OPEN SOURCE-SOFTWARE PRODUCEREN

Met succes een open source-softwareproject runnen

Karl Fogel

OPEN SOURCE-SOFTWARE PRODUCEREN

Met succes een open source-softwareproject runnen

Karl Fogel

Dit boek is opgedragen aan twee dierbare vrienden zonder wie het niet tot stand zou zijn gekomen: Karen Underhill en Jim Blandy.

Colofon

Deze vertaling van het boek Producing Open Source Software is mogelijk gemaakt door Stichting Kennisnet, SURFfoundation, SURFdiensten en SURFnet.

Er zijn geen inhoudelijke wijzigingen in de tekst van de auteur aangebracht, met uitzondering van het onderdeel 'vrij' versus 'open' in Hoofdstuk 1. De tekst is hier ingekort, omdat het beschreven probleem in het Engels wel, maar in het Nederlands niet van toepassing is. Dit is gedaan in overleg met en met toestemming van de auteur.

Oorspronkelijke titel
Producing Open Source Software
How to Run a Successful Free Software Project
Auteur *Karl Fogel*

Vertaling *Tekom Vertalers BV.*
Ontwerp en opmaak *Vrije Stijl grafisch ontwerp*
Beeld *MegapixelMedia, Istockphoto.com*
Druk *Drukkerij Libertas*

INHOUDSOPGAVE

VOORWOORD

Als je verandering wilt in de wereld, begin dan bij jezelf. Onder dat motto ben ik dit jaar begonnen met de uitvoering van het politiek breed gesteunde Actieplan Nederland Open in Verbinding. Overheden en de publieke sector zijn nu verplicht om open standaarden te gebruiken en worden sterk aangeraden om voor open source software te kiezen.

Het gebruik van open source software stimuleert innovatie, omdat softwareontwikkelaars kunnen voortbouwen op software die eerder is ontwikkeld door concurrenten. Daarnaast maakt het klanten minder afhankelijk van leveranciers, omdat ze de vrijheid hebben om bestaande software uit te breiden en niet vastzitten aan jarenlange contracten. Ten slotte zorgt de beschikking over de broncode ervoor dat de toepassing in de toekomst altijd gebruikt en uitgebreid kan worden, ongeacht of de softwareleverancier nog bestaat.

In de praktijk blijkt echter dat onbekendheid met dit type software potentiële klanten afschrikt. Elk softwareproject kent zijn uitdagingen en die van open source zijn nu eenmaal anders dan die van gesloten software. Karl Fogel vertelt in zijn boek precies wat je moet doen om deze uitdagingen te overwinnen.

Ik ben dan ook blij met het initiatief van SURF en Kennisnet om het boek van deze ervaren softwareontwikkelaar in het Nederlands te vertalen en deze kennis vrij beschikbaar te stellen via internet. De kracht van het boek is, wat mij betreft, vooral dat het zo leesbaar is. Doordat het de voor- en nadelen van open source softewareontwikkeling zo helder uitlegt, is het een goede remedie tegen drempelvrees.

Ik weet daarom zeker dat dit boek u zal aanmoedigen om open source software te maken en te gebruiken.

Frank Heemskerk
Staatssecretaris van Economische Zaken

EEN WOORD VOORAF

WAAROM SCHRIJF IK DIT BOEK?

Op feestjes kijken mensen mij niet langer met een niet-begrijpende blik aan als ik zeg dat ik open source-software schrijf. “Oh, ja, open source, zoals Linux?” zeggen ze. Ik knik geestdriftig ter instemming. “Ja, inderdaad! Dat is wat ik doe.” Het is prettig geen compleet randverschijnsel meer te zijn. In het verleden was de volgende vraag altijd erg voorspelbaar: “Maar hoe verdien je je boterham daarmee?” Ik antwoordde dan met een samenvatting van de logica van open source: dat er organisaties zijn in wiens belang het is dat er bepaalde software bestaat, maar dat het niet noodzakelijk is er exemplaren van te verkopen. Ze willen gewoon zeker weten dat de software beschikbaar is en onderhouden wordt, meer als gereedschap dan als verhandelbaar product.

De laatste tijd gaat de vervolgvraag echter niet altijd over het geld. De business case voor open source software[1] is niet meer zo geheimzinnig als voorheen en veel niet-programmeurs begrijpen inmiddels (of zijn op zijn minst niet meer verbaasd) dat er mensen zijn die hier een fulltime baan aan hebben. In plaats daarvan is de vraag die ik steeds vaker hoor “Oh, en hoe gaat dat in zijn werk?”

Ik had daar geen bevredigend antwoord op en hoe meer ik m’n best deed om een antwoord te vinden, des te meer ik mij realiseerde hoe ingewikkeld dit onderwerp eigenlijk is. Een open source-softwareproject runnen verschilt nogal van leiding geven aan een bedrijf. Stel je voor dat je constant over de hoedanigheid van je product moet onderhandelen met een groep vrijwilligers van wie je de meeste nog nooit hebt ontmoet! Om allerlei redenen lijkt het evenmin op het leiden van een traditionele non-profitorganisatie of een overheid. Er zijn overeenkomsten met al deze zaken, maar ik kom langzaam maar zeker tot de conclusie dat open source-software een geval apart is. Er zijn veel zaken waarmee het vergeleken kan worden, maar geen enkele is echt gelijkwaardig. En inderdaad, zelfs de veronderstelling dat open source-softwareprojecten ‘gerund’ kunnen worden, doet de werkelijkheid enigszins geweld aan. Een open source-softwareproject kan worden gestart en worden beïnvloed door geïnteresseerde partijen, vaak tamelijk sterk. Maar de waardevolle eigenschappen ervan kunnen niet het bezit worden van een enkele eigenaar en zo lang er mensen zijn die ergens, waar dan ook, geïnteresseerd zijn in het voortzetten van het project, kan het niet eenzijdig stopgezet worden. Iedereen heeft onbegrensde macht en iedereen heeft geen macht. Ingrediënten voor een interessante dynamiek.

En dat is waarom ik dit boek wilde schrijven. Open source-softwareprojecten hebben zich ontwikkeld tot een specifieke cultuur, een levensopvatting waarbij de vrijheid om software datgene te laten doen wat je wilt dat software moet doen een dogma is. Toch is het resultaat van deze vrijheid niet een groepje losstaande individuen die hun eigen weg gaan met de code, maar enthousiaste samenwerking. En inderdaad, het vermogen om samen te werken op zich is een van de meest gewaardeerde vaardigheden in de wereld van de open source-software. Om deze projecten te kunnen managen moet iemand deel gaan uitmaken van een soort opgezwollen samenwerking, waar iemands vermogen om niet alleen samen te werken met een ander maar ook nieuwe manieren te bedenken voor deze samenwerking kan resulteren in een tastbaar voordeel voor de software. Dit boek probeert de technieken te beschrijven waarmee dit gedaan kan worden. Het is in geen geval compleet, maar het is in ieder geval een begin.

Goede open source-software is een waardig doel op zich en ik hoop dat lezers die zoeken naar een manier dit te realiseren blij zullen zijn met wat ze hier vinden. Maar daarnaast hoop ik iets over te brengen van het grote plezier dat je kunt beleven aan het werken met een gemotiveerd team van open source-ontwikkelaars en aan de interactie met gebruikers op de heerlijk directe manier die open source stimuleert. Deelnemen in een succesvol open source-softwareproject is leuk; uiteindelijk is dat wat het hele systeem in stand houdt.

WIE ZOU DIT BOEK MOETEN LEZEN?

Dit boek is bedoeld voor ontwikkelaars en managers van open software die overwegen een open source-project te beginnen, en voor degenen die al een project gestart zijn en zich afvragen hoe ze verder moeten. Het kan ook handig zijn voor mensen die alleen willen participeren in een open source-project, maar dit nooit eerder gedaan hebben.

De lezer hoeft geen programmeur te zijn, maar moet wel de technische basisconcepten van software kennen, zoals broncode, compilers en patches. Ervaring met open source-software, als gebruiker of ontwikkelaar, is niet nodig. Mensen die al aan open source-softwareprojecten hebben meegewerkt, zullen sommige delen van het boek waarschijnlijk als een open deur beschouwen en deze waarschijnlijk willen overslaan. Omdat er enorm grote verschillen kunnen zijn wat betreft de ervaring die de lezers hebben, heb ik geprobeerd de secties van duidelijke koppen te voorzien en te vermelden wanneer iets overgeslagen kan worden door degenen die al bekend zijn met deze materie.

BRONNEN

Veel van het basismateriaal voor dit boek heb ik verzameld gedurende de vijf jaar dat ik aan het Subversion-project (http://subversion.tigris.org/) gewerkt heb. Subversion is een open source-versiecontrolesysteem, dat helemaal nieuw geschreven is. Het is bedoeld om CVS te vervangen als het door de open source-gemeenschap geprefereerde versiecontrolesysteem. Het project is begin 2000 gestart door mijn werkgever, CollabNet (http://www.collab.net/). Godzijdank begreep CollabNet meteen vanaf het begin hoe dit project moest worden gerund als een werkelijk coöperatief en breed gedragen project. In het begin boden veel vrijwillige ontwikkelaars zich aan; op dit moment zijn er zo'n vijftig ontwikkelaars werkzaam aan het project, van wie slechts enkele medewerkers van CollabNet zijn.

Subversion is in veel opzichten een klassiek voorbeeld van een open source-project en ik voelde me er uiteindelijk veel meer door geïnspireerd dan ik oorspronkelijk verwacht had. Dit was gedeeltelijk een kwestie van gemak: telkens wanneer ik een voorbeeld nodig had van een bepaald fenomeen, schoot me er meestal meteen een van Subversion te binnen. Maar het was ook een kwestie van verificatie. Alhoewel ik op verschillende manieren betrokken ben bij andere open source-softwareprojecten en met vrienden en kennissen praat die betrokken zijn bij veel andere projecten, realiseerde ik me snel dat beweringen geverifieerd moeten worden op feitelijke juistheid als ze zwart op wit komen te staan. Ik wilde geen uitspraken doen over gebeurtenissen betreffende andere projecten die alleen gebaseerd zijn op wat ik kon lezen in de archieven van hun openbare mailinglijsten. Als iemand dat met Subversion zou doen, dan weet ik dat hij de helft van de tijd goed zou zitten en de andere helft fout. Dus wanneer ik inspiratie of voorbeelden gebruikte van een ander project waar ik geen directe ervaring mee had, dan probeerde ik eerst te praten met een informant van dat project, iemand waarvan ik zeker wist dat hij me kon vertellen wat er echt speelde.

Ik heb de afgelopen vijf jaar aan Subversion gewerkt, maar ik doe dit werk al twaalf jaar. Andere projecten die dit boek hebben beïnvloed zijn:

- Het *GNU Emacs text editor-project* van de Free Software Foundation, waarvan ik van een paar kleine onderdelen het onderhoud verricht.
- *Concurrent Versions System (CVS)*, waaraan ik in 1994–1995 intensief gewerkt heb met Jim Blandy, maar waarbij ik sindsdien slechts met tussenpozen betrokken ben.
- De verzameling open source-projecten die bekend staat als de *Apache Software Foundation* en in het bijzonder de *Apache Portable Runtime (APR)- en Apache HTTP-server.*
- *OpenOffice.org*, de *Berkeley Database* van Sleepycat en de *MySQL Database*. Ik ben niet persoonlijk betrokken geweest bij deze projecten, maar ik heb ze geobserveerd en, in sommige gevallen, gesproken met de mensen daar.
- *GNU Debugger (GDB)* (idem).
- *Het Debian-project* (idem).

Dit is uiteraard geen volledige lijst. Zoals de meeste open source-programmeurs houd ik op afstand verschillende projecten in de gaten, gewoon om een idee te hebben van de stand van zaken. Ik zal ze niet allemaal noemen, maar ze worden in de tekst vermeld wanneer dat van toepassing is.

DANKBETUIGING

Het schrijven van dit boek duurde vier keer zo lang als ik aanvankelijk dacht. Vaak voelde het als het zwaard van Damocles dat, waar ik ook was, boven mijn hoofd hing. Zonder de hulp van vele mensen zou ik niet in staat geweest zijn het boek af te maken en tegelijkertijd mijn verstand te bewaren.

Andy Oram, mijn redacteur bij O'Reilly, was de droom van elke schrijver. Behalve grondige kennis van de branche (hij opperde veel van de onderwerpen), bezit hij de zeldzame gave om te begrijpen wat je wil zeggen en je te helpen de juiste manier te vinden om het te zeggen. Het was een eer met hem te mogen werken. Mijn dank gaat ook uit naar Chuck Toporek, die dit plan direct naar Andy stuurde.

Brian Fitzpatrick heeft de revisie gedaan van bijna al het materiaal dat ik schreef. Dat heeft het boek niet alleen beter gemaakt, maar me ook aan het schrijven gehouden op de momenten dat ik overal elders wilde zijn behalve achter mijn computer. Ben Collins-Sussman en Mike Pilato hielde eveneens de voortgang in de gaten en waren altijd bereid om - soms zeer uitvoerig - te discussiëren, ongeacht het onderwerp dat ik die week onder handen had. Ze merkten het ook als mijn tempo omlaag ging en plaagden me er zo nodig op een vriendschappelijke manier mee. Bedankt jongens.

Biella Coleman schreef haar proefschrift in dezelfde periode dat ik dit boek schreef. Ze weet wat het is om te gaan zitten en iedere dag te schrijven en ze fungeerde voor mij als een inspirerend voorbeeld en een gewillig oor. Ze heeft ook een fascinerende antropologische kijk op de open source-softwarebeweging en ze heeft me ideeën en referenties aan de hand gedaan die ik voor het boek kon gebruiken. Alex Golub, nog een antropoloog met een voet in de open source-softwarewereld, die ook op dat moment zijn proefschrift schreef, was daarvoor al buitengewoon behulpzaam geweest.

Micah Anderson maakte ondanks zijn eigen schrijfklus altijd een opgewekte indruk, hetgeen op een misselijk en jaloers makende manier inspirerend was, en stond altijd klaar met zijn vriendschap, een gesprek en (in ieder geval één keer) technische ondersteuning. Bedankt Micah!

Jon Trowbridge en Sander Striker gaven zowel bemoedigende als concrete hulp. Dankzij hun brede ervaring op het gebied van open source-software leverde materiaal op waar ik op geen enkele andere manier aan had kunnen komen.

Mijn dank ook aan Greg Stein, niet alleen voor zijn vriendschap en aanmoediging op het juiste moment, maar ook voor het feit dat hij aan het Subversion-project liet

zien hoe belangrijk regelmatige evaluatie van de code is bij het opzetten van een programmeergemeenschap. Mijn dank gaat ook uit naar Brian Behlendorf, die het belang van openbare discussies tactvol in onze hoofden gestampt heeft. Ik hoop dat dit in het hele boek naar voren komt.

Dank aan Benjamin 'Mako' Hill en Seth Schoen, voor diverse gesprekken over open source-software en de daarbij horende politiek; aan Zack Urlocker en Louis Suarez-Potts voor het vrijmaken van tijd voor een interview ondanks hun volle agenda; aan Shane op de Slashcode-lijst voor zijn toestemming om zijn posts te mogen citeren en aan Haggen So voor zijn enorme behulpzaamheid bij het vergelijken van canned hosting-sites.

Mijn dank aan Alla Dekhtyar, Polina en Sonya voor hun niet-aflatende en geduldige aanmoedigingen. Ik ben erg blij dat ik onze avonden niet langer vroeg hoef te beëindigen (of beter gezegd, zonder succes vroeg probeer te beëindigen) om naar huis te gaan en aan 'Het Boek' te werken.

Mijn dank aan Jack Repenning voor zijn vriendschap, gesprekken en koppige weigering om een makkelijke foutieve analyse te accepteren als er een moeilijkere goede voorhanden is. Ik hoop dat er iets van zijn lange ervaring met zowel het ontwikkelen van software als de software-industrie aan dit boek is blijven plakken.

CollabNet was uitzonderlijk royaal en stond me een flexibel rooster toe om te schrijven en klaagde niet toen het veel langer ging duren dan oorspronkelijk de bedoeling was. Ik ken de fijne kneepjes niet van hoe management tot zulke beslissingen komt, maar ik vermoed dat Sandhya Klute en later Mahesh Murthy er iets mee te maken hebben gehad. Mijn dank aan beiden.

Het hele Subversion-ontwikkelingsteam is de afgelopen vijf jaar een grote bron van inspiratie voor me geweest en veel van wat in het boek staat, heb ik geleerd door met hen samen te werken. Ik zal ze hier niet allemaal met naam en toenaam bedanken, omdat het er gewoon teveel zijn, maar ik verzoek elke lezer die iemand die aan Subversion meewerkt tegen het lijf loopt om deze onmiddellijk een drankje naar zijn of haar keuze aan te bieden. Ik ben dat zeer beslist van plan.

Vaak zeurde ik tegen Rachel Scollon over de stand van zaken van het boek. Ze was altijd bereid te luisteren en speelde het op de een of andere manier klaar om het probleem kleiner te laten lijken dan het leek voordat we praatten. Dat heeft enorm geholpen, bedankt.

Mijn dank aan (alweer) Noel Taylor. Hij zal zich, gezien al mijn geklaag de vorige keer, wel afgevraagd hebben waarom ik per se nog een boek wilde schrijven. Dankzij zijn vriendschap en leiding van Golosá is er muziek en kameraadschap in mijn leven gebleven, zelfs in de drukste tijden. Mijn dank ook aan Matthew Dean en Dorothea Samtleben, vrienden en lankmoedige muziekpartners, die begripvol bleven ook al stapelden de smoesjes waarom ik niet geoefend had zich op. Megan Jennings steunde me voortdurend en was oprecht geïnteresseerd in het onderwerp, ook al was ze er niet mee vertrouwd. Een geweldig elixer voor de onzekere schrijver. Bedankt, maatje!

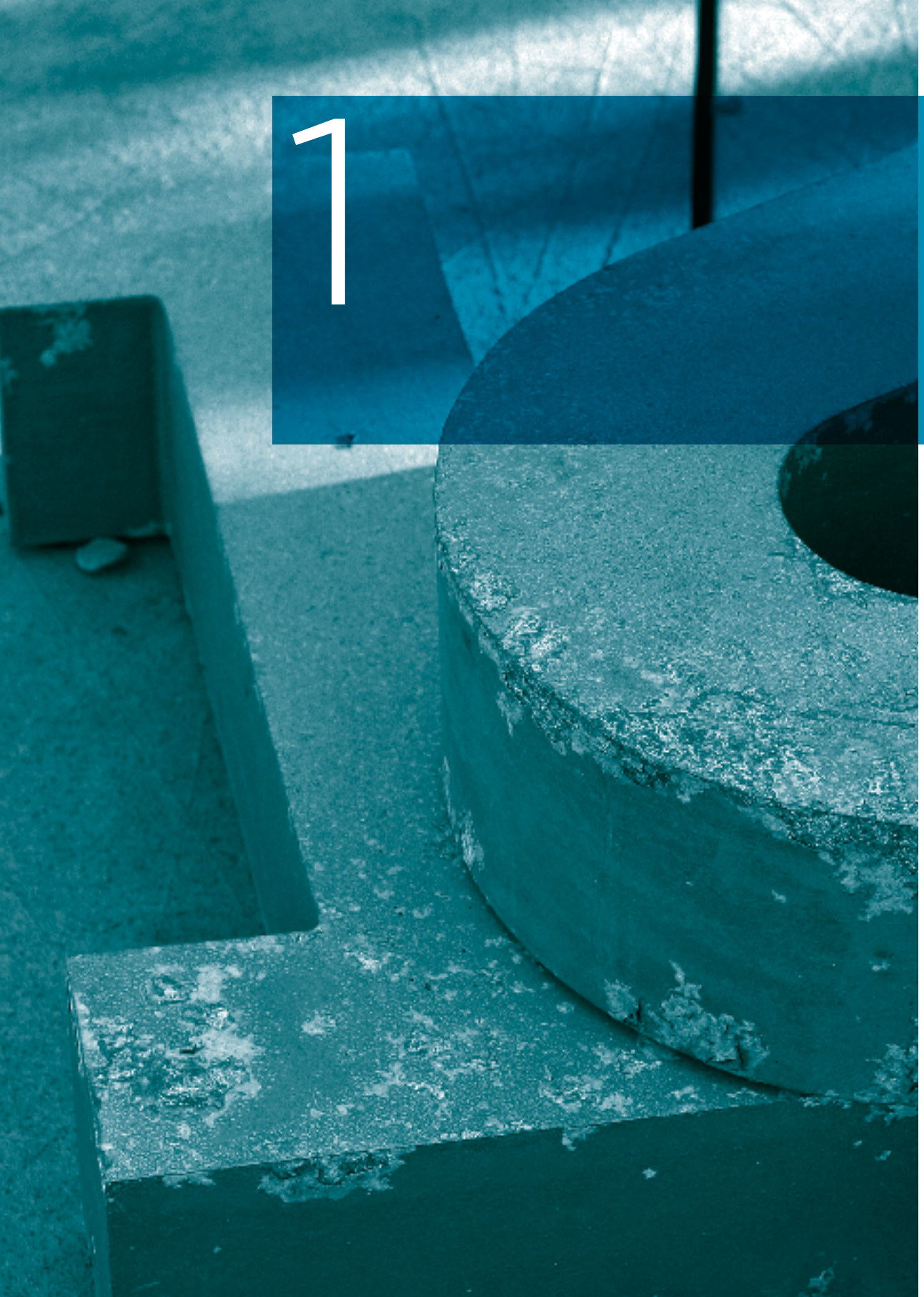
Ik beschikte voor dit boek over vier goed geïnformeerde en toegewijde redacteurs: Yoav Shapira, Andrew Stellman, Davanum Srinivas en Ben Hyde. Als ik in staat was geweest om al hun uitstekende suggesties over te nemen, dan was dit een beter boek geworden. Helaas kon ik door tijdsgebrek alleen de krenten uit de pap pikken, maar de verbeteringen zijn al met al toch aanzienlijk. De resterende fouten komen geheel op mijn conto.

Mijn ouders, Frances en Henry, hebben me zoals altijd erg gesteund en omdat dit boek minder technisch is dan het voorgaande hoop ik dat ze het wat leesbaarder vinden.

Ten slotte wil ik degenen bedanken aan wie dit boek is opgedragen, Karen Underhill en Jim Blandy. Karens vriendschap en welwillendheid betekenen alles voor me, niet alleen tijdens het schrijven van dit boek, maar sowieso de afgelopen zeven jaar. Ik had het boek eenvoudigweg niet af kunnen krijgen zonder haar hulp. Hetzelfde geldt voor Jim, een ware vriend en een hackers hacker, die me de eerste beginselen bijbracht van open source-software, op een manier zoals een vogel een vliegtuig iets zou leren over vliegen.

DISCLAIMER

De gedachten en opvattingen die tot uitdrukking gekomen zijn in dit boek zijn geheel de mijne. Ze geven niet per definitie de opvattingen weer van CollabNet of van het Subversion-project.



1

INLEIDING

De meeste open source-softwareprojecten mislukken.

Over de mislukkingen horen we meestal weinig. De meeste aandacht gaat nu eenmaal uit naar geslaagde projecten. En het aantal open source-softwareprojecten is zo groot, dat er ondanks hun lage slagingskans heel veel projecten[2] overblijven die opvallen. Een andere reden waarom we weinig over de mislukkingen horen, is dat een mislukking geen gebeurtenis is. Projecten mislukken doorgaans niet op één bepaald moment; programmeurs verliezen geleidelijk hun interesse en gaan iets anders doen. Er kán wel een moment komen waarop een project voor het laatst wordt bijgewerkt, maar degenen die dat deden, wisten op dat moment doorgaans niet dat het de laatste veranderingen betrof. Het is zelfs nauwelijks vast te stellen wanneer een project nu precies afgelopen is. Is het einde gekomen als er zes maanden niemand aan heeft gewerkt? En wat als er naast de ontwikkelaars zelf geen gebruikers bijkomen? En wat als de ontwikkelaars het ene project verlaten voor een ander project met hetzelfde doel — en wat als ze hun eerdere bijdragen in dat andere project inbrengen? Is het eerste project dan afgelopen of heeft het zich alleen verplaatst? Dergelijke complexe vraagstukken maken het onmogelijk om het aantal mislukte projecten exact vast te stellen. Anekdotisch bewijs op grond van meer dan tien jaar open source, wat snuffelen op SourceForge.net en een beetje googelen kunnen echter tot slechts één conclusie leiden: de kans op mislukking is extreem groot, waarschijnlijk in de orde van 90–95%. En dit percentage wordt nog hoger als we de actieve projecten meerekenen die niet goed functioneren: dergelijke projecten leveren *wél* uitvoerbare code op maar zijn niet populair of hun vooruitgang is te langzaam of te onregelmatig.

Dit boek gaat over het voorkomen van mislukkingen. We richten ons niet alleen op de oplossingen, maar ook op wat er onderweg verkeerd kan gaan zodat u problemen tijdig kunt herkennen en corrigeren. Mijn hoop is, dat u na het lezen van dit boek niet alleen over een schat aan technieken beschikt om de gangbare valkuilen van het ontwikkelen van open source te vermijden, maar dat u ook weet hoe u moet omgaan met het onderhoud van een succesvol, groeiend project. Succes is geen kwestie van optellen en aftrekken, en dit boek gaat niet over het behalen van overwinningen of het verslaan van de concurrentie. Want een belangrijk aspect van

het uitvoeren van open source-projecten, is een goede samenwerking met andere, gerelateerde projecten. Op de lange termijn draagt elk succesvol project bij aan de belangen van vrije software in het algemeen.

Het is verleidelijk om te denken dat open source-softwareprojecten om dezelfde redenen falen als propriëtaire-softwareprojecten. Vrije software heeft zeker niet het alleenrecht op onrealistische verwachtingen, vage specificaties, slecht gebruik van middelen, tijdgebrek en alle andere bekende struikelblokken van de software-industrie. Over deze onderwerpen zijn boeken volgeschreven, en ik zal proberen dat hier niet nogmaals te doen. Ik wil juist proberen om de problemen te beschrijven die specifiek zijn voor vrije software. Want als open source-softwareprojecten vastlopen, gebeurt dat vaak omdat de ontwikkelaars (of de managers) geen rekening hebben gehouden met de unieke problemen van het ontwikkelen van open source-software, terwijl ze misschien wél goed weten wat de mogelijk obstakels zijn bij het ontwikkelen van closed source-software.

Eén van de grootste fouten is, dat er onrealistische verwachtingen bestaan over de voordelen van open source zelf. Een open licentie is geen garantie dat er ineens grote groepen ontwikkelaars aan uw project gaan werken, en het publiceren van de broncode van een slechtlopend project is geen wondermiddel tegen alle ontwikkelingsproblemen. Integendeel: het openstellen van een project voegt allerlei moeilijkheden toe en kan op de korte termijn juist *meer* gaan kosten dan een project inhouse te houden. Openbaar maken betekent ook dat u de code inzichtelijk moet maken voor volslagen vreemden, dat u een ontwikkelingswebsite moet opzetten, e-maillijsten moet bijhouden en -vaak voor het eerst- documentatie moet schrijven. Dat alles is veel werk. En *als* er al ontwikkelaars interesse hebben in uw project, dan moet u de eerste tijd veel vragen beantwoorden voordat u resultaten ziet. Ontwikkelaar Jamie Zawinski zegt het volgende over de moeizame aanloop van het Mozilla-project:

Open source werkt wel, maar het is zeker geen wondermiddel. Als er al lessen te leren zijn, besef dan dat slechtlopende projecten niet ineens op wonderbaarlijke wijze opbloeien door het toverwoord 'open source' uit te spreken. Software schrijven is moeilijk, punt uit. Er zijn geen eenvoudige oplossingen.
(bron http://www.jwz.org/gruntle/nomo.html)

Een bijkomende fout is het negeren van de presentatie en de verpakking. Men denkt dat dit later nog wel kan, als het project eenmaal goed op weg is. Presentatie en verpakking zijn veelomvattende taken, met als doel om de toegankelijkheid te verbeteren. Om het project aantrekkelijk te maken voor nieuwkomers, moet u documentatie schrijven voor de gebruikers en de ontwikkelaars, een informatieve website opzetten, het compileren en installeren van de software zo veel mogelijk automatiseren enz. Helaas beschouwen programmeurs dit werk als minder belangrijk dan het schrijven van de code zelf. Hier zijn verschillende redenen voor. Ten eerste wordt het als overbodig huiswerk werk gezien, omdat het vooral voordelen oplevert voor mensen die het project niet kennen. De ontwikkelaars zelf hebben zo'n verpakking immers niet nodig. Zij weten al hoe ze de software moeten installeren, beheren en gebruiken, omdat ze hem zelf hebben geschreven. Ten tweede vereisen presentatie en verpakking veelal compleet andere vaardigheden dan het schrijven van code. En mensen hebben nu eenmaal de neiging om zich te concentreren op waar ze goed in zijn, zelfs als het project bij iets anders meer baat zou hebben. Hoofdstuk 2, *Aan de slag* gaat dieper in op presentatie en verpakking, en legt uit waarom dit vanaf het begin één van de prioriteiten moet zijn.

Een volgende denkfout is, dat open source vrijwel geen projectmanagement vereist, of omgekeerd, dat de gebruikte beheermethoden voor interne ontwikkeling even goed zullen werken voor een open source-project. Het is niet altijd goed zichtbaar hoe open source-projecten worden beheerd, maar bij succesvolle projecten is er achter de schermen meestal wel degelijk sprake van een vorm van beheer. Een klein gedachte-experiment zegt genoeg. In een open source-project werkt een willekeurige groep — doorgaans zéér eigenzinnige — programmeurs samen, die elkaar waarschijnlijk nooit hebben ontmoet en die allen verschillende persoonlijke drijfveren hebben om aan het project mee te werken. Als gedachte-experiment vraag ik u om te denken aan wat er gebeurt met zo'n groep *zonder* enige vorm van management. Het zou een regelrecht wonder zijn als zo'n groep niet uit elkaar zou vallen. De meeste dingen gaan niet vanzelf, hoe graag we dat soms ook willen. Het soort management dat is vereist, is echter informeel, subtiel en onopvallend, terwijl het wel degelijk zeer actief kan zijn. Want het enige dat een groep van ontwikkelaars bij elkaar houdt, is hun gedeelde overtuiging dat ze samen meer kunnen bereiken dan individueel. De beheerdoelstelling richt zich dan ook voornamelijk op het in stand houden van deze overtuiging, door afspraken te maken over de communicatie, door te zorgen dat waardevolle programmeurs niet te lijden hebben van persoonlijke wrijvingen, en door in het algemeen een zodanig project te creëren dat ontwikkelaars eraan willen blijven bijdragen. De specifieke technieken waarmee u deze doelstelling kunt realiseren, komen later in dit boek aan de orde.

Tot slot is er een algemene categorie problemen die we het predicaat 'cultuurverschillen' zullen geven. Tot tien jaar geleden, en zelfs tot vijf jaar geleden, bestonden er nauwelijks mondiale opvattingen over de vrijesoftwarecultuur, maar dat is veranderd. Langzaam heeft zich een cultuur afgetekend die zeker niet eenduidig is — er heersen net zo veel interne discussies en meningsverschillen als in andere geografisch gebonden culturen — maar er is wel een gemeenschappelijke basis. De meeste geslaagde open source-projecten vertonen één of meer van de volgende basiskenmerken. Ze belonen bepaald gedrag en bestraffen juist ander gedrag; ze creëren een sfeer van ongedwongen samenwerking, soms ten koste van de centrale aansturing; en hun opvattingen over wat onbeschoft en beleefd is kan soms nogal afwijken van de algemeen gangbare gedragsregels. Het is een belangrijk gegeven dat bijna alle ervaren deelnemers deze uitgangspunten kennen en accepteren, zodat er min of meer consensus bestaat over het verwachte gedrag. Mislukte projecten wijken doorgaans in belangrijke mate af van deze kernwaarden, vaak onbedoeld, zodat er geen consensus is over welk gedrag normaal is en welk niet. Als er dan problemen ontstaan, kan de situatie snel uit de hand lopen omdat de deelnemers niet kunnen terugvallen op een gedeelde cultuur voor het bijleggen van conflicten.

Dit boek is een praktische handleiding, geen antropologische studie of geschiedenisboek. Enige achtergrondkennis over de oorsprong van de huidige vrijesoftware-

cultuur is echter onontbeerlijk om praktische adviezen te kunnen geven. Wie deze cultuur begrijpt, komt ver in de wereld van open source met zijn vele lokale variaties en gebruiken, en zal overal van harte welkom zijn als deelnemer. Wie de cultuur echter niet begrijpt, zal het verrassend lastig vinden om een project te organiseren of om er aan deel te nemen. En aangezien het aantal ontwikkelaars dat zich op vrije software stort enorm blijft stijgen, vallen er helaas veel mensen in deze laatste categorie — dit is in belangrijke mate een cultuur van nieuwe immigranten en zal dat nog wel even blijven. Als u denkt dat u bij de laatstgenoemde categorie hoort, biedt het volgende hoofdstuk de nodige achtergrond voor de discussies die later zullen volgen, zowel in dit boek als op internet. (Als u daarentegen al enige tijd met open source werkt, heeft u waarschijnlijk al veel over de ontstaansgeschiedenis gehoord en kunt u het volgende hoofdstuk wel overslaan.)

1.1 GESCHIEDENIS

Het met elkaar delen van software is al zo oud als er software bestaat. In de begindagen van het computertijdperk verdienden de fabrikanten hun geld vooral met hardware en werd software niet gezien als een product waarmee je geld kon verdienen. De meeste klanten van het eerste uur waren wetenschappers en technici, die zelf in staat waren de software die bij het apparaat geleverd werd aan te passen. De verbeteringen in de software werden niet alleen naar de fabrikant gestuurd, maar soms ook direct naar andere eigenaars van dezelfde machines. De meeste fabrikanten hadden hier geen probleem mee of moedigden dit juist aan. In hun ogen zorgden deze verbeteringen in de software, ongeacht wie daarvoor zorgde, alleen maar voor meer klanten voor hun machines.

Hoewel deze vroege periode veel lijkt op de vrijesoftwarecultuur van vandaag, zijn er twee essentiële verschillen. Ten eerste was er nauwelijks standaardhardware. De ontwikkelingen op het gebied van computers gingen razendsnel, maar door alle verschillen was niets compatibel. Met als gevolg dat de software die voor de ene machine was geschreven, meestal niet werkte op andere machines. De programmeurs specialiseerden zich doorgaans op één bepaalde architectuur of hardwarefamilie, terwijl de meeste programmeurs zich tegenwoordig een programmeertaal eigen maken die onafhankelijk is van de computerhardware waar ze mee werken. Hoe meer kennis en ervaring er werd opgedaan met een bepaald type computer, des te aantrekkelijker het voor hen en hun collega's werd om dat type computer te blijven gebruiken. De fabrikant had er dus belang bij dat de programma's voor hun machines zo veel mogelijk mensen bereikten.

Ten tweede bestond internet nog niet. Hoewel er minder wettelijke beperkingen waren op verspreiding van software dan vandaag de dag, waren er wel meer technische beperkingen: het kostte relatief veel moeite om gegevens van de ene plaats naar de andere te transporteren. Er bestonden wel enkele lokale netwerken, waarmee werknemers van één en hetzelfde onderzoekslaboratorium of bedrijf hun gegevens konden delen. Maar voor grootschalige verspreiding over een bredere doelgroep bestonden er flinke barrières. Vaak werden deze barrières *toch* overwonnen. Verschillende groepen onderhielden onderling contact en stuurden elkaar schijfjes of tapes met de post. Ook fungeerden sommige fabrikanten zelf als centraal distributiecentrum voor patches. Wat hielp, was dat veel van de oorspronkelijke computergebruikers aan universitaire instellingen werkten, waar het gebruikelijk is om kennis met elkaar te delen. Maar de fysieke beperkingen van de toenmalige datatransmissie vormden een natuurlijke barrière tegen het delen van software, een barrière die rechtevenredig was met de afstand (fysiek of qua organisatie) die de software moest afleggen. Het grootschalig, moeiteloos delen van gegevens zoals we dat nu kennen, bestond in die tijd gewoonweg niet.

De opkomst van propriëtaire en vrije software

Toen de bedrijfstak volwassener werd, vonden gelijktijdig een aantal veranderingen plaats die onderling verband hielden. De aanvankelijke wildgroei aan hardwarearchitectuur resulteerde gaandeweg in enkele duidelijke winnaars — op grond van hun superieure technologie, hun superieure marketing of een combinatie van beide. Tegelijkertijd werden, niet geheel toevallig, de zogenaamde 'hogere' programmeertalen ontwikkeld waarmee elk programma maar één keer hoeft te worden geschreven, in één taal, waarna het automatisch kan worden vertaald ('gecompileerd') om op verschillende typen computers te werken. De hardwarefabrikanten zagen al snel in wat de gevolgen hiervan zouden zijn: Klanten kunnen grote softwareprojecten opzetten zonder zich aan één bepaalde hardwarearchitectuur te binden. Toen bovendien de verschillen in prestaties tussen de verschillende computersystemen minder werden omdat de minder efficiënte computertypen het onderspit delfden, zag het er naar uit dat een fabrikant die alleen hardware verkocht een onzekere toekomst tegemoet ging. Puur rekenvermogen was niet langer het unieke verkoopargument en software begon het verschil uit te maken. Het leek een goede strategie om ook software te gaan verkopen of om op zijn minst de software in combinatie met de hardware te leveren.

Dit betekende dat fabrikanten de auteursrechten op hun code beter moesten gaan beschermen. Want als gebruikers de programmacode gewoon onderling zouden blijven delen en aanpassen, zouden er door derden verbeteringen kunnen worden bedacht die de verkoper ook als 'toegevoegde waarde' had kunnen verkopen. Of erger nog, gedeelde programmacode zou in handen van de concurrentie kunnen vallen. Het ironische is dat dit zich allemaal afspeelde op het moment dat internet zich begon te ontwikkelen. Net op het moment dat de technische barrières tegen het delen van software eindelijk wegvielen, zorgden veranderingen in de computerindustrie ervoor dat het economisch onwenselijk werd om dit te doen, tenminste vanuit het oogpunt van de afzonderlijke bedrijven. De leveranciers werden streng en ontzegden de gebruikers de toegang tot de programmacode op hun computers, of ze dwongen geheimhouding af, hetgeen het delen van de code feitelijk onmogelijk maakte.

Bewuste weerstand

Terwijl het steeds moeilijker werd om programmacode ongehinderd te delen, besloot ten minste één programmeur om zich hiertegen te verzetten. In de jaren zeventig en begin jaren tachtig was Richard Stallman werkzaam in het *Artificial Intelligence Lab* van het *Massachusetts Institute of Technology*, tijdens wat achteraf de ideale tijd en plaats is gebleken voor het delen van programmacode. Het AI-lab

kende veel echte 'hackers'[3] die alle verbeteringen aan het systeem met elkaar deelden, hetgeen ook van ze werd verwacht. Later schreef Stallman hierover:

We noemden onze software geen 'free software'*, omdat die term nog niet bestond; maar het was het wel. Als er mensen van andere universiteiten of bedrijven bij ons aanklopten om een programma te gebruiken, dan kon dat gewoon. En je ergens een nieuw en interessant programma zag, dan kon je de programmeur altijd om de broncode vragen, zodat je die kon lezen, aanpassen of zelfs gedeeltelijk hergebruiken voor je eigen programma.* **(bron: http://www.gnu.org/gnu/thegnuproject.html)**

Dit programmeursparadijs rond Stallman stortte kort na 1980 in, toen de veranderende tijdgeest binnen de computerindustrie ook het AI-lab bereikte. Op een gegeven moment nam een klein IT-bedrijfje veel van de programmeurs van het lab in dienst om aan een soortgelijk besturingssysteem te gaan werken als in het lab, maar nu onder een gesloten licentie. En in dezelfde periode schakelde het AI-lab over op nieuwe apparatuur met een propriëtair besturingssysteem.

Stallman begreep welk mechanisme achter de gebeurtenissen zat:
De moderne computers van die tijd, zoals de VAX of de 68020, hadden elk hun eigen besturingssysteem, maar dat was geen vrije software. Er moest zelfs een geheimhoudingsverklaring worden getekend voordat er een uitvoerbaar bestand werd verstrekt.
Dat hield in, dat het elke computergebruiker in principe werd verboden om andere gebruikers te helpen. Samenwerken was niet langer toegestaan. De makers van propriëtaire software redeneerden als volgt: 'Als u iets deelt met uw buren, dan bent u een softwarepiraat. Als u veranderingen wilt, smeek ons dan maar om die te maken.'

Stallman besloot zich tegen deze trend te gaan verzetten. In plaats van te blijven werken in het uitgeklede AI-lab of programmeur te worden bij een van de nieuwe bedrijfjes waar zijn werk achter slot en grendel zou verdwijnen, nam hij ontslag en richtte het *GNU-project* en de *Free Software Foundation* (FSF) op. Het doel van GNU[4] was het ontwikkelen van een volledig vrij en open besturingssysteem met allerlei toepassingen waarin gebruikers zelf aanpassingen konden blijven maken en met elkaar mochten delen. Hij probeerde in feite opnieuw te creëren wat er in het AI-lab verloren was gegaan, maar dan op mondiale schaal en zonder de kwetsbaarheden die de cultuur van het AI-lab zo snel de das om hadden gedaan.

Naast het werk aan zijn nieuwe besturingssysteem, ontwierp Stallman een licentie die ervoor moest zorgen dat zijn programmacode tot in lengte van dagen vrij zou zijn. Die GNU General Public License (GPL) is een knap staaltje juridisch judo: Deze licentie bepaalt dat de code onbeperkt mag worden vermenigvuldigd en aangepast, maar dat zowel de kopieën als het afgeleide werk (de aanpassingen dus) onder dezelfde licentievoorwaarden moeten worden uitgebracht, zonder aanvullende beperkingen. Hij gebruikt hier de wet op de auteursrechten om een effect te bewerkstelligen dat tegengesteld is aan het gangbare auteursrecht: In plaats de verspreiding van de software te beperken, mag juist niemand, zelfs de auteur niet, de verspreiding ervan tegengaan. Stallman verwachtte van deze licentie meer dan van het gewoonweg vrijgeven van zijn programmacodes. Openbaar gemaakte programmacode zou immers in propriëtaire software terecht kunnen komen (wat in de praktijk inderdaad veel is gebeurd met code die niet goed was beschermd). Hoewel de beschikbaarheid van de originele code er niet mee werd aangetast, zou Stallman er wel zijn vijand — de makers van propriëtaire software — mee hebben geholpen. U kunt de GPL zien als een protectionistische maatregel voor vrije software, omdat het voorkomt dat de makers van niet-vrije software de GPL-code ongestraft voor hun eigen doeleinden kunnen gebruiken. De GPL in relatie tot andere vrije softwarelicenties wordt in meer detail besproken in Hoofdstuk 9, *Licenties, auteursrechten en patenten.*

Met de hulp van vele programmeurs, waarvan slechts enkele de ideeën van Stallman aanhingen en de rest eenvoudigweg over veel vrije programmacode wilde kunnen beschikken, begon het GNU-project met het schrijven van vrije programmacode voor de meest kritieke onderdelen van een besturingssysteem. Vanwege de inmiddels verregaande standaardisatie van hardware en software voor computers was het mogelijk om de GNU-programma's te gebruiken op verder onvrije systemen, en velen deden dit ook. Met name de GNU-teksteditor (Emacs) en C-compiler (GCC) waren zeer succesvol. Ze kregen een grote schare trouwe fans, niet op idealistische gronden maar vanwege de technische mogelijkheden. Rond 1990 had het GNU-project een nagenoeg volledig vrij besturingssysteem opgeleverd, met uitzondering van de kernel — dat is de code die de computer opstart en zorgt voor het beheer van het geheugen, de schijfstations en de overige systeembronnen.

Ongelukkigerwijze had het GNU-project een kernelontwerp gekozen dat moeilijker te implementeren bleek dan verwacht. Dit leverde zo veel vertraging op dat de Free Software Foundation geen compleet vrij besturingssysteem kon uitbrengen. Het laatste stukje van de puzzel kwam van Linus Torvalds, een Finse student computerwetenschappen die met hulp van vrijwilligers uit de hele wereld een vrije kernel wist te ontwikkelen met een minder vooruitstrevend ontwerp. Hij noemde dit Linux en in combinatie met de bestaande GNU-programma's leverde dit een volledig vrij besturingssysteem op. Voor het eerst was het nu mogelijk om een computer op te starten en er mee te werken zonder enige vorm van propriëtaire software te hoeven gebruiken.[5]

Veel van de software in dit nieuwe besturingssysteem was niet door het GNU-project geproduceerd. GNU was ook niet de enige groep die aan een vrij besturingssysteem werkte. Zo werd er op dat moment ook al gewerkt aan de code die uiteindelijk in NetBSD en FreeBSD terecht zou komen. Het belang van de Free Software Foundation lag niet alleen in de programmacode die ze schreven, maar vooral in de politieke taal die ze uitten. Door het over vrije software te hebben als ideologie en niet als gemaksartikel, *moesten* programmeurs er zich wel een mening over vormen. Zelfs degenen die het niet met de FSF eens waren, moesten zich ermee bezig houden, al was het maar om er afstand van te nemen. Op deze manier kon de FSF effectief propaganda bedrijven door met hun code een boodschap mee te geven, via de GPL en via andere teksten. Tegelijk met de code werd ook hun boodschap verspreid.

Onbewuste weerstand

In de ontluikende wereld van vrije software gebeurde echter nog veel meer, zonder een achterliggende expliciete ideologie als het GNU-project van Stallman. Een van de belangrijkste initiatieven was de *Berkeley Software Distribution* (*BSD*), een geleidelijke herwaardering van het Unix-besturingssysteem — dat tot het einde van de jaren zeventig gold als een min of meer propriëtair onderzoeksproject voor AT&T — door programmeurs van de *University of California* in Berkeley. De BSD-groep mengde zich niet openlijk in politieke discussies waarin programmeurs werden opgeroepen tot samenwerking en het delen van code, maar ze *deed* het wel met volle inzet en enthousiasme, door een grootschalig gedistribueerd ontwikkelingsproject te coördineren waarbij de commandoregelfuncties en programmabibliotheken van Unix, en uiteindelijk ook de kernel van dat besturingssysteem, volledig opnieuw werden geprogrammeerd, voornamelijk door vrijwilligers. Het BSD-project werd bij uitstek een voorbeeld van de niet-ideologische ontwikkeling van vrije software en was het voorportaal van veel ontwikkelaars die later in de wereld van open source actief zijn gebleven.

Een ander samenwerkingsproject was het *X Window System*, een vrije grafische netwerkomgeving die in het midden van de jaren tachtig aan het MIT werd ontwikkeld in samenwerking met hardwareleveranciers die hun klanten met schermvensters wilden laten werken. De X-licentie maakte het echter met opzet mogelijk om de vrije kern uit te breiden met propriëtaire software, omdat alle leden van het samenwerkingsverband met hun eigen toevoegingen aan de X-distributie een concurrentievoordeel wilden behalen ten opzichte van de andere leden. X Windows[6] zelf was vrije software, maar het diende voornamelijk als gemeenschappelijke basis voor verder tegenstrijdige zakelijke belangen. Het was zeker niet gericht op het beëindigen van de dominantie van propriëtaire software. Nog een ander voorbeeld, dat enkele jaren voor het GNU-project speelde, was *TeX*. Een vrij tekstverwerkingsprogramma voor drukwerkproducties van Donald Knuth. Hij verspreidde dit met een licentie die bepaalde dat iedereen de programmacode mocht aanpassen en verspreiden, maar dat dit geen 'TeX' mocht worden genoemd tenzij er aan een strikte reeks van compatibiliteitseisen was voldaan (dit is een voorbeeld van een vrije licentie met 'handelsmerkbescherming', waar u meer over kunt lezen in Hoofdstuk 9, *Licenties, auteursrechten en patenten*). Knuth nam geen stelling in het debat tussen vrije en propriëtaire software, hij had alleen maar een betere tekstverwerker nodig om zijn *echte* doel te bereiken — het schrijven van een boek over programmeren — en hij zag geen reden om zijn systeem daarna voor zichzelf te houden.

Zonder op elk project en op elke vorm van licentie in te gaan, kunnen we met zekerheid zeggen dat er eind jaren tachtig allerlei vrije software beschikbaar was onder uiteenlopende licentievoorwaarden. Deze veelheid aan licentievoorwaarden is een afspiegeling van de bijbehorende veelheid aan achterliggende motivaties. Zelfs niet alle programmeurs die voor de GNU GPL kozen, waren zo idealistisch als het GNU-project zelf. Veel ontwikkelaars werken nu eenmaal graag aan vrije software, maar zien propriëtaire software niet als een maatschappelijk ongewenst verschijnsel. Er waren wel mensen die een morele drang voelden om de wereld te bevrijden van de 'softwarehamsteraars' (Stallmans term voor de makers van niet-vrije software), maar anderen haalden hun motivatie gewoon uit de techniek of uit het plezier van samenwerken met gelijkgestemden, of zelfs uit een basale behoefte als drang naar erkenning. Deze uiteenlopende motivaties bleken elkaar echter niet in de weg te zitten. Dat is deels te danken aan het feit dat software, in tegenstelling tot andere creatieve uitingsvormen zoals proza of de beeldende kunsten, aan semiobjectieve criteria moet voldoen om een succes te zijn: het moet werken en er mogen niet te veel fouten in zitten. Dit geeft alle deelnemers aan een project automatisch een soort gemeenschappelijke basis en creëert een reden en een kader voor samenwerking, zonder zich zorgen te hoeven maken over andere dan technische criteria.

Bovendien hadden veel ontwikkelaars nóg een reden mee te doen: Vrije software bleek programmacode van hoge kwaliteit op te kunnen leveren. In sommige gevallen was het aantoonbaar technisch superieur aan de vergelijkbare commerciële alternatieven. Met andere woorden, het was minstens zo goed en de prijs was altijd lager. Terwijl er weinig mensen zijn die vrije software alleen op idealistische gronden willen gebruiken, zijn er altijd genoeg mensen die dergelijke software graag willen gebruiken als dit betere resultaten oplevert. En van die gebruikers is altijd een klein percentage bereid om hun tijd en kennis ter beschikking te stellen voor het onderhouden en verbeteren van de software.

Niet alle projecten leveren hoogwaardige code op, maar het gebeurt bij open source-softwareprojecten overal ter wereld steeds vaker. Dit begon bedrijven die sterk afhankelijk zijn van software ook op te vallen. Veel bedrijven ontdekten dat ze allang vrije software gebruikten in hun dagelijkse bedrijfsvoering, maar het eenvoudigweg niet wisten (de directie weet nu eenmaal niet altijd wat de IT-afdeling precies doet). Bedrijven werden actiever en deden steeds minder geheimzinnig over hun rol in open source-softwareprojecten. Ze begonnen tijd en apparatuur beschikbaar te stellen voor de ontwikkeling van vrije programma's en ze soms zelfs te financieren. In het gunstigste geval betalen dergelijke investeringen zichzelf namelijk vele malen terug. De sponsor hoeft slechts enkele programmeurs te betalen die zich voltijds met het project bezig houden, maar profiteert wel van de inbreng van *iedereen*, inclusief het werk van de onbetaalde vrijwilligers en de programmeurs die door andere bedrijven worden betaald.

'Vrij' versus 'Open source'

De term open source-software is in het Nederlands taalgebied onomstreden. In het Engels werd open source-software eerst 'free software' genoemd, een term die nog steeds gebruikt wordt. Die term zorgde na verloop van tijd voor verwarring door de dubbele betekenis die het woord 'free' heeft in het Engels. 'Free' kan worden vertaald als 'vrij' en als 'gratis'. Beide termen waren lastig. Open Source Software was immers niet per se kosteloos en vrij betekende niet dat gebruikers de code in alle gevallen kenden of mochten aanpassen.

Uiteindelijk kwam het Open Source initiative (OSI) met de term 'open source', om een oplossing te bieden voor de verwarring rondom de term 'free', maar ook als marketingstrategie. 'Free software' lag niet goed in het bedrijfsleven. Het werd gezien als niet-commercieel en soms zelfs als diefstal. Dat concept wilden bedrijven dus niet kopen. Dezelfde software met het label 'open source' was veel beter te vermarkten.

Zoals gezegd speelt deze discussie in het Nederlands taalgebied niet. De term open source is hier gangbaar en volledig geaccepteerd.
(bron: http://opensource.feratech.com/advocacy/faq.php en http://opensource.feratech.com/advocacy/case_for_hackers.php#marketing)

1.2 DE HUIDIGE STAND VAN ZAKEN

Als u een project voor vrije software beheert, hoeft u zich echt niet dagelijks bezig te houden met zware filosofische vraagstukken. Programmeurs stellen niet als voorwaarde dat alle deelnemers aan het project over alles hetzelfde denken als zij (wie dat wel doet, is al snel in geen enkel project meer welkom). Maar u moet wel weten dat er een tweestrijd bestaat tussen 'vrije software' en 'open source', al was het maar om te voorkomen dat u iemand tegen zich in het harnas jaagt en omdat het doorgronden van de motieven van de ontwikkelaars de beste manier is — en vaak de *enige* — om een project te beheren.

Vrije software heeft een eigen cultuur. Om er succesvol in te kunnen werken, moet u in de eerste plaats begrijpen waarom mensen er aan deel willen nemen. Dwang werkt niet. Als het ene project ze niet meer bevalt, gaan ze gewoon naar het volgende. De vrijesoftwarecultuur valt zelfs tussen andere vrijwilligersprojecten op door de losse binding die de deelnemers hebben. De deelnemers ontmoeten elkaar vrijwel nooit in het echt en steken er alleen tijd in als ze er zin in hebben. De normale wegen waarlangs mensen zich met elkaar kunnen verbinden om een hechte groep te vormen, zijn daarmee aanzienlijk versmald: het beperkt zich uitsluitend tot het geschreven woord, dat elektronisch wordt verzonden. Om deze reden kan het lang duren voordat er zich een samenhangende en toegewijde groep vormt. Daarentegen is het heel makkelijk om vrijwilligers al na vijf minuten weer te verliezen. Als een project geen goede eerste indruk maakt, zijn nieuwkomers vaak direct weer weg.
De vergankelijkheid, of liever gezegd de *potentiële* vergankelijkheid, van relaties is wellicht het moeilijkste aspect bij het opzetten van een nieuw project. Wat houdt al deze mensen lang genoeg bij elkaar om iets bruikbaars te produceren? Met het beantwoorden van die vraag zou ik de rest van dit boek kunnen vullen, maar samengevat in één regel ziet het er als volgt uit:

Men moet voelen dat de betrokkenheid bij een project en de invloed die men kan uitoefenen rechtevenredig is aan de bijdrage die men levert.

Sluit nooit ontwikkelaars of potentiële ontwikkelaars uit om niet-technische redenen. Het mag duidelijk zijn dat door het bedrijfsleven gesponsorde projecten, al dan niet met betaalde ontwikkelaars, op dit punt extra voorzichtig moeten opereren (zie ook Hoofdstuk 5, *Geld voor meer details*). Dat wil niet zeggen dat er zonder sponsoring geen vuiltje aan de lucht is. Geld is slechts één van de vele factoren die invloed hebben op het welslagen van een project. Het is ook belangrijk welke taal u kiest en welke licentievorm, net als het ontwikkelingsproces, de beschikbare infrastructuur, een effectieve aankondiging van het begin van het project en nog veel meer. Het volgende hoofdstuk gaat over het opzetten van een kansrijk project.

2| SourceForge.net is een populaire hosting-website en bood in april 2004 onderdak aan 79.225 geregistreerde projecten. Dat zijn natuurlijk lang niet alle open source-softwareprojecten waar via internet aan wordt gewerkt; dit zijn alleen de projecten bij SourceForge.

3| Stallman gebruikt het begrip 'hacker' in de zin van 'iemand die van slim programmeren houdt', en niet in de relatief nieuwe betekenis van 'iemand die in computers inbreekt'.

4| Dit staat voor 'GNU's Not Unix', en het 'GNU' in die afkorting staat voor ... weer precies hetzelfde.

5| Technisch gezien was Linux niet de eerste. Kort voor Linux kwam er een vrij besturingssysteem uit voor IBM-compatibele computers, onder de naam 386BSD. Het was echter erg moeilijk om 386BSD goed werkend te krijgen. Linux werd niet alleen populair omdat het vrij was, maar voornamelijk omdat de kans groot was dat je computer na het installeren ook daadwerkelijk opstartte.

6| De eigenlijke naam is 'X Window System', maar in de praktijk wordt het doorgaans 'X Windows' genoemd omdat drie woorden gewoon teveel zijn.

7| Er mogen kosten in rekening worden gebracht voor het leveren van vrije software, maar aangezien het niet verboden is om deze software gratis verder te verspreiden, is de prijs feitelijk direct nul.

8| De broncode van Netscape Navigator is in 1998 uiteindelijk wél vrijgegeven onder een open source-licentie en vormt nu de basis van de Mozilla-webbrowser. Zie http://www.mozilla.org/.

9| De OSI-website is http://www.opensource.org/.

2

AAN DE GANG

Het klassieke model over hoe van start te gaan met open source-softwareprojecten is van de hand van Eric Raymond, uit een inmiddels beroemd artikel over open source-processen getiteld The Cathedral and the Bazaar. Hij schreef:

Ieder goed softwareproject begint bij de persoonlijke passie van een ontwikkelaar.
(uit: http://www.catb.org/~esr/writings/cathedral-bazaar/)

Wat opvalt is dat Raymond niet zegt dat open source-projecten alleen mogelijk zijn wanneer een individu er zin in heeft. Wat hij zegt, is dat goede software het gevolg is van een programmeur die er persoonlijk belang bij heeft dat het probleem wordt opgelost. Deze persoonlijk passie is relevant voor open source-software omdat dit in de meeste gevallen de motivatie vormt om een open source-softwareproject te beginnen.

De meeste open source-softwareprojecten beginnen nog steeds zo, maar in mindere mate dan in 1997, toen Raymond zijn wijze woorden schreef. Vandaag de dag zien we dat organisaties, waaronder commerciële bedrijven, centraal beheerde open source-projecten vanaf het eerste begin op poten zetten. De eenzame programmeur, die wat code uit zijn mouw schudt om een lokaal probleem op te lossen en zich vervolgens realiseert dat het veel breder inzetbaar is, is nog steeds een bron van veel nieuwe open source-software, maar zeker niet de enige.

Raymonds insteek is echter nog steeds waardevol. Een belangrijke voorwaarde is dat de producenten van de software zelf direct belang hebben bij het succes ervan, omdat ze het zelf gebruiken. Wanneer de software niet doet wat het zou moeten doen, zal de persoon of de organisatie daar in hun dagelijkse werkzaamheden last van hebben. Van het *OpenAdapter-project* (http://www.openadapter.org/) bijvoorbeeld, dat werd opgezet door de investeringsbank Dresdner Kleinwort Wasserstein en dat een open source-framework is voor het integreren van ongelijksoortige financiële informatiesystemen, kan nauwelijks worden gezegd dat er sprake was van persoonlijke passie van de programmeurs. Maar de instelling in kwestie had die passie wel. Die passie had direct te maken met de ervaringen van de instelling en haar partners en als het project niet in zou spelen op die ervaring, zou dat zeker

gevoeld worden. Deze manier van werken resulteert in goede software, omdat de feedbacklus in de juiste richting loopt. Het programma wordt niet geschreven om aan een ander te worden verkocht, het is bedoeld om een eigen probleem op te lossen. Het wordt geschreven om iemands eigen probleem op te lossen, waarna het met anderen wordt gedeeld. Het is alsof het probleem een ziekte is, en de software het geneesmiddel waarvan de distributie gericht is op het volledig uitroeien van de epidemie.

Dit hoofdstuk gaat over de manier waarop een nieuw open source-softwareproject bij de buitenwereld kan worden geïntroduceerd. Veel van de aanbevelingen lijken echter op die van een gezondheidsorganisatie die een geneesmiddel distribueert. De doelen komen sterk overeen: U wilt duidelijk maken wat het geneesmiddel doet en ervoor zorgen dat het in de handen van de juiste mensen komt en dat degene die het gebruikt, weet hoe het moet worden toegepast. In het geval van software wilt u echter ook sommige gebruikers overhalen deel uit te gaan maken van het lopende onderzoek om het geneesmiddel te verbeteren.

De distributie van vrije software is een tweeledige taak. Er moeten gebruikers voor de software worden gevonden, maar ook ontwikkelaars. Deze twee elementen zijn niet per definitie in strijd met elkaar, maar ze maken de eerste presentatie van het project wel gecompliceerder. Sommige informatie is bruikbaar voor beide doelgroepen, andere informatie is echter alleen bruikbaar voor de één of de ander. Beide soorten informatie moeten voldoen aan het principe van de 'geschaalde' presentatie. Dat wil zeggen dat het detailniveau van de gepresenteerde informatie in iedere fase direct overeen moet komen met de hoeveelheid tijd en moeite die de lezer erin stop. Meer moeite moet altijd meer worden beloond. Wanneer er geen zeer sterk verband is tussen de twee kunnen mensen hun vertrouwen snel verliezen en willen ze er geen moeite meer voor doen.

Een logisch gevolg hiervan is dat het uiterlijk er wel degelijk toe doet. Met name programmeurs zijn niet altijd geneigd dit te geloven. Hun liefde voor inhoud boven vorm kan bijna worden gezien als een vorm van beroepseer. Het is geen toeval dat veel programmeurs een grote hekel hebben aan marketing en pr, en ook niet dat grafisch ontwerpers vaak ontzet zijn door wat programmeurs er op hun eigen houtje van hebben gebakken.

En dat is jammer, want er zijn situaties waarin de vorm gelijk staat aan de inhoud, en de presentatie van een project is daar één van. Het eerste bijvoorbeeld dat een bezoeker te weten komt van een project is hoe de website eruitziet. Deze informatie wordt opgenomen voordat er ook maar enig inzicht is in de feitelijke inhoud van de site, dus voordat er ook maar iets van de tekst is gelezen of links zijn aangeklikt. Hoe onrechtvaardig dit ook kan lijken, mensen zullen nou eenmaal altijd een mening vormen op grond van de eerste indruk. De uitstraling van de website geeft aan of er zorg is besteed aan de presentatie van het project. Mensen hebben een extreem gevoelige antenne voor de hoeveelheid aandacht die is besteed. De meesten van ons kunnen in één oogopslag zeggen of een website snel in elkaar is geflanst of dat er serieus aandacht is besteed. Dit is het eerste stukje informatie dat je project publiceert en de indruk die dit maakt, is door associatie van invloed op de rest van het project.

Hoewel een groot deel van dit hoofdstuk gaat over de inhoud waarmee uw project van start zou moeten gaan, mag u dus niet vergeten dat het ook uitmaakt hoe het eruit ziet en aanvoelt. Omdat de website van het project bedoeld is voor twee verschillende soorten bezoekers, namelijk gebruikers en ontwikkelaars, moet er bijzondere aandacht worden besteed aan duidelijkheid en doelgroepgerichtheid. Hoewel dit niet de juiste tijd en plaats is voor een algemene verhandeling over webdesign, is één principe ervan de moeite van het vermelden waard, met name wanneer de site bestemd is voor meerdere doelgroepen (eventueel met overlap): bezoekers moeten min of meer kunnen zien waar een link naartoe gaat voordat ze erop klikken. Het moet bijvoorbeeld *alleen al door te kijken* naar de links naar gebruikersdocumentatie duidelijk zijn, dat ze naar gebruikersdocumentatie leiden en niet naar bijvoorbeeld ontwikkelaarsdocumentatie. Een project opzetten gaat voor een deel over het bieden van informatie, maar voor een deel ook over het bieden van gemak. Alleen de aanwezigheid al van bepaald standaardelementen op plaatsen waar deze worden verwacht, wekt vertrouwen bij gebruikers en ontwikkelaars. Uiteindelijk moeten zij beslissen of ze wel of niet aan het project willen deelnemen. Het laat zien dat het project zijn zaakjes goed heeft geregeld, dat het voorbereid is op de vragen die mensen gaan stellen en moeite heeft gedaan deze zo te beantwoorden dat er niet teveel inspanning van de vraagsteller wordt verwacht. Door uit te stralen dat het overal op voorbereid is, geeft het project een boodschap af: "Het is geen verspilling van je tijd wanneer je hieraan meedoet," en dat is precies wat mensen willen horen.

Maar eerst kijkt u rond

Voordat u met een open source-project van start gaat eerst een belangrijk waarschuwing:

Kijk altijd eerst heel goed rond of er al een project bestaat dat doet wat u wilt. Er bestaat een redelijke kans dat het probleem dat u nu wilt oplossen, iemand anders vóór u ook al heeft willen oplossen. Als ze het probleem hebben opgelost en de code uitgebracht hebben onder een vrije licentie, hoeft u het wiel niet opnieuw uit te vinden. Natuurlijk zijn er uitzonderingen: als u een project wilt opzetten als educatieve ervaring heeft u natuurlijk niets aan bestaande code. Ook kan het project dat u in gedachten hebt zo specialistisch zijn dat u zeker weet dat nog niemand dat heeft gedaan. Maar in het algemeen kan rondkijken echt geen kwaad en kunnen de voordelen enorm zijn. Wanneer de gebruikelijke zoekacties op internet niets opleveren, kunt u een kijkje nemen op http://freshmeat.net/ (een nieuwssite voor open source-projecten, waarover later meer), http://www.sourceforge.net/ en in de lijst van vrije software van de Free Software Foundation op http://directory.fsf.org/.
Zelfs als u niet precies kunt vinden wat u zoekt, kan het zijn dat u iets vindt dat zo dicht in de buurt komt dat het zinvoller is aan dat project deel te gaan nemen en daar functionaliteit aan toe te voegen dan helemaal bij nul te beginnen.

2.1 BEGINNEN MET WAT U HEBT

OK, u hebt rondgekeken, niets gevonden dat past bij wat u nodig hebt en besloten een nieuw project op te zetten.

Wat nu?

Het moeilijkste stuk van het lanceren van een open source-softwareproject is uw privévisie te transformeren naar een publieke visie. U of de organisatie waar u voor werkt, mag dan wel precies weten wat u wilt, het kan een hele opgave zijn om uw bedoelingen ook op een begrijpelijke manier aan de buitenwereld duidelijk te maken. Het is echter van cruciaal belang dat u hiervoor voldoende tijd uittrekt. U, en andere oprichters, moeten beslissen waar het project om draait, dat wil zeggen beslissen waar de grenzen liggen, wat het *niet* doet en wat wel, en dit in de vorm van een missieverklaring op papier zetten. Dit gedeelte is meestal niet echt moeilijk, hoewel er soms onuitgesproken vooronderstellingen en zelfs onenigheid aan het licht kunnen komen over de aard van het project, wat natuurlijk prima is: het is beter dit nu op te lossen dan later. De volgende stap is om het project zo te verpakken, dat het geschikt is voor consumptie. Dit is eerlijk gezegd een eentonig werkje.

Wat het zo bewerkelijk maakt is dat het voornamelijk bestaat uit het organiseren en documenteren van dingen die iedereen al lang weet; dat wil zeggen, iedereen die tot dat moment bij het project betrokken is geweest. Voor de mensen die het feitelijke werk doen, heeft dit dus eigenlijk geen direct voordeel. Zij hebben namelijk niet zoveel aan een README-bestand waarin een overzicht van het project wordt gegeven, noch aan een ontwerpersdocument of een gebruikershandleiding. Zij hebben geen behoefte aan een zorgvuldig opgezet codeschema waarmee wordt geconformeerd aan de informele maar wijdverspreide normen voor de distributie van softwarebroncode. De broncode is, ongeacht de opzet, voor hen sowieso in orde. Uiteindelijk zijn zij er al aan gewend. En als de code functioneert, weten ze ook hoe ze die moeten gebruiken. Het maakt hun zelfs niet uit als de fundamentele architecturale uitgangspunten van het project helemaal niet worden gedocumenteerd. Ook daaraan zijn ze al gewend.

Nieuwkomers aan de andere kant, hebben hier wel degelijk behoefte aan. Gelukkig hebben ze dit niet allemaal tegelijk nodig. Het is niet nodig om iedere mogelijke bron op papier te zetten voordat het project publiekelijk wordt gelanceerd. Alleen in een perfecte wereld gaat ieder nieuw open source-project van start met een gedetailleerd ontwerpdocument, een complete gebruikershandleiding (met speciale verwijzingen naar toepassingen die wel gepland maar nog niet geïmplementeerd zijn), prachtige en portbaar verpakte code die op ieder platform draait enz.

In de praktijk zou het aan elkaar knopen van al deze losse eindjes hinderlijk veel tijd kosten. Dit zijn dingen waarvan toch wel mag worden gehoopt dat vrijwilligers hiermee een handje zullen helpen zodra het project eenmaal loopt.

Wat echter *wel* nodig is, is dat er voldoende tijd en moeite wordt gestopt in de presentatie, waardoor nieuwkomers de eerste drempel van het onbekende over worden geholpen. Dit moet worden gezien als de eerste stap in het opstartproces, waarin het project de eerste minimale energiestoot krijgt toegediend. Ik heb deze eerste drempel ook wel *hacktivation energy* horen noemen: de hoeveelheid energie die een nieuwkomer erin moet steken voordat hij er iets voor terugkrijgt. Hoe lager de hacktivation energy van een project is, hoe beter. Uw eerste taak bestaat eruit deze hacktivation energy terug te brengen naar een niveau dat mensen aanspoort deel te nemen aan het project.

Iedere afzonderlijke subsectie hieronder beschrijft een belangrijk aspect van het opstarten van een nieuw project. Ze zijn in grote lijnen geordend in de volgorde waarmee een nieuwe bezoeker ermee wordt geconfronteerd, hoewel de volgorde waarin u ze uiteindelijk implementeert hiervan kan afwijken. U kunt de secties gebruiken als checklist. Wanneer u met een project van start gaat, ga dan gewoon de lijst langs en zorg ervoor dat ieder onderwerp aan bod komt, of dat u in ieder geval geen problemen hebt met de mogelijke gevolgen mocht u er een weglaten.

Een goede naam kiezen

Verplaats uzelf in iemand die net over uw project heeft gehoord, misschien doordat hij het toevallig tegen het lijf liep terwijl hij naar software zocht om een probleem op te lossen. Het eerste wat iemand dan ziet, is de naam van het project.

Een goede naam maakt uw project niet meteen tot een succes en een slechte naam zal echt niet de ondergang ervan betekenen (nou ja, een echt slechte naam zal dat misschien wel, maar we gaan er maar van uit dat niemand expres probeert een project te laten mislukken). Een slechte naam kan de acceptatie van het project echter vertragen, of omdat mensen hem niet serieus nemen, of gewoon omdat ze hem niet kunnen onthouden.

Een goede naam:

- geeft een idee van wat het project doet of heeft er in ieder geval een duidelijke relatie mee, zodat de naam hem later snel weer te binnen schiet;
- is makkelijk te onthouden. In dit geval kunt u niet om het feit heen dat Engels de standaardtaal is geworden van internet: ‘makkelijk te onthouden’ betekent ‘makkelijk te onthouden voor iemand die Engels kan lezen.’ Namen met woordspelingen die afhangen van de uitspraak van iemand met Engels als moedertaal kunnen abracadabra zijn voor mensen voor wie Engels niet de moedertaal is. Als de woordspeling bijzonder goed in het gehoor ligt en makkelijk te onthouden is, kan het wel de moeite waard zijn. Vergeet alleen niet dat veel mensen die de naam zien de woordspeling niet op dezelfde manier in hun hoofd horen als iemand met Engels als moedertaal;
- is niet gelijk aan namen van andere projecten en maakt geen inbreuk op handelsmerken. Dit is niet alleen fatsoenlijk maar ook juridisch verstandig. U wilt geen verwarring creëren over de identiteit. Het is vandaag de dag al moeilijk genoeg om bij te houden wat er allemaal op internet beschikbaar is zonder dat verschillende dingen dezelfde naam hebben.
  De bronnen die eerder zijn genoemd in het gedeelte ‘Maar eerst kijkt u rond’ kunnen goed van pas komen als u wilt uitzoeken of er al een project bestaat met de naam die u in gedachten hebt. U kunt gratis op handelsmerken zoeken op

http://www.nameprotect.org/ en http://www.uspto.gov/.
- is zo mogelijk beschikbaar als domeinnaam in de domeinen .com, .net en .org. U moet er één kiezen, waarschijnlijk .org, om mee te adverteren als de officiële website van het project. De andere twee moeten worden doorgelinkt naar deze site en zijn alleen bedoeld om te voorkomen dat anderen verwarring kunnen veroorzaken over de identiteit rond de naam van het project. Zelfs als u van plan bent de hosting van het project bij een andere site onder te brengen (zie het gedeelte 'Canned hosting') kunt u nog steeds specifieke domeinen registreren en deze doorlinken naar de hostingsite. Het helpt gebruikers als de URL makkelijk te onthouden is.

Zorg voor een duidelijke missieverklaring
Zodra mensen de website van het project hebben gevonden, gaan ze eerst op zoek naar een korte beschrijving, een missieverklaring, om (binnen 30 seconden) te kunnen beslissen of ze geïnteresseerd zijn. Deze moet op een prominente plaats op de eerste pagina vermeld staan, bij voorkeur direct onder de naam van het project.
De missieverklaring moet concreet, beperkend en vooral kort zijn. Hier een voorbeeld van een goede missieverklaring, van http://www.openoffice.org/:

We willen als een gemeenschap *hét internationale kantoorpakket ontwikkelen, dat beschikbaar is voor de belangrijkste besturingssystemen, met toegang tot alle functies en gegevens via open, componentgebaseerde API's en een op XML gebaseerd bestandsformaat.*

In slechts een paar woorden slaan ze de spijker op zijn kop, voornamelijk door in te spelen op de voorkennis van de lezer. Door 'als gemeenschap' te zeggen geven ze aan dat geen van de deelnemende partijen een dominante rol speelt in de ontwikkeling; 'internationaal' betekent dat de software mensen de kans geeft in meerdere talen en op verschillende plaatsen te werken en 'de belangrijkste besturingssystemen' houdt in dat het beschikbaar is voor Unix, Mac en Windows. De rest van de verklaring laat zien dat open interfaces en makkelijk te begrijpen bestandsindelingen een belangrijk onderdeel zijn van de doelstelling. Ze zeggen niet direct dat ze proberen een gratis alternatief te bieden voor Microsoft Office, maar de meeste mensen kunnen dit waarschijnlijk wel tussen de regels door lezen. Hoewel de missieverklaring op het eerste gezicht breed lijkt, worden de grenzen in feite duidelijk aangegeven: het woord '*kantoorpakket*' is heel concreet voor degene die bekend is met dergelijke software. Ook hier wordt de voorkennis van de lezer (in dit geval waarschijnlijk van MS Office) gebruikt om de missieverklaring beknopt te houden.

De aard van de missieverklaring hangt voor een deel af van de persoon die hem schrijft en niet alleen van de software die het beschrijft. Het is bijvoorbeeld logisch dat OpenOffice.org de woorden 'als gemeenschap' gebruikt, omdat het project werd gestart en nog steeds grotendeels wordt gesponsord door Sun Microsystems. Door deze woorden in de verklaring op te nemen geeft Sun aan dat het bedrijf zich bewust is van het feit dat mensen zich zorgen kunnen maken dat zij zullen proberen het proces te domineren. Met een dergelijke opmerking, die louter aangeeft dat men zich bewust is van een *potentieel* probleem, wordt het probleem al grotendeels ondervangen. Aan de andere kant hoeven projecten die niet worden gesponsord door één bedrijf dergelijke taal helemaal niet te gebruiken. Ontwikkeling door een gemeenschap is per slot van rekening de norm, dus normaal gesproken is er geen reden om dit in de missieverklaring te vermelden.

Vermeld dat het project vrij is
De mensen die nog steeds geïnteresseerd zijn nadat ze de missieverklaring hebben gelezen, zullen hierna meer informatie willen hebben, misschien gebruikers- of ontwerpersdocumentatie, en uiteindelijk iets willen downloaden. Maar voordat ze dat doen, willen ze zeker weten dat het om open source gaat.

De homepagina moet ondubbelzinnig duidelijk maken dat het om een open source-project gaat. Dit lijkt misschien voor de hand te liggen, maar u zult verbaasd zijn te zien hoeveel projecten dit vergeten. Ik heb open source-softwareprojecten gezien waarvan de homepagina niet alleen vergat te vermelden met welke afzonderlijke vrije licenties de software wordt gedistribueerd, maar zelfs dat de software vrij is. Soms werd deze informatie pas vermeld op de downloadpagina of de ontwikkelaarspagina, of een andere plaats die pas met een extra muisklik kon worden bereikt. In extreme gevallen werd er helemaal geen licentie vermeld op de website en was de enige manier om die te vinden de software te downloaden en er in te kijken.

Maak deze fout niet. Een dergelijke slordigheid kan u vele potentiële ontwikkelaars en gebruikers kosten. Vermeld direct aan het begin, direct onder de missieverklaring, dat het project gaat om 'vrije software' of 'open source software' en vermeld de exacte licentie. Voor een snelle handleiding over het kiezen van een licentie kunt u terecht in het gedeelte 'Een licentie kiezen en toepassen' verderop in dit hoofdstuk. Licentiekwesties worden uitgebreid behandeld in Hoofdstuk 9, *Licenties, auteursrechten en patenten.*

Op dit punt aangekomen heeft onze hypothetische bezoeker, waarschijnlijk in minder dan een minuut, besloten dat hij geïnteresseerd is om, laten we zeggen, nog eens een minuut of vijf te spenderen aan het project. In de volgende gedeelten wordt beschreven wat ze tegen zou moeten komen gedurende die vijf minuten.

Lijst met functies en vereisten
Er moet een korte lijst beschikbaar zijn met door de software ondersteunde functies (als iets nog niet klaar is, kunt u het wel in de lijst opnemen, maar met de vermelding '*gepland*' of '*in uitvoering*' ernaast) en de soort omgeving die vereist is om de software te kunnen gebruiken. De lijst met functies/vereisten kan worden gezien als de lijst die u iemand zou geven die vraagt om een korte samenvatting van de software. In de meeste gevallen is het een logische uitbreiding van de missieverklaring. De missieverklaring kan bijvoorbeeld de volgende zijn:

Een full-text indexer en zoekmachine creëren met een rich API voor gebruik door programmeurs door het bieden van zoekdiensten voor een groot aantal tekstbestanden.

De lijst met functies en vereisten bevat in dit geval de details, waarmee de draagwijdte van de missieverklaring wordt verduidelijkt:

Functies:

- Zoekt in platte tekst, HTML en XML
- Zoeken op woorden of zinnen
- (gepland) Fuzzy matching
- (gepland) Voortdurende updates van de indexen
- (gepland) Indexing van websites op afstand

Vereisten:

- Python 2.2 of hoger
- Voldoende schijfruimte voor de indexen (ongeveer 2x de omvang van de oorspronkelijke gegevens)

Met deze informatie kunnen lezers snel inschatten of de software voor hen zou kunnen werken en kunnen ze overwegen om ook als ontwikkelaar bij het project betrokken te zijn.

Ontwikkelingsstatus

Mensen willen altijd weten hoe ver een project gevorderd is. Voor nieuwe projecten willen ze weten hoe groot het gat is tussen de beloften van het project en de huidige realiteit. Voor gevorderde projecten willen ze weten hoe actief het wordt onderhouden, hoe vaak er nieuwe releases uitkomen, hoe snel er op bugmeldingen wordt gereageerd enz.

Om deze vragen te beantwoorden dient u te zorgen voor een pagina met de ontwikkelingsstatus, waarin de doelen en behoeften van het project voor de korte termijn worden weergegeven (het project kan bijvoorbeeld ontwikkelaars met een bepaalde expertise nodig hebben). Deze pagina kan ook een overzicht geven van de releases uit het verleden, met de functies, zodat bezoekers een indruk kunnen krijgen van hoe het project 'voortgang' definieert en hoe snel het daadwerkelijk vordert in verhouding met die definitie.

Wees niet bang om niet-klaar over te komen en geef niet toe aan de verleiding de ontwikkelingsstatus op te blazen. Iedereen weet dat software in fasen wordt ontwikkeld en niemand hoeft zich schamen om te zeggen: "Dit is alfasoftware met bekende bugs. Het loopt en werkt, soms, maar gebruik is voor eigen risico." Dergelijke taal schrikt het soort ontwikkelaar dat u in dit stadium nodig heeft niet af. En wat betreft de gebruikers, er is niet vervelender dan een project dat gebruikers lokt voordat de software klaar voor ze is. Als er eenmaal een reputatie van instabiliteit en bugs is gevestigd, komt u er niet zo makkelijk meer vanaf. Op de lange duur kunt u beter conservatief zijn. Het is altijd beter wanneer de software *stabieler* is dan de gebruiker verwacht dan *instabieler*, en prettige verrassingen zorgen voor de beste mond-tot-mondreclame.

Alfa en Bèta

De term *alfa* betekent over het algemeen de eerste release, waarmee gebruikers echt kunnen werken, en die over de bedoelde functionaliteit beschikt, maar waarvan ook bekend is dat hij bugs bevat. Het belangrijkste doel van alfasoftware is het genereren van feedback, zodat ontwikkelaars weten waar ze aan moeten werken. De volgende fase, *bèta*, betekent dat alle ernstige bugs in de software zijn hersteld, maar dat hij nog niet voldoende is getest voor de officiële release. Het doel van bètasoftware is dat dit de officiële release wordt wanneer er geen bugs meer worden gevonden, of dat er gedetailleerde feedback op komt voor de ontwikkelaars zodat ze de officiële release snel kunnen realiseren. Het verschil tussen alfa en bèta is vooral een kwestie van inzicht.

Downloads

De software moet gedownload kunnen worden als broncode in standaardbestandsindelingen. Wanneer het project zich nog in de opstartfase bevindt, zijn binaire (executable) pakketten niet nodig, behalve als de software zulke gecompliceerde kenmerken of afhankelijkheden heeft dat alleen het aan de gang krijgen al veel werk zou zijn voor de meeste mensen. (Als dat het geval is, zal het project sowieso moeite hebben om ontwikkelaars te vinden!)

Het distributiemechanisme moet zo gemakkelijk, standaard en goedkoop mogelijk zijn. Als u een ziekte wilt uitroeien, dan zult u er bij de distributie van het medicijn op letten dat het met een standaardinjectiespuit kan worden toegediend. Evenzo zou software moeten voldoen aan standaardbouw- en installatiemethodes. Hoe meer het van de normen afwijkt, des te groter de kans is dat potentiële gebruikers en ontwikkelaars het halverwege opgeven en afhaken.

Dit lijkt logisch, maar veel projecten doen tot aan het einde van het proces geen moeite hun installatieprocedures te standaardiseren, omdat ze zichzelf wijsmaken dat ze dit altijd later nog kunnen doen: *"Dat regelen we wel als de code bijna klaar is."* Zij realiseren zich niet dat door uitstel van het saaie werk aan de *build* en de installatieprocedures, het voltooien van de code feitelijk nog langer duurt, omdat ze ontwikkelaars afschrikken die anders misschien wel mee hadden willen werken aan de code. En wat nog verraderlijker is, is ze niet eens *weten* dat ze deze ontwikkelaar verliezen, omdat het proces uit een aaneenschakeling van non-events bestaat: iemand bezoekt de website, downloadt de software, probeert het te installeren, slaagt hier niet in, geeft het op en gaat weg. Maar wie anders dan de persoon zelf weet dat dit is gebeurd? Niemand van de mensen die aan het project werkt, realiseert zich dat iemands interesse en goede bedoelingen zomaar zijn verspild.

Saai werk met een hoog rendement moet altijd in een vroeg stadium worden gedaan. Het aanzienlijk verlagen van de toegangsdrempel van het project door het goed te verpakken geeft een zeer hoog rendement.

Wanneer u een downloadpakket uitbrengt, is het van extreem belang dat u de release een uniek versienummer toekent, zodat mensen altijd twee verschillende versies kunnen vergelijken en kunnen zien welke versie de laatste is. Een meer gedetailleerde uiteenzetting over versienummers vindt u in het gedeelte 'Releasenummers'. Meer informatie over het standaardiseren van build- en installatieprocedures vindt u in het gedeelte 'Downloadpakketten maken'. Beide onderwerpen zijn ook terug

te vinden in Hoofdstuk 7, *Downloadpakketten maken, releases en dagelijkse ontwikkeling.*

Toegang tot versiecontrole en bug tracker
Het downloaden van pakketten met de broncode is prima voor mensen die de software alleen willen installeren en gebruiken, maar het is niet gevoeg voor diegenen die hem willen debuggen of nieuwe functies willen toevoegen.

Screenshots van de broncode kunnen helpen, maar ze zijn nog steeds niet gedetailleerd genoeg voor een bloeiende ontwikkelaarsgemeenschap.

Mensen moet realtime toegang krijgen tot de laatste broncode. De manier om dat te doen is door gebruik te maken van een versiecontrolesysteem. De aanwezigheid van een anoniem toegankelijke versiegecontroleerde broncode is een signaal, zowel voor gebruikers als voor ontwikkelaars, dat dit project moeite doet de mensen te geven wat ze nodig hebben om te participeren. Als u niet meteen versiecontrole kunt aanbieden, plaats dan een melding dat u van plan bent dit snel te implementeren. De infrastructuur van versiecontrole wordt besproken in het gedeelte 'Versiecontrole' in Hoofdstuk 3, *Technische infrastructuur*.

Hetzelfde geldt voor de bug tracker van het project. Het belang van een bugopsporingssysteem zit 'm niet alleen in de bruikbaarheid voor de ontwikkelaars, maar ook in de boodschap die het geeft aan mensen die het project in de gaten houden. Voor veel mensen is een toegankelijke bugdatabase één van de sterkste signalen dat een project serieus moet worden genomen. Bovendien is het zo dat hoe groter het aantal bugs in de database is, des te beter het project eruitziet. Dit lijkt onlogisch, maar vergeet niet dat het aantal geregistreerde bugs van drie dingen afhangt: het absolute aantal aanwezige bugs in de software, het aantal gebruikers dat de software gebruikt en het gemak waarmee deze gebruikers nieuwe bugs kunnen melden. Van deze drie factoren spelen de laatste twee een grotere rol dan de eerste. Alle software van een bepaalde omvang en complexiteit heeft een in feite willekeurig aantal bugs, die er alleen nog maar op wachten te worden ontdekt. De feitelijk vraag is hoe goed een project in staat is deze bugs te registreren en er prioriteiten aan toe te kennen. Een project met een goed onderhouden bugdatabase (wat betekent dat er direct op bugs wordt gereageerd, dat dubbele bugs worden samengevoegd enz.) maakt daarom een betere indruk dan een project zonder bugdatabase of een bijna lege database.

Natuurlijk zal uw bugdatabase slechts een klein aantal bugs bevatten als uw project nog maar net is opgestart; daar is niet veel aan te doen. Als de statuspagina echter benadrukt dat het om een jong project gaat en als mensen naar de bugdatabase kijken en zien dat de meeste meldingen van recente datum zijn, kunnen zij daar uit afleiden dat het project beschikt over een in verhouding goed aantal registraties en zullen ze niet onnodig gealarmeerd worden door het lage absolute aantal geregistreerde bugs.

Merk op dat bugvinders vaak niet alleen worden gebruikt voor bugs in de software, maar ook voor verbeteringsverzoeken, wijzigingen in de documentatie, openstaande taken en nog veel meer. Meer informatie over het gebruik van een bug tracker vindt u in het gedeelte 'Bugvinder' in Hoofdstuk 3, *Technische infrastructuur*, daarom ga ik er hier niet op in. Het belangrijkste voor de presentatie van het project is het hebben van een bug tracker en ervoor te zorgen dat dit feit zichtbaar is op de homepagina van het project.

Communicatiekanalen
Bezoekers willen meestal weten hoe ze de mensen die bij het project betrokken zijn, kunnen bereiken. Verstrek de adressen van mailinglijsten, chatrooms en IRC-kanalen en alle andere mogelijke forums waar contact opgenomen kan worden met anderen die betrokken zijn bij de software. Maak duidelijk dat u en de andere auteurs van het project ook lid zijn van deze mailinglijsten, zodat mensen zien dat ze feedback kunnen geven die de ontwikkelaars ook bereikt. Uw aanwezigheid op deze lijst houdt niet in dat u zich verplicht alle vragen te beantwoorden of alle verzoeken om functies te implementeren. Uiteindelijk zullen de meeste gebruikers waarschijnlijk nooit lid worden van de forums, maar ze zullen gerustgesteld zijn door het feit dat het *kan* als het nodig is.

In de beginfase van het project is het niet nodig om afzonderlijke forums te hebben voor gebruikers en ontwikkelaars. Het is veel beter om iedereen die bij de software betrokken is ook met elkaar te laten praten, in één 'room'. Bij de vroege deelnemers is de grens tussen gebruiker en ontwikkelaar vaak niet zo duidelijk. Voor zover dit onderscheid wel kan worden gemaakt, is de ontwikkelaar-gebruikerverhouding in de beginfase van het project over het algemeen meer in het voordeel van eerstgenoemde dan later. Hoewel u er niet van uit kunt gaan dat iedere vroege deelnemer een programmeur is die wil meewerken aan het verbeteren van de software, kunt u er wel van uitgaan dat hij er in ieder geval in geïnteresseerd is om de ontwikkelingsdiscussies te volgen en een idee te krijgen van de richting van het project.

Omdat dit hoofdstuk alleen gaat over het opstarten van een project volstaat het te vermelden dat dergelijke communicatieforums dienen te bestaan. Later, in de sectie 'Omgaan met groei' in Hoofdstuk 6, *Communicatie*, gaan we uitzoeken waar en hoe u dergelijke forums kunt opzetten, de manier waarop ze gemodereerd of op een andere manier beheerd kunnen worden en hoe, wanneer de tijd daarvoor rijp is, het gebruikersforum moet worden afgesplitst van het ontwikkelaarsforum zonder een onoverbrugbare kloof te creëren.

Richtlijnen voor ontwikkelaars
Als iemand overweegt een bijdrage te leveren aan het project, zal hij op zoek gaan naar de richtlijnen voor ontwikkelaars. Richtlijnen voor ontwikkelaars zijn niet zozeer technisch als wel sociaal: ze geven aan hoe ontwikkelaars met elkaar en met de gebruikers omgaan en uiteindelijk hoe de dingen geregeld worden.

Dit onderwerp wordt meer gedetailleerd behandeld in het gedeelte 'Alles op papier zetten' in Hoofdstuk 4, *Sociale en politieke infrastructuur,* maar de belangrijkste elementen van de richtlijnen voor ontwikkelaars zijn:

• suggesties voor forums voor interactie met andere ontwikkelaars;

- instructies over *hoe* bugs moeten worden gerapporteerd en patches kunnen worden ingediend;
- een indicatie van hoe de ontwikkeling meestal plaatsvindt. Is het project een vriendelijke dictatuur, een democratie of nog iets anders.
- Met het woord 'dictatuur' wordt hier overigens niets negatiefs bedoeld. Er is helemaal niets mis met een dictatuur, waarbij één ontwikkelaar vetorecht heeft over alle wijzigingen. Veel succesvolle projecten werken op deze manier. Het belangrijkste is dat het project hier ook voor uitkomt. Een dictatuur die zich voordoet als democratie zal mensen tegen zich in het harnas jagen, terwijl een dictatuur die zegt een dictatuur te zijn prima kan functioneren, zolang de dictator competent is en vertrouwd wordt.
- Zie http://svn.collab.net/repos/svn/trunk/www/hacking.html voor een voorbeeld van bijzonder gedetailleerde ontwikkelaarsrichtlijnen, of http://www.openoffice.org/dev_docs/guidelines.html voor bredere richtlijnen, die zich meer richten op de supervisie en de geest van het meedoen en minder op technische zaken.
- Een programmeur een introductie geven over de software is een afzonderlijke kwestie, die wordt behandeld in het gedeelte 'Ontwikkelaarsdocumentatie' later in dit hoofdstuk.

Documentatie

Documentatie is van essentieel belang. Er moet altijd iets zijn dat mensen kunnen lezen, zelfs al is het elementair en onvolledig. Dit valt duidelijk onder het eerder genoemde 'eentonige' werk en is vaak waar een nieuw open source-project over struikelt. Een missieverklaring en een lijst met functies opstellen, een licentie kiezen, een overzicht geven van de ontwikkelingsstatus, dit zijn allemaal relatief kleine klusjes die geheel kunnen worden afgerond en waar meestal later niet meer op terug hoeft worden gekomen. Documentatie is echter nooit helemaal klaar, hetgeen een reden kan zijn waarom sommige mensen het maken ervan steeds maar weer uitstellen.

Het meest verraderlijke is dat de bruikbaarheid van de documentatie voor de schrijvers ervan vaak omgekeerd evenredig is aan de bruikbaarheid ervan voor de lezers. De meest belangrijke documentatie voor eerste gebruikers bestaat uit de beginselen: hoe de software snel kan worden geïnstalleerd, een overzicht van hoe hij werkt en misschien enkele instructies over het verrichten van veelvoorkomende taken. Dit is echter precies wat de *schrijvers* van de documentatie maar al te goed weten; zo goed, dat het moeilijk voor hen kan zijn om de dingen vanuit het perspectief van de lezer te bekijken. Zaken die voor de lezer uitvoerig beschreven zouden moeten worden, vinden zij vaak niet de moeite van het vermelden waard.

Er is helaas geen pasklare oplossing voor dit probleem. Iemand moet ervoor gaan zitten en alles opschrijven en vervolgens laten lezen door typisch nieuwe gebruikers om de kwaliteit ervan te testen. Gebruik een eenvoudig en makkelijk op te maken bestandsindeling zoals HTML, platte tekst, Texinfo of een variant van XML, wat soms de meest praktische oplossing is voor kleine, snelle en spontane verbeteringen. Dit is niet alleen goed om de drempels te weg te halen die de oorspronkelijke schrijvers ervan weerhouden periodieke verbeteringen aan te brengen, maar ook voor degenen die later bij het project komen en aan de documentatie willen werken.

Een manier om ervoor te zorgen dat de eerste basisdocumentatie wordt gemaakt, is om van tevoren de reikwijdte ervan te beperken. Daardoor lijkt het in ieder geval niet op een taak waaraan geen einde komt. Een goede vuistregel is dat deze minimaal moet voldoen aan de volgende criteria:

Vertel de lezer duidelijk hoeveel technische expertise van hem wordt verwacht.

Beschrijf helder en grondig hoe de software moet worden geïnstalleerd en vertel de gebruiker, ergens aan het begin van de documentatie, hoe een soort diagnostische test of enkelvoudig commando kan worden gedaan om te kijken of alles correct is geïnstalleerd. De documentatie over het opstarten van de software is in sommige gevallen nog belangrijker dan de feitelijke gebruikersdocumentatie. Hoe meer moeite iemand heeft moeten doen om de software te installeren en op te starten, des te vasthoudender hij zal zijn bij het uitzoeken van geavanceerde functionaliteit die niet goed is gedocumenteerd. Wanneer mensen afhaken, doen ze dat in de beginfase. Daarom hebben ze bij de eerste fasen, zoals de installatie, de meeste ondersteuning nodig.

Geef één voorbeeld in de vorm van een praktische oefening over hoe een veelvoorkomende taak kan worden uitgevoerd. Natuurlijk, veel oefeningen met veel taken zou nog beter zijn, maar als de tijd beperkt is, kiest u één taak die u grondig uitwerkt. Wanneer iemand gezien heeft dat de software gebruikt kan worden voor één ding, zullen ze op eigen houtje proberen uit te vinden wat ze nog meer kunnen en, als u geluk hebt, de documentatie zelf gaan schrijven. Wat ons bij het volgende punt brengt.

Geef de plaatsen aan waarvan bekend is dat de documentatie onvolledig is. Door de lezers te laten zien dat u zich bewust bent van de tekortkomingen laat u zien dat u de dingen vanuit hun perspectief bekijkt. Dit verzekert hen dat zij het project er niet van hoeven te overtuigen van wat belangrijk is en wat niet. Deze opmerkingen hoeven geen toezegging te zijn om de hiaten voor een bepaalde datum opgevuld te hebben, ze kunnen evengoed worden gezien als een open oproep voor hulp van vrijwilligers.

Het laatste punt is in feite van toepassing op het hele project en niet alleen op de documentatie. Het is in de wereld van open source normaal om de onvolkomenheden nauwkeurig bij te houden. U hoeft de tekortkomingen van het project niet te overdrijven, u moet ze alleen nauwgezet en feitelijk identificeren wanneer de context daarom vraagt (dat kan zijn in de documentatie, de bugtrackingdatabase of in een discussie via de mailinglijst). Niemand zal dit zien als probleem van het project, noch als toezegging om het probleem voor een bepaalde datum opgelost te hebben, behalve wanneer het project deze toezegging expliciet doet. Omdat iedereen die de software gebruikt de tekortkomingen zelf zal tegenkomen, is het beter dat zij daar psychologisch op voorbereid zijn. Daarmee maakt het project de indruk dat het precies weet waar het mee bezig is.

Bijhouden van veelgestelde vragen (FAQ)

Een *FAQ* (*Frequently Asked Questions*-document of veelgestelde vragen) kan één

van de meest waardevolle investeringen van een project zijn wat betreft educatief rendement. FAQ's zijn optimaal afgestemd op de feitelijke vragen van gebruikers en ontwikkelaars - dit in tegenstelling tot de vragen waarvan u verwacht dat ze worden gesteld. Daarom is een goed onderhouden FAQ vaak precies datgene waarnaar men op zoek is. De FAQ is vaak de eerste plaats waar gebruikers kijken wanneer ze een probleem tegenkomen, vaak zelfs nog voordat de officiële handleiding wordt geraadpleegd. Het is waarschijnlijk het document van uw project met de meest links vanuit andere websites.

Helaas kunt u geen FAQ opstellen aan het begin van het project. Goede FAQ's worden niet geschreven, ze groeien. Dit zijn per definitie reactieve documenten, die zich na verloop van tijd ontwikkelen in reactie op het praktische gebruik van de software. Omdat het onmogelijk is precies te voorspellen welke vragen mensen zullen stellen, is het ook onmogelijk een FAQ te schrijven vanaf nul.

Verspil daarom geen tijd door dit toch te proberen. Het kan echter wel handig zijn om een vrijwel geheel deel FAQ-sjabloon te maken, zodat mensen een duidelijk plek hebben waar ze vragen en antwoorden kunnen indienen nadat het project van start is gegaan. In deze fase is de belangrijkste eigenschap niet dat ze volledig zijn, maar dat ze bruikbaar zijn: als het toevoegen van FAQ's makkelijk is, zullen mensen het ook daadwerkelijk doen. (Goed onderhoud van FAQ's is een serieus en intrigerend probleem en wordt uitvoeriger behandeld in het gedeelte 'FAQ-manager' in Hoofdstuk 8, *Het managen van vrijwilligers.*)

Beschikbaarheid van documentatie

Documentatie moet op twee plaatsen beschikbaar zijn: online (direct vanaf de website) en in het downloadpakket van de software (zie het gedeelte 'Downloadpakketten maken' in Hoofdstuk 7, *Downloadpakketten maken, releases en dagelijkse ontwikkeling*). Documentatie moet online beschikbaar zijn in bladerbare vorm, omdat mensen documentatie vaak lezen *voordat* ze de software voor de eerste keer downloaden, om hen te helpen beslissen of ze de software sowieso willen downloaden. Zij moet echter ook bij de software zijn gevoegd, omdat bij het downloaden van het pakket in principe dat alles wat nodig is voor gebruik van het pakket aanwezig moet zijn (d.w.z. lokaal beschikbaar).

Voor online documentatie moet u ervoor zorgen dat er een link beschikbaar is die *alle* documentatie op één HTML-pagina laat zien (zet een opmerking als 'monolithisch', 'alles-in-één' of 'één grote pagina' bij de link, zodat mensen weten dat het even kan duren om de pagina te laden). Dit is van belang omdat mensen vaak naar een specifiek woord of zinsdeel willen zoeken in de hele documentatie. Over het algemeen weten mensen waar ze naar zoeken, ze zijn alleen vergeten in welke gedeelte het stond. Voor hen is het zeer frustrerend een HTML-pagina voor zich te krijgen met alleen de inhoud, vervolgens een andere pagina voor de inleiding en weer een andere pagina voor de installatie-instructies enz. Wanneer de pagina's zo zijn opgesplitst, is de zoekfunctie van de browser nutteloos. De indeling met afzonderlijke pagina's is handig voor mensen die al weten in welke gedeelte ze moeten zoeken, of die het gehele document van A tot Z willen doorlezen. Dit is echter *niet* de manier waarop de documentatie meestal wordt gebruikt. Het komt veel vaker voor dat iemand die vertrouwd is met de basisprincipes van de software terugkomt om te zoeken naar een specifiek woord of zinsdeel. Als u hun geen enkelvoudig en doorzoekbaar document biedt, maakt u het hun alleen maar moeilijker.

Documentatie voor ontwikkelaars

Documentatie voor ontwikkelaars wordt geschreven om programmeurs te helpen om de code te begrijpen, zodat ze deze kunnen repareren en uitbreiden. Dit is wat anders dan de ontwikkelaars*richtlijnen* die eerder aan bod zijn gekomen. Die hebben meer een sociaal dan een technisch karakter. Ontwikkelaarsrichtlijnen vertellen programmeurs hoe ze met elkaar moeten omgaan; ontwikkelaarsdocumentatie laat hun zien hoe ze met de code zelf om moeten gaan. Vaak worden de twee gemakshalve samengevoegd (zoals in het voorbeeld http://svn.collab.net/repos/svn/trunk/www/hacking.html dat eerder werd vermeld), maar het hoeft niet.

Hoewel ontwikkelaarsdocumentatie erg nuttig kan zijn, hoeft met de release niet te worden gewacht totdat deze klaar is. Zolang de oorspronkelijke auteurs beschikbaar (en bereid) zijn om vragen te beantwoorden over de code is dat voldoende om van start te gaan. Eerlijk gezegd is het feit dat men dezelfde vraag steeds weer moet beantwoorden de reden voor het schrijven van de documentatie. Maar ook voordat deze is geschreven, zullen mensen die vastbesloten zijn een bijdrage te leveren een manier vinden om de code te leren kennen. De drijfveer die ertoe leidt dat mensen de tijd nemen om een code te leren kennen, is het feit dat de code iets voor ze doet. Als mensen daar vertrouwen in hebben, zullen ze ook de tijd nemen dingen uit te zoeken. Als dit vertrouwen ontbreekt, zal zelfs de meest uitgebreide ontwikkelaarsdocumentatie ze niet overhalen om deel te nemen of te blijven.

Als u dus alleen tijd heeft om documentatie te schrijven voor één doelgroep, kies dan voor de gebruikers. Alle gebruikersdocumentatie is in feite ook ontwikkelaarsdocumentatie. Iedere programmeur die aan een deel van de software gaat werken, moet op de hoogte te zijn van het gebruik ervan. Wanneer programmeurs in een later stadium steeds dezelfde vragen stellen, kunt u de tijd nemen om enkele afzonderlijke documenten voor hen te schrijven.

Sommige projecten gebruiken 'wiki's' voor hun eerste documentatie, of zelfs als hun belangrijkste documentatie. In mijn ervaring werkt dit alleen als de wiki actief wordt geredigeerd door een aantal personen die het eens zijn over hoe de documentatie moet worden georganiseerd en welke 'toon' deze moet hebben. Raadpleeg voor meer informatie het gedeelte 'Wiki's' in Hoofdstuk 3, *Technische infrastructuur.*

Voorbeelden van output en screenshots

Als het project een grafische gebruikersinterface bevat of als het grafische of anderszins kenmerkende output genereert, plaats dan enkele voorbeelden hiervan op de website van het project. Voor een gebruikersinterface is dat een screenshot, voor de output kunnen dat screenshots of bestanden zijn. Beide spelen in op de behoefte van mensen aan directe bevrediging: een enkele screenshot kan overtuigender zijn dan eindeloze beschrijvende teksten en geklets op mailinglijsten, omdat een screenshot onbetwistbaar bewijs is dat de software echt werkt. Hij kan bugs bevatten, hij kan moeilijk te installeren zijn, de documentatie kan onvolledig zijn,

maar de screenshot bewijst dat als iemand genoeg moeite doet hij de software aan de gang kan krijgen.

Screenshots
Omdat het maken van screenshots een onmogelijke opgave kan lijken voordat u er werkelijk een paar hebt gemaakt, vindt u hier een paar instructies. Met behulp van Gimp (http://www.gimp.org/): ga naar File->Acquire->Screenshot, kies Single Window of Whole Screen en klik op OK. Uw eerstvolgende klik van de muis legt nu het venster of scherm vast waarop u klikt als afbeelding in Gimp. U kunt de afbeelding zo nodig bijsnijden of groter of kleiner maken, met behulp van de instructies op http://www.gimp.org/tutorials/Lite_Quickies/#crop.

Er zijn vele andere dingen die u op de website van het project kunt plaatsen als u daar de tijd voor heeft, of als dit om wat voor reden dan ook bijzonder nuttig blijkt: een nieuwspagina, een pagina met de projectgeschiedenis, een pagina met toepasselijke links, een zoekfunctie voor de website, een link voor donaties, etc. Geen van deze dingen is noodzakelijk tijdens de startfase, maar houdt u ze in gedachten voor de toekomst.

Canned Hosting
Er zijn enkele sites die vrije hosting en infrastructuur aanbieden voor open source-projecten: een weblocatie, versiecontrole, een bug tracker, een downloadlocatie, chatforums, regelmatige back-ups enz. De details variëren per site, maar de basisdiensten zijn bij allemaal gelijk. Door één van deze sites te gebruiken krijgt u veel dingen gratis. U geeft daar natuurlijk wel wat voor op, en dat is de fijnmazige controle over de gebruikerservaring. De hostingservice bepaalt welke software wordt gebruikt op de site en kan bepalen, of in ieder geval beïnvloeden, hoe de webpagina's van het project eruitzien en overkomen.
Raadpleeg de sectie 'Canned Hosting' in Hoofdstuk 3, *Technische infrastructuur* voor meer gedetailleerde informatie over de voor- en nadelen van canned hosting en een lijst met sites die dit aanbieden.

2.2 EEN LICENTIE KIEZEN EN TOEPASSEN

De bedoeling van dit gedeelte is om u een snelle, globale handleiding te geven voor het kiezen van een licentie. Lees Hoofdstuk 9, *Licenties, auteursrechten en patenten* voor de gedetailleerde wettelijke implicaties van de verschillende licenties en hoe de licentie die u kiest van invloed kan zijn op de mate waarin anderen uw software kunnen samenvoegen met andere vrije software.

Er bestaan vele, uitstekende licenties voor vrije software waaruit u kunt kiezen. De meeste daarvan hoeven hier niet aan bod te komen, omdat ze zijn opgesteld voor de specifieke juridische behoeften van een enkele organisatie of persoon en niet geschikt zijn voor uw project. We zullen ons hier beperken tot de meest gebruikte licenties. In de meeste gevallen zult u één van deze licenties willen gebruiken.

De 'Alles mag'-licenties
Als het voor u geen probleem is als de code van uw project wordt gebruikt in propriëtaire programma's kunt u een MIT/X-licentie gebruiken. Het is de eenvoudigste van verschillende minimale licenties, die niet veel meer doen dan het vastleggen van symbolische auteursrechten (zonder feitelijk het kopiëren te verbieden) en verklaren dat er geen garanties van toepassing zijn op de code. Raadpleeg voor meer informatie het gedeelte 'De MIT / X Window System-licentie'.

De GPL
Als u niet wilt dat uw code wordt gebruikt in propriëtaire programma's, gebruikt u de GNU General Public License (http://www.gnu.org/licenses/gpl.html). De GPL is waarschijnlijk de meest algemeen geaccepteerde licentie voor vrije software ter wereld. Dit is op zich al een groot voordeel, omdat veel potentiële gebruikers en mensen die eraan meewerken er reeds mee bekend zijn en daarom geen extra tijd hoeven te spenderen aan het lezen en begrijpen van uw licentie. Raadpleeg voor meer informatie het gedeelte 'De GNU General Public License' in Hoofdstuk 9, *Licenties, auteursrechten en patenten.*

Hoe u een licentie toepast op uw software
Zodra u een licentie hebt gekozen, moet u deze vermelden op de homepagina van uw project. U hoeft de feitelijke tekst van uw licentie hier niet te vermelden. Het is voldoende de naam van de licentie te noemen en een link op te nemen naar de volledige tekst van de licentie op een andere pagina.

Dit laat uw gebruikers weten onder welke licentie u van plan bent de software uit te brengen, maar het is niet voldoende voor wettelijke doeleinden. Daarvoor moet de software zelf voorzien zijn van de licentie. De standaardmanier om dit te doen is de volledige tekst van de licentie op te nemen in een bestand met de naam COPYING (of LICENSE) en een korte verwijzing op te nemen bovenaan ieder source-bestand, onder vermelding van de datum van het auteursrecht, de eigenaar en de licentie, en een vermelding van de plaats waar de volledige tekst van de licentie te vinden is.
Er zijn diverse manieren om dit te doen, we zullen hier slechts één voorbeeld bekijken. De GNU GPL vermeldt dat er bovenaan ieder source-bestand een mededeling moet worden opgenomen die er zo uit kan zien:

```
Copyright (C) <jaar>  <naam van de auteur>

Dit programma is vrije software. U kunt het distribueren en/of
wijzigen onder de voorwaarden van de GNU General Public License
zoals gepubliceerd door de Free Software Foundation, waarbij u
kunt kiezen uit versie 2 van de licentie of (naar eigen keus)
iedere latere versie.

Dit programma wordt gedistribueerd in de hoop dat het bruikbaar
is, maar ZONDER ENIGE VORM VAN GARANTIE, zelfs zonder de im-
pliciete garantie van VERKOOPBAARHEID of GESCHIKTHEID VOOR EEN
BEPAALD DOEL. Zie de GNU General Public License voor meer infor-
matie.
```

```
Samen met dit programma zou u een exemplaar ontvangen moeten
hebben van de GNU General Public License. Als dit niet het geval
is, schrijf dan naar Free Software Foundation, Inc., 59 Temple
Place, Suite 330, Boston, MA  02111-1307  USA
```

Er wordt niet specifiek vermeld dat het exemplaar van de licentie die u samen met het programma ontvangt het bestand COPYING is, maar over het algemeen wordt het daar ondergebracht. (U kunt de bovenstaande mededeling aanpassen en dit expliciet vermelden.) Dit voorbeeld geeft tevens een postadres waar een exemplaar van de licentie kan worden opgevraagd. Een andere veel gebruikte methode is om een link te geven naar een webpagina met de betreffende licentie. U kunt zelf het beste bepalen waar het meest permanente exemplaar van de licentie wordt bijgehouden en daar naar verwijzen. Dit zou een pagina op de website van uw project kunnen zijn. Over het algemeen hoeft de mededeling die u in ieder source-bestand opneemt niet exact gelijk te zijn aan de mededeling hierboven, zolang het begint met dezelfde vermelding van de houder en datum van de auteursrechten en de naam van de licentie en de plek waar de volledige licentie kan worden bekeken vermeld zijn.

2.3 DE TOON ZETTEN

Tot nu toe hebben we de eenmalige taken behandeld tijdens het opzetten van het project: het kiezen van een licentie, het maken van de website enz. De belangrijkste aspecten van het opstarten van een nieuw project zijn echter dynamisch. Het kiezen van een adres voor een mailinglijst is makkelijk. Ervoor zorgen dat de discussies op de lijst bij het onderwerp en productief blijven, is echter een hele andere zaak. Als het project voor buitenstaanders wordt geopend nadat het jaren binnenshuis is ontwikkeld, zullen de ontwikkelingsprocessen veranderen en zult u de bestaande ontwikkelaars moeten voorbereiden op die veranderingen.

De eerste stappen zijn de moeilijkste, omdat er nog geen gewoontes en verwachtingen zijn gecreëerd voor toekomstig gedrag. Stabiliteit binnen een project wordt niet bepaald door het formele beleid, maar door gedeelde, weinig tastbare inzichten die zich door de tijd ontwikkelen. Er zijn vaak ook schriftelijke regels, maar deze zijn vaak in wezen niet meer dan een aftreksel van de ongeschreven, zich steeds verder ontwikkelende onderlinge afspraken die sturing geven aan het project. Het schriftelijk beleid geeft geen definitie van het projectcultuur, maar eerder een beschrijving, en zelfs dan alleen nog bij benadering.

Er zijn een paar redenen waarom de dingen zo werken. Groei en een hoge omzet zijn niet zo schadelijk voor de ontwikkeling van sociale normen als wel gedacht wordt. Zo lang veranderingen niet te snel plaatsvinden, is er voldoende tijd voor instromers om te leren hoe alles in zijn werk gaat. Nadat ze dit hebben geleerd, zullen ze zelf ook helpen deze werkwijzen verder te versterken. Bedenk maar eens hoe kinderliedjes eeuwenlang hebben overleefd. Er zijn vandaag de dag kinderen die ongeveer dezelfde rijmpjes zingen als kinderen honderden jaren geleden, hoewel er nu geen kinderen in leven zijn die dat toen ook waren. Jongere kinderen horen de liedjes van oudere kinderen en zodra ze zelf ouder zijn, zullen zij ze op hun beurt ze weer voorzingen aan de andere jongere kinderen. Natuurlijk maken de kinderen geen deel uit van een bewust leerprogramma, maar de reden waarom liedjes de eeuwen overleven is dat ze regelmatig en herhaaldelijk worden overgedragen. De duur van een open source-softwareproject beslaat dan misschien geen eeuwen (dat weten we nu nog niet), maar de dynamiek van de overdracht is precies dezelfde. De verloopsnelheid is echter veel hoger en moet worden gecompenseerd door een meer actieve en doelbewuste overdracht.

Deze overdracht wordt ondersteund door het feit dat mensen, wanneer ze het project instappen, over het algemeen op zoek zijn naar sociale normen. Zo zitten mensen nou eenmaal in elkaar. Binnen iedere groep die bij elkaar wordt gehouden door een gemeenschappelijk inspanning gaan mensen instinctief op zoek naar gedragingen die hun kenmerken als onderdeel van die groep. Het doel van het vaststellen van deze gewoontes is ervoor te zorgen dat dit ‘groepsgedrag’ nuttig is voor het project. Zodra dit is vastgesteld houdt het zichzelf grotendeels in stand.

Hieronder volgen enkele voorbeelden van specifieke dingen die u kunt doen om goede gewoontes te creëren. De lijst is niet bedoeld als complete opsomming, maar als illustratie van het feit dat het creëren van een collectieve sfeer in de beginfase enorm nuttig kan zijn voor een project. Hoewel iedere ontwikkelaar feitelijk alleen in zijn kamertje aan het werk is, kunt u ze allemaal in belangrijke mate het gevoel geven dat ze allemaal samenwerken in dezelfde ruimte. Hoe meer ze dit gevoel krijgen, des te meer tijd ze aan het project zullen willen besteden. Ik heb deze specifieke voorbeelden gekozen omdat ik ze tegen ben gekomen in het Subversion-project (http://subversion.tigris.org/), waaraan ik heb deelgenomen en dat ik vanaf het allereerste begin heb geobserveerd. Ze zijn echter niet uniek voor Subversion. Situaties als deze komen voor in de meeste open source-projecten en moeten worden gezien als kansen om een project goed te beginnen.

Voorkom privédiscussies

Ook nadat u het project naar buiten hebt gebracht, zullen u en de andere oprichters er behoefte aan hebben moeilijke vragen onderling in besloten kring op te lossen. Dit is met name een feit in de eerste fase van het project, omdat er dan veel belangrijke beslissingen moeten worden genomen en er over het algemeen weinig vrijwilligers beschikbaar zijn die voldoende gekwalificeerd zijn om ze te beantwoorden. Alle overduidelijke nadelen van discussies via publieke lijsten doemen in alle mogelijke vormen voor u op: de vertraging die inherent is aan e-maildiscussies, de noodzaak voldoende tijd te geven aan het vormen van consensus, het gedoe met naïeve vrijwilligers die ten onrechte denken dat ze alles begrijpen (ieder project heeft zulke vrijwilligers, soms zijn ze de sterprogrammeurs van de toekomst, soms blijven ze voor altijd naïef), de persoon die niet kan begrijpen waarom u alleen probleem X wilt oplossen terwijl het overduidelijk deel uitmaakt van een groter probleem Y, en ga zo maar door. De verleiding om beslissingen achter gesloten deuren te nemen en ze later als voldongen feit te presenteren, of in ieder geval als resolute aanbeveling van een hechte en invloedrijke groep, zal in ieder geval zeer groot zijn.

Geef hier niet aan toe.

Hoe tijdrovend en zenuwslopend publieke discussies ook kunnen zijn, ze verdienen op de langere termijn bijna altijd de voorkeur. Belangrijke beslissingen in besloten kring nemen kan programmeurs bij uw project wegjagen. Geen enkele serieuze vrijwilliger blijft lang geïnteresseerd in een omgeving waar een geheime raad de belangrijke beslissingen neemt. Bovendien hebben openbare discussies positieve bijwerkingen die groter zijn dan het antwoord op de betreffende technische vraag:

- De discussie helpt nieuwe ontwikkelaars te trainen en op te leiden. U weet nooit hoeveel mensen meekijken bij de discussie. Zelfs als de meesten niet deelnemen, kunnen velen van hen het gesprek in stilte volgen en daarbij informatie verzamelen over de software.

- De discussie traint *u* in de kunst van het uitleggen van technische kwesties aan mensen die niet zo op de hoogte zijn van de software als u dat bent. Dit is een vaardigheid waarvoor veel oefening nodig is en u kunt dit niet oefenen door met mensen te praten die al weten wat u weet.

- De discussies en conclusies zullen altijd beschikbaar blijven in openbare archieven, waardoor voorkomen wordt dat toekomstige discussies in herhaling vallen. Zie het gedeelte 'Opvallend gebruik van archieven' in Hoofdstuk 6, *Communicatie*.

Ten slotte bestaat er natuurlijk ook nog de mogelijkheid dat iemand op de lijst een werkelijke bijdrage kan leveren aan de discussie door met een idee te komen dat u niet had verwacht. De kans hierop is moeilijk in te schatten. Het hangt af van de complexiteit van de code en de het vereiste specialisatieniveau van de deelnemers. Maar als anekdotisch bewijs acceptabel zou zijn, zou ik durven stellen dat de kans hierop groter is dan je intuïtief zou verwachten. In het Subversion-project waren wij (de oprichters) ervan overtuigd dat we te maken hadden met een aantal complexe problemen waarover we al maanden lang aan het tobben waren. Eerlijk gezegd betwijfelden we of er wel iemand op de nieuwe mailinglijst zou zijn die een daadwerkelijke bijdrage zou kunnen leveren aan de discussie. Daarom kozen we de weg van de minste weerstand en begonnen door middel van privé-e-mails te brainstormen over enkele technische kwesties, totdat een toeschouwer van het project[10] hier lucht van kreeg en vroeg of we de discussie wilden voeren via de publieke mailinglijst. We waren nogal sceptisch en daarom blij verrast door het aantal scherpzinnige opmerkingen en suggesties dat al vrij snel kwam. In veel gevallen hadden mensen ideeën waar wij niet eens op waren gekomen. Er bleek een aantal zeer slimme mensen op onze lijst te staan, die alleen maar zaten te wachten op een moment om toe te happen. Het klopt dat de daarop volgende discussies langer duurden dan wanneer we de discussie intern hadden gehouden, maar ze waren zoveel productiever dat het absoluut de moeite van de extra tijd waard was.

Zonder te vervallen in clichés als 'de groep is altijd slimmer dan het individu' (we hebben allemaal genoeg groepen gezien om wel beter te weten), moeten we wel erkennen dat er bepaalde activiteiten zijn waarin de groep beter is. Grootschalige controle door experts is hier één voorbeeld van, het snel genereren van een groot aantal ideeën een andere. De kwaliteit van de ideeën hangt natuurlijk af van de kwaliteit van het denkwerk dat eraan voorafging, maar u zult nooit weten welke soorten denkers u in huis heeft totdat u ze stimuleert met een uitdagend probleem.

Natuurlijk zijn er ook discussies die privé moeten blijven. In dit boek zullen we hiervan voorbeelden tegenkomen. Maar als uitgangspunt moet altijd gelden: *als er geen reden is om de discussie intern te voeren, moet hij publiekelijk worden gevoerd.*

Om dit te realiseren moet actie worden genomen. Het is niet voldoende om er alleen voor te zorgen dat uw eigen posts op de publieke lijst terechtkomen. U moet ook andermans onnodig privé gehouden onderonsjes een duwtje geven in de richting van de publieke lijst. Als iemand probeert een privédiscussie te beginnen, en er is geen reden de discussie privé te voeren, dan is het aan u om onmiddellijk een publieke discussie daarover te starten. U zou niet eens moeten reageren op het onderwerp voordat u erin bent geslaagd de discussie door te sluizen naar een publiek platform of hebt kunnen vaststellen dat het echt noodzakelijk is de discussie privé te voeren. Als u hier consequent in bent, zullen mensen snel doorhebben hoe het in zijn werk gaat en standaard de publieke forums gaan gebruiken.

Grofheden in de kiem smoren

Vanaf het allereerste begin van het publieke bestaan van uw project moet u een nultolerantiebeleid voeren ten aanzien van grof of beledigend gedrag op de forums. Nultolerantie betekent niet per se technische handhaving. U hoeft mensen niet van een mailinglijst te verwijderen wanneer ze andere deelnemers beledigen, of hun *commit access* af te nemen omdat ze kleinerende opmerkingen maken. (Theoretisch zou u uiteindelijk op dergelijke maatregelen moeten terugvallen, maar alleen nadat alle andere middelen hebben gefaald, waar aan het begin van het project per definitie geen sprake van kan zijn.) Nultolerantie betekent alleen dat incorrect gedrag niet onopgemerkt blijft. Wanneer iemand bijvoorbeeld een technisch bericht verstuurt waarin ook een *op de persoon gerichte* aanval is opgenomen tegen een andere ontwikkelaar van het project, is het noodzakelijk dat u de *persoonlijke* aanval *eerst* behandelt in uw reactie en pas daarna ingaat op de technische kwestie.

Helaas gebeurt het maar al te makkelijk en maar al te vaak dat constructieve discussies vervallen in destructieve ruzies. Mensen zeggen in e-mails dingen die ze iemand nooit recht in het gezicht zouden zeggen. Het onderwerp van de discussie versterkt dit effect alleen maar: voor technische kwesties geloven mensen vaak dat er slechts één correct antwoord is op de meeste vragen en dat onenigheid over dat antwoord alleen kan worden verklaard door onwetendheid of domheid. Het is maar een klein stapje tussen zeggen dat iemands idee stom is en zeggen dat de persoon zelf stom is. Het is in feite zelfs moeilijk te herkennen waar een technische discussie eindigt en de persoonlijke aanval begint. Dit is één van de redenen waarom drastische maatregelen of bestraffingen geen goed idee is. Wanneer u merkt dat zoiets gebeurt, plaatst u in plaats daarvan een post waarin u aangeeft hoe belangrijk het is de discussie vriendelijk te houden, zonder dat u iemand ervan beschuldigt opzettelijk gemeen te zijn. Dergelijke posts van de 'vriendelijke politie' hebben helaas de neiging te klinken als een kleuterjuf die een klas de les leest over goed gedrag:

laten we allereerst stoppen met (mogelijke) persoonlijke kritiek. Een voorbeeld hier-

van is zeggen dat J's ontwerp voor de Security Layer "naïef is en geen rekening houdt met de basisprincipes van computerbeveiliging." Dit kan wel of niet waar zijn, maar in beide gevallen is dit niet de manier om een discussie te voeren. J heeft zijn voorstel in goed vertrouwen gedaan. Als het tekortkomingen bevat, geef ze dan aan. Wij kunnen ze herstellen of een nieuw ontwerp maken. Ik weet zeker dat M J niet persoonlijk wilde beledigen, maar de woordkeuze was wat ongelukkig en we proberen de gesprekken hier opbouwend te houden.

En nu verder over het voorstel. Ik denk dat M gelijk had met ...

Hoe gekunsteld een dergelijke reactie ook kan klinken, het effect is opmerkelijk. Als u slecht gedrag consequent benoemt maar geen excuses of bevestiging verwacht van de beledigende partij, laat u mensen vrij om af te koelen en zich van hun betere kant te laten zien door zich de volgende keer fatsoenlijk te gedragen. En dat zullen ze dan ook doen. Eén van de geheimen om dit tot goed te doen, is ervoor te zorgen dat de discussie zelf niet het belangrijkste thema wordt. Het moet altijd een zijdelingse opmerking zijn, een korte inleiding tot het belangrijke deel van uw reactie. Merk terloops op "dat we discussies hier niet op deze manier voeren," maar stap dan over op het werkelijke onderwerp, zodat u mensen iets geeft waar het werkelijk over gaat en waar ze op kunnen reageren. Als iemand protesteert dat hij uw berisping niet heeft verdiend, weiger u dan mee te laten slepen in de woordenwisseling. U kunt óf helemaal niet reageren (als u denkt dat ze alleen stoom hoeven af te blazen en niet echt een antwoord nodig hebben), óf u zegt, voordat u overgaat tot het hoofdonderwerp, dat het u spijt dat u te sterk hebt gereageerd en dat het moeilijk is nuances te zien in e-mails. Dring nooit aan op een iemands bevestiging dat hij of zij zich incorrect heeft gedragen, publiekelijk noch privé. Als ze er zelf voor kiezen zich te verontschuldigen is dat geweldig, maar dit van ze eisen kan alleen maar tot wrok leiden.

Het algemene doel is om ervoor te zorgen dat goede omgangsvormen worden gezien als behorend bij de 'groepscultuur'. Dit is goed voor het project, omdat ontwikkelaars kunnen worden afgeschrikt (zelfs bij projecten waar ze graag aan zouden meewerken) door vuilbekkerij. Misschien weet u niet eens dat ze worden afgeschrikt. Iemand kan stilletjes lid zijn van de mailinglijst, tot de ontdekking komen dat je wel een erg dikke huid moeten hebben voor dit project en besluiten helemaal niet aan het project deel te willen nemen. De toon van de forums vriendelijk houden is een overlevingsstrategie voor de lange termijn, en veel makkelijker te realiseren zolang het project nog klein is. Zodra het onderdeel geworden is van de cultuur hoeft u niet de enige meer te zijn die dit propageert. Het zal door alle deelnemers in stand worden gehouden.

De code nakijken op verdachte onderdelen

Eén van de beste manieren om een productieve ontwikkelaarsgemeenschap te creëren is mensen naar elkaars code te laten kijken.

Er is enige technische infrastructuur voor nodig om dit doeltreffend te realiseren, in het bijzonder moet 'commit e-mails' zijn ingeschakeld. Raadpleeg voor meer informatie het gedeelte 'Commit e-mails'.

Het effect van commit e-mails is dat iedere keer wanneer iemand een wijziging doorvoert aan de broncode er een e-mail wordt verstuurd met het logbericht en de 'diff's' van de wijziging (zie diff in het gedeelte 'Woordenlijst versiecontrole'). Code review is de gewoonte om commit e-mails na te kijken op het moment dat ze binnenkomen, op zoek naar bugs of mogelijke verbeteringen.[11]

Code review heeft een aantal voordelen. Het is het meest voor de hand liggende voorbeeld van controle door experts in de wereld van open source en draagt direct bij aan de kwaliteit van de software. Iedere bug die binnensluipt in een onderdeel van de software is daar terechtgekomen doordat hij is gecommit en niet is ontdekt. Dus hoe meer mensen naar de doorgevoerde wijzigingen kijken des te minder bugs er binnen kunnen komen. Code review heeft echter ook een indirect voordeel: het laat mensen weten dat wat ze doen van belang is, omdat niemand de tijd zou nemen iets na te kijken tenzij die persoon belang heeft bij het effect. Mensen werken het beste wanneer ze weten dat anderen de tijd nemen om hun werk te evalueren. Reviews moeten publiek toegankelijk zijn. Zelfs in situaties waarbij ik in dezelfde ruimte zat als de andere ontwikkelaars en één van ons een commit had gemaakt, zorgden we ervoor om deze niet mondeling te evalueren maar in plaats daarvan de review naar de ontwikkelaarsmailinglijst te sturen. Iedereen heeft er baat bij dat de review daadwerkelijk zichtbaar is. Mensen volgen de commentaren en vinden daar soms onjuistheden in. Maar zelfs wanneer dat niet het geval is, herinnert het hun in ieder geval aan het feit dat de review een te verwachten standaardactiviteit is, zoals afwassen en grasmaaien.

In het Subversion-project was code review in het begin geen vaste gewoonte. Niemand kon garanderen dat iedere commit geëvalueerd zou worden, hoewel mensen soms wel wijzigingen bekeken als zij speciaal geïnteresseerd waren in dat deel van de code. Er slopen bugs in de software die echt opgespoord hadden kunnen en moeten worden. Een ontwikkelaar genaamd Greg Stein, die door voorgaande projecten bekend was met het belang van code review, besloot dat hij een voorbeeld zou stellen door *iedere regel van iedere commit* die in de code werd ingevoerd na te kijken. Op iedere commit, van wie dan ook, volgde al snel een e-mail naar de ontwikkelaarslijst van Greg, waarin de commit grondig werd ontleed en mogelijke problemen geanalyseerd werden. Van tijd tot tijd kreeg een bijzonder vernuftig stukje code een complimentje. Meteen vanaf het begin onderving hij bugs en niet-optimale code, die anders ongemerkt in de software zouden zijn geslopen. Opvallend genoeg klaagde hij nooit over het feit dat hij de enige was die iedere commit evalueerde, ook al koste het hem veel tijd. Maar hij liet niet na code review de hemel in te prijzen wanneer hij maar de kans kreeg. Al snel begonnen ook anderen, onder wie ikzelf, regelmatig commits te evalueren. Wat was onze motivatie daarvoor? Het was niet zo dat we wel moesten omdat Greg er voor zorgde dat wij ons zo schaamden. Maar hij had wel laten zien dat het evalueren van code een waardevolle tijdsbesteding was en dat iemand net zoveel aan het project kon bijdragen door andermans code te evalueren als door het zelf schrijven van nieuwe code. Toen hij dit eenmaal had aangetoond, werd dit de normale manier van werken, zelfs in die mate dat als er geen reactie kwam op een commit, de committer zich zorgen begon te maken en zelfs op de mailinglijst ging navragen of iemand al kans had gezien de commit te evalueren. Toen Greg later een andere baan kreeg, waardoor hij veel

minder tijd had voor Subversion, moest hij stoppen met de reviews. Tegen die tijd had de gewoonte echter zo wortel geschoten bij de rest van ons, dat het leek alsof we nooit iets anders hadden gedaan.

Begin vanaf de allereerste commit met reviews. Het soort problemen dat het makkelijkst op te sporen is door het evalueren van diff's zijn beveiligingskwetsbaarheden, *memory leaks*, ontoereikende uitleg of API-documentatie, off-by-one-fouten, caller-callee discipline mismatches en andere problemen waarvoor maar weinig context nodig is om ze op te kunnen sporen. Zelfs grootschaliger kwesties, zoals het feit dat verzuimd is herhaalde patronen samen te vatten naar één enkele locatie, zijn echter makkelijker op te sporen door iemand die regelmatig reviews doet, omdat diff's uit het verleden informatie geven voor huidige diff's.

Maakt u zich niet druk als u niets kunt vinden waar u commentaar op kunt leveren, of als u niet genoeg weet van ieder onderdeel van de code. Over iedere commit is over het algemeen wel iets te zeggen. Zelfs als u niets kunt vinden om vragen over te stellen, is er misschien wel iets waar u uw complimenten over wilt uitspreken. Het belangrijkste is dat u duidelijk maakt aan alle committers dat wat zij doen wordt gezien en begrepen. Natuurlijk ontslaat het doen van code reviews de programmeurs niet van de verantwoordelijkheid om hun eigen wijzigingen te evalueren en te testen voordat ze ze doorvoeren. Niemand mag het aan de code reviewers overlaten dingen te ontdekken die hij zelf had moeten ontdekken.

Wees bij het openen van een aanvankelijk gesloten project alert op de impact van de wijziging

Als u overgaat tot het openen van een bestaand project waarbinnen de ontwikkelaars eraan zijn gewend te werken in een gesloten source-omgeving, moet u ervoor zorgen dat iedereen begrijpt dat er een grote verandering aan staat te komen. Zorg er ook voor dat u begrijpt hoe zij zich voelen.

Probeert u zich voor te stellen hoe de situatie er voor hen uitziet. Voorheen werden alle beslissingen over de code en het ontwerp gemaakt binnen een groep van programmeurs die allemaal ongeveer evenveel van de software afwisten, die allemaal onder dezelfde druk stonden van het management en die op de hoogte waren van elkaars sterke en zwakke kanten. Nu vraagt u hen hun code bloot te stellen aan de kritische blik van willekeurige onbekenden, die hun mening alleen op de code baseren, zonder zich bewust te zijn van de kwesties binnen het bedrijf die misschien tot bepaalde beslissingen hebben geleid. Deze onbekenden zullen een heleboel vragen stellen; vragen die de oude groep ontwikkelaars heftig met de neus op de feiten drukken dat de documentatie waaraan ze zo hard hebben gewerkt nog steeds niet goed is (dit is onvermijdelijk). En alsof dat nog niet genoeg is, zijn de nieuwkomers ook nog eens onbekende, anonieme entiteiten. Als één van uw ontwikkelaars al onzeker was over zijn vaardigheden, stelt u zich dan eens voor hoe hij zich zal voelen wanneer nieuwkomers vallen over fouten in de code die hij heeft geschreven en, erger nog, dat ook nog in aanwezigheid van zijn collega's. Behalve wanneer u een team van perfecte programmeurs hebt, is dit onvermijdelijk. Sterker nog, dit is waarschijnlijk bij al uw programmeurs het geval. Dat is niet omdat ze slechte programmeurs zijn. De reden is gewoon dat ieder programma van een bepaalde omvang bugs bevat en een review door collega-programmeurs zal er daarvan altijd een aantal aan het licht brengen (zie het gedeelte 'De code nakijken op verdachte onderdelen' eerder in dit hoofdstuk). Tegelijkertijd zal er in het begin niet veel code van de nieuwkomers zelf worden geëvalueerd door de andere programmeurs, omdat ze pas zelf code kunnen gaan schrijven als ze wat meer vertrouwd zijn met het project. Op uw ontwikkelaars komt dit over alsof er alleen maar kritiek op hen afkomt en dat ze zelf geen kritiek kunnen leveren. Daardoor bestaat het gevaar dat de oude garde zich in gaat graven.

De beste manier om dit te voorkomen is om iedereen van tevoren te waarschuwen over wat er komen gaat, het uit te leggen, te vertellen dat het aanvankelijke ongemakkelijke gevoel volkomen normaal is, en te verzekeren dat het daarna beter zal gaan. Sommige van deze waarschuwingen zouden in een privégesprek gegeven moeten worden, voordat het project opengesteld wordt. Maar het kan ook nuttig zijn om mensen op de publieke mailinglijst eraan te herinneren dat dit een nieuwe manier van werken is voor het project, en dat het even kan duren voordat iedereen eraan is gewend. Het beste dat u kunt doen, is zelf het goede voorbeeld geven. Als u ziet dat uw ontwikkelaars niet genoeg vragen van nieuwkomers beantwoorden, helpt het nauwelijks als u ze zegt dat ze meer moeten antwoorden. Ze voelen misschien nog niet aan waarop ze wel en waarop ze niet moeten reageren. Het kan echter ook zijn dat ze nog niet goed hebben geleerd prioriteiten te stellen tussen het codeerwerk en de nieuwe verplichting tot externe communicatie. De beste manier om ze beter te laten participeren is door zelf te participeren. Wees actief op de publieke mailinglijst en zorg ervoor dat u ook enkele vragen beantwoordt. Wanneer u niet de expertise heeft om een vraag te beantwoorden, kunt u deze zichtbaar overdragen aan een ontwikkelaar die dat wel heeft. Let er daarbij op dat hij de vraag beantwoordt of in ieder geval reageert. Het is heel normaal dat de oude garde van ontwikkelaars geneigd is terug te vallen in privédiscussies; ze zijn niet anders gewend. Zorg ervoor dat u ook lid bent van eventuele interne mailinglijsten waar dit zou kunnen gebeuren, zodat u kunt vragen om dergelijke discussies naar de publieke lijst te verplaatsen.

Er zijn ook nog andere zaken die op de lange termijn een rol spelen bij het openen van voorheen gesloten projecten. Hoofdstuk 4, *Sociale en politieke infrastructuur* behandelt hoe betaalde en onbetaalde ontwikkelaars op succesvolle wijze kunnen samenwerken en Hoofdstuk 9, *Licenties, auteursrechten en patenten* bespreekt de noodzaak van juridische zorgvuldigheid bij het openen van een particuliere codebase die software kan bevatten die is geschreven door of 'eigendom' is van derden.

2.4 AANKONDIGEN

Zodra het project presentabel is — niet perfect, alleen persentabel — is het tijd om het aan de buitenwereld bekend te maken. In feite is dit een heel eenvoudig proces: ga naar http://freshmeat.net/, klik op Submit in de bovenste navigatiebalk en vul een formulier in waarin u uw nieuwe project bekendmaakt. Freshmeat is de plaats waar iedereen kijkt voor aankondigingen van nieuwe projecten. U hoeft slechts de aandacht te trekken van een paar mensen en uw project wordt via mond-tot-mondreclame bekend.

Als u op de hoogte bent van mailinglijsten of nieuwsgroepen waar een aankondiging van uw project binnen de discussie en het interessegebied valt, kunt u daar een post plaatsen. Zorg er echter voor om niet meer dan één post per forum te plaatsen en de mensen naar uw eigen forums door te sturen voor verdere discussie (door het reply-to-veld in te stellen). De post moet kort en to the point zijn:

```
To: discuss@lists.example.org
Subject: [ANN] Scanley full-text indexer project
Reply-to: dev@scanley.org

Dit is een eenmalige post om de lancering aan te kondigen van
het Stanley-project, een open source full-text indexer en zoek-
machine met een 'rich API', voor gebruik door programmeurs, die
een zoekfunctie biedt voor een groot aantal tekstbestanden.
Scanley is nu een werkende code, wordt actief ontwikkeld en is
op zoek naar zowel ontwikkelaars als testers.

Website: http://www.scanley.org/

Functies:
   - Zoekt in platte tekst, HTML en XML
   - Zoeken op woorden of zinnen
   - (gepland) Fuzzy matching
   - (gepland) Voortdurende updates van de indexen
   - (gepland) Indexing van websites op afstand

Vereisten:
   - Python 2.2 of hoger
   - Voldoende schijfruimte voor de indexen (ongeveer 2x de om-
     vang van de oorspronkelijke gegevens)

Voor meer informatie brengt u een bezoek aan scanley.org.

Hartelijk dank,
-J. Random
```

(Raadpleeg het gedeelte 'Publiciteit' in Hoofdstuk 6, *Communicatie voor advies over de aankondiging van volgende releases en andere gebeurtenissen*.)
Er is in de wereld van de vrije software veel discussie gaande over of een project pas gepubliceerd mag worden wanneer er werkende code beschikbaar is, of dat het handig kan zijn een project al te openen tijdens de ontwerp- en discussiefase. Ik heb altijd gedacht dat het belangrijk was om te beginnen met werkende code, dat dat het onderscheid maakte tussen een succesvol project en speeltje, en dat serieuze ontwikkelaars zich alleen in zouden willen laten met software die al iets concreets zou kunnen.

Dit bleek niet het geval te zijn. Bij het Subversion-project begonnen we met een ontwerpdocument, een clubje geïnteresseerde en goed communicerende ontwikkelaars, een heleboel poeha en geen enkele werkende code. Tot mijn grote verrassing trok het project vanaf het eerste begin actieve deelnemers aan en op het moment dat we beschikten over werkende code waren er al heel wat vrijwillige ontwikkelaars nauw betrokken bij het project. Subversion is niet het enige voorbeeld. Het Mozilla-project werd ook zonder werkende code gelanceerd en is nu een succesvolle en populaire webbrowser.

Door dit soort voorbeelden moet ik dus afstand nemen van de aanname dat werkende code een absolute noodzaak is voor het lanceren van een project. Werkende code is nog steeds één van de beste uitgangspunten voor succes en het is een goede vuistregel om te wachten met het aankondigen van het project totdat u werkende code hebt. Er zijn echter omstandigheden waarin het lanceren van het project in een eerder stadium verstandig is. Ik denk wel dat minimaal een goed ontwikkeld ontwerpdocument, of anders één of andere vorm van een codekader noodzakelijk is. Natuurlijk kan dit worden gewijzigd aan de hand van publieke reacties, maar er moet wel iets concreets zijn, iets dat tastbaarder is dan goede bedoelingen, iets waar mensen hun tanden in kunnen zetten.

Wanneer u een project aankondigt, verwacht dan niet meteen dat er hordes vrijwilligers op af komen. Meestal is het gevolg van een aankondiging dat u een paar sporadische verzoeken om informatie krijgt en dat wat meer mensen lid worden van uw mailinglijst, maar verder blijft alles ongeveer bij het oude. Na verloop van tijd zult u echter merken dat het aantal programmeurs en gebruikers geleidelijk toeneemt. De bekendmaking is meer het planten van een zaadje. Het kan lang duren voordat het nieuws bekend wordt. Als het project de betrokkenen consequent beloont, zal het nieuws uiteindelijk steeds bekender worden. Mensen willen het namelijk graag aan anderen laten weten als ze iets goeds hebben gevonden. Als alles goed gaat zal de dynamiek van de exponentiële communicatienetwerken uw project langzaam transformeren in een complexe gemeenschap, waar u niet per se iedereen kent en niet langer iedere conversatie kunt bijhouden. De volgende hoofdstukken gaan over het werken in een dergelijke omgeving.

10| We zijn nog niet aangekomen bij het gedeelte over het bedanken van mensen, maar hier vast een oefening over wat ik later uit zal leggen: de naam van degene die dit heeft opgemerkt is Brian Behlendorf. Hij was ook degene die het algemene belang aangaf van het publiek houden van alle discussies tenzij er een bijzondere reden voor privacy is.

11| Hoe dan ook, dit is hoe code review normaal gesproken wordt gedaan in open source-projecten. In meer gecentraliseerde projecten kan 'code review' ook betekenen dat meerdere mensen bij elkaar komen en prints van de broncode bestuderen, op zoek naar specifieke problemen of patronen.

3

TECHNISCHE INFRASTRUCTUUR

Open source-softwareprojecten zijn afhankelijk van technologieën die het selectief bemachtigen en de selectieve integratie van informatie ondersteunen. Hoe vaardiger u bent in het gebruik van deze technologieën en in het overreden van anderen om ze te gebruiken, des te geslaagder uw project zal zijn. Naarmate het project groeit, gaat dit steeds meer op. Wat ervoor zorgt dat een open source-project niet instort onder het gewicht van Brooks' Law,[12] is goede informatie. Brooks' Law luidt dat menskracht toevoegen aan een softwareproject dat al achterloopt, het project alleen nog maar meer vertraagt. Fred Brooks merkte op dat de complexiteit van een project toeneemt met het kwadraat van het aantal deelnemers. Wanneer er slechts enkele mensen bij een project betrokken zijn, kan iedereen makkelijk met elkaar communiceren. Wanneer er honderden mensen bij betrokken zijn, is het niet langer mogelijk dat iedereen voortdurend op de hoogte is van wat de anderen doen. Als goed management van een open source-softwareproject inhoudt dat iedereen het gevoel heeft dat iedereen in één kamer samenwerkt, dan dringt de volgende vraag zich op: wat gebeurt er als iedereen tegelijk gaat praten in die volle kamer?

Dit probleem is niet nieuw. In niet-figuurlijke volle kamers heet de oplossing *parlementaire procedure:* formele richtlijnen voor het houden van realtime discussies in grote groepen, voorkomen dat afwijkende meningen verloren gaan in de vloed van 'ik-ook'-opmerkingen, hoe subcommissies te vormen, hoe te herkennen wanneer er een beslissing genomen is enz. Een belangrijk onderdeel van de parlementaire procedure is het specificeren van hoe de wisselwerking is tussen de groep en het informatiemanagementsysteem. Sommige opmerkingen worden gemaakt 'voor het verslag', andere niet. Het verslag zelf is het voorwerp van manipulatie. Het wordt geacht geen letterlijke transcriptie te zijn van wat er is gebeurd, maar een weergave van wat de groep bereid is overeen te komen dat gebeurd is.Het verslag is niet monolithisch, maar neemt voor verschillende doeleinden verschillende gedaanten aan. Het bevat de notulen van afzonderlijke vergaderingen, de volledige verzameling van alle notulen van alle vergaderingen, samenvattingen, agenda's plus aantekeningen, commissieverslagen, rapporten van afwezige correspondenten, actielijsten enz.
Omdat internet fysiek geen kamer is, hoeven we ons geen zorgen te maken over die aspecten van de parlementaire procedure die ervoor moeten zorgen dat andere mensen zwijgen als iemand het woord voert. Maar als het gaat om informatiema-

nagementtechnieken, dan zijn goed geleide open source-projecten parlementaire procedures op steroïden. Aangezien vrijwel alle communicatie bij open source-projecten schriftelijk plaatsvindt, zijn er uitgebreide systemen ontwikkeld: voor het correct routeren en labelen van gegevens, voor het minimaliseren van herhalingen om pseudoafwijkingen te voorkomen, voor de opslag en het terughalen van gegevens, voor het corrigeren van foutieve of overbodige informatie en voor het met elkaar in verband brengen van ongelijksoortige stukjes informatie wanneer relaties worden gezien. Actieve deelnemers aan open source-projecten maken zich deze technieken vaak eigen en voeren dikwijls ingewikkelde handmatige taken uit om te zorgen dat informatie juist gerouteerd wordt. Maar de hele operatie hangt uiteindelijk af van geavanceerde softwarematige ondersteuning. De communicerende media zouden het routeren, labelen en vastleggen zo veel mogelijk zelf moeten doen en de informatie op de makkelijkst mogelijke manier beschikbaar stellen aan de mens. In de praktijk moet de mens natuurlijk nog steeds op veel punten tijdens het proces ingrijpen. Het is belangrijk dat de software ook dergelijke interventies eenvoudig toelaat. Over het algemeen is het echter zo dat als de mens de informatie meteen bij de eerste invoering in het systeem nauwgezet labelt en routeert, de software zo geconfigureerd zou moeten zijn dat optimaal gebruik kan worden gemaakt van die metagegevens.

De adviezen in dit hoofdstuk zijn sterk praktisch gericht, gebaseerd op ervaringen met specifieke software en gebruikspatronen. Het is echter belangrijk om niet alleen een bepaald arsenaal aan technieken te onderwijzen. Ook moet, aan de hand van veel kleine voorbeelden, worden gedemonstreerd welke algemene houding de beste stimulans is voor goed informatiemanagement in uw project. Deze houding behelst een combinatie van technische vaardigheden en *people skills*. De technische vaardigheden zijn essentieel, want informatiemanagementsoftware vereist immers altijd configuratie, plus een zekere hoeveelheid doorlopend onderhoud en bijstellen naar aanleiding van nieuwe behoeften die ontstaan (zie bijvoorbeeld de uiteenzetting over de aanpak van projectgroei in het gedeelte ‘De bug tracker voorfilteren’ verderop in dit hoofdstuk. De people skills zijn vereist omdat ook menselijke communicatie onderhoud vereist: het is niet altijd meteen duidelijk hoe deze gereedschappen ten volle benut kunnen worden en in sommige gevallen bestaan er tegenstrijdige conventies binnen een project (zie bijvoorbeeld de uiteenzetting over het plaatsen van `Reply-to`-headers in uitgaande posts van mailinglijsten in het gedeelte ‘Mailinglijsten’). Iedereen die bij het project betrokken is, moet op het juiste moment en op de juiste manieren worden gestimuleerd om zijn steentje bij te dragen aan het goed georganiseerd houden van de projectinformatie. Hoe meer de contribuant betrokken is, des te gecompliceerder en gespecialiseerder de technieken zijn die hij geacht mag worden te leren.

Informatiemanagement kent geen pasklare oplossingen; er zijn te veel variabelen. Een voorbeeld: je hebt eindelijk alles precies zo geconfigureerd als je wilt en de meeste betrokkenen zitten op één lijn. Dan blijkt dat door de groei van het project een aantal van de werkwijzen niet opgeschaald kan worden. Een ander voorbeeld: de projectgroei stabiliseert en de ontwikkelaar en de gebruikersgemeenschappen ontwikkelen een prettige relatie met de technische infrastructuur. Maar dan ontwikkelt iemand een geheel nieuwe informatiemanagementservice en al gauw vragen nieuwkomers waarom uw project daar geen gebruikt van maakt. Dit gebeurt momenteel dikwijls met open source-softwareprojecten die dateren van voor de uitvinding van de wiki (zie http://en.wikipedia.org/wiki/Wiki). Bij veel vraagstukken is het een kwestie van opvatting, waarbij het gemak van diegenen die de informatie produceren wordt afgewogen tegen het gemak van diegenen die de informatie consumeren, of waarbij de tijd die nodig is om informatiemanagementsoftware te configureren wordt afgewogen tegen het nut ervan voor het project.

Kijk uit voor de verleiding om te overautomatiseren, dat wil zeggen dingen te automatiseren die eigenlijk de aandacht van mensen vereisen. Technische infrastructuur is belangrijk, maar waardoor een open source-softwareproject werkt, is serieuze zorg – en de intelligente uitwerking daarvan door de betrokken personen. De technische infrastructuur dient voornamelijk als prettig instrument waarmee de mensen hun zorg kunnen uitwerken.

3.1 PROJECTBENODIGDHEDEN

De meeste open source-projecten bieden voor het informatiemanagement ten minste een minimale set van standaardtools aan:

Website
: Voornamelijk een gecentraliseerd eenrichtingskanaal met informatie van het project richting het publiek. De website kan ook fungeren als administratieve interface voor andere projecttools.

Mailinglijsten
: Doorgaans het meest actieve communicatiemiddel van het project en het ‘medium voor verslaglegging’.

Versiebeheer
: Hiermee kunnen ontwikkelaars codewijzigingen makkelijk beheren, met inbegrip van *reverting* en *change porting*. Hierdoor kan iedereen zien wat er met de code gebeurt.

Bugs volgen
: Stelt ontwikkelaars in staat bij te houden waaraan ze precies werken, zaken met elkaar af te stemmen en releases te plannen. Hierdoor kan iedereen informeren naar de status van bugs en de informatie (bijv. *reproduction recipes*) over bepaalde bugs. Kan, naast het bijhouden van bugs, ook worden gebruikt voor taken, releases, nieuwe functies enz.

Realtime chat
: Een plek voor snelle, luchtige discussies en vraag-en-antwoorduitwisselingen. Wordt niet altijd volledig gearchiveerd.

Elke tool uit dit lijstje voorziet in een bepaalde behoefte, maar hun functies zijn ook onderling gerelateerd. Er moet dan ook voor worden gezorgd dat de tools kunnen

samenwerken. Verderop zullen we onderzoeken hoe we dit kunnen bereiken en – belangrijker – hoe we mensen ertoe kunnen bewegen ze te gebruiken. De website wordt pas op het einde behandeld, aangezien deze meer functioneert als een bindmiddel voor de andere componenten dan als zelfstandige tool.
Je kunt je waarschijnlijk veel moeite van het kiezen en configureren van deze tools besparen door een *canned hosting* site te gebruiken: een server die de benodigde tools voor het runnen van een open source-softwareproject in sjablonen levert . Zie de paragraaf 'Canned Hosting' verderop in dit hoofdstuk over de voor- en nadelen van *canned hosting*.

3.2 MAILINGLIJSTEN

Mailinglijsten zijn van vitaal belang voor de projectcommunicatie. Als een gebruiker te maken krijgt met een medium anders dan de webpagina's, is dat zeer waarschijnlijk een mailinglijst van het project. Maar voordat ze te maken krijgen met de mailinglijst zelf, worden ze geconfronteerd met de interface van de mailinglijst, dat wil zeggen het mechanisme om deel te nemen aan *(subscribe to)* de lijst. Daarmee zijn we beland bij regel nummer één betreffende mailinglijsten:

Probeer mailinglijsten niet handmatig te beheren: haal list management software in huis.

Het is verleidelijk om dit uit te stellen. In het begin lijkt het opzetten van list management software namelijk overbodige luxe. Het handmatig beheren van kleine lijsten met weinig berichten lijkt verleidelijk eenvoudig: je hoeft alleen maar een *subscribe to*-adres in te stellen dat naar jou forwardt en wanneer iemand daar naartoe mailt, voeg je zijn e-mailadres toe aan een tekstbestand dat alle adressen op de lijst bevat (of je verwijdert er een adres uit). Kan het nog simpeler?

De waarheid is echter dat goed beheer van mailinglijsten – en mensen verwachten nu eenmaal goed beheer – helemaal niet eenvoudig is. Het gaat niet alleen om het op verzoek tot subscriben en unsubscriben van gebruikers. Het gaat ook om moderatie ter voorkoming van spam, het aanbieden van mailinglijsten in *digest* versus *message-by-message*-vorm, het aanbieden van standaardlijsten en projectinformatie door middel van *autoresponders* en andere zaken. Een persoon die een subscription-adres in de gaten houdt, kan slechts uiterst minimale functionaliteit leveren, en dan nog lang niet zo betrouwbaar en consequent als met software.
Moderne list management software beschikt meestal ten minste over de volgende functionaliteiten:

Subscriben via zowel e-mail als website
: Als een gebruiker zich abonneert op een lijst, dient hij *meteen* een geautomatiseerd welkomstbericht als antwoord te krijgen, met informatie waarop hij zich heeft geabonneerd, hoe verder om te gaan met de mailinglijstsoftware en (meest belangrijk) hoe te unsubscriben. Dit automatiseerde antwoord kan worden aangepast met projectspecifieke informatie zoals, vanzelfsprekend, de website van het project, de plek waar de FAQ's staan enz.

Abonneren in digest-modus of message-by-message.
: In digest-modus ontvangt de abonnee per dag één e-mail, met alle lijstactiviteiten van die dag. Voor mensen die een lijst niet nauwgezet volgen en er niet actief aan deelnemen, verdient de digest-modus vaak de voorkeur. Immers, ze kunnen in een keer alle onderwerpen doorkijken en worden niet afgeleid door e-mails die op willekeurige momenten van de dag binnenkomen.

Moderatiefuncties
: Moderatie betekent in deze context het controleren van *posts* om er zeker van te zijn dat ze a) geen spam zijn en b) over het onderwerp gaan, voordat ze worden verzonden naar de hele lijst. Moderatie vereist noodzakelijkerwijs mensenwerk, maar software kan dat werk enorm vereenvoudigen. Op moderatie komen we later nog terug.

Administratieve interface
: Hiermee kan een beheerder - onder andere - verouderde adressen eenvoudig verwijderen. Dit kan urgent worden als het adres van een ontvanger automatisch antwoorden gaat versturen als 'Dit adres bestaat niet meer' in antwoord op iedere post naar de lijst. (Sommige mailinglijstsoftware herkent dit fenomeen en unsubscribet deze persoon vervolgens automatisch.)

Manipulatie van headers
: Veel mensen hebben geavanceerde filter- en antwoordregels ingesteld in hun mail client. Mailinglijstsoftware is in staat om bepaalde standaard-headers toe te voegen en te manipuleren, waarvan deze mensen voordeel hebben (meer details verderop).

Archivering
: Alle posts naar een beheerde lijst worden opgeslagen en beschikbaar gemaakt op internet. Een andere mogelijkheid die geboden wordt door sommige mailinglijstsoftwarepakketten is een speciale interface om een externe archiveringstool in te pluggen, zoals MHonArc (http://www.mhonarc.org/). Zoals in het gedeelte 'Opvallend gebruik van archieven' in *Hoofdstuk 6, Communicatie* wordt besproken, is archivering van cruciaal belang.

De bedoeling van het bovenstaande is om duidelijk te maken dat het beheer van mailinglijsten een ingewikkelde zaak is, waarover veel is nagedacht en ten aanzien waarvan de meeste problemen zijn opgelost. U hoeft beslist geen expert op dit gebied te worden, maar u moet wel beseffen dat u altijd meer kunt leren en dat bij het runnen van een open source-softwareproject het beheer van mailinglijsten van tijd tot tijd de aandacht zal opeisen. Hieronder bespreken we enkele van de meest voorkomende aspecten van het configureren van mailinglijsten.

Spampreventie

Vanaf het moment dat deze zin is geschreven en het moment dat hij gelezen wordt, is het spamprobleem op internet waarschijnlijk twee keer zo groot geworden, althans zo zal het aanvoelen. Nog niet zo heel lang geleden kon je een mailinglijst runnen zonder enige spampreventiemaatregelen. Af en toe kwam er verdwaalde

post, maar zo sporadisch dat er slechts lichte ergernis ontstond. Die tijd is voorgoed voorbij. Als je op dit moment een mailinglijst begint zonder spampreventiemaatregelen te nemen, wordt die lijst bedolven onder de junkmails en is deze volledig onbruikbaar. Spampreventie is verplicht.

We verdelen spampreventie onder in twee categorieën: voorkomen dat spamberichten op uw mailinglijsten verschijnen en voorkomen dat uw mailinglijst geen bron wordt van nieuwe e-mailadressen voor de adressenzoekers *(harvesters)* van de spamverspreiders. De eerste categorie is het belangrijkst, dus die behandelen we eerst.

Filteren van posts

Er bestaan drie basistechnieken om spamposts te voorkomen en de meeste mailinglijstensoftware beschikt over alledrie. U kunt ze het beste in tandem gebruiken:

1. *Laat op geautomatiseerde basis alleen posts toe van subscribers van de lijst.*
   Dit is in principe doeltreffend. Ook vereist het weinig administratieve overhead omdat het doorgaans alleen een kwestie is van het wijzigen van een instelling in de configuratie van de lijstsoftware. Let op dat posts die niet automatisch worden goedgekeurd niet meteen worden weggegooid. In plaats daarvan moeten ze worden doorgetuurd ter moderatie, en wel om twee redenen. Op de eerste plaats wilt u dat ook niet-subscribers kunnen posten: iemand met een vraag of aanbeveling moet eenmalig een bericht kunnen posten op de mailinglijst zonder te hoeven subscriben. Ten tweede kan het gebeuren dat een subscriber een bericht verstuurt vanaf een ander adres dan waarmee hij geregistreerd staat. E-mailadressen vormen geen betrouwbaar middel om mensen mee te identificeren en dienen dan ook niet als zodanig behandeld te worden.

2. *Filter posts met behulp van spamfiltersoftware.*
   Als de mailinglijstsoftware dit toelaat (en de meeste pakketten doen dat) kunt u posts laten filteren door spamfiltersoftware. Automatische spamfilters zijn niet perfect en zullen dat ook niet worden. Er is immers een voortdurende wapenwedloop aan de gang tussen spammers en filterschrijvers. Het kan echter de hoeveelheid spam die door de mazen glipt richting de moderatiewachtrij enorm verminderen. En hoe langer die wachtrij is, des meer tijd mensen eraan kwijt zijn. Elke vorm van automatisch spam filteren is dus nuttig.

   Op deze plek is geen ruimte voor een gedetailleerde instructie over het instellen van spamfilters. Daartoe dient u de documentatie van uw mailinglijstsoftware te raadplegen (zie voor meer informatie het gedeelte ‘Software’ verderop in dit hoofdstuk). Lijstsoftware beschikt vaak standaard over ingebouwde spampreventiefuncties. Wellicht wilt u echter een of meer filters van een andere partij toevoegen. Ik heb goede ervaringen met deze twee: SpamAssassin (http://spamassassin.apache.org/) en SpamProbe (http://spamprobe.sourceforge.net/). Dit is geen oordeel over de vele andere open source-spamfilters. Sommige daarvan zijn blijkbaar ook erg goed. Toevallig heb ik de genoemde twee naar tevredenheid gebruikt.

3. *Moderatie.*
   Voor mails die niet automatisch worden toegelaten, omdat ze van niet-subscribers zijn en die wel door de spamfiltersoftware komen – als u die hebt – is er een laatste fase: *moderatie*. Het bericht wordt gerouteerd naar een speciaal adres waar het door iemand wordt onderzocht en wordt goedgekeurd of afgekeurd.

   Goedkeuring kan twee gedaanten aannemen: u kunt deze post voor deze ene keer accepteren of u kunt de lijstsoftware opdragen deze en alle toekomstige berichten van dezelfde afzender te accepteren. Om de druk op toekomstige moderatie te verminderen, zult u bijna altijd het laatste willen doen. Hoe u goedkeurt verschilt van systeem tot systeem, maar doorgaans is het alleen een kwestie van een antwoord sturen aan een speciaal adres met de opdracht ‘accept’ (alleen deze post accepteren) of ‘allow’ (deze en toekomstige posts accepteren).

   Afwijzen gebeurt doorgaans door gewoon de moderatiemail te negeren. Als de lijstsoftware nooit bevestiging krijgt dat iets een geldige post is, stuurt deze de post niet door naar de lijst. De moderatiemail negeren sorteert dus het gewenste effect. Soms bestaat er ook de mogelijkheid om met ‘reject’ of ‘deny’ te antwoorden. Hiermee keurt u automatisch toekomstige mails van dezelfde afzender af, zonder ze door moderatie te laten gaan. Het heeft vrijwel nooit zin om dit laatste te doen, omdat moderatie voornamelijk betrekking heeft op spampreventie en spammers sowieso zelden twee keer spam vanaf hetzelfde adres verzenden.

Zorg ervoor dat u moderatie *alleen* gebruikt voor het wegfilteren van spam en berichten die duidelijk niet over het onderwerp gaan, bijvoorbeeld wanneer iemand een bericht stuurt naar de verkeerde mailinglijst. In een moderatiesysteem is het meestal mogelijk om de afzender direct te antwoorden. Gebruik die mogelijkheid niet om vragen te beantwoorden die eigenlijk op de mailinglijst zelf thuishoren, ook al weet u het antwoord uit uw hoofd. Als u dat namelijk wel doet, krijgt de projectgemeenschap een onvolledig beeld van het soort vragen dat mensen stellen, en heeft men niet de mogelijkheid om zelf op de vraag te antwoorden en/of de antwoorden van anderen in te zien. Mailinglijstmoderatie is louter bedoeld om de lijst te vrijwaren van junk en e-mails over andere onderwerpen.

Adressen in archief verbergen

Om te voorkomen dat uw mailinglijsten een bron worden van adressen voor spammers bestaat er een veelgebruikte techniek om e-mailadressen in de archieven te verbergen. U vervangt bijvoorbeeld

`naam@domein.nl` door `naam_AT_domein.nl` of `naamGEENSPAM@domein.nl`

Of gebruik een soortgelijke, voor een mens eenvoudig te ontcijferen codering. Omdat adres-harvesters vaak te werk gaan door webpagina’s door te ploegen – ook de online archieven van uw mailinglijst – op zoek naar @’s, is het coderen van adressen een manier om een e-mailadres onzichtbaar of onbruikbaar te maken voor spammers. Dit voorkomt natuurlijk niet dat er spam naar de mailinglijst zelf verstuurd wordt. Het verhindert echter wel dat de hoeveelheid spam die direct naar de privéadressen van de gebruikers van de lijst wordt gestuurd, toeneemt.

Het verbergen van adressen kan controversieel zijn. Sommigen zijn er tuk op en zijn verbaasd als uw archieven dat niet automatisch doen. Anderen vinden het juist onhandig omdat ook zij de adressen moeten decoderen voordat ze ze kunnen gebruiken. Sommigen beweren dat de techniek niet doeltreffend is, omdat een harvester zich theoretisch kan aanpassen aan ieder consistent coderingspatroon. De praktijk leert echter dat het verbergen van adressen wel degelijk doeltreffend is.
Zie http://www.cdt.org/speech/spam/030319spamreport.shtml.

Idealiter laat lijstenbeheersoftware de keuze over aan iedere afzonderlijke subscriber, hetzij door een speciale ja/nee-header, hetzij door een instelling in de accountvoorkeuren van die subscriber. Ik ken echter geen software die beschikt over de keuze per subscriber of per bericht. Voorlopig is het dus de lijstbeheerder die voor iedereen de keuze moet maken, aangenomen dat de *archiver* over deze functie beschikt. Dat is namelijk niet altijd het geval. Ik neig er licht toe om 'adres verbergen' aan te zetten. Sommige mensen zijn erg voorzichtig, om te voorkomen dat hun e-mailadres op een webpagina of elders gepost wordt en een spam-harvester dat ziet. Zij zien niet graag dat hun voorzichtigheid teniet wordt gedaan door een mailinglijstarchief. Daarnaast is het ongemak dat het verbergen van adressen de archiefgebruiker oplevert zeer gering. Het is een klusje van niks om een verborgen e-mailadres om te zetten naar een geldig adres als je die persoon wilt bereiken. Vergeet echter niet dat het uiteindelijk allemaal een wapenwedloop is. Tegen de tijd dat u dit leest, hebben harvesters wellicht de meest gangbare manieren van verbergen gekraakt en moeten we iets nieuws bedenken om harvesten te voorkomen.

Management van identificatie en headers

Lijst-subscribers willen vaak mails van de lijst in een projectspecifieke map kunnen opbergen, apart van hun andere mail. Hun mailclient kan dit automatisch doen door de *headers* van de mail te bekijken. De headers zijn de velden bovenaan de mail die de afzender, de ontvanger, het onderwerp, de datum en verschillende andere details van het bericht weergeven. Bepaalde headers zijn bekend en in principe verplicht:

```
Van:...
Aan:...
Onderwerp:...
Datum:...
```

Andere zijn optioneel, echter vrij gebruikelijk. Bijvoorbeeld: e-mails hoeven de header

```
Reply-to: zendernaam@domein.nl
```

niet te hebben. Toch gebruiken de meeste mailclients deze header wel, omdat de ontvanger zo extra informatie ontvangt over hoe de schrijver bereikt kan worden. Dit is met name nuttig als de schrijver de email van een ander adres verstuurt dan waar naartoe moet worden geantwoord.

Sommige mailclients beschikken over een makkelijk te gebruiken interface om mails op basis van de 'Onderwerp-header' te archiveren. Dat leidt ertoe dat mensen verzoeken om een door de mailinglijst automatisch aan alle onderwerpen toegevoegd voorvoegsel, waardoor zij hun e-mailprogramma zo kunnen instellen dat het dat voorvoegsel herkent en de desbetreffende mails automatisch in de juiste map belanden. Dit is het idee: de oorspronkelijke schrijver schrijft:

```
Onderwerp: Versie 2.5 maken.
```

De mail ziet echter in de lijst als volgt uit:

```
Onderwerp: [discussie@lijsten.domein.nl] Versie 2.5 maken.
```

Hoewel bijna alle lijstbeheersoftware over deze functie beschikt, raad ik u ten zeerste aan deze niet in te schakelen. Het probleem dat deze functie zou oplossen, kan op een veel minder opdringerige manier worden opgelost, en de kosten van het ruimteverbruik in het onderwerpveld zijn veel te hoog. Ervaren mailinglijstgebruikers scannen doorgaans de onderwerpen van de inkomende mail van die dag af en beslissen vervolgens wat ze gaan lezen en/of beantwoorden. Door tekst voor de naam van het onderwerp te plakken, kan de rechterkant van het onderwerp van het scherm verdwijnen, waardoor het gedeeltelijk onleesbaar wordt. Hierdoor verdwijnt informatie waarvan mensen afhankelijk zijn voor de beslissing om mail al dan niet te openen en neemt de totale functionaliteit van de mailinglijst voor iedereen dus af.
In plaats van met de onderwerp-header te gaan knutselen, is het verstandig om uw gebruikers te leren de andere standaard-headers beter te gebruiken, te beginnen met de Aan-header, waarin de naam van de mailinglijst hoort te staan:

```
Aan: <discussie@lijsten.domein.nl>
```

Elk mailprogramma dat op onderwerp kan filteren, kan ook filteren op de Aan-header.

Er worden nog andere, vrij veel gebruikte headers door de deelnemers verwacht bij mailinglijsten. Filteren op deze headers is nog betrouwbaarder dan het gebruik van de Aan- of Cc-headers. Aangezien deze headers aan iedere post worden toegevoegd door de mailinglijstbeheersoftware, rekenen gebruikers wellicht op hun aanwezigheid:

```
list-help: <mailto:discuss-help@ lists.example.org>
list-unsubscribe: <mailto:discuss-unsubscribe@lists.example.org>
list-post: <mailto:discuss@lists.example.org>
Delivered-To: mailing list discuss@lists.example.org
Mailing-List: contact discuss-help@lists.example.org; run by ezmlm
```

In grote lijnen behoeven de genoemde headers geen uitleg. Raadpleeg http://www.nisto.com/listspec/list-manager-intro.html voor aanvullende informatie. Als u de gedetailleerde, formele specificatie nodig hebt, gaat u naar http://www.faqs.org/rfcs/rfc2369.html.

U ziet dat deze headers impliceren dat als u een mailinglijst opzet met de naam ‘list’, er ook administratieve adressen ‘list-help’ en ‘list-unsubscribe’ bestaan. Daarnaast is het normaal om ‘list-subscribe’ te hebben voor het subscriben en ‘list-owner’ om de lijstautoriteiten te bereiken. Al naar gelang de list management software die u gebruikt, kunnen deze en/of andere beheeradressen worden ingesteld. U vindt hierover meer in de documentatie van de software. Gewoonlijk wordt als onderdeel van de welkomstmail aan iedere nieuwe gebruiker nadat hij zich heeft aangemeld een volledige uitleg van de betekenis van al deze speciale adressen meegestuurd. Waarschijnlijk krijgt u deze welkomstmail zelf ook. Als dat niet het geval is, vraag iemand anders dan om een kopie. Dan weet u wat uw gebruikers te zien krijgen als ze zich aanmelden voor de lijst. Houd een kopie van het welkomstbericht bij de hand, zodat u kunt antwoorden over de mailinglijstfuncties. Nog beter, plaats ze op een webpagina. Op die manier hoeft u mensen die hun exemplaar kwijt zijn en een mail sturen met de vraag “Hoe kan ik unsubscriben van de lijst?” alleen maar een URL te sturen.

Sommige mailinglijstsoftware beschikt over de mogelijkheid om onderaan iedere post de unsubscribe-instructies toe te voegen. Als u over die optie beschikt, zet ‘m dan aan. Dit genereert per bericht een paar extra regels op een onschadelijke plek, maar het kan u veel tijd besparen omdat het aantal mensen dat naar u mailt – of nog erger, dat naar de lijst mailt! – met de vraag hoe te unsubscriben afneemt.

Het grote ‘reply-to’-debat

Eerder, in het gedeelte ‘Voorkom privédiscussies’, benadrukte ik het belang om ervoor te zorgen dat discussies publiek blijven. Ook sprak ik over actieve maatregelen om te voorkomen dat discussies verdwijnen naar privé-threads. Dit hoofdstuk gaat over het zodanig instellen van de projectcommunicatiesoftware, dat deze u zo veel mogelijk werk uit handen neemt. Als de mailinglijstsoftware daarom over de mogelijkheid beschikt om er automatisch voor te zorgen dat de discussies op de lijst blijven, lijkt het logisch om die functie dan maar aan te zetten.

Dat is het echter niet. Er bestaat misschien wel een dergelijke functie, maar die heeft enkele behoorlijk ernstige bijverschijnselen. De vraag of deze functie nu wel of niet gebruikt moet worden, is een van de heftigste discussies op het terrein van mailinglijstbeheer. Toegegeven, het geschil zal de dagbladen niet halen, maar bij open source-softwareprojecten laait hij van tijd tot tijd op. Hieronder beschrijf ik de functie, de belangrijkste argumenten ervoor en ertegen, en doe ik zo goed als ik kan een aanbeveling.

De functie zelf is heel simpel: de mailinglijstsoftware kan, als u dat wilt, de Reply-to-header zo instellen dat bij iedere post alle antwoorden automatisch naar de mailinglijst worden teruggestuurd. Dat wil zeggen, ongeacht wat de oorspronkelijk afzender in de Reply-to-header zet (ook wanneer ze hem helemaal niet gebruiken), tegen de tijd dat de lijst-subscribers de post zien, bevat de header het lijstadres:

```
Reply-to: discuss@lists.example.org
```


Op zich lijkt dat een goede zaak. Omdat vrijwel alle mailprogramma’s op de Reply-to-header letten, wordt een antwoord automatisch gericht aan de hele lijst, niet alleen aan de afzender waarop geantwoord wordt. Natuurlijk kan degene die antwoordt *handmatig* de ontvanger van zijn antwoord instellen, maar belangrijk is dat antwoorden *standaard* naar de lijst gaan. Het is een perfect voorbeeld van hoe technologie ingezet wordt om samenwerking te stimuleren.

Helaas kleven er enkele nadelen aan deze optie. Het eerste staat bekend als het *Can’t Find My Way Back Home*-probleem *(Ik ben verdwaald!):* soms zet de oorspronkelijke afzender zijn ‘echte’ e-mailadres in het Reply-to-veld, omdat hij mail vanaf een ander adres wil versturen dan het adres waarop hij normaliter mail ontvangt. Wie altijd met hetzelfde adres verstuurt en ontvangt, heeft dit probleem niet en is er wellicht verbaasd over dat het überhaupt voorkomt. Maar wie een ongebruikelijke e-mailconfiguratie heeft, of zelf niet kan bepalen hoe het Van-adres van zijn eigen e-mails eruitziet - misschien omdat er verzonden wordt vanaf een werkadres en de zender geen invloed heeft op zijn IT-afdeling - is gebruik van Reply-to wellicht de enige manier om ervoor te zorgen dat antwoorden bij hem terechtkomen. Wanneer zo iemand een mail stuurt naar een mailinglijst waarvan hij geen subscriber is, wordt zijn Reply-to-instelling essentiële informatie. Als de lijstsoftware over deze informatie vervangt, ziet hij zelf wellicht nooit een antwoord op zijn bericht.

Het tweede, en naar mijn oordeel meest zwaarwegende argument tegen Reply-to-geknutsel, heeft te maken met verwachtingspatronen. De meeste ervaren mailgebruikers zijn gewend aan twee basismethoden om te antwoorden: *reply-to-all* en *reply-to-author (Beantwoorden aan allen* en *Beantwoorden)*. Alle moderne mailprogramma’s hebben een aparte knop voor deze twee handelingen. Gebruikers weten dat Beantwoorden aan allen betekent dat de hele lijst het antwoord krijgt en dat bij Beantwoorden alleen de schrijver het antwoord ontvangt. Hoewel u mensen wilt stimuleren om zo veel mogelijk naar de lijst te antwoorden, zijn er beslist situaties waarin degene die antwoordt de keuze moet hebben om privé te antwoorden. Het kan bijvoorbeeld gaan om een vertrouwelijk opmerking aan de schrijver van de oorspronkelijke boodschap, iets waarvan het ongepast zou zijn om het op de openbare lijst te plaatsen.

Bedenk nu wat er gebeurt als de lijst de oorspronkelijke Reply-to van de afzender vervangt. Degene die antwoordt drukt op de knop Beantwoorden en verwacht dat zijn antwoord alleen bij de oorspronkelijke schrijver terechtkomt. Omdat dit conform het redelijke verwachtingspatroon is, controleert hij waarschijnlijk niet of het adres van de ontvanger het bedoelde adres is. Hij schrijft zijn vertrouwelijke privébericht, gepaard met wellicht genante details over iemand op de lijst, en drukt op Verzenden. Onverwacht staat zijn bericht een paar minuten later *op de mailinglijst!* Zeker, hij zou beter hebben kunnen kijken naar het veld Aan en zou geen aannamen hebben moeten maken over de Reply-to-header. Maar schrijvers zetten vrijwel altijd Reply-to op hun privéadres (of beter gezegd, dat doet hun e-mailprogramma voor ze) en zeer veel ervaren e-mailers verwachten dat ook. Het is zelfs zo dat iemand die Reply-to expres op een ander adres zet, zoals de lijst, hij dat meestal vermeld in de tekst, opdat mensen niet voor verrassingen komen te staan wanneer ze antwoorden.

Vanwege de ernstige gevolgen die dit onverwachte gedrag kan hebben, gaat mijn voorkeur ernaar uit om lijstbeheersoftware nooit aan de Reply-to-header te laten rommelen. Dit is een voorbeeld van technologie die bedoeld is om mensen te laten samenwerken, maar – naar mijn mening – gevaarlijke bijwerkingen heeft. De voorstanders van het gebruik van deze functie hebben echter ook enkele sterke argumenten. Wat u ook kiest, van tijd tot tijd zullen er mensen die naar uw lijst mailen met de vraag waarom u het niet andersom doet. Aangezien u dit onderwerp beslist niet op uw lijst ter discussie wilt stellen, is het verstandig om een ingeblikt antwoord klaar te hebben, en wel van het type dat de discussie beëindigt en niet aanzwengelt. Zorg ervoor dat u *niet* beweert dat uw keuze, welke dat ook moge zijn, de enige juiste is (ook al vindt u dat wel). Geef in plaats daarvan aan dat dit een oude discussie is, dat er goede argumenten voor beide oplossingen zijn, dat u het niet iedereen naar de zin kunt maken en dat u daarom een naar uw oordeel zo verstandig mogelijke keuze hebt gemaakt. Vraag beleefd of men niet terug wil komen op dit onderwerp tenzij men echt nieuwe informatie heeft. Blijf vervolgens weg uit de thread en hopelijk sterft dit onderwerp dan een natuurlijke dood.

Iemand stelt misschien voor om over dit onderwerp te stemmen. Daaraan kunt u natuurlijk toegeven, maar ik ben van mening dat koppen tellen geen bevredigende oplossing voor dit probleem is. De straf voor iemand die verrast is over wat er gebeurt, is dermate groot (per ongeluk een privémail naar een openbare lijst sturen) en het ongemak voor alle anderen zo miniem (af en toe iemand eraan herinneren om naar de hele lijst te antwoorden in plaats van alleen naar uzelf) dat het de vraag is of de meerderheid is staat zou mogen zijn een minderheid bloot te stellen aan een dergelijk risico.

Ik ben hier niet op alle argumenten ingegaan, alleen op degene die doorslaggevend lijken. Raadpleeg voor de volledige discussie de twee volgende gezaghebbende documenten, die men altijd citeert in deze discussie:

Leave Reply-to alone, *by Chip Rosenthal*
http://www.unicom.com/pw/reply-to-harmful.html
Set Reply-to to list, *by Simon Hill*
http://www.metasystema.net/essays/reply-to.mhtml

Hoewel ik hierboven een voorkeur heb uitgesproken, vind ik niet dat er een 'juist' antwoord op deze kwestie is. Ik neem dan ook met plezier deel aan veel lijsten die Reply-to *wel* instellen. Belangrijk is dat u in een vroeg stadium voor het een of het ander kiest en zich dan niet meer laat verleiden tot een discussie erover.

Twee fantasieën

Op een dag zal iemand het slimme idee krijgen om in een mailprogramma een knop reply-to-list toe te voegen. Het programma zou aan de hand van enkele van de eerder genoemde standaard-headers het adres van de mailinglijst weten uit te dokteren, en vervolgens het antwoord direct aan de lijst richten. Alle andere ontvanger-adressen worden dan achterwege gelaten, omdat zij waarschijnlijk allemaal subscribers van de lijst zijn. Uiteindelijk zouden andere e-mailprogramma's deze functie overnemen en zou deze discussie tot het verleden behoren. (NB, het mailprogramma Mutt beschikt over deze functie.[13])

Een betere oplossing zou zijn om van het Reply-to-geknutsel een voorkeur te maken die iedere subscriber zelf in kan stellen. Wie wil dat de lijst Reply-to instelt (hetzij op de mail van anderen, hetzij op de eigen mail), kan daarom vragen. Wie dat niet wil, krijgt een Reply-to-functie waaraan niet gesleuteld is. Ik ken echter geen list management software die deze functie heeft per subscriber. Voorlopig zitten we dus opgescheept met een algemene instelling.[14]

Archivering

De technische details voor het opzetten van archivering van mailinglijsten zijn afhankelijk de list management software. Dit valt buiten de reikwijdte van dit boek. Bij het kiezen of configureren van een *archiver* doet u er verstandig aan op de volgende eigenschappen te letten.

Onmiddellijke archivering

Mensen willen vaak verwijzen naar een gearchiveerde post die de afgelopen twee uur gedaan is. Indien mogelijk dient een archiver elke post onmiddellijk archiveren, zodat de post ook in het archief zit wanneer hij op de mailinglijst verschijnt. Als die optie niet beschikbaar is, stel de *archiver* dan zo in dat hij ongeveer elk uur een update draait. (Sommige archivers voeren hun updateprocessen standaard eens per nacht uit, maar dat is in de praktijk veel te weinig voor een actieve mailing lijst.)

Referentiestabiliteit

Als een bericht eenmaal op een bepaalde URL is gearchiveerd, moet het op precies die URL eeuwig toegankelijk blijven, ten minste 'zo lang mogelijk'. Ook als archieven opnieuw worden opgebouwd, worden hersteld van een back-up of op een andere manier worden gerepareerd, moeten alle URL's die publiekelijk toegankelijk zijn ongewijzigd blijven. Dankzij stabiele referenties kunnen internetzoekmachines archieven indexeren, hetgeen voor gebruikers die op zoek zijn naar antwoorden van onschatbare waarde is. Vaste referenties zijn ook belangrijk, omdat vanuit de bug tracker (zie het gedeelte Bug tracker verderop in dit hoofdstuk) en vanuit andere projectdocumenten vaak gelinkt wordt naar posts en threads van mailinglijsten.

Idealiter zou de mailinglijstsoftware in een header de archief-URL van een bericht of ten minste het berichtspecifieke deel van de URL moeten opnemen, wanneer de software het bericht verspreidt naar ontvangers. Op die manier zouden mensen die een kopie van het bericht hebben de archieflocatie ervan kennen, zonder het archief daadwerkelijk te bezoeken. Dat zou handig zijn omdat iedere handeling waarvoor de webbrowser nodig is automatisch tijdrovend is. Ik weet niet of er mailinglijstsoftware bestaat met deze functie. Helaas beschikken de programma's die ik gebruik er niet over. Het is echter iets om naar te zoeken, of, als u mailinglijstsoftware schrijft, op te nemen als functie... alstublieft!

Back-ups

Het moet duidelijk zijn hoe u de archieven kunt backuppen en het herstelproce-

dé zou ook niet moeilijk moeten zijn. Met andere woorden, gebruik uw *archiver* niet als een *black box*. U (of iemand anders binnen het project) hoort te weten waar de archiver de berichten opslaat en hoe u de feitelijke archiefpagina's kunt herstellen uit de opgeslagen berichten, mocht dat ooit nodig zijn. Deze archieven bevatten kostbare gegevens. Een project dat zijn archief verliest, verliest een belangrijk gedeelte van zijn collectief geheugen.

Thread-ondersteuning
Het dient mogelijk te zijn om van elk afzonderlijk bericht naar de *thread* te gaan (groep gerelateerde berichten) waarvan dat oorspronkelijke bericht deel uitmaakt. Ook iedere thread moet zijn eigen URL hebben, afgezien van de URL's van de verschilende berichten in de thread.

Doorzoekbaarheid
Een archiver zonder zoekfunctie – zowel naar de tekst als naar de schrijvers en onderwerpen van berichten – is vrijwel onbruikbaar. Houd wel in gedachten dat sommige archivers 'hun' zoekfunctie gewoon uitbesteden aan externe zoekmachines zoals Google. Dat is weliswaar aanvaardbaar, maar een eigen zoekfunctie is doorgaans meer diepgaand, bijvoorbeeld omdat het de zoeker in staat stelt te specificeren dat de exacte match moet voorkomen in een onderwerpregel en niet in de tekst van het bericht zelf.

Het bovenstaande is niet meer dan een technische checklist om u te helpen een archiver te beoordelen en in te stellen. Hoe u mensen zover krijgt om de archiver daadwerkelijk te *gebruiken* ten gunste van het project wordt in latere hoofdstukken besproken, voornamelijk in het gedeelte 'Opvallend gebruik van archieven'.

Software
Hier volgen enkele open source-tools voor lijstbeheer en archivering. Als de site waarop u uw project host al een standaardinstelling heeft, hoeft u wellicht nooit een tool te kiezen. Maar als u er zelf een moet installeren, volgen hier enkele mogelijkheden. De tools die ik gebruikt heb zijn Mailman, Ezmlm, MHonArc en Hypermail. Dat wil niet zeggen dat andere niet goed zijn. Ook zijn er waarschijnlijk tools die ik niet eens heb kunnen vinden, dus denk niet dat deze lijst volledig is.

Beheersoftware voor mailinglijsten:

Mailman — http://www.list.org/
(Heeft ingebouwde archiver en hooks om externe archivers in te pluggen.)
SmartList — http://www.procmail.org/
(Bedoeld om te worden gebruikt samen met het mailverwerkingssysteem Procmail.)
Ecartis — http://www.ecartis.org/
ListProc — http://listproc.sourceforge.net/
Ezmlm — http://cr.yp.to/ezmlm.html
(Bedoeld om te werken met het mailbezorgingssysteem Qmail.)
Dada — http://mojo.skazat.com/
(Ondanks de bizarre pogingen van de website om dit te verbergen, is dit vrije software, uitgebracht onder de GNU General Public License. Dada beschikt ook over een ingebouwde archiver.)

Archiveringssoftware voor mailinglijsten:

MHonArc — http://www.mhonarc.org/
Hypermail — http://www.hypermail.org/
Lurker — http://sourceforge.net/projects/lurker/
Procmail — http://www.procmail.org/
(Zustersoftware van SmartList. Dit is een algemeen mailverwerkingssysteem dat, klaarblijkelijk, kan worden geconfigureerd als archiver.)

3.3 VERSIEBEHEER

Een *versiebeheersysteem* of *revisiebeheersysteem* is een combinatie van technologieën en praktijken voor het volgen en beheren van wijzigingen aan projectbestanden, met name de broncode, documentatie en webpagina's. Als u nog nooit met versiebeheer te maken hebt gehad, is het eerste wat u moet doen iemand vinden die dat wel heeft gedaan, en deze persoon zien te betrekken bij uw project. Tegenwoordig verwacht iedereen dat tenminste de broncode van uw project onderworpen is aan versiebeheer. Waarschijnlijk neemt niemand uw project serieus als er geen sprake is van behoorlijk versiebeheer.

De reden dat versiebeheer zo algemeen ingeburgerd is, is dat vrijwel elk aspect van het runnen van een project er eenvoudiger van wordt: communicatie tussen ontwikkelaars, releasebeheer, bugbeheer, codestabiliteit, experimentele ontwikkelingspogingen en toewijzing en autorisatie van veranderingen door bepaalde ontwikkelaars. Het versiebeheersysteem fungeert als centrale coördinatie-eenheid temidden van al deze werkterreinen. De kern van versiebeheer is *veranderingsmanagement:* iedere afzonderlijke verandering aan de projectbestanden identificeren, iedere verandering annoteren met metagegevens als de datum en de auteur van de verandering, en vervolgens de feiten opnieuw afspelen voor iedereen die daarom vraagt en op elke manier waarop ze het vragen. Het is een communicatiemechanisme waarbij verandering de basale informatie-eenheid is.

In dit gedeelte komen niet alle aspecten van het gebruik van een versiebeheersysteem aan de orde. Deze zijn dusdanig veelomvattend dat ze verspreid over het hele boek per onderwerp aan de orde komen. Hier concentreren we ons op: het zo kiezen en instellen van een versiebeheersysteem dat het verderop in het project coöperatieve samenwerking stimuleert.

Verklarende woordenlijst versiebeheer
Als u nog nooit met versiebeheer hebt gewerkt, kan dit boek u dat niet leren. Maar het is onmogelijk om het onderwerp te bespreken zonder een paar essentiële termen de revue te laten passeren. Deze termen zijn nuttig, ongeacht het versiebeheersysteem dat u gebruikt. Het gaat om de basiswerkwoorden en zelfstandig naamwoorden van digitale samenwerking, die in de rest van het boek in generieke

zin worden gebruikt. Ook als er in de wereld geen versiebeheersystemen zouden bestaan, dan nog zou het probleem van veranderingsmanagement blijven bestaan. Met deze woorden beschikken we over een taal waarin we bondig over dat probleem kunnen praten.

'Versie' versus 'revisie'

Het woord *versie* wordt soms gebruikt als synoniem voor *revisie*. Ik doe dat in dit boek niet. Het zou dan namelijk te makkelijk verward kunnen worden met 'versie' in de betekenis van een versie van een stuk software, dat wil zeggen het release- of editienummer, bijvoorbeeld 'versie 1.1'. Omdat het begrip 'versiebeheer' reeds ingeburgerd is, blijf ik het gebruiken als synoniem voor 'revisiebeheer' en 'veranderingsbeheer'.

commit (committen)

Een verandering maken aan het project, formeler gezegd: een verandering op zo'n manier opslaan in de versiebeheerdatabase zodat deze kan worden ingepast in toekomstige releases van het project. 'Commit' is een zelfstandig naamwoord. Het werkwoord is 'committen'. Als zelfstandig naamwoord is het in principe synoniem met 'verandering'. Bijvoorbeeld: "Ik heb net een fix gecommit voor de servercrashbug die men heeft gemeld in Mac OS X. Jay, kun jij de commit alsjeblieft beoordelen en controleren of ik de allocator daar niet verkeerd gebruik?"

logbericht (log message)

Een stukje commentaar dat bij iedere commit wordt gevoegd en waarin de aard en het doel van de commit wordt beschreven. Logberichten behoren in elk project tot de belangrijkste documenten. Ze vormen een verbinding tussen de zeer technische taal van afzonderlijke codeveranderingen en meer gebruikersvriendelijke taal van functies, bugfixes en projectvoortgang. Verderop in dit gedeelte bekijken we manieren om logberichten de verspreiden naar de juiste partijen. Daarnaast wordt in het gedeelte 'Gewoontes voor codificeren' in Hoofdstuk 6 aandacht besteed aan manieren om contribuanten te stimuleren beknopte en bruikbare logberichten te schrijven.

update

Vragen dat andermans veranderingen (commits) in uw lokale kopie van het project worden opgenomen, dat wil zeggen om uw eigen kopie up-to-date te maken. Dit is een zeer normale operatie. De meeste ontwikkelaars updaten hun codes verscheidene malen per dag, zodat ze zeker weten dat ze min of meer hetzelfde runnen als de andere ontwikkelaars en dat als ze bug vinden, ze erg vrij zeker van kunnen zijn dat die niet al gerepareerd *(gefixt)* is. Bijvoorbeeld: "Hé, ik heb gemerkt dat de indexeringscode altijd de laatste byte weglaat. Is dit een nieuwe bug?" "Ja, maar hij is vorige week gerepareerd. Probeer te updaten, dan moet 'ie weg zijn."

repository

Een database waarin de veranderingen worden opgeslagen. Sommige versiebeheersystemen zijn gecentraliseerd: er bestaat één enkele master-repository waarin alle veranderingen aan het project worden opgeslagen. Andere systemen zijn gedecentraliseerd: elke ontwikkelaar heeft zijn eigen repository en verandering kunnen over en weer willekeurig geruild worden tussen repositories. Het versiebeheersysteem houdt de afhankelijkheden tussen veranderingen bij en wanneer het tijd is voor een release, wordt een bepaalde set veranderingen goedgekeurd voor die release. De vraag wat beter is, gecentraliseerd of gedecentraliseerd, is een van de heilige oorlogen die woeden in de wereld van softwareontwikkeling. Laat u niet verleiden om erover op de projectmailinglijsten in discussie te gaan.

checkout

Het proces van het verkrijgen van een kopie van het project uit een repository. Een checkout produceert doorgaans een *directory tree* die een *working copy* wordt genoemd (zie hieronder), vanwaar veranderingen terug naar de oorspronkelijke repository wordt gecommit. Bij sommige gedecentraliseerde versiebeheersystemen is elk working copy zelf een repository en kunnen veranderingen weggeschoven worden naar (of binnengehaald worden in) elke repository die ze wil accepteren.

working copy

De eigen directory tree van een ontwikkelaar, die de broncodebestanden van het project bevat en eventueel de webpagina's en andere documenten ervan. Een working copy bevat ook wat metagegevens die door het versiebeheersysteem beheerd worden en die de working copy vertellen uit welke repository zij afkomstig is, welke 'revisies' (zie hieronder) van de bestanden aanwezig zijn enz. Over het algemeen beschikt elke ontwikkelaar over zijn eigen working copy, waarin hij verandering aanbrengt en test en van waaruit hij commit.

revisie, *verandering*, *changeset*

Een revisie is meestal één specifieke incarnatie van een bepaald bestand of een bepaalde map. Als het project bijvoorbeeld begint met revisie 6 van bestand F en iemand commit een verandering aan F, dan ontstaat revisie 7 van F. Sommige systemen gebruiken revisie, verandering en changeset ook om te verwijzen naar een set wijzigingen die samen zijn gecommit als een conceptuele eenheid.

Deze termen hebben bij verschillende versiebeheersystemen soms afzonderlijke technische betekenissen, maar het basisidee is altijd gelijk ze bieden een manier om nauwkeurig te kunnen spreken over exacte momenten in de geschiedenis van een bestand of een set bestanden, zoals meteen voordat en meteen nadat een bug is gefixt. Bijvoorbeeld: "O ja, zij heeft dat in revisie 10 gerepareerd" of "Hij heeft dat in revisie 10 van foo.c gerepareerd."

Wanneer men het over een of meerdere bestanden heeft zonder een bepaalde revisie te vermelden, heeft men het meestal over de meest recente versie ervan.

diff

Een tekstuele voorstelling van een verandering. Een diff laat zien welke regels

zijn veranderd en hoe, plus een paar regels context aan weerskanten van de verandering. Een ontwikkelaar die al bekend is met een gedeelte van de code kan een diff meestal lezen in relatie tot die code en begrijpen wat de verandering heeft veroorzaakt; hij kan zelfs bugs herkennen.

tag
: Een label voor een bepaalde verzameling bestanden bij specifieke revisies. Tags worden meestal gebruikt om interessante momentopnamen van het project te bewaren. Zo wordt bij elke openbare release meestal een tag gemaakt, zodat men direct van het versiebeheersysteem de exacte set bestanden/revisies kan krijgen waaruit die release bestaat. Een doorsnee-tag-naam ziet eruit als `Release_1_0, Delivery_00456`.

branch (vertakking)
: Dit is een kopie van het project, weliswaar onder versiebeheer, maar geïsoleerd, zodat veranderingen aan de betreffende branch geen invloed hebben op de rest van het project en vice versa, behalve wanneer veranderingen bewust worden 'gemerged' van de ene kant naar de andere,zie hieronder. Branches staan ook bekend als *lines of development* (ontwikkeltrajecten). Ook als een project geen expliciete branches heeft, vindt de ontwikkeling toch plaats op de *main branch* (hoofdlijn), ook wel *main line* of *trunk* geheten.

  Branches bieden de mogelijkheid om verschillende ontwikkelingstrajecten los van elkaar te volgen. Men kan bijvoorbeeld een branch gebruiken voor een experimentele ontwikkeling, die voor de main trunk te destabiliserend zou zijn. Anderzijds kan een branch worden gebruikt om een nieuwe release te stabiliseren. Tijdens het releaseproces gaat de gewone ontwikkeling rustig door in de main branch van de repository. Tegelijkertijd zijn in de release-branch geen veranderingen toegestaan, met uitzondering van diegene die door de releasebeheerders zijn goedgekeurd. Op die manier hoeft een release het lopende ontwikkelwerk niet te dwarsbomen. Raadpleeg het gedeelte 'Branches gebruiken om bottlenecks te voorkomen' verderop in dit hoofdstuk voor een meer gedetailleerde bespreking van de voor- en nadelen van canned hosting.

merge(n) (oftewel porten)
: Een verandering verplaatsen van de ene naar de andere branch. Dit behelst het mergen van de main trunk naar een branch en vice versa. Dat zijn de meest gangbare manieren van mergen. Een verandering porten tussen twee niet-main branches komt zelden voor. Raadpleeg voor meer informatie over dit soort mergen het gedeelte 'Enkelvoudigheid van informatie'.

  'Merge(n)' heeft tweede, gerelateerde betekenis: het is datgene wat het versiebeheersysteem doet als het ziet dat twee mensen hetzelfde bestand hebben gewijzigd, maar op een elkaar niet overlappende manier. Aangezien de twee veranderingen elkaar niet in de weg zitten, worden wanneer een van die mensen zijn kopie van het bestand – dat zijn eigen verandering reeds bevat – update, de veranderingen van de andere persoon automatisch ingevoegd. Dit komt vaak voor, met name bij projecten waarbij verschillende mensen aan dezelfde code werken. Als twee verschillende veranderingen wel overlappen, is het resultaat een *conflict*; zie hieronder.

conflict
: Datgene wat ontstaat als twee mensen verschillende veranderingen proberen aan te brengen op dezelfde plek in de code. Alle versiebeheersystemen sporen conflicten automatisch op en melden aan een van de betrokkenen dat zijn verandering conflicteert met die van iemand anders. Het is aan die persoon om het probleem op te lossen *(resolve)* en die oplossing mee te delen aan het versiebeheersysteem.

lock
: Een manier om de exclusieve intentie kenbaar te maken om een verandering aan te brengen aan een bepaald bestand of een bepaalde map, bijvoorbeeld: "Ik kan op dit moment geen veranderingen aan de webpagina's aanbrengen. Blijkbaar heeft Alfred ze allemaal gelockt zolang hij met de achtergrondafbeeldingen bezig is." Niet alle versiebeheersystemen bieden locken aan als mogelijkheid en de systemen die dat wel doen, stellen niet allemaal verplicht dat de lockfunctie wordt gebruikt. Dat komt omdat gelijktijdige, parallelle ontwikkeling de norm is, en mensen uitsluiten van bestanden (meestal) tegen dit principe indruist.

  Van versiebeheersystemen die locken verplicht stellen om commits te maken, wordt gezegd dat ze het model *lock-modify-unlock* gebruiken. De systemen die dat niet doen, gebruiken het model *copy-modify-merge*. Een uitstekende, diepgaande uitleg en vergelijking van de twee modellen vindt u op http://svnbook.red-bean.com/svnbook-1.0/ch02s02.html. Over het algemeen is het copy-modify-merge-model beter voor de ontwikkeling van open source en alle versiebeheersystemen die in dit boek worden besproken gebruiken dit model.

Een versiebeheersysteem kiezen

Op het moment van schrijven zijn in de wereld van de vrije software *Concurrent Versions System (CVS*, http://www.cvshome.org/) en *Subversion (SVN*, http://subversion.tigris.org/) de twee populairste versiebeheersystemen.

CVS bestaat al heel lang. De meeste ervaren ontwikkelaars kennen het en het doet min of meer wat u nodig heeftt. Aangezien het al lang populair is, verzandt u waarschijnlijk niet in ellenlange debatten of dit systeem de juiste keuze was of niet. Er kleven echter enkele nadelen aan CVS. Het programma biedt geen makkelijke manier om te verwijzen naar veranderingen aan meerdere bestanden; het staat niet toe dat bestanden onder versiebeheer worden hernoemd of gekopieerd (als je de code-tree moet reorganiseren na de start van het project, kan dat echt vervelend zijn); de merge-ondersteuning is matig; het kan niet zo goed overweg met grote en binaire bestanden; en bepaalde handelingen zijn traag als er veel bestanden bij betrokken zijn.

Geen van de minpunten van CVS is dodelijk en het programma is nog steeds erg populair. De laatste jaren echter heeft het nieuwere Subversion terrein gewonnen, vooral bij nieuwere projecten[15]. Als u met een nieuw project begint, adviseer ik Subversion.

U kunt mijn onpartijdigheid in twijfel trekken omdat ik betrokken ben bij het Subversion-project. De laatste jaren is er een aantal nieuwe versiebeheersystemen voor open source verschenen. Zie Bijlage A, Gratis versiebeheersystemen. Daarin worden ze allemaal – voor zover ik ze ken – vermeld, ruwweg in volgorde van populariteit. Zoals uit de lijst blijkt, kan het kiezen van een versiebeheersysteem gemakkelijk ontaarden in een levenswerk. Wellicht hoeft u de beslissing niet te nemen omdat uw hosting-site dat voor u doet. Maar als u wel zelf moet kiezen, raadpleeg dan uw andere ontwikkelaars, informeer in uw omgeving wat de ervaringen van anderen zijn, en kies er dan een uit. Kom daar niet meer op terug. Elk stabiel, productieklaar versiebeheersysteem voldoet. U hoeft zich niet al te veel zorgen te maken dat u een volstrekt foute beslissing neemt. Als u niet kunt beslissen, ga dan voor Subversion. Het is redelijk eenvoudig te leren en blijft waarschijnlijk de komende jaren de norm.

Het versiebeheersysteem gebruiken
De aanbevelingen in dit gedeelte zijn niet gekoppeld aan een bepaald versiebeheersysteem en horen in elk systeem eenvoudig geïmplementeerd te kunnen worden. Raadpleeg voor details de documentatie van het systeem dat u gebruikt.

Houd alles onder versiebeheer
Houd niet alleen de broncode van uw project onder versiebeheer, maar ook de webpagina's, documentatie, FAQ's, ontwerpnotities en alles waaraan mensen veranderingen zouden willen aanbrengen. Bewaar ze naast de broncode, in dezelfde repository tree. Ieder brokje informatie dat het opschrijven waard is - dat wil zeggen, ieder brokje dat veranderd zou kunnen worden- is het ook waard om te 'versioneren',. Dingen die niet veranderen moeten gearchiveerd worden, niet geversioneerd. Een al geposte e-mail bijvoorbeeld verandert niet meer. Versies ervan bijhouden is derhalve onzin, tenzij hij deel gaat uitmaken van een groter, zich verder ontwikkelend document.

De reden dat alles samen op één plek versioneren zo belangrijk is, is dat mensen zich maar één mechanisme eigen hoeven te maken voor het indienen van veranderingen. Vaak begint een projectmedewerker met het aanbrengen van veranderingen aan de webpagina's of aan de documentatie en groeit hij bijvoorbeeld door naar het veranderen van kleine beetjes code. Als het project één systeem gebruikt voor allerlei soorten indienen, hoeft men het kunstje maar één keer te leren. Door alles samen te versioneren kunnen nieuwe functies samen met hun documentatie-updates worden gecommit, en wordt bij het maken van branches met de code ook de documentatie etc. meegenomen.

Pas geen versiebeheer toe op *gegenereerde bestanden*. Dit zijn gegevens die niet echt veranderen, aangezien ze programmatisch zijn gemaakt vanuit andere bestanden. Sommige build-systemen bijvoorbeeld maken `configure` op basis van de template `configure.in`. Om een verandering aan te brengen in `configure`, is het gebruikelijk `configure.in` te veranderen en het dan opnieuw te genereren. Daarom is alleen de template `configure.in` een veranderend bestand. Beheer alleen de versies van de templates. Als u de resultaatbestanden ook versioneert, gaan mensen beslist vergeten om te regenereren nadat ze een verandering aan een template hebben gecommit. De resulterende inconsistentie zullen ongekende verwarring zaaien.[16]

De gulden regel dat alle veranderbare gegevens onder versiebeheer gehouden moeten worden, kent één ongelukkige uitzondering: de bug tracker. Bugdatabases bevat veel veranderbare gegevens, maar om technische redenen kunnen ze die gegevens meestal niet opslaan in het hoofdversiebeheersysteem. (Sommige bug trackers beschikken zelf over primitieve versioneringsfuncties. Deze zijn echter onafhankelijk van de hoofdrepository van het project.)

Doorzoekbaarbaarheid
De repository van het project zou op het web doorgebladerd moeten kunnen worden. Dat betekent dat je niet alleen de meest recente revisies van de projectbestanden moet kunnen bekijken, maar ook eerdere revisies, de verschillen tussen revisies, dat je logbestanden moet kunnen lezen op geselecteerde wijzigingen enz.

Doorzoekbaarheid (bladerbaarheid) is belangrijk omdat het een lichtgewichtportal is naar de projectgegevens. Als de repository niet kan worden bekeken met een webbrowser, dan moet iemand die een bepaald bestand wil inspecteren (bijvoorbeeld om te kijken of een bepaalde bugfix aan de code is uitgevoerd) eerst lokaal clientsoftware voor versiebeheer installeren. Daardoor wordt een simpel zoekklusje van twee minuten al gauw een taak van een half uur of langer.

Doorzoekbaarheid betekent ook behoefte aan vaste URL's om bepaalde revisies van bestanden te bekijken en om de meest recente revisie op een gegeven moment te bekijken. Dit kan zeer nuttig zijn bij technische discussies of om mensen naar de documentatie te leiden. In plaats van bijvoorbeeld te zeggen "Raadpleeg voor het debuggen van de server het bestand www/hacking.html in uw werkkopie", kunt u zeggen "Raadpleeg voor het debuggen van de server *http://svn.collab.net/repos/svn/trunk/www/hacking.html*." Daarmee geeft u een URL die altijd naar de laatste revisie van het bestand *hacking.html* leidt. Zo'n URL is beter omdat hij niet voor tweeërlei uitleg vatbaar is en het irrelevant is of de geadresseerde een up-to-date werkkopie heeft.

Sommige versiebeheersystemen zijn standaard uitgerust met ingebouwde bladermechanismes voor repositories, andere laten dat over aan tools van derden. Drie van deze tools zijn *ViewCVS* (http://viewcvs.sourceforge.net/), *CVSWeb* (http://www.freebsd.org/projects/cvsweb.html) en *WebSVN* (http://websvn.tigris.org/). De eerste werkt met zowel CVS als Subversion, de tweede alleen met CVS en de derde alleen met Subversion.

E-mails committen
Iedere commit naar de repository zou een e-mail moeten genereren waarin staat wie de verandering gemaakt heeft, wanneer deze gemaakt is, welke bestanden en mappen zijn veranderd en hoe ze zijn veranderd. De e-mail dient dan naar een speciale mailinglijst te gaan, die uitsluitend bestemd is voor commit-e-mails en losstaat van de mailinglijsten waar personen naartoe mailen. Ontwikkelaars en andere geïnteresseerden zouden moeten worden gestimuleerd om te subscriben op de commit-lijst omdat dit de meest doeltreffende manier is om bij te blijven met wat er op codeniveau in het project gebeurt. Afgezien van het duidelijke voordeel van *peer review* (beoordeling door gelijken - zie het gedeelte 'De code nakijken op verdachte

onderdelen'), helpen commit-e-mails een gevoel van gemeenschap te bevorderen omdat ze een gemeenschappelijke omgeving creëren waarin mensen op gebeurtenissen (commits) reageren waarvan ze weten dat anderen die ook kunnen zien.

De specifieke instellingen van commit-e-mails hangen af van welk versiebeheersysteem u gebruikt, maar meestal is het een script of een andere verpakte faciliteit die voor de uitvoering zorgen. Als u het niet kunt vinden, zoek dan naar documentatie over *hooks*, met de *post-commit hook*, in CVS *loginfo hook* geheten. Post-commit hooks zijn een veelgebruikte manier om geautomatiseerde taken te lanceren op basis van commits. De hook wordt geactiveerd door een afzonderlijke commit, wordt voorzien van alle gegevens over die commit en is dan vrij om die gegevens te gebruiken om actie te ondernemen, bijvoorbeeld een e-mail versturen.

Het is verstandig om van voorverpakte commit-e-mailsystemen een aantal standaardinstellingen te wijzigen:

1. Sommige commit-mailers bevatten de feitelijke *diffs* in de e-mail niet, maar geven in plaats daarvan een URL om de verandering op het net te bekijken met behulp van het bladersysteem van de repository. Hoewel het een goed idee is om de URL vermelden zodat later naar de verandering verwezen kan worden, is het tevens *zeer* belangrijk dat de commit-e-mail de diffs zelf bevat. E-mails lezen is inmiddels zo gewoon geworden, dat als de inhoud van de verandering meteen in de commit-e-mail zichtbaar is, ontwikkelaars er per kerende post op zullen reageren, zonder hun e-mailprogramma te verlaten. Als je op een link moet klikken om de verandering te beoordelen, zullen de meesten dat niet doen, omdat het een nieuwe actie vereist in plaats van een voorzetting van wat je al aan het doen was. Bovendien is het voor de beoordelaar die iets over de verandering wil vragen vele malen makkelijker om op de knop 'Beantwoorden met tekst' te drukken en de vermelde diff van commentaar te voorzien, dan het internet op te gaan en met veel moeite bepaalde gedeelten van de diff te knippen en plakken van de browser naar het e-mailprogramma en vervolgens te reageren.

   (Als de diff enorm groot is, zoals wanneer een grote hoeveelheid nieuwe code aan de repository is toegevoegd, dan is natuurlijk verstandiger om de diff achterwege te laten en alleen te verwijzen naar een URL. De meeste commit-mailers kunnen dit soort beperkingen automatisch uitvoeren. Als de uwe dat niet kan, dan is het beter de diffs erin te laten en af en toe geconfronteerd te worden met een enorme e-mail, dan de diffs helemaal achterwege te laten. Op een simpele manier kunnen bekijken en reageren is een hoeksteen van het stimuleren van samenwerking en dat is veel te belangrijk om te laten vallen.)

2. De Reply-to-header van de commit-e-mails moet zijn ingesteld op de gewone ontwikkelingslijst, niet op de commit-e-maillijst. Dat betekent dat wanneer iemand commits bekijkt en een antwoord schrijft, zijn antwoord automatisch naar de ontwikkelingslijst met personen gaat, want daar worden technische onderwerpen normaliter besproken. Hier zijn enkele redenen voor: ten eerste wilt u alle technische discussie op een lijst houden. Iedereen verwacht namelijk dat ze daar plaatsvinden en op die manier hoeft men slechts in één archief te zoeken. Ten tweede zijn er wellicht geïnteresseerden die niet geabonneerd zijn op de commit-e-maillijst. Ten derde afficheert de commit-e-maillijst zichzelf als een manier om commits te bekijken, niet om commits te bekijken *en* af te toe een technische discussie. Wie zich geabonneerd heeft op de commit-e-maillijst heeft zich alleen ingeschreven voor commit-e-mails. Door ze via die lijst ander materiaal te sturen, wordt als het ware een overeenkomst geschonden. Ten vierde gebruikt men vaak schrijfprogramma's die de commit-e-maillijst lezen en de resultaten verwerken, bijvoorbeeld om op een webpagina af te beelden. Die programma zijn ingesteld op de verwerking van consistent geformatteerde commit-e-mails, niet op inconsistente, door mensen geschreven mails.

   Het advies om Reply-to zo in te stellen, is niet in tegenspraak met de aanbevelingen in het gedeelte 'Het grote 'reply-to'-debat' eerder in dit hoofdstuk. De *afzender* van een bericht mag Reply-to altijd zelf instellen. In dit geval is de afzender het versiebeheersysteem zelf, dat Reply-to instelt om aan te geven dat de geëigende plek voor antwoorden de ontwikkelingsmailinglijst is, en in ieder geval niet de commit-lijst.

CIA: Nog een manier om veranderingen te publiceren

Commit-e-mails zijn niet de enige manier om nieuws over veranderingen te spreiden. Onlangs is hiervoor een mechanisme ontwikkeld genaamd CIA (http://cia.navi.cx/). CIA is een realtime verzamelaar en distributeur van commit-statistieken. Het meest populaire gebruik van CIA is om commit-mededelingen naar IRC-kanalen te verzenden, zodat de mensen die zijn ingelogd op die kanalen de commits realtime kunnen zien gebeuren. Hoewel dit een wat minder technisch hulpmiddel is dan commit-e-mails, omdat de toeschouwer wellicht niet aanwezig is als de commit-mededeling in IRC verschijnt, is deze techniek een geweldig *sociaal* hulpmiddel. Mensen krijgen namelijk het gevoel dat ze deel uitmaken van iets levendigs en actiefs en hebben het gevoel dat de vooruitgang met eigen ogen zien gebeuren.

Het werkt als volgt: u roept het CIA-mededelingprogramma op vanuit uw post-commit hook. Het mededelingprogramma formatteert de commit-informatie in een xml-bericht en verstuurt dat naar een centrale server (meestal `cia.navi.cx`). Die server distribueert de commit-informatie dan naar andere plaatsen.

CIA kan ook worden geconfigureerd om RSS-feeds te verzenden. Raadpleeg voor details hierover de documentatie op http://cia.navi.cx/.

Om een voorbeeld te zien van CIA in actie, zet u uw IRC client op `irc.freenode.net`, kanaal `#commits`.

Branches gebruiken om bottlenecks te voorkomen

Onervaren gebruikers van versiebeheer zijn soms een beetje wars van branching en merging. Dit is waarschijnlijk een van de nevenwerkingen van de populariteit van CVS. De CVS-interface voor branching en merging is bepaald niet intuïtief. Veel gebruikers hebben derhalve geleerd deze handelingen te vermijden.

Als u een van hen bent, neem dan resoluut het besluit om uw angsten te overwinnen en de tijd te nemen om branching en merging onder de knie te krijgen. Het zijn geen moeilijke handelingen als u eenmaal aan ze gewend bent, en de branches worden steeds belangrijker naarmate het project meer ontwikkelaars aantrekt.

Branches zijn waardevol omdat ze van een schaarse bron - ruimte om te werken in de code van het project - juist een overvloedige maken. Gewoonlijk spelen alle ontwikkelaars samen in dezelfde zandbak en werken ze aan hetzelfde zandkasteel. Wanneer iemand een nieuwe ophaalbrug wil toevoegen maar niet iedereen ervan kan overtuigen dat dit een verbetering zou zijn, kan hij dankzij branching naar een afgezonderd hoekje gaan en het uitproberen. Als de poging slaagt, kan hij de andere ontwikkelaars erbij roepen om het resultaat te bekijken. Als iedereen het erover eens is dat het resultaat geslaagd is, kunnen ze het versiebeheersysteem de opdracht geven om de ophaalbrug van het branch-kasteel naar het hoofdkasteel te verhuizen ('merge').

Het is duidelijk hoe deze faciliteit bijdraagt aan het samen uitvoeren van een project. Mensen moeten over de vrijheid kunnen beschikken om nieuwe dingen te ontwikkelen zonder dat ze het gevoel hebben dat ze het werk van anderen in de weg zitten. Net zo belangrijk is het dat er momenten zijn dat de code afgezonderd dient te worden van de dagelijkse ontwikkelingsproductie, bijvoorbeeld om een bug te repareren of een release te stabiliseren, zonder je zorgen te hoeven maken om steeds veranderende code (zie het gedeelte 'Een release stabiliseren' en het gedeelte 'Meervoudige releasetrajecten' in Hoofdstuk 7, *Downloadpakketten maken, releases en dagelijkse ontwikkeling*).

Gebruik branches veel en vaak, en spoor anderen aan dat ook te doen. Maar zorg er ook voor dat een branche niet langer actief is dan absoluut noodzakelijk. Iedere actieve branch vraagt namelijk een beetje aandacht van de ontwikkelgemeenschap. Zelfs diegenen die niet in een branch bezig zijn, hebben een soort perifeer besef van wat er daar gebeurt. Een dergelijk besef is natuurlijk gewenst en voor een branch-commit moet net als voor andere commits een commit-e-mail worden verzonden. Branches mogen echter geen middel worden om de ontwikkelgemeenschap op te splitsen. Zeldzame uitzonderingen daargelaten, dient het mergen van de veranderingen in de main line en daarna het opheffen van de branch, het uiteindelijke doel van een branch te zijn.

Enkelvoudigheid van informatie

Merging kent een belangrijke voorwaarde: commit dezelfde verandering nooit tweemaal. Dat wil zeggen dat een bepaalde verandering het versiebeheersysteem precies één keer in mag binnenkomen. De revisie (of set van revisies) waarin de verandering is binnengekomen, geldt vanaf dat moment als unieke *identifier.* Als de verandering moet worden toegepast op andere branches dan degene waarop zij binnenkwam, moet zij vanaf het oorspronkelijke punt van binnenkomst worden gemerged naar de andere bestemmingen, in plaats van een tekstueel identieke verandering te committen. Dat laatste zou hetzelfde effect op de code hebben, maar accuraat boekhouden en releasebeheer onmogelijk maken.

De praktische implicaties van dit advies verschillen per versiebeheersysteem. Bij sommige systemen zijn merges speciale events, fundamenteel anders dan commits, die zijn voorzien van hun eigen metagegevens. Bij andere worden de resultaten van merges net zo gecommit als andere veranderingen. Het primaire middel om een merge-commit te onderscheiden van een commit van een nieuwe verandering is zodoende het logbericht. Herhaal in een logbericht van een merge *niet* het logbericht van de oorspronkelijke verandering. Geef in plaats daarvan alleen aan dat het hier om een merge gaat en geef de identificerende revisie van de oorspronkelijke verandering met een samenvatting van tenminste één zin over de gevolgen van de verandering. Indien iemand het volledige logbericht wil zien, dient hij de oorspronkelijke revisie te raadplegen.

De reden waarom het belangrijk is om het herhalen van het logbericht te voorkomen, is dat logberichten soms worden bewerkt nadat ze zijn gecommit. Als een logbericht van een verandering op elke merge-bestemming zou worden herhaald, zou iemand, ook als die het oorspronkelijke bericht zou bewerken, nog altijd alle herhaalde berichten onbewerkt laten. En dat zou later in het project alleen maar voor verwarring zorgen.

Hetzelfde principe geldt voor het terugdraaien van een verandering Als een verandering wordt teruggetrokken uit de code, dan moet het logbericht louter meedelen dat een bepaalde revisie wordt teruggedraaid. Het bericht mag de feitelijke codeverandering die het resultaat is van het terugdraaien *niet* beschrijven, aangezien de betekenis van de verandering uit het oorspronkelijke logbericht en de oorspronkelijke verandering kan worden gehaald. Het logbericht van de terugdraaiing moet ook de reden vermelden waarom de verandering wordt teruggedraaid, maar mag niets dupliceren uit het logbericht van de oorspronkelijke verandering. Ga indien mogelijk terug naar het logbericht van de oorspronkelijke verandering en bewerk het zo dat duidelijk wordt dat de verandering is teruggedraaid.

Het bovenstaande impliceert dat u voor het verwijzen naar revisies een consistente syntaxis dient te hanteren. Dat is niet alleen handig in logberichten, maar ook in e-mails, de bug tracker en elders. Als u CVS gebruikt, stel ik "`path/to/file/in/project/tree:REV`" voor, waarbij REV een CVS-revisienummer is zoals '1.76'. Als u Subversion gebruikt, is de standaardsyntaxis voor revisie 1729 'r1729' (een bestandspad is niet nodig omdat Subversion globale revisienummers gebruikt). Andere systemen hebben gewoonlijk een standaardsyntaxis voor de changeset-naam. Wat voor uw systeem dan ook de juiste syntaxis moge zijn, stimuleer mensen die te gebruiken als ze verwijden naar veranderingen. Door de gedaante van naamsveranderingen consistent te houden, wordt de projectboekhouding een stuk makkelijker (wat zal blijken in Hoofdstuk 6, *Communicatie* en Hoofdstuk 7, *Downloadpakketten maken, releases en dagelijkse ontwikkeling*) en aangezien een groot deel van de boekhouding door vrijwilligers gedaan wordt, moet die zo eenvoudig mogelijk zijn.

Autorisatie

De meeste versiebeheersystemen beschikken over een functie waarmee bepaalde mensen toestemming of juist geen toestemming krijgen om te committen in specifieke subgebieden van de repository. Uitgaande van het principe dat mensen die

een hamer in de hand geduwd krijgen op zoek gaan naar spijkers, gebruiken veel projecten deze functie overvloedig. Ze selecteren zorgvuldig en geven mensen alleen toegang tot die gebieden waarnaar ze mogen committen en zorgen ervoor ze dat nergens anders kunnen. (Raadpleeg om te zien hoe projecten beslissen wie waar mag committen het gedeelte '*Committers*' in Hoofdstuk 8, *Het managen van vrijwilligers*.)

Waarschijnlijk kan een dergelijk strak beleid geen kwaad, maar een wat soepeler aanpak is ook goed. Sommige projecten gebruiken een soort beleefdheidssysteem. Wanneer iemand commit-toegang krijgt, ook al is het maar voor een subgebied van de repository, krijgt hij een wachtwoord waarmee hij overal in het project kan committen. Hem wordt alleen gevraagd om zich te beperken tot zijn subgebied. Vergeet niet dat hier geen echte risico's aan verbonden zijn; uiteindelijk worden in een actief project alle commits sowieso geëvalueerd. Als iemand ergens commit waar hij niet hoort te committen, zien anderen dat en trekken ze aan de bel. Als een verandering ongedaan moet worden gemaakt, is dat simpel genoeg. Alles zit onder versiecontrole, dus gewoon terugdraaien.

Een meer relaxte aanpak heeft zo zijn voordelen. Ten eerste, als ontwikkelaars hun vleugels uitslaan naar andere gebieden (hetgeen meestal het geval is als ze bij het project betrokken blijven) is er geen administratieve rompslomp in verband met het aanpassen van de privileges. Zodra iemand toestemming krijgt om ergens anders ook te committen, kan diegene meteen aan de slag.

Ten tweede kan het vleugels uitslaan op een fijnmaziger manier worden uitgevoerd. Meestal gaat het zo dat een committer in gebied X die ook actief wil worden in gebied Y, begint met het posten van patches naar gebied Y en vraagt om een evaluatie. Als iemand met commit-toegang tot Y zo'n patch ziet, hoeven ze de committer alleen maar te vragen om de verandering direct te committen (onder vermelding van de naam van de beoordelaar/goedkeurder in het logbericht natuurlijk). Op die manier is de commit afkomstig van degene die de verandering daadwerkelijk geschreven heeft, hetgeen vanuit het oogpunt van informatiemanagement en crediting de voorkeur verdient.

Ten slotte, en wellicht niet als onbelangrijkste, bevordert zo'n beleefdheidssysteem een gevoel van vertrouwen en wederzijds respect. Iemand commit-toegang geven tot een subdomein zegt iets over het technische niveau van die persoon: "We zien dat je de deskundigheid bezit om commits te maken in een bepaald domein, dus ga je gang maar." Maar het hanteren van een strikt autorisatieregime voegt daaraan toe: "Niet alleen vinden we dat je deskundigheid beperkt is, we zetten ook wat vraagtekens bij je *bedoelingen*." Zoiets wilt u natuurlijk liever niet zeggen. Als u iemand bij het project betrekt als committer is dat een kans om hem te initiëren in een sfeer van wederzijds vertrouwen. Hem meer macht geven dan hij moet hebben, is een goede manier om dat te doen. Laat hem vervolgens weten dat het aan hem is om zich binnen de gestelde grenzen te blijven bewegen.

Het Subversion-project werkt al meer dan vier jaar met dit beleefdheidssysteem en op het moment van schrijven heeft het 33 volledige en 43 gedeeltelijke committers.

Het enige onderscheid dat het systeem eigenlijk maakt, is dat tussen committers en niet-committers; verdere onderverdelingen worden geregeld door mensen. Toch hebben we nog nooit een probleem gehad met committers die expres buiten hun domein committen. Er is een paar keer een misverstand geweest over de omvang van iemands commit-privileges, maar dat is altijd snel en in goede harmonie opgelost.

Het spreekt voor zich dat u in situaties waarin zelfhandhaving onpraktisch is een strikt autorisatiebeleid moet hanteren. Maar die situaties komen nauwelijks voor. Zelfs al is er sprake van miljoenen regels broncode en honderdduizenden ontwikkelaars, dan nog hoort een commit naar een willekeurige module geëvalueerd te worden door diegenen die aan die module werken. Zij horen te signaleren dat iemand gecommit heeft die dat niet had mogen doen. Als dit soort standaardevaluatie van commits niet plaatsvindt, heeft het project grotere problemen dan alleen het autorisatiesysteem.

Kort gezegd, besteed niet te veel tijd aan de autorisatie van het versiebeheersysteem, tenzij u daar een specifieke reden voor hebt. Meestal levert een strikte aanpak geen tastbaar voordeel op en bent u beter af door op de terughoudendheid van de mensen te vertrouwen.

Dit alles betekent niet dat de beperkingen op zich onbelangrijk zijn, integendeel. Het zou slecht zijn voor een project om mensen te stimuleren in gebieden te committen waarvoor ze niet gekwalificeerd zijn. Bovendien heeft volledige (onbeperkte) commit-toegang een bijzondere status. Het houdt namelijk stemrecht in over projectbrede kwesties. Dit politieke aspect van commit-toegang wordt uitgebreider besproken in het gedeelte 'Wie mag er stemmen?' in Hoofdstuk 4, *Sociale en politieke infrastructuur.*

3.4 BUG TRACKER

Bugs opsporen en volgen is een uitgebreid en gevarieerd onderwerp. In dit boek worden diverse aspecten ervan besproken. In deze sectie zal ik me proberen te concentreren op de installatie en overwegingen van technische aard. Om daarmee echter aan de slag te kunnen, moeten we eerst een beleidsvraag stellen: wat voor gegevens moeten er nu precies in een bug tracker worden bewaard?

De term *bug tracker* is misleidend. Vaak worden bugopsporingssystemen gebruikt voor het volgen van verzoeken om nieuwe functies, eenmalige taken, ongevraagde patches, kortom, van vrijwel alles met een duidelijke begin- en eindtoestand, met optionele overgangstoestanden ertussenin, en wat gedurende zijn levenscyclus aanvullende gegevens verzamelt. Daarom worden bug trackers ook wel *issue trackers*, *defect trackers*, *artifact trackers*, *request trackers*, *trouble ticket systems,* enz. genoemd. Zie Bijlage B, *Gratis bug trackers* voor een lijst met software.

In dit boek zal ik de term bug tracker blijven gebruiken voor de software die het daadwerkelijke volgen doet, omdat de meesten dat nu eenmaal doen. Ik gebruik

het woord *issue* om te verwijzen naar een enkel item in de database van de bug tracker. Daarmee kunnen we onderscheid maken tussen het gedrag of wangedrag dat de gebruiker is tegengekomen (dat wil zeggen de bug zelf) en de *record* die de opspoorder heeft gemaakt van de ontdekking, de diagnose en de mogelijke oplossing van de bug. Vergeet niet dat hoewel de meeste issues over feitelijke bugs gaan, issues ook gebruikt kunnen worden om andere zaken op de sporen.

De kenmerkende levenscyclus van een issue ziet er als volgt uit:

1. Iemand rapporteert een issue. Deze persoon verstrekt een samenvatting, een eerste beschrijving (met inbegrip van een reproductierecept, indien van toepassing; als u wilt zien hoe u goede bugrapporten kunt stimuleren, raadpleeg dan het gedeelte 'Behandel iedere gebruiker als een mogelijke vrijwilliger' in Hoofdstuk 8, *Het managen van vrijwilligers*) en alle andere gegevens waar de bug tracker om vraagt. Degene die het issue rapporteert, is mogelijk een volledig onbekende van het project: bugrapporten en verzoeken om functies kunnen namelijk net zo goed afkomstig zijn van de gebruikersgemeenschap als van de ontwikkelaarsgemeenschap.

   Eenmaal gerapporteerd, verkeert het issue in de zogenaamde *open* toestand. Omdat nog geen actie is ondernemen, labellen sommige trackers deze status als *niet-geverifieerd* en/of *niet-gestart*. Het issue is nog niet aan iemand toegewezen, of - iets dat sommige systemen doen - is bij gebrek aan een echte toewijzing toegewezen aan een pseudogebruiker. Op dit punt bevindt het issue zich in een *holding*-gebied: het issue is vastgelegd, maar nog niet doorgedrongen tot het bewustzijn van het project.

2. Anderen lezen het issue, voegen er commentaar aan toe en vragen de oorspronkelijke rapporteur wellicht om uitleg over enkele aspecten.

3. De bug wordt *gereproduceerd*. Dit is wellicht het belangrijkste moment in het leven van de bug. Hoewel de bug de facto nog niet verholpen is, betekent het feit dat iemand anders dan de oorspronkelijke rapporteur hem kan reproduceren, dat de bug bestaat en, niet minder belangrijk, het bevestigt dat de oorspronkelijke rapporteur een bijdrage heeft geleverd aan het project door een echte bug te rapporteren.

4. De bug wordt *gediagnosticeerd*: de oorzaak wordt vastgesteld en, indien mogelijk, wordt de inspanning ingeschat die nodig is om hem te repareren. Zorg ervoor dat deze zaken worden vastgelegd in het issue. Als degene die de bug heeft gediagnosticeerd namelijk plotseling niet meer aan het project kan deelnemen – hetgeen met vrijwilliger-ontwikkelaars snel kan gebeuren – moet iemand anders de draad kunnen oppakken.

   In dit stadium, of soms al in het vorige, kan een ontwikkelaar *ownership* op zich nemen van het issue en dit aan zichzelf *toewijzen (assign)*. (In het gedeelte 'Onderscheid maken tussen informeren en toewijzen' in Hoofdstuk 8, *Het managen van vrijwilligers* wordt het toewijzingsproces gedetailleerder beschreven.) In dit stadium kan ook de *prioriteit* van het issue gesteld worden. Als de bug bijvoorbeeld zo ernstig is dat de volgende release erdoor vertraagd zou worden, dan moeten de feiten snel boven water komen en moet de tracker op de een of andere manier beschikken over een manier om dat vast te leggen.

5. Het issue wordt ingepland *(scheduled)* om te worden opgelost. Inplannen betekent niet per se dat er een datum wordt genoemd waarop de bug moet zijn verholpen. Soms betekent het alleen maar dat besloten wordt met welke volgende release (niet noodzakelijkerwijs de eerstvolgende) de bug gefixt moet zijn. Als de bug eenvoudig te verhelpen is, wordt soms helemaal afgezien van inplanning.

6. De bug wordt verholpen (of de taak voltooid, de patch aangebracht, of wat dan ook). De verandering of de reeks *(set)* van veranderingen waarmee de bug is gefixt, moet worden vastgelegd in een opmerking bij het issue, waarna het issue wordt *gesloten (closed)* en/of als *opgelost (resolved)* gemarkeerd.

Er bestaan enkele gangbare variaties op deze levenscyclus. Soms wordt een issue al vrij snel nadat het is gerapporteerd gesloten omdat blijkt dat er helemaal geen sprake is van een bug, maar van een misverstand aan de kant van de gebruiker. Naarmate een project meer gebruikers aantrekt, komt dit soort ongeldige issues steeds vaker voor. Ontwikkelaars sluiten ze af met commentaren die de neiging hebben steeds chagrijniger te worden. Probeer dat laatste te voorkomen; niemand heeft er iets aan. De statistische trend is alleen zichtbaar voor de ontwikkelaar, niet voor de gebruiker. (In het gedeelte *'De bug tracker voorfilteren'* verderop in dit hoofdstuk besteden we aandacht aan de technieken om het aantal ongeldige issues te verminderen.) Indien bij verschillende gebruikers dezelfde misverstanden steeds weer optreden, moet wellicht dat gedeelte van de software opnieuw ontworpen worden. Dit soort patronen valt het beste te constateren als de bugdatabase in de gaten gehouden wordt door een issuemanager; zie het gedeelte 'Issuemanager' in Hoofdstuk 8, *Het managen van vrijwilligers*.

Een andere variatie op de levenscyclus is dat het issue al snel na stap 1 wordt gesloten als *duplicaat (duplicate)*. Een duplicaat is een issue dat wordt gerapporteerd maar reeds bekend is bij het project. Duplicaten beperken zich niet tot open issues. Een bug kan, nadat hij verholpen is, terugkeren. Dit staat bekend als *regressie (regression)*. In dat geval bestaat het voorkeurstraject uit het heropenen van het oorspronkelijke issue en het sluiten van alle nieuwe rapporten als duplicaten van het oorspronkelijke. De bug tracker hoort deze relatie in beide richtingen bij te kunnen houden, zodat reproductiegegevens in de duplicaten beschikbaar zijn voor het oorspronkelijke issue en vice versa.

Een derde variant is dat de ontwikkelaars het issue sluiten in de veronderstelling dat het opgelost is, maar dat de oorspronkelijke rapporteur de fix afwijst en het issue heropent. Meestal gebeurt dit omdat de ontwikkelaars geen toegang hebben tot de omgeving die nodig is om de bug te reproduceren of omdat ze de fix niet hebben getest met hetzelfde reproductierecept als de rapporteur.

Naast deze variaties kunnen er, afhankelijk van de opsporingssoftware, nog andere kleine verschillen optreden in de levenscyclus van een bug. Maar de basisvorm is dezelfde en hoewel de levenscyclus zelf niet specifiek is voor open source-software, heeft ze wel gevolgen voor de manier waarop open source-projecten hun bug trackers gebruiken.

Uit stap 1 blijkt al dat de tracker net zo zeer onderdeel uitmaakt van het publieke gezicht van het project als de mailinglijsten of de webpagina's. Iedereen kan een issue rapporteren, iedereen kan een issue bekijken en iedereen kan door de lijst met open issues bladeren. Daaruit volgt dat u nooit weet hoeveel mensen de voortgang van een bepaald issue in de gaten houden. Hoewel de omvang en deskundigheid van de ontwikkelgemeenschap een remmende factor kan zijn op het tempo waarin issues worden opgelost, moet in ieder geval geprobeerd worden om elk issue te beantwoorden zodra het verschijnt. Ook al blijft het issue zelf een tijd liggen, dan nog prikkelt een reactie de betrokkenheid van de rapporteur. Immers, iemand heeft geregistreerd wat hij gemeld heeft. Vergeet niet dat een issue rapporteren meer moeite kost dan alleen maar een mailtje versturen. Bovendien betreedt een issue, zodra dit is gezien door een ontwikkelaar, het bewustzijn van het project, bijvoorbeeld doordat die ontwikkelaar vanaf dat moment zijn ogen openhoudt voor andere voorvallen van dat issue, er met andere ontwikkelaars over praat enz.

De noodzaak om tijdig te reageren heeft twee consequenties:

- De bug tracker moet gekoppeld zijn aan een mailinglijst, zodat iedere verandering aan een issue, met inbegrip van de eerste rapportage, een uitgaande mail genereert die beschrijft wat er is gebeurd. Dat is meestal een andere mailinglijst dan de gewone ontwikkelmailinglijst, omdat misschien niet alle ontwikkelaars geautomatiseerde bugmails willen ontvangen. De Reply-to-header moet echter net als bij commit-mails ingesteld worden op de ontwikkelingsmailinglijst.

- Het formulier voor het rapporteren van issues moet het e-mailadres van de rapporteur kunnen vastleggen zodat deze kan worden benaderd voor meer informatie. (Het e-mailadres van de rapporteur mag echter niet *verplicht* zijn. Sommigen geven er de voorkeur aan issues anoniem te rapporteren. Raadpleeg voor meer informatie over het belang van anonimiteit het gedeelte 'Anonimiteit en betrokkenheid' verderop in dit hoofdstuk).

Interactie met mailinglijsten

Zorg ervoor dat de bug tracker geen discussieforum wordt. Hoewel het belangrijk is om in de bug tracker enige menselijke aanwezigheid te houden, is deze functie in principe niet geschikt voor een realtime discussie. Zie de bug tracker meer als een archief, een manier om feiten en verwijzingen te organiseren voor andere discussies, voornamelijk die discussies die plaatsvinden in mailinglijsten.

Er zijn twee redenen voor dit onderscheid. Allereerst is de bug tracker omslachtiger in het gebruik dan een mailinglijst (en ook dan realtime chatforums). Dat komt niet doordat bug trackers slecht ontworpen gebruikersinterfaces hebben, maar omdat hun interfaces zijn ontworpen om 'statische toestanden' vast te leggen en te presenteren, geen vrije discussies. Ten tweede houdt niet iedereen die betrokken zou moeten zijn bij een discussie over een bepaald issue de bug tracker in de gaten. Het is een onderdeel van goed issuemanagement (zie het gedeelte 'Zowel managementtaken als technische taken delen' in Hoofdstuk 8, *Vrijwilligers managen*) om ervoor te zorgen dat elk issue ter attentie komt van de juiste mensen en dat niet iedere ontwikkelaar alle issues in de gaten moet houden. In het gedeelte 'Geen discussies in de bug tracker' in Hoofdstuk 6, *Communicaties,* kijken we naar manieren om te voorkomen dat mensen discussies wegzuigen uit de geëigende forums en verplaatsen naar de bug tracker.

Sommige bug trackers zijn in staat mailinglijsten in de gaten te houden en automatisch alle e-mails te loggen die over een bekend issue gaan. Meestal doen ze dat door de ID van het issue in de onderwerpregel van de mail te herkennen als onderdeel van een speciale tekenreeks (string); ontwikkelaars leren om deze strings bij te sluiten in hun mails zodat ze de aandacht van de tracker trekken. De bug tracker slaat of de hele mail op, of (nog beter) legt alleen een link naar de mail vast in het gewone mailinglijstarchief. Beide functies zijn zeer nuttig. Als uw tracker over deze functie beschikt, zet hem dan aan en herinner anderen eraan om er gebruik van te maken.

De bug tracker voorfilteren

De meeste issuedatabases krijgen uiteindelijk met hetzelfde probleem te maken: een verpletterende hoeveelheid dubbele of ongeldige issues die worden gerapporteerd door goedbedoelende maar onervaren of slecht geïnformeerde gebruikers. De eerste stap om dit probleem aan te pakken is meestal om een duidelijke mededeling op de voorpagina van de bug tracker te zetten, waarin wordt uitgelegd hoe je kunt zien dat een bug echt een bug is, hoe je erachter kunt komen of de bug al eerder gerapporteerd is en, ten slotte, hoe je een bug doeltreffend rapporteren kan als je na dit alles er nog steeds van overtuigd bent dat het een echte bug is.

Dit vermindert de hoeveelheid ruis voor een tijdje, maar naarmate het aantal gebruikers toeneemt, komt het probleem allengs weer terug. Je kunt een afzonderlijke gebruiker er niet de schuld van geven. Ze proberen allemaal bij te dragen aan het welslagen van het project en ook al is hun eerste bugrapport geen succes, dan nog wilt u ze stimuleren om betrokken te blijven en in de toekomst issues beter te rapporteren. In de tussentijd dienen de projectmedewerkers de issuedatabase zo rommelvrij mogelijk te houden.

Twee technieken zijn zeer effectief om dit probleem te voorkomen. Ten eerste moet u ervoor zorgen dat de bug tracker in de gaten gehouden wordt door mensen met voldoende kennis van zaken, zodat zij een issue kunnen sluiten als duplicaat of ongeldig zodra het binnenkomt. Ten tweede moet u van gebruikers verlangen (of dit in ieder geval zeer sterk stimuleren) dat ze een vermoeden van een bug laten controleren door iemand anders voordat ze deze daadwerkelijk als bug rapporteren in de tracker.

De eerste techniek wordt vrijwel overal toegepast. Zelfs projecten met enorme databases, zoals de Debian bug tracker op http://bugs.debian.org/, die op het moment waarop ik dit schrijf 315.929 issues bevat, regelen het zo dat *iemand* elk issue ziet dat binnenkomt. Al naar gelang het soort issue kan dat telkens iemand anders

zijn. Het Debian-project bijvoorbeeld is een verzameling softwarepakketten, dus routeert Debian elk issue automatisch naar de persoon die het pakket onderhoudt. Natuurlijk kan een gebruiker de verkeerde categorie kiezen voor een issue, met als gevolg dat het issue eerst naar de verkeerde persoon wordt gestuurd, die het weer moet herrouteren. Het belangrijkste echter is dat de verantwoordelijkheid wordt gedeeld. Of de gebruiker nu goed of fout rapporteert, het monitoren van binnenkomende issues wordt min of meer gelijk over de ontwikkelaars verdeeld, waardoor elk issue een tijdig antwoord kan krijgen.

De tweede techniek is minder wijdverbreid, waarschijnlijk omdat hij moeilijker te automatiseren is. Het basisidee is dat ieder nieuw issue in de database ge-*buddied* wordt. Als een gebruiker denkt dat hij een probleem gevonden heeft, wordt hem verzocht om het op een van de mailinglijsten of in een IRC-kanaal te beschrijven, en iemand te vragen om te bevestigen dat het inderdaad om een bug gaat. Door in een vroeg stadium een extra paar ogen in te schakelen, kan een hoop overbodig rapporteren worden voorkomen. Soms stelt de tweede partij vast dat het niet om een bug gaat of dat de bug in een recente release al is verholpen. Of de partij in kwestie is met de symptomen bekend vanwege een eerder issue en kan door de gebruiker op het eerdere issue te wijzen een duplicaat voorkomen. Vaak is het al voldoende om de gebruiker te vragen: “Heeft u in de bug tracker gezocht of deze bug al gerapporteerd is?” Veel mensen denken daar simpelweg niet aan, maar zijn beslist bereid om te zoeken wanneer ze weten dat iemand dat van hen *verwacht*.

Het buddysysteem kan ervoor zorgen dat de issuedatabase echt schoon blijft, maar heeft ook enkele nadelen. Veel mensen gaan toch op de solotour, of omdat ze de instructie om voor elk nieuw issue een buddy te zoeken niet zien, of omdat ze die naast zich neerleggen. Daarom blijft het nodig dat er vrijwilligers zijn die de database in de gaten houden. Bovendien is het niet eerlijk om rapporteurs te streng te berispen voor het negeren van de richtlijnen. Immers, de meeste nieuwe rapporteurs hebben geen idee hoe moeilijk het is om een issuedatabase bij te houden. En dus moeten de vrijwilligers waakzaam zijn en toch terughoudendheid in acht nemen bij het terugkoppelen van dit soort ongeverifieerde issues aan de rapporteur. Het doel is om de rapporteur te trainen, zodat hij in de toekomst het buddysysteem wel gebruikt en er een steeds toenemende groep mensen ontstaat die het issuefiltersysteem begrijpt. Bij het constateren van een issue dat niet *gebuddied* is, neemt u idealiter de volgende stappen:

1. Reageer meteen op het issue onder beleefde dankzegging voor de rapportage. Wijs de rapporteur op de buddyrichtlijnen, die natuurlijk prominent zichtbaar horen te zijn op de website.

2. Als het duidelijk om een geldig issue gaat en geen duplicaat, keur het dan goed en zet de normale levenscyclus in werking. De rapporteur is immers geïnformeerd over het buddyprincipe en het is dus zinloos om het tot dusver gedane werk teniet te doen door een geldig issue te sluiten.

3. Anderzijds, als het issue duidelijk ongeldig is, sluit het dan, maar vraag de rapporteur om het te heropenen zodra hij bevestiging krijgt van een buddy. Als hij dat doet, moet hij een referentie naar de bevestigings-thread plaatsen (bijv. een URL naar de mailinglijstarchieven).

Bedenkt dat dit systeem weliswaar de signaal-ruisverhouding van de issuedatabase zal verbeteren, maar nooit het foutief rapporteren volledig zal stoppen. De enige manier om dat voor elkaar te krijgen, is de bug tracker afsluiten voor alle gebruikers en alleen toegankelijk maken voor ontwikkelaars. Dit middel is echter bijna altijd schadelijker dan de kwaal. Het is maar beter om te accepteren dat het verwijderen van ongeldige issues een vast onderdeel uitmaakt van het routineonderhoud van het project en te zorgen voor zo veel mogelijk vrijwilligers die daarbij helpen.

Zie ook het gedeelte ‘Issuemanager’ in Hoofdstuk 8, *Het managen van vrijwilligers*.

3.5 IRC- / REALTIME CHATSYSTEMEN

Veel projecten beschikken over realtime chatrooms die gebruik maken van *Internet Relay Chat* (*IRC*), een kanaal waar gebruikers en ontwikkelaars elkaar vragen kunnen stellen en meteen antwoord kunnen krijgen. Hoewel u een IRC-server vanaf uw eigen website *kunt* draaien, is het over het algemeen niet de moeite waard. Doe in plaats daarvan wat iedereen doet: run je IRC-kanalen op Freenode (http://freenode.net/). Freenode biedt de controle die u nodig hebt om de IRC-kanalen van uw project te beheren,[16] terwijl u de niet onaanzienlijke moeite van het zelf bijhouden van een IRC-server bespaard blijft.

Het eerste wat u doet, is een naam kiezen voor het kanaal. De meest voor de hand liggende keuze is de naam van uw project. Als die op Freenode nog vrij is, gebruik hem dan. Is dat niet het geval, kies dan een naam die zo veel mogelijk op uw projectnaam lijkt en, zo mogelijk, makkelijk te onthouden is. Maak op de website van uw project reclame voor het feit dat er een kanaal beschikbaar is en wel zo dat een bezoeker die snel even een vraag wil stellen dat meteen ziet. Dit staat bijvoorbeeld in een prominent geplaatste box bovenaan de homepage van Subversion:

If you’re using Subversion, we recommend that you join the `users@subversion.tigris.org` *mailing list, and read the Subversion Book and FAQ. You can also ask questions on IRC at* `irc.freenode.net` *channel* `#svn`.

Sommige projecten beschikken over meerdere kanalen, één voor iedere subtopic: bijvoorbeeld voor installatieproblemen, voor gebruiksvragen, voor ontwikkel-chats, enz. (in het gedeelte ‘Omgaan met groei’ in Hoofdstuk 6, *Communicatie* wordt dit en hoe op te delen in meervoudige kanalen besproken). Als uw project zich nog in de beginfase bevindt, hoort het maar één kanaal te hebben, waarop iedereen met elkaar praat. Later, als de verhouding gebruikers/ontwikkelaars verandert in het voordeel van de eerstgenoemde, kunnen er meer kanalen nodig zijn.

Hoe weten mensen welke kanalen allemaal beschikbaar zijn, laat staan op welke kanalen zij moeten chatten? En als ze chatten, hoe weten ze dan wat de plaatselijke gebruiken zijn?

De manier om ze dat te weten te laten komen, is het *kanaalonderwerp (channel topic)* in te stellen.[18] Het kanaalonderwerp is een kort bericht dat elke gebruiker ziet als hij voor het eerst dat kanaal gebruikt. Het geeft nieuwkomers enkele snelle richtlijnen en aanwijzingen voor verdere informatie. Bijvoorbeeld:

```
You are now talking on #svn

Topic for #svn is Forum for Subversion user questions, see also
http://subversion.tigris.org/. || Development discussion hap-
pens in #svn-dev. || Please don't paste long transcripts here,
instead use a pastebin site like http://pastebin.ca/. || NEWS:
Subversion 1.1.0 is released, see http://svn110.notlong.com/ for
details.
```

Dat is kort en bondig, maar het vertelt nieuwkomers wat ze moeten weten. Het bericht zegt precies waar het kanaal voor is, geeft de homepage van het project (voor het geval iemand op het kanaal verzeild raakt zonder eerst op de website van het project te zijn geweest), vermeldt een gerelateerd kanaal en geeft wat adviezen over *pasting (plakken)*.

Paste-sites

Een IRC-kanaal is een gedeelte ruimte: iedereen kan zien wat iedereen zegt. Normaliter is dat een goede zaak, omdat het mensen toestaat in te breken in een discussie als ze denken dat ze wat kunnen bijdragen en omdat toeschouwers kunnen leren door toe te kijken. Maar het wordt problematisch als iemand een grote hoeveelheid informatie in één keer moet verstrekken, zoals een transcriptie van een debuggingsessie. Te veel outputregels in het kanaal plakken verstoort namelijk de andere discussies.

De oplossing is om een van de *pastebin*- of *pastebot*-sites te gebruiken. Wanneer u iemand om een grote hoeveelheid gegevens verzoekt, vraag dan of hij die gegevens niet in het kanaal plakt, maar in plaats daarvan naar (bijvoorbeeld) http://pastebin.ca/ gaat, zijn gegevens daar in het formulier plakt en van daaruit het IRC-kanaal op de hoogte stelt van de URL die het gevolg is van deze actie. Iedereen kan dan die URL bezoeken en de gegevens bekijken.

Er zijn diverse gratis paste-sites beschikbaar op dit moment, teveel voor een volledig overzicht. Hier volgen enkele van de sites die ik heb gebruikt: http://www.nomorepasting.com/, http://pastebin.ca/, http://nopaste.php.cd/ http://rafb.net/paste/ http://sourcepost.sytes.net/, http://extraball.sunsite.dk/notepad.php en http://www.pastebin.com/.

Bots

Veel technisch georiënteerde IRC-kanalen beschikken over een niet-menselijke deelnemer, een zogenaamde *bot*, die in staat is om informatie op te slaan en weer op te hoesten in antwoord op specifieke opdrachten. In principe wordt de bot aangesproken als ieder ander lid van het kanaal. Dat wil zeggen, de opdrachten worden gegeven door tegen de bot te 'spreken'. Een voorbeeld:


```
<kfogel> ayita: learn diff-cmd =
http://subversion.tigris.org/faq.html#diff-cmd
<ayita>  Thanks!
```

Dat commando gaf de bot, die op het kanaal is ingelogd als ayita, de opdracht om een bepaalde URL te onthouden als het antwoord op de query 'diff-cmd'. Nu kunnen we ayita aanspreken en de bot opdracht geven om een andere gebruiker op de hoogte te stellen van diff-cmd:

```
<kfogel> ayita: tell jrandom about diff-cmd
<ayita>  jrandom: http://subversion.tigris.org/faq.html#diff-cmd
```

Hetzelfde wordt bereikt met een handige verkorte versie:

```
<kfogel> !a jrandom diff-cmd
<ayita>  jrandom: http://subversion.tigris.org/faq.html#diff-cmd
```

De exacte set opdrachten en gedragingen verschilt van bot tot bot. Het bovenstaande voorbeeld is met `ayita` (http://hix.nu/svn-public/alexis/trunk/), waarvan gewoonlijk een instance loopt in `#svn` op freenode. Andere bots zijn Dancer (http://dancer.sourceforge.net/) en Supybot (http://supybot.com/). NB: Er zijn geen speciale serverprivileges vereist om een bot te runnen. Een bot is een client-programma; iedereen kan er een opzetten en het opdragen om naar een bepaalde server/bepaald kanaal te luisteren.

Als uw kanaal steeds opnieuw dezelfde vragen krijgt, raad ik u ten zeerste aan een bot op te zetten. Slechts een klein percentage van de kanaalgebruikers zal de deskundigheid verwerven om een bot te manipuleren, maar dat percentage zal een onevenredig hoog percentage vragen beantwoorden omdat de bot hun in staat stelt veel efficiënter te antwoorden.

IRC archiveren

Hoewel het mogelijk is om alles wat er op een IRC-kanaal gebeurt te archiveren, wordt dat niet noodzakelijkerwijs verwacht. IRC-discussies mogen dan in naam openbaar zijn, veel mensen beschouwen ze niettemin als informele privégesprekken. Gebruikers zijn daardoor vaak wat slordig met de grammatica en ventileren meningen -bijvoorbeeld over andere software of programmeurs- die ze liever niet in een online archief bewaard zien voor het nageslacht.

Natuurlijk zijn er soms *uittreksels (excerpts)* die bewaard dienen te worden en daar is niets mis mee. De meeste IRC-clients zijn in staat om op verzoek van een gebruiker een discussie naar een bestand te archiveren. Als dat niet mogelijk is, kunt u altijd de discussie uit de IRC knippen en in een meer permanent medium plakken (meestal de bug tracker). Maar van onbeperkt archiveren gaan sommige gebruikers zich ongemakkelijk voelen. Als u niettemin alles archiveert, verklaar dat dan duidelijk in het kanaalonderwerp en geef het archief een URL.

3.6 RSS FEEDS

RSS (Really Simple Syndication) is een mechanisme om samenvattingen van nieuws dat rijk is aan metagegevens te distribueren naar alle 'subscribers', dat wil zeggen, mensen die hebben aangegeven belangstelling te hebben om deze samenvattingen te ontvangen. Een gegeven RSS-source wordt gewoonlijk een *feed* genoemd en de subscription-interface van de gebruiker heet *feed reader* of *feed aggregator*. RSS Bandit en het toepasselijk geheten Feedreader zijn bijvoorbeeld twee open source RSS readers.

Er is in dit boek geen ruimte voor een gedetailleerde technische uitleg over RSS[19], maar u dient de twee belangrijkste zaken in de gaten te houden. Ten eerste wordt de software om de feed te lezen uitgekozen door de subscriber. Deze is *dezelfde* voor alle feeds die waarop de subscriber is geabonneerd. Dit is overigens het belangrijkste verkoopargument van RSS: dat de subscriber een interface kiest voor al zijn feeds, waardoor elke feed zich kan concentreren op het leveren van inhoud. Ten tweede is RSS nu alomtegenwoordig. Zelfs zozeer, dat de meeste mensen die het gebruiken niet eens weten *dat* ze het gebruiken. Voor de buitenwereld lijkt RSS op een knopje op een webpagina, met een label dat zegt 'Subscribe to this site' of 'News feed'. U klikt op de knop en vanaf dat moment updatet uw feed reader - die net zo goed een applet kan zijn die in uw website *geëmbed* is- automatisch wanneer er nieuws is van de betreffendesite.

Dit betekent dat uw open source-project waarschijnlijk een RSS feed zou moeten bevatten. Vergeet overigens niet dat de meeste canned hosting-sites een kant-en-klare aanbieden (zie het gedeelte '*Canned hosting*'). Zorg ervoor dat u dagelijks niet zo veel nieuwsberichten verstuurt dat de subscribers het kaf niet meer van het koren kunnen onderscheiden. Als er teveel nieuws-events zijn, negeren subscribers de feed gewoon of ze unsubscriben uit pure wanhoop. Idealiter biedt een project afzonderlijke feeds aan, één voor belangrijke mededelingen, een andere die bijvoorbeeld volgt wat er in de issue tracker gebeurt, weer een andere voor elke mailinglijst enz. In de praktijk is het moeilijk om dit goed te doen omdat het kan resulteren in interfaceverwarring voor zowel bezoekers aan de projectwebsite als de beheerders. Maar een project zou minimaal één RSS feed op de voorpagina moeten hebben voor het versturen van belangrijke mededelingen zoals releases en beveiligingswaarschuwingen.[20]

3.7 WIKI'S

Een *wiki* is een website die de bezoeker toestaat de inhoud ervan te wijzigen of uit te breiden. De term 'wiki' (van een Hawaiiaans woord dat 'vlug' of 'supersnel' betekent) wordt ook gebruikt voor de software die dat soort bewerken mogelijk maakt. De wiki werd uitgevonden in 1995, maar werd pas echt populair vanaf het jaar 2000-2001, voornamelijk dankzij het succes van de 'Wikipedia' (http://www.wikipedia.org/), een webencyclopedie op basis van wiki's met vrije inhoud. U moet een wiki zien als het midden tussen IRC en webpagina's: wiki's vinden niet realtime plaats, waardoor mensen de kans krijgen om na te denken en hun bijdragen bij te schaven, maar bijdragen zijn ook erg makkelijk toe te voegen en vragen daarom minder interface-overhead dan het bewerken van een gewone webpagina.

Wiki's behoren nog niet tot de standaarduitrusting van open source-projecten, maar dat zal waarschijnlijk niet lang meer duren. Omdat het om een relatief nieuwe technologie gaat en men nog experimenteert met verschillende manieren om ze te gebruiken, raad ik u hier aan enige voorzichtigheid in acht te nemen. Momenteel vallen er veel meer ongelukken met wiki's te analyseren dan successen.

Als u besluit om een wiki te runnen, besteed dan veel aandacht aan een duidelijke pagina-indeling en een aangename visuele lay-out, zodat bezoekers (m.a.w. potentiële editors) intuïtief weten hoe ze hun bijdrage moeten plaatsen. Net zo belangrijk is het om deze normen op de wiki zelf te posten, zodat mensen ergens naartoe kunnen voor advies. Te vaak houden wikibeheerders zichzelf voor de gek. Omdat de hordes bezoekers allemaal afzonderlijk inhoud van hoge kwaliteit aan de site toevoegen, hebben ze het idee dat de som van al die bijdragen ook van hoge kwaliteit is. Zo werkt het echter niet met websites. Iedere pagina of paragraaf kan op zichzelf beschouwd goed zijn, maar is dat niet als hij wordt ingebouwd in een rommelig of verwarrend geheel. Te vaak leiden wiki's aan:

Gebrek aan navigatieprincipes.
Een goed georganiseerde website geeft bezoekers het gevoel dat ze altijd weten waar ze zijn. Als de pagina's goed zijn ontworpen bijvoorbeeld, voelen mensen intuïtief het verschil tussen een index- en een inhoudgedeelte. Contribuanten aan een wiki zullen dergelijke verschillen ook respecteren, mits de verschillen überhaupt aanwezig zijn.

Dubbele informatie.
Wiki's krijgen uiteindelijk verschillende pagina's waar dezelfde dingen worden gezegd, omdat de afzonderlijke contribuanten de verdubbeling niet hebben opgemerkt. Dit kan ten dele het gevolg zijn van het hierboven vermelde gebrek aan navigatieprincipes, in die zin dat men wellicht de dubbele inhoud niet vindt omdat deze zich niet bevindt waar men het verwacht.

Inconsistente doelgroep.
Als er zoveel auteurs zijn, is dit probleem ten dele onvermijdelijk. Het kan echter worden verminderd als er geschreven richtlijnen bestaan over het opstellen van nieuwe inhoud. Wat ook helpt, is om nieuwe bijdragen in het begin agressief te bewerken, om zo een voorbeeld te stellen in de hoop dat men allengs besef van de normen begint te ontwikkelen.

Voor al deze problemen is de oplossing identiek: hanteer strikte redactionele normen en maak die niet alleen duidelijk door ze te posten maar ook door de manier van het bewerken van pagina's. Over het algemeen zullen wiki's tekortkomingen in het oorspronkelijke materiaal uitvergroten, omdat contribuanten de patronen die ze voorgeschoteld krijgen na-apen. Zet dus niet alleen maar een wiki op in de hoop dat alles wel goed komt. U dient hem met goed geschreven inhoud in de grondverf te zetten, zodat men een sjabloon heeft om op verder te bouwen.

Het lichtende voorbeeld van een goedgerunde wiki is natuurlijk Wikipedia, alhoewel dit wellicht deels het geval is omdat de inhoud (encyclopedische lemma's) van nature buitengewoon geschikt is voor het wiki-formaat. Maar als u Wikipedia onder de loep neemt, zult u zien dat de beheerders een *zeer* grondige basis voor samenwerking hebben gecreëerd. Er is uitgebreide informatie beschikbaar over hoe je lemma's moet schrijven, hoe je het perspectief bewaart, welke soort correcties je kunt uitvoeren, welke correcties je moet vermijden, er is een proces om meningsverschillen over betwiste correcties op te lossen (met verschillende stadia, inclusief uiteindelijke arbitrage), enz. Ze beschikken over autorisatiemiddelen, zodat ze een pagina die het onderwerp is van herhaalde onjuiste correcties, kunnen afsluiten tot het probleem is opgelost. Met andere woorden: ze gooien niet zomaar wat templates op een website in de hoop dat alles goed gaat. Wikipedia werkt omdat de grondleggers zorgvuldig hebben nagedacht over de manier waarop ze duizenden onbekenden zover konden krijgen om hun geschriften aan te passen aan een gedeelde visie. U heeft waarschijnlijk geen voorbereiding nodig van dit niveau om een wiki in een open source-softwareproject te runnen, maar het is de moeite waard om het wezen ervan na te bootsen.

Kijk voor meer informatie over wiki's op http://en.wikipedia.org/wiki/Wiki. Ook leeft en ademt de eerste wiki nog steeds. Deze bevat veel discussies over het onderhouden van wiki's: zie http://www.c2.com/cgi/wiki?WelcomeVisitors, http://www.c2.com/cgi/wiki?WhyWikiWorks en http://www.c2.com/cgi/wiki?WhyWikiWorksNot voor allerlei standpunten.

3.8 WEBSITE

Er valt weinig te zeggen over het opzetten van de projectwebsite vanuit een technisch gezichtspunt. Het opzetten van een webserver en het schrijven van de webpagina's is tamelijk simpel en de meeste aspecten van lay-out en indeling zijn al in het vorige hoofdstuk behandeld. De belangrijkste functie van de website is om een helder en uitnodigend overzicht van het project te bieden en als bindmiddel te fungeren voor de andere tools, zoals het versiebeheersysteem, de bug tracker enz. Als u niet over de deskundigheid beschikt om zelf een website op te zetten, dan kost het doorgaans niet al te veel moeite om iemand te vinden die dat wel kan en graag wil helpen. Om tijd en moeite te besparen, kiezen velen er niettemin voor om canned hosting-sites te gebruiken.

Canned Hosting

Een canned-site gebruiken heeft twee grote voordelen. Het eerste is servercapaciteit en bandbreedte: hun servers zijn dikke dozen met vette verbindingen. Hoe succesvol uw project ook wordt, u zult geen schijfruimte tekortkomen of de netwerkverbinding vol laten lopen. Het tweede voordeel is eenvoud. Zij hebben al een bug tracker gekozen, en een versiebeheersysteem, een mailinglijstmanager, een archiver en al het andere wat u nodig heeft om een site te draaien. Ze hebben de tools geconfigureerd en zorgen voor de backups van alle gegevens die in de tools opgeslagen zijn. U hoeft geen enkele beslissing te nemen. Het enige wat u hoeft te doen is een formulier invullen en op een knop drukken, en ineens beschikt u over een projectwebsite.

Dat zijn behoorlijk prettige voordelen. Het nadeel is vanzelfsprekend dat u *hun* keuzes en configuraties dient te accepteren, ook al zou iets anders beter zijn voor uw project. Meestal zijn canned-sites binnen bepaalde marges wel instelbaar, maar u krijgt nooit de fijnmazige controle die u met een eigen site en volledige beheerderstoegang tot de server hebt.

Een perfect voorbeeld is de verwerking van gegenereerde bestanden. Bepaalde webpagina's van het project zijn gegenereerde bestanden. Er zijn bijvoorbeeld systemen om FAQ-gegevens in een gemakkelijk te bewerken masterformat te houden, van waaruit html-, pdf- en andere presentatieformats kunnen worden gegenereerd. Zoals eerder in dit hoofdstuk werd uitgelegd in het gedeelte '*Houd alles onder versiebeheer*' wilt u de gegenereerde formaten niet onder versiebeheer houden, alleen het hoofdbestand. Maar als uw website wordt gehost op andermans server, kan het onmogelijk zijn om een *custom hook* op te zetten om de online html-versie van de FAQ te regenereren telkens wanneer het hoofdbestand wordt gewijzigd. De enige manier om hier een mouw aan te passen is om gegenereerde bestanden ook onder versiebeheer te stoppen, zodat zij te zien zijn op de website.

Er kunnen ook grotere consequenties zijn. Wellicht hebt u niet de mate van controle over de presentatie die u graag zou willen hebben. Bij sommige canned hosting-sites kunt u uw webpagina's aanpassen, maar de standaardlay-out van de site is meestal op allerlei merkwaardige manieren zichtbaar. Sommige projecten bijvoorbeeld die zichzelf hosten op SourceForge hebben hun homepages helemaal op maat gesneden, maar verwijzen ontwikkelaars nog steeds naar hun 'SourceForge page' voor meer informatie. De SourceForge-pagina zou de homepage van het project zijn, ware het niet dat het project gebruik maakt van een aangepaste homepage. De SourceForge-pagina heeft links met de bug tracker, de CVS repository, downloads, et cetera. Helaas bevat een SourceForge-pagina ook veel externe ruis. De top bestaat uit een *banner ad*, vaak bewegende beelden. De linkerkant is een verticale sortering linkjes die voor iemand die geïnteresseerd is in het project nauwelijks relevant zijn. De rechterkant is vaak ook een advertentie. Alleen het midden van de pagina is helemaal gewijd aan projectspecifiek materiaal en dat is dan nog gerangschikt op een manier die bezoekers in verwarring brengt over waar ze op moeten klikken.

Er is ongetwijfeld een goede reden voor elk afzonderlijk aspect van het SourceForge-ontwerp: goed vanuit het perspectief van SourceForge, zoals de advertenties. Maar vanuit het gezichtpunt van een afzonderlijk project kan het eindresultaat bestaan uit een minder ideale webpagina. Ik wil hier geen kwaad spreken over SourceForge, want soortgelijke bezwaren gelden ook voor andere canned hosting-sites. Het is een afweging. De technische sores van een projectsite wordt u bespaard, maar de prijs die u betaalt is dat u de manier waarop iemand anders de site runt, moet accepteren.

Alleen u kunt beslissen of canned hosting voor uw project de beste oplossing is. Als u voor een canned site kiest, laat dan de mogelijkheid open om later naar uw eigen servers te verhuizen door voor het 'home address' van het project een *aangepaste* domeinnaam te kiezen. U kunt de URL doorsturen naar de canned site of u kunt een volledig aangepaste home page op de openbare URL hebben en gebruikers

overdragen naar de canned site voor geavanceerde functionaliteit. Doe het in ieder geval zo dat u het adres van het project niet hoeft te wijzigen als u later besluit over te stappen op een andere hosting-oplossing.

Een canned hosting site kiezen
De grootste en bekendste hosting site is SourceForge. Twee andere sites die dezelfde of soortgelijke diensten verlenen, zijn savannah.gnu.org en BerliOS.de. Enkele organisaties, zoals de Apache Software Foundation en Tigris.org[21], verlenen gratis hosting aan open source-projecten die passen bij hun missie en hun bestaande projectengemeenschap.

Haggen So voerde als onderdeel van zijn promotieonderzoek een grondige evaluatie uit van verschillende canned hosting-sites, *Construction of an Evaluation Model for Free/Open Source Project Hosting (FOSPHost) sites*. U vindt de resultaten op http://www.ibiblio.org/fosphost/. Kijk vooral naar het zeer leesbare vergelijkingsdiagram op http://www.ibiblio.org/fosphost/exhost.htm.

Anonimiteit en betrokkenheid
Een probleem, dat weliswaar niet beperkt is tot de canned sites maar daar het meest wordt aangetroffen, is misbruik van de loginfunctionaliteit van gebruikers. De functionaliteit op zich is simpel. De site laat iedere gebruiker zichzelf registreren met een gebruikersnaam en een wachtwoord. Vanaf dat moment bewaart de site een profiel van die gebruiker en projectbeheerders kunnen bepaalde permissies aan een gebruiker toewijzen, bijvoorbeeld het recht om naar de repository te committen.

Dit kan buitengewoon nuttig zijn: het is zelfs een van de grootste voordelen van canned hosting. Het probleem is dat gebruikerslogin soms verplicht gesteld wordt voor taken die ongeregistreerde gebruikers zouden moeten kunnen uitvoeren, met name de mogelijkheid issues te rapporteren in de bug tracker en te reageren op bestaande issues. Door voor dergelijke acties een ingelogde gebruikersnaam te vereisen, legt het project de betrokkenheidslat hoger voor taken die snel en eenvoudig horen te zijn. Natuurlijk wilt u contact kunnen opnemen met iemand die gegevens heeft ingevoerd in de bug tracker, maar een veld waar diegene al zijn gegevens kan invoeren (als hij dat al wil) is daarvoor voldoende. Als een nieuwe gebruiker een bug vaststelt en eerst een compleet account dient te creëren voordat hij de bug kan melden in de bug tracker, raakt hij alleen maar geïrriteerd. Wellicht besluit zo iemand om de bug maar helemaal niet te melden.

De voordelen van gebruikersmanagement zijn doorgaans groter dan de nadelen. Maar als u kunt kiezen welke acties anoniem mogen worden gedaan, zorg er dan voor dat niet alleen *alle* read-only-acties toegestaan zijn aan niet-ingelogde bezoekers, maar ook enkele acties die gegevensinvoer behelzen, met name in de bug tracker en – indien u die hebt – de wikipagina's.

12| Uit zijn boek The Mythical Man Month, 1975. Zie http://en.wikipedia.org/wiki/The_Mythical_Man-Month en http://en.wikipedia.org/wiki/Brooks_Law.

13| Vlak nadat zijn boek was verschenen, schreef Michael Bernstein me: "Er zijn andere e-mail-clients behalve Mutt die een reply-to-list-functie implementeren. Evolution bijvoorbeeld heeft deze functie onder een toetscombinatie, maar niet onder een knop (Ctrl+L)."

14| Sinds ik dat schreef, ben ik te weten gekomen dat ten minste één lijstbeheersysteem over deze functie beschikt: Siesta. Lees hierover ook dit artikel: http://www.perl.com/pub/a/2004/02/05/siesta.html

15| Zie http://cia.vc/stats/vcs en http://subversion.tigris.org/svn-dav-securityspace-survey.html voor bewijs van deze groei.

16| Raadpleeg voor een andere mening voor het onder versie brengen van configure-bestanden, Alexey Makhotkins post 'configure.in and version control' op http://versioncontrolblog.com/2007/01/08/configurein-and-version-control/.

17| Er is geen vereiste of verwachting om te doneren aan Freenode, maar overweeg niettemin een bijdrage te doen als u of uw project het zich kunnen veroorloven. Ze zijn in de VS een van belasting vrijgesteld goed doel en verlenen een waardevolle dienst.

18| Om een channel topic in te stellen, gebruikt u de opdracht /topic. Alle opdrachten in IRC beginnen met "/". Zie http://www.irchelp.org/ als u niet bekend bent met het gebruik en het beheer van IRC. Met name http://www.irchelp.org/irchelp/irctutorial.html is een uitstekend leerboek.

19| Zie hiervoor http://www.xml.com/pub/a/2002/12/18/dive-into-xml.html.

20| Ere wie ere toekomt: dit gedeelte stond niet in de eerste gepubliceerde uitgave van het boek, maar Brian Akers blogstukje 'Release Criteria, Open Source, Thoughts On...' herinnerde me aan het feit dat RSS feeds voor open source-projecten nuttig zijn.

21| Disclaimer: Ik ben in dienst van CollabNet, dat Tigris.org sponsort, en ik gebruik Tigris regelmatig.

4

SOCIALE EN POLITIEKE INFRASTRUCTUUR

De eerste vragen die mensen meestal stellen over open source software zijn: “Hoe werkt het? Wat houdt het project gaande? Wie neemt de beslissingen?”. Ik voel met niet zo prettig bij nietszeggende antwoorden over meritocratie, de geest van samenwerking, de code die voor zichzelf spreekt enz. Feit is nou eenmaal dat deze vraag moeilijk te beantwoorden is. Meritocratie, samenwerken en werkende code maken er allemaal deel van uit. Toch zeggen ze weinig over hoe een project van dag tot dag werkelijk functioneert en ze zeggen helemaal niets over hoe conflicten worden opgelost.

In dit hoofdstuk probeer ik de onderliggende principes te laten zien die alle succesvolle projecten over het algemeen hebben. Ik bedoel succesvol niet alleen wat betreft de technische kwaliteit, maar ook ten aanzien van de operationele gezondheid en het overlevingsvermogen. De operationele gezondheid is het voortdurende vermogen van het project om nieuwe codebijdragen in te passen, nieuwe ontwikkelaars aan te trekken en te reageren op binnenkomende bugrapporten. Het overlevingsvermogen is het vermogen van het project om onafhankelijk van een individuele participant of sponsor te blijven bestaan. Anders gezegd, het is de kans dat het project blijft voortbestaan ook als alle leden die het project gestart zijn andere bezigheden vinden. Technisch succes is niet moeilijk te bewerkstelligen, maar zonder een solide basis van ontwikkelaars en een sociaal fundament kan het gebeuren dat een project de groei die het gevolg is van het aanvankelijke succes of het vertrek van charismatische figuren niet aankan.

Er zijn verscheidene manieren om dit soort succes te bewerkstelligen. Voor sommige daarvan is een formele bestuurlijke structuur nodig, waarbij discussies worden opgelost, nieuwe ontwikkelaars uitgenodigd (soms om te vertrekken) en nieuwe functies gepland enz. Voor andere is een minder formele structuur nodig en meer bewuste terughoudendheid om een atmosfeer van eerlijkheid te creëren waarop mensen kunnen vertrouwen. Beide manieren leiden tot hetzelfde resultaat: een besef van institutionele duurzaamheid, ondersteund door gewoontes en procedures die door alle deelnemers goed begrepen worden. Deze eigenschappen zijn zelfs nog belangrijker in zelforganiserende systemen dan in centraal gecontroleerde, omdat in een zelforganiserend systeem iedereen zich bewust is van het feit - in ieder

geval gedurende een bepaalde tijd - dat een paar rotte appels de hele mand kunnen bederven.

Afsplitsbaarheid of forkability
Het onmisbare ingrediënt dat ontwikkelaars in een open source-softwareproject samenbindt en ze bereidwillig maakt om compromissen te sluiten als dat nodig is, is de *forkability* van de code: dit is de mogelijkheid dat iedereen een kopie van de broncode kan maken en daarmee een concurrerend project *(fork)* kan beginnen. Het tegenstrijdige is dat bij een open source-softwareproject de *mogelijkheid* van een fork belangrijker is dan daadwerkelijke forks, die zeer zeldzaam zijn. Omdat een fork slecht is voor iedereen (de redenen hiervoor worden in detail behandeld in het gedeelte 'Forks' in Hoofdstuk 8*, Het managen van vrijwilligers*) zullen naarmate de dreiging van een fork toeneemt steeds meer mensen bereid zijn om compromissen te sluiten om die te voorkomen.

Forks, of beter gezegd de mogelijkheid voor het ontstaan van een fork, zijn de reden dat er bij een open source-softwareproject geen echte dictator kan bestaan. Dat klinkt misschien verrassend, met name gezien het feit dat het vrij gebruikelijk is dat iemand in een open source-project de 'dictator' of de 'tiran' genoemd wordt. Maar dit soort tirannie is anders en verschilt nogal van de conventionele betekenis van het woord. Stelt u zich eens voor: de onderdanen van een koning kunnen op ieder willekeurig moment een kopie maken van diens hele koninkrijk, daar naartoe verhuizen en naar believen zelf regeren. Zou zo'n koning niet heel anders regeren dan een collega wiens onderdanen gedwongen waren onder zijn bewind te blijven, ongeacht wat hij doet?

Daarom zijn zelfs projecten die formeel niet zijn georganiseerd als een democratie, in de praktijk toch een democratie als het op het nemen van belangrijke beslissingen aankomt. Vermenigvuldigbaarheid impliceert forkability, forkability impliceert consensus. Het kan zijn dat iedereen bereid is zich op één leider te verlaten (het bekendste voorbeeld daarvan is Linus Torvalds in Linux kernel development), maar dit is omdat men daarvoor *gekozen* heeft, op een volkomen niet-cynische, niet-kwaadaardige manier. De dictator heeft geen magische greep op het project. Een belangrijke eigenschap van alle open source-licenties is dat ze de ene partij niet meer zeggenschap geven over hoe de code veranderd of gebruikt mag worden dan de andere. Als de dictator plotseling slechte beslissingen gaat nemen, heeft dit rusteloosheid tot gevolg, gevolgd door opstand en een fork. Uitzonderingen daargelaten, lopen de zaken zelden zo hoog op, omdat de dictator nagenoeg altijd eerst een compromis zoekt.

Maar alleen omdat forkability de zeggenschap beperkt die één persoon kan uitoefenen, wil dat nog niet zeggen dat er geen belangrijke verschillen zijn in hoe projecten bestuurd worden. U wilt niet bij iedere beslissing met de ultieme vraag geconfronteerd worden wie er een fork overweegt. Dat zou erg vermoeiend worden en alleen maar de aandacht afleiden van het echte werk. De volgende twee gedeeltes gaan over verschillende manieren om projecten op zo'n manier te organiseren dat het nemen van de meeste beslissingen soepel verloopt. Het gaat om twee enigszins geïdealiseerde, extreme voorbeelden. Veel projecten zitten hier ergens tussenin.

4.1 VRIENDELIJKE DICTATORS

Het *vriendelijkedictatormodel* is precies wat het lijkt. De uiteindelijke beslissingsbevoegdheid ligt bij één persoon, van wie op basis van zijn persoonlijkheid en ervaring verwacht wordt dat hij deze verstandig gebruikt.

Alhoewel 'vriendelijke dictator' (VD) de standaardterm is voor deze functie, is het beter te denken in termen van een 'door de gemeenschap goedgekeurde scheidsrechter' of 'rechter'. Over het algemeen nemen VD's niet alle beslissingen; vaak nemen ze zelfs niet eens de meeste beslissingen. Het is onwaarschijnlijk dat één persoon over voldoende kennis beschikt om constant de juiste beslissingen te nemen ten aanzien van alle aspecten van het project. Wat zeker is, is dat goede ontwikkelaars niet bij een project betrokken blijven, tenzij ze invloed kunnen uitoefenen op de koers ervan. Daarom valt er voor een VD over het algemeen weinig te dicteren. In plaats daarvan laat hij, wanneer mogelijk, de problemen zichzelf oplossen door middel van discussies en experimenten. Aan deze discussies neemt hij zelf deel, maar dan als gewone ontwikkelaar. Vaak voegt hij zich naar een ander die meer kennis heeft van dat specifieke gebied. Alleen wanneer het duidelijk wordt dat er geen consensus bereikt gaat worden en dat de meerderheid *wil* dat iemand de beslissing neemt ter wille van de voortgang van de ontwikkeling, dan houdt hij zijn poot stijf en zegt 'zo gaan we het doen'. Terughoudendheid in het nemen van beslissingen op deze manier is een kenmerk van vrijwel alle succesvolle VD's en het is één van de redenen waarom ze hun positie kunnen behouden.

Wie is geschikt als vriendelijke dictator?
VD zijn vraagt om een combinatie van eigenschappen. In de eerste plaats heeft men een goed ontwikkeld gevoel nodig voor de eigen invloed op het project, wat op zijn beurt weer leidt tot zelfbedwang. In het begin van een discussie moet de VD vermijden om met grote stelligheid meningen en conclusies naar voren te brengen. Anders krijgen anderen het gevoel dat het zinloos is om een afwijkende mening te uiten. Mensen moeten zich vrij voelen om ideeën te kunnen ventileren, ook stomme ideeën. Het is onvermijdelijk dat de VD zelf van tijd tot tijd ook met een dom idee op de proppen komt. Daarom moet hij tevens over het vermogen beschikken om een eigen fout te herkennen en te erkennen (*iedere* goede ontwikkelaar zou die eigenschap natuurlijk moeten hebben, vooral als hij lang bij het project betrokken blijft). Het verschil is dat de VD zich af en toe een foutje kan permitteren zonder zich zorgen te hoeven maken zijn geloofwaardigheid op lange termijn. Ontwikkelaars met minder ervaring voelen zich misschien minder zeker. Daarom moet de VD zijn kritiek of die beslissingen die negatief uitpakken voor de minderervaren ontwikkelaar goed wegen, zowel op technisch als op psychologisch vlak.

De VD hoeft *niet* degene te zijn met de beste technische vaardigheden binnen het project. Hij moet bekwaam genoeg zijn om aan de code zelf te kunnen werken en een verandering te kunnen begrijpen, te bekritiseren en af te wegen, maar dat is alles. De VD-functie wordt niet verkregen of behouden op basis van intimiderend goede programmeervaardigheden. Wat *wel* belangrijk is, is ervaring en een zesde zintuig voor ontwerp als zodanig, niet noodzakelijkerwijs de vaardigheid om een goed ontwerp te produceren, maar de vaardigheid om het te herkennen, ongeacht waar het vandaan komt.

Het is meestal zo dat de VD ook de oprichter van het project is, maar dit is meer een logisch dan een causaal verband. De verschillende kwaliteiten die iemand in staat stellen om met succes een project te beginnen (technische deskundigheid, het vermogen andere mensen over te halen zich bij het project aan te sluiten enz.) zijn precies de kwaliteiten die een VD nodig heeft. En oprichters van een project hebben natuurlijk automatisch al het voordeel van anciënniteit, waardoor voor alle betrokkenen VD-schap van deze persoon vaak de weg van de minste weerstand is.

Vergeet niet dat wat betreft een fork het mes aan twee kanten snijdt. Een VD kan net zo makkelijk een fork van een project maken als iemand anders en sommige VD's hebben dat dan ook gedaan, toen ze het gevoel kregen de richting die zij met het project op wilden niet strookte met die van de meerderheid van de ontwikkelaars. Omdat forkability bestaat, maakt het niet uit of de VD systeembeheerdersprivileges van de belangrijkste servers van het project of niet. Mensen denken vaak dat controle over de server de ultieme macht binnen een project betekent, maar in feite is het niet relevant. De mogelijkheid om op een bepaalde server wachtwoorden van mensen toe te voegen of te verwijderen, heeft alleen betrekking op de kopie die zich op die server bevindt. Voortdurend misbruik van die macht, of dat nu door de VD is of door iemand anders, zou alleen maar leiden tot verplaatsing van de ontwikkeling naar een andere server.

Of uw project een VD zou moeten hebben of misschien beter geleid zou worden met een minder gecentraliseerd systeem, hangt voornamelijk af van wie beschikbaar is voor de functie. Een algemene regel is dat als het voor iedereen overduidelijk is wie de VD zou moeten zijn, dat de weg is die u in moet slaan. Maar als er geen overduidelijke kandidaat is voor het VD-schap, kan het project beter gebruik maken van een gedecentraliseerd beslissingsproces zoals wordt beschreven in het volgende gedeelte.

4.2 CONSENSUSDEMOCRATIE

Wanneer projecten een tijdje lopen, hebben ze de neiging het model van VD-schap te willen verlaten ten gunste van een open, democratisch systeem. Dit is niet noodzakelijkerwijs uit ontevredenheid over een bepaalde VD. Een op de groep gebaseerd bestuur is nou eenmaal 'evolutionair stabieler', om maar eens een metafoor uit de biologie te gebruiken. Iedere keer dat een VD aftreedt of probeert de verantwoordelijkheid voor het nemen van beslissingen gelijkmatiger te verdelen, is dat een kans voor de groep om een nieuw, niet-dictatoriaal systeem te vestigen: om een nieuwe grondwet vast te stellen als het ware.

De groep zal misschien niet van de eerste gelegenheid gebruik maken, of van de tweede, maar uiteindelijk gebeurt het toch. En als dat eenmaal het geval is, is er meestal geen weg terug meer.

Gezond verstand zegt waarom: als een groep van N mensen een persoon speciale macht toekent, betekent dat dat N - 1 mensen allemaal instemden met het feit dat hun individuele invloed minder zou worden. Mensen doen dat over het algemeen liever niet. En zelfs als ze het doen, dan nog is het resulterende dictatorschap slechts voorwaardelijk. De groep heeft de dictator gezalfd en het is duidelijk dat de groep de dictator ook kan afzetten. Wanneer het project eenmaal overgestapt is van een charismatisch leider naar een meer formeel, op de groep gebaseerd leiderschap, keert men maar zelden op zijn schreden terug.

Hoewel deze systemen in detail zeer verschillend functioneren, hebben ze twee elementen gemeen. Ten eerste werkt de groep meestal met consensus; ten tweede kan men terugvallen op een formele stemprocedure mocht geen consensus bereikt kunnen worden.

Consensus betekent alleen maar een overeenkomst waarmee iedereen bereid is te leven. Een groep heeft consensus bereikt over een bepaalde kwestie wanneer iemand voorstelt dat consensus is bereikt en niemand dit tegenspreekt. De persoon die consensus voorstelt moet uiteraard helder verwoorden wat deze consensus inhoudt en welke acties er ondernomen moeten worden, voor zover dit niet reeds duidelijk is.

De meeste discussies in een project gaan over technische onderwerpen, zoals de juiste manier om een bepaalde 'bug' te herstellen, het wel of niet toevoegen van een nieuwe functie, hoe strikt interfaces gedocumenteerd moeten worden enz. Op consensus gebaseerd bestuur werkt goed omdat het naadloos aansluit op de technische discussie zelf. Aan het eind van de discussie is er vaak eenstemmigheid over de te volgen koers. Meestal zal iemand een concluderende post voor de mailinglist maken. Dit is tegelijkertijd een samenvatting is van wat er beslist is en impliciet een voorstel tot consensus. Dit is tevens de laatste kans om te zeggen 'Wacht even, ik ben het daar niet mee eens. We moeten hier nog eens goed naar kijken.'

Voor kleine en niet-controversiële beslissingen is het voorstel tot consensus impliciet. Wanneer een ontwikkelaar bijvoorbeeld spontaan een bugfix commit, dan is deze commit op zichzelf al een voorstel tot consensus: "Ik neem aan dat we het er allemaal over eens zijn dat deze bug hersteld moet worden en dat dit de manier is om hem te herstellen." Natuurlijk zegt de ontwikkelaar dit niet daadwerkelijk. Hij repareert de bug gewoon. De anderen binnen het project hoeven niet expliciet akkoord te gaan: wie zwijgt, stemt toe. Als iemand een verandering commit waarover achteraf *geen* consensus blijkt te bestaan, dan is de oplossing simpelweg, dat deze verandering binnen het project opnieuw ter discussie wordt gesteld alsof de aanpassing nog niet was gecommit. Waarom dit werkt is het onderwerp van het volgende gedeelte.

Versiecontrole betekent dat u kunt relaxen

Het feit dat de broncode van het project onder versiebeheer wordt gehouden, betekent dat de meeste beslissingen makkelijk ongedaan gemaakt kunnen worden. Dit gebeurt meestal als iemand een verandering commit in de veronderstelling dat iedereen daar blij mee is, en achteraf blijkt dat er bezwaren tegen bestaan. Kenmerkend voor zulke bezwaren is dat ze meestal beginnen met een verontschuldiging voor het missen van de voorafgaande discussie. Dit blijft echter achterwege als degene die het bezwaar maakt in de archieven van de mailinglist geen sporen van een

dergelijke discussie vindt. Er is hoe dan ook geen reden om, nadat de verandering gecommit is, de inhoudelijke discussie op een andere toon te voeren dan daarvoor. Iedere verandering kan worden teruggedraaid, in ieder geval tot het moment dat daarvan afhangende nieuwe veranderingen worden ingevoerd (d.w.z. nieuwe code die niet meer zou werken als die eerdere verandering zou verdwijnen). Het versiebeheersysteem geeft het project de mogelijkheid om de gevolgen van verkeerde of haastig genomen beslissingen ongedaan te maken. Dit geeft mensen op hun beurt de vrijheid om op hun instinct te vertrouwen ten aanzien van de hoeveelheid feedback die nodig is voordat ze iets doen.

Het betekent ook dat het proces om consensus te verkrijgen niet al te formeel hoeft te zijn. Binnen de meeste projecten wordt dit op gevoel gedaan. Minimale veranderingen kunnen zonder discussie geaccepteerd worden, of na een minimale discussie gevolgd door enkele instemmende knikjes. Bij meer ingrijpende veranderingen, vooral als die veel code kunnen destabiliseren, moet men een dag of twee wachten voordat aangenomen kan worden dat er consensus is. Niemand mag natuurlijk buiten een belangrijke discussie gehouden worden alleen omdat hij zijn e-mail niet vaak genoeg heeft gelezen.

Wanneer iemand er dus zeker van is dat hij weet wat er gedaan moet worden, dan moet hij dat gewoon doen. Dit geldt niet alleen voor softwarefixes, maar ook voor updates van websites, veranderingen in documenten en verder alles waarvan aangenomen mag worden dat het niet controversieel is. Normaal gesproken komt het maar zelden voor dat een actie ongedaan gemaakt moet worden en deze gevallen kunnen van geval tot geval behandeld worden. Het spreekt vanzelf dat u mensen niet moet aanmoedigen om koppig te zijn. Er is altijd een psychologisch verschil tussen een besluit waarover nog gediscussieerd wordt en een besluit dat al is doorgevoerd, ook al is het technisch terug te draaien. Mensen hebben altijd het gevoel dat daadkracht en actie hand in hand gaan en zullen een verandering liever willen voorkomen dan hem later terugdraaien. Als een ontwikkelaar misbruik maakt van dit feit door potentieel controversiële veranderingen te snel te committen, kunnen en moeten mensen hierover klagen en moet deze ontwikkelaar strenger worden gecontroleerd totdat er verbetering zichtbaar is.

Als geen consensus bereikt wordt, stem dan

Het is niet te vermijden dat in sommige discussies geen consensus bereikt wordt. Als alle andere manieren om een patstelling op te lossen falen, dan moet er gestemd worden. Maar voordat er kan worden gestemd, moet er een aantal duidelijke keuzemogelijkheden op het stembiljet worden gezet. Ook hier loopt het normale proces van de technische discussie op vloeiende wijze over in de beslissingsprocedures van het project. Het soort vragen waarover gestemd moet worden betreft vaak complexe zaken met vele facetten. In dergelijke complexe discussies zijn er gewoonlijk een of twee mensen die de rol spelen van *eerlijke bemiddelaars.* Ze posten regelmatig samenvattingen van de verschillende argumenten en houden de belangrijkste punten van onenigheid (en overeenstemming) bij. Dankzij deze samenvattingen heeft iedereen een beter idee van de voortgang die is geboekt en welke zaken nog aandacht behoeven. De samenvattingen kunnen ook als prototype dienen voor het stembiljet, mocht er gestemd moeten worden. Als de eerlijke bemiddelaars hun werk goed doen, kunnen ze geloofwaardig om een stemming vragen als de tijd daar is en zal de groep bereid zijn het stembiljet te gebruiken dat gebaseerd is op hun samenvattingen van de problematiek. De bemiddelaars zelf kunnen deelnemen aan de discussie. Ze hoeven niet buiten het gekrakeel te blijven, zolang ze de mening van de anderen maar begrijpen en die eerlijk weergeven, en zolang ze de discussie neutraal blijven samenvatten en niet hun eigen standpunten laten prevaleren.

De uiteindelijke inhoud van het stembiljet is gewoonlijk niet controversieel. Tegen de tijd dat er gestemd moet worden, is de onenigheid meestal tot een paar essentiële kwesties teruggebracht, met herkenbare labels en korte beschrijvingen. Af en toe protesteert een ontwikkelaar tegen het stembiljet zelf. Soms is dit bezwaar legitiem, bijvoorbeeld als een belangrijke keuze niet is opgenomen of niet correct is beschreven. Maar het komt voor dat een ontwikkelaar alleen maar het onvermijdelijke probeert af te wenden, wetende dat de stemming waarschijnlijk niet in het voordeel van zijn keuze zal uitvallen. Raadpleeg het gedeelte ‘Moeilijke mensen’ in *Hoofdstuk 6, Communicatie* om te zien hoe u om moet gaan met dit soort tegenwerking.

Vergeet niet het kiessysteem vast te leggen. Er bestaan allerlei systemen en men zou een verkeerd idee kunnen krijgen over welke procedure gevolgd wordt. Een goede keus in de meeste gevallen is *instemmingsverkiezing*, waarbij iedere kiezer voor zoveel opties op het stembiljet kan kiezen als hij wil. Instemmingsverkiezingen zijn makkelijk uit te leggen en te tellen en hebben in tegenstelling tot andere methodes maar één verkiezingsronde. Zie http://en.wikipedia.org/wiki/Voting_system#List_of_systems voor meer informatie over instemmingsverkiezingen en andere kiessystemen. Probeer echter uitgebreide discussies over het te gebruiken kiessysteem te vermijden, want voor u het weet raakt u in een discussie verzeild over welk kiessysteem te gebruiken om het kiessysteem te kiezen!). Eén van de redenen waarom een instemmingsverkiezing een goede optie is, is omdat er moeilijk iets tegenin te brengen valt: veel eerlijker dan dit kiessysteem kan bijna niet.

Voer ten slotte de stemming openbaar uit. Er is geen reden tot geheimhouding of anonimiteit in een stemming over zaken waarvan de discussie toch al publiekelijk gevoerd is. Laat iedere kiezer zijn stem posten op de mailinglijst van het project, zodat iedereen de stemming zelf kan natellen en alles in het archief opgenomen kan worden.

Wanneer te stemmen?

Het moeilijkste van verkiezingen is beslissen wanneer u ze gaat houden. In principe zou een stemming een uitzondering moeten zijn. Het is een laatste redmiddel wanneer alle andere opties gefaald hebben. Denk niet dat stemmen een geweldige manier is om een discussies op te lossen. Dat is het niet. Het maakt weliswaar een einde aan de discussie, maar daarmee ook aan het creatief denken over het probleem. Zolang de discussie gaande is, is er een mogelijkheid dat iemand met een nieuwe oplossing komt die iedereen bevalt. Verrassend genoeg gebeurt dit vaak: een levendige discussie kan tot een nieuwe manier van denken over het probleem en een voorstel leiden waarmee uiteindelijk iedereen tevreden is. Zelfs wanneer er geen nieuw voorstel komt dan is het nog steeds beter om een compromis te sluiten dan een stemming te houden. Na een compromis is *iedereen* een klein beetje onte-

vreden; na een stemming zijn sommigen blij, terwijl anderen zeer ontevreden zijn. Vanuit een politiek standpunt gezien heeft de eerste situatie de voorkeur. Daarbij kan iedereen ten minste het gevoel hebben dat hij een prijs bedongen heeft voor zijn ontevredenheid. Hij is weliswaar ontevreden, maar dat zijn alle anderen ook.

Het belangrijkste voordeel van stemmen is dat het uiteindelijk een probleem oplost, waardoor iedereen weer verder kan. Maar de oplossing komt tot stand door de koppen te tellen, in plaats door een rationele dialoog die iedereen uit doet komen bij dezelfde conclusie. Hoe ervarener mensen zijn met open source-projecten, des minder happig ze zijn om problemen op te lossen door middel van een stemming. In plaats daarvan zullen ze proberen om eerder voorgestelde oplossingen nog eens grondig te bekijken of hun eigen standpunt nog wat verder bij te stellen dan ze aanvankelijk gepland hadden om een compromis te bereiken. Er bestaan diverse technieken om een voorbarige stemming te voorkomen. Het meest voor de hand liggende is om te zeggen “Ik denk niet dat we al een stemming toe zijn” en dan uitleggen waarom niet. Een ander manier is om te vragen om een informele (niet bindende) stemming door handopsteken. Als de respons duidelijk overhelt naar een bepaalde kant, dan zijn sommigen eerder geneigd tot een compromis, waardoor een echte stemming niet meer nodig is. Het meest effectief is echter het aanbieden van een nieuwe oplossing of een nieuwe kijk op een oud idee, zodat men zich opnieuw over de kwestie buigt in plaats van alleen maar dezelfde argumenten te herhalen.

In bepaalde zeldzame gevallen is iedereen het erover eens dat alle beschikbare compromissen stuk voor stuk slechter zijn dan de afzonderlijke oplossingen zonder compromis. Wanneer dat gebeurt, valt er minder in te brengen tegen een stemming, enerzijds omdat het waarschijnlijk is dat het tot een betere oplossing leidt en anderzijds omdat mensen, ongeacht de uitkomst, niet bijzonder ontevreden zullen zijn. Maar zelfs dan mag de stemming nooit overhaast plaatsvinden. De discussie die plaatsvindt in de aanloop naar de stemming is hetgeen waarvan de stemmers leren. Die discussie vroegtijdig stoppen komt de kwaliteit van het resultaat niet ten goede.

(NB: Het advies om liever geen stemming te houden, gaat niet op voor het stemmen over het opnemen van een verandering zoals beschreven in het gedeelte ‘Een release stabiliseren’ in Hoofdstuk 7*, Downloadpakketten maken, releases en dagelijkse ontwikkeling*. In dat geval is stemmen meer een communicatiemechanisme, een manier om iemands betrokkenheid te registreren in het proces van het evalueren van veranderingen, zodat iedereen kan zien hoeveel feedback een bepaalde verandering ontvangen heeft.)

Wie mag er stemmen?

Als u eenmaal een kiessysteem hebt gekozen, dient de kwestie van het electoraat zich aan: wie mag er stemmen? Deze vraag ligt gevoelig, omdat op dit moment officieel erkend moet worden dat binnen het project sommigen meer betrokken zijn of een betere beoordeling kunnen maken dan anderen.

De beste oplossing is om een al bestaand onderscheid te nemen, namelijk dat voor commit access en daaraan stemprivileges toe te voegen. In projecten waar zowel gehele als gedeeltelijke toegang mogelijk is, hangt de vraag of mensen met een gedeeltelijke toegang mogen stemmen grotendeels af van het proces waarmee de toegang verleend wordt. Als het project deze toegang makkelijk verleent, bijvoorbeeld als een manier om de vele door derden aangedragen tools in de repository te onderhouden, dan zou duidelijk gemaakt moeten worden dat de gedeeltelijke toegang alleen geldt voor het aandragen van veranderingen en niet het stemmen. Omgekeerd is dit natuurlijk eveneens van toepassing: omdat volwaardige committers stemprivileges *zullen* hebben, moeten ze niet alleen als programmeurs gekozen worden maar ook als stemgerechtigden. Als iemand op de mailinglijst verstorend of dwars gedrag vertoont, dan moet de groep zeer terughoudend zijn in het benoemen van deze persoon tot committer, ook al is hij technisch bekwaam.

Het kiezen van nieuwe committers – zowel met volledige als gedeeltelijke toegang - zou eigenlijk ook met het kiessysteem moeten gebeuren. Maar dit is een van die zeldzame gevallen waarbij geheimhouding gewenst is. U kunt niet op een publieke mailinglijst stemmen over potentiële committers, omdat de kandidaat gekwetst en zijn reputatie geschaad kan worden. In plaats daarvan is het gebruikelijk dat een committer een voorstel om iemand anders commit-toegang te verlenen op een mailinglijst post waartoe alleen andere committers toegang hebben. De andere committers kunnen vrijuit spreken, omdat ze weten dat de discussie privé is. Vaak is er geen verschil van mening en is er dus geen stemming nodig. Na een paar dagen gewacht te hebben om er zeker van te zijn dat iedere committer de kans heeft gehad te reageren, zendt degene die het voorstel gedaan heeft de kandidaat bericht en biedt hem volledige toegang aan. Als er wel verschil van mening is dan volgt er, zoals voor elke kwestie, een discussie, die kan uitmonden in een stemming. Om dit proces eerlijk en open te kunnen laten verlopen, zou het feit dat erover wordt gediscussieerd al geheim moeten zijn. Als de desbetreffende persoon zou weten wat er gaande was en daarna geen volledige toegang aangeboden zou krijgen, zou hij kunnen concluderen dat hij de stemming verloren heeft en zou hij zich waarschijnlijk gekwetst voelen. Als iemand nadrukkelijk om volledige toegang vraagt, is er uiteraard geen andere keus dan het voorstel te overwegen en iemand expliciet te accepteren of af te wijzen. Als dat laatste het geval is, dan moet dat zo beleefd mogelijk gebeuren en met een duidelijke uitleg: “We stellen je patches zeer op prijs, maar we hebben er nog niet genoeg gezien om te kunnen oordelen”, of “We waarderen al je patches, maar ze hadden nog aanzienlijke aanpassingen nodig voordat ze in gebruik genomen konden worden. We zijn dus nog niet voldoende gerustgesteld om je volledige toegang te geven. We hopen dat dit na verloop van tijd zal veranderen”. Vergeet niet dat wat u zegt een klap in iemands gezicht kan zijn, afhankelijk van het zelfvertrouwen van een persoon. Probeer het bericht te schrijven vanuit zijn perspectief.

Omdat het toevoegen van een nieuwe committer meer consequenties heeft dan de meeste andere eenmalige beslissingen, hebben sommige projecten speciale voorwaarden voor de stemming. Een voorstel moet bijvoorbeeld minimaal *n* positieve stemmen krijgen en geen negatieve, of een grote meerderheid moet voor een voorstel stemmen. De exacte parameters doen er niet toe. Het belangrijkste is dat de groep voorzichtig moet zijn met het toelaten van nieuwe committers. Gelijke, of zelfs strengere bijzondere voorwaarden kunnen van toepassing zijn op stemmingen

om een committer te *verwijderen*, hoewel dit hopelijk nooit nodig zal zijn. Zie het gedeelte 'Committers' in Hoofdstuk 8*, Het managen van vrijwilligers* voor meer informatie over de niet aan het stemmen gerelateerde aspecten van het toevoegen of verwijderen van committers.

Opinie peilen versus stemmen
Voor bepaalde stemmingen kan het nuttig zijn het aantal stemgerechtigden uit te breiden. Als de ontwikkelaars bijvoorbeeld niet kunnen bepalen of een bepaalde keus voor een interface past bij de manier waarop mensen daadwerkelijk de software gebruiken, kan het een oplossing zijn om aan alle deelnemers van de mailinglijst van het project te vragen om te stemmen. Dit is eigenlijk meer een *peiling* dan een stemming, maar de ontwikkelaars kunnen ervoor kiezen om het resultaat als bindend te beschouwen. Zoals met elke peiling moet u aan de deelnemers duidelijk maken dat ze kunnen reageren. Als iemand met een beter idee komt dan wat in de vragen van de peiling staat, dan kan deze respons achteraf het belangrijkste resultaat van de peiling blijken te zijn.

Veto's
Sommige projecten kennen een vetorecht. Een veto is een mogelijkheid voor een ontwikkelaar om een haastige of niet goed overwogen verandering een halt toe te roepen, of deze in ieder geval zo lang te vertragen dat men er over kan discussiëren. Een veto zit tussen een zeer sterk bezwaar en obstructie in. De exacte betekenis ervan varieert van project tot project. Binnen sommige projecten is het erg moeilijk een veto terzijde te schuiven, bij andere kan een veto ongeldig worden verklaard met een meerderheid van stemmen, eventueel na een geforceerd uitstel om verder te kunnen discussiëren. Ieder veto moet vergezeld gaan van een grondige uitleg. Een veto zonder zo'n uitleg zou meteen ongeldig moeten worden verklaard.

Vetorecht leidt altijd tot de mogelijkheid van vetomisbruik. Soms willen ontwikkelaars door middel van een veto de druk verhogen, terwijl er eigenlijk alleen behoefte bestond aan meer discussie. U kunt misbruik van veto's voorkomen door zelf terughoudend te zijn met het uitspreken van veto's en door het vriendelijk te melden wanneer iemand zijn vetorecht te vaak gebruikt. Indien nodig kunt u de groep er ook aan herinneren dat veto's alleen bindend zijn zolang de groep bepaalt dat ze dat zijn. Als een duidelijke meerderheid van de ontwikkelaars uiteindelijk X wil, dan is X wat hoe dan ook gebeurt. Of de ontwikkelaar die het veto uitsprak trekt dit terug, of de groep besluit het gewicht van het veto te verminderen.

Het komt voor dat mensen '-1' schrijven om hun veto uit te spreken. Dit gebruik komt van de Apache Software Foundation, die gebruik maakt van een zeer gestructureerd stem- en vetoproces. Het staat beschreven op http://www.apache.org/foundation/voting.html. De Apache-normen hebben zich verspreid over andere projecten en u kunt zien dat hun afspraken, in verschillende gradaties, gebruikt worden op veel plaatsen in de open source-wereld. Technisch gezien betekent '-1' niet altijd een formeel veto, zelfs niet volgens de Apache-normen, maar informeel wordt het meestal opgevat als een veto, of op zijn minst als een zeer sterk bezwaar ergens tegen.

Net zoals stemmen kunnen veto's met terugwerkende kracht gelden.
Het is niet correct om tegen een veto bezwaar te maken op grond van het feit dat de aanpassing in kwestie al toegepast is of de betreffende actie al ondernomen (tenzij het iets is dat niet terug te draaien is, zoals een persbericht).

Aan de andere kant is de kans dat een veto dat weken later uitgesproken wordt serieus wordt genomen erg klein; en zo hoort het ook.

4.3 ALLES OPSCHRIJVEN

Op een gegeven moment wordt het aantal afspraken en overeenkomsten binnen het project zo groot dat u ze ergens vast moet leggen. Om zo'n document geldigheid te geven moet u duidelijk maken dat het gebaseerd is op discussies uit de mailinglijst en op besluiten die al in werking zijn getreden. Refereer bij het samenstellen van dit document aan de betreffende threads in de archieven van de mailinglijst. Wanneer er een punt is waar u niet zeker van bent, vraag het dan opnieuw. Het document mag niet voor verassingen zorgen. Het is niet het uitgangspunt van de afspraken maar slechts een beschrijving ervan. Als het succesvol is, is het natuurlijk wel zo dat mensen het zullen citeren alsof het een bron met autoriteit is. Dat betekent eigenlijk alleen maar dat het de globale mening van de groep goed weergeeft.

Dit is het document waaraan gerefereerd wordt in het gedeelte 'Ontwikkelaarsrichtlijnen' in Hoofdstuk *2, Aan de gang* Wanneer het project zich nog in de beginfase bevindt, kunt u bij het bepalen van de richtlijnen niet terugvallen op een lange projectgeschiedenis. Naarmate de ontwikkelaarsgemeenschap echter groeit en volwassener wordt, kunt u de tekst aanpassen aan de hand van de feitelijke gang van zaken.

Probeer niet alles op te schrijven. Geen enkel document kan alles bevatten wat mensen moeten weten over hoe ze deel moeten nemen aan een project. Veel van de afspraken die betrekking hebben op een project blijven altijd onuitgesproken en worden nooit expliciet benoemd, maar worden toch door iedereen nageleefd. Andere zaken zijn gewoon zo overduidelijk dat ze niet genoemd hoeven worden en alleen maar afleiden van de belangrijke, minder voor de hand liggende zaken. Het heeft bijvoorbeeld geen zin richtlijnen te schrijven als "Wees beleefd en respectvol naar anderen op de mailinglijst en houd u verre van vuilbekkerij" of "Schrijf goede en leesbare codes zonder bugs". Natuurlijk zijn dit allemaal wenselijke zaken maar omdat er geen universum denkbaar is waarin dit *niet* zo is heeft het geen zin ze te noemen. Als mensen onbeleefd zijn op de mailinglijst of een code vol bugs schrijven, dan zullen ze daar niet mee stoppen omdat de richtlijnen van het project dat zeggen. Dergelijke situaties moeten opgelost worden op het moment dat ze de kop opsteken en niet door algemene aansporingen om braaf te zijn. Aan de andere kant, als het project specifieke richtlijnen heeft *hoe* een goede code moet worden geschreven, bijvoorbeeld regels om iedere API te documenteren in een bepaalde format, dan moeten deze richtlijnen zo compleet mogelijk vastgelegd worden.

Een goede manier om te bepalen wat u op moet nemen in dit document is om het te baseren op de vragen die nieuwkomers het meest stellen en op de klachten die ervaren ontwikkelaars het vaakst uiten. Dit houdt niet automatisch in dat dit document een FAQ-pagina wordt. Het moet waarschijnlijk een meer consistente en beschrijvende structuur hebben dan een FAQ kan bieden. Ook in dit document zou de aanpak van de kwesties echter op de realiteit gebaseerd moet zijn, in plaats van te anticiperen op de kwesties zie zich eventueel zouden kunnen voordoen.

Als het project geleid wordt door een VD of als er mensen zijn met speciale bevoegdheden (manager, voorzitter, wat dan ook), dan biedt dit document tevens een goede gelegenheid om de opvolgingsprocedures vast te leggen. Soms kan dit zo simpel zijn als een lijstje met namen van mensen die de VD kunnen vervangen, mocht hij het project om wat voor reden dan ook verlaten. Over het algemeen kan alleen de VD een opvolger benoemen. Als er gekozen verantwoordelijken zijn, dan moet de procedure hoe deze mensen zijn genomineerd en gekozen beschreven worden in het document. Als er aanvankelijk geen procedure was, dan moet er consensus worden bereikt over een procedure in de mailinglijst *voordat* u erover schrijft. Mensen kunnen soms lange tenen hebben als het om hiërarchische structuren gaat. Daarom moet hier omzichtig mee worden omgegaan.

Misschien is het wel het belangrijkst om duidelijk te maken dat regels heroverwogen kunnen worden. Als de afspraken zoals beschreven in het document het project gaan hinderen, herinner iedereen er dan aan dat het verondersteld wordt een reflectie te zijn van de intenties van de groep en geen bron van frustratie en blokkades. Als iemand er een gewoonte van maakt om de regels iedere keer dat ze hem in de weg staan te willen herzien, dan hoeft u hier niet altijd met deze persoon over in discussie te gaan. Soms is zwijgen de beste tactiek. Als andere mensen het eens zijn met de klachten, dan zullen ze de klager bijvallen en is het duidelijk dat er iets moet veranderen. Maar als niemand het met hem eens is, dan zal de klager weinig respons krijgen en blijven de regels zoals ze zijn.

Twee goede voorbeelden van richtlijnen binnen een project zijn het Subversion-bestand `hacking.html`, op http://svn.collab.net/repos/svn/trunk/www/hacking.html en de governance-documenten van de Apache Software Foundation, op http://www.apache.org/foundation/how-it-works.html en http://www.apache.org/foundation/voting.html. De ASF is in feite een verzameling van softwareprojecten, wettelijk georganiseerd als een organisatie zonder winstoogmerk. De bijhorende documenten beschrijven dus meer de governance-procedures dan de afspraken over de ontwikkeling van programmatuur. Maar ze blijven de moeite van het lezen waard, omdat ze de verzamelde ervaringen van veel open source-projecten vertegenwoordigen.

5

GELD

In dit hoofdstuk wordt gekeken hoe een open source-softwareomgeving aan fondsen kan komen. Dit is niet alleen bestemd voor ontwikkelaars die worden betaald om aan open source-softwareprojecten te werken, maar ook voor hun managers die inzicht moeten hebben in de sociale dynamiek van de ontwikkelomgeving. In dit hoofdstuk wordt ervan uitgegaan dat de aangesproken persoon ('u') een betaalde ontwikkelaar is, of iemand die leiding geeft aan deze ontwikkelaars. Voor beiden is het advies vaak hetzelfde en wanneer dit niet zo is, wordt in de context duidelijk gemaakt voor wie het bedoeld is.

Commerciële financiering van de ontwikkeling van open source-software is geen nieuw verschijnsel. Veel van deze ontwikkeling is altijd op informele wijze bekostigd. Wanneer een systeembeheerder een netwerkanalyse-instrument schrijft om hem te helpen zijn werk te doen, dit vervolgens online zet en bugfixes en functiebijdragen krijgt van andere systeembeheerders, is er in feite een informeel consortium gevormd. De financiële middelen van het consortium komen uit het salaris van de systeembeheerders. Kantoorruimte en netwerkbandbreedte worden, zonder dat men dit beseft, gedoneerd door de organisaties waarvoor zij werken. Die organisaties hebben natuurlijk baat bij de investering, ook al zijn ze zich daar in het begin niet van bewust.

Het verschil is dat tegenwoordig veel van deze inspanningen worden geformaliseerd. Bedrijven zijn zich bewust geworden van de voordelen van open source-software en zijn zelf directer betrokken geraakt bij de ontwikkeling ervan. Ook ontwikkelaars zijn gaan verwachten dat zeer belangrijke projecten ten minste donaties aantrekken en mogelijk zelfs lange-termijnsponsors. Hoewel de aanwezigheid van geld de basisdynamiek van open source-softwareontwikkeling niet heeft gewijzigd, heeft het wel in hoge mate de schaal veranderd waarop dingen gebeuren, zowel wat betreft het aantal ontwikkelaars als de tijd per ontwikkelaar. Het heeft tevens invloed gehad op de wijze waarop projecten worden georganiseerd en op de interactie van de daarbij betrokken partijen. Het gaat er niet alleen om hoe het geld wordt uitgegeven of hoe het rendement op de investering wordt gemeten. Het gaat tevens over management en processen: hoe kunnen de hiërarchische leidinggevende structuren van bedrijven en de semigedecentraliseerde commissies van vrijwilligers van open

source-softwareprojecten op productieve wijze met elkaar samenwerken? Zullen ze het eens kunnen worden over de betekenis van 'op productieve wijze'?

Financiële steun wordt doorgaans toegejuicht door open source-ontwikkelgemeenschappen. Het kan de kwetsbaarheid van een project voor de 'macht van de chaos' verminderen waardoor zoveel projecten worden weggevaagd voordat ze echt van de grond zijn gekomen. Daardoor kan het mensen bereidwilliger maken om de software een kans te geven: ze hebben het gevoel dat ze hun tijd investeren in iets wat er over zes maanden nog steeds zal zijn. Geloofwaardigheid immers, is tot op zekere hoogte aanstekelijk. Als bijvoorbeeld IBM een open source-project steunt, gaan mensen er gevoeglijk vanuit dat het project niet zal mislukken. De daaruit voortvloeiende bereidheid om zich voor het project in te zetten, kan ertoe bijdragen dat de voorspelling zichzelf verwezenlijkt.

Financiering gaat echter ook gepaard met het concept controle. Als niet voorzichtig met het fenomeen geld wordt omgegaan, kan het een project verdelen in twee groepen, groepsontwikkelaars en ontwikkelaars buiten de groep. Als de onbetaalde vrijwilligers het gevoel krijgen dat ontwerpbesluiten of functietoevoegingen eenvoudigweg beschikbaar zijn voor de hoogste bieder, stappen ze over op een project dat meer op een meritocratie lijkt en minder op onbetaald werk ten gunste van iemand anders. Het kan zijn dat ze nooit openlijk klagen op de mailinglijsten. In plaats daarvan houden de vrijwilligers geleidelijk op met hun pogingen om serieus te worden genomen en komt er daardoor steeds minder geluid vanuit externe bronnen. De bedrijvigheid van kleinschalige activiteiten gaat door, in de vorm van bugrapporten en af en toe wat kleine fixes. Maar er is dan geen sprake van grote codebijdragen of externe deelname aan ontwerpbesprekingen. Mensen voelen wat van hen wordt verwacht en voldoen (of niet) aan deze verwachtingen.

Hoewel geld voorzichtig moet worden gebruikt, betekent dat niet dat het geen invloed kan kopen. Dat kan het namelijk beslist wel. De essentie is dat geld niet rechtstreeks invloed kan kopen. Bij een simpele handelstransactie ruil je geld voor datgene wat je wilt hebben. Als je een extra functie nodig hebt, onderteken je een contract, betaal je ervoor en wordt het gedaan. Bij een open source-project ligt dat niet zo eenvoudig. U kunt een overeenkomst sluiten met een aantal ontwikkelaars, maar ze zouden zichzelf – en u – voor de gek houden als ze zouden garanderen dat het werk waarvoor u hebt betaald door de ontwikkelaarsgemeenschap zou worden aanvaard alleen maar omdat u ervoor hebt betaald. Het werk kan alleen worden aanvaard om de intrinsieke waarde ervan en om hoe het past in de visie die de open source-gemeenschap van de software heeft. Mogelijk hebt u wat inbreng in die visie, maar u bent niet de enige.

Geld kan dus geen invloed kopen, maar het kan wel zaken kopen die tot invloed *leiden*. Het duidelijkste voorbeeld zijn programmeurs. Als er goede programmeurs worden ingehuurd, die lang genoeg blijven om ervaring met de software op te doen en geloofwaardigheid bij de gemeenschap te krijgen, kunnen deze via dezelfde middelen het project beïnvloeden als alle andere leden. Ze hebben een stem, of als ze met velen zijn, hebben ze een stemcoalitie. Als ze in het project gerespecteerd worden, hebben ze meer invloed dan alleen via hun stemmen. Betaalde ontwikkelaars hoeven hun motieven ook niet te verbergen. Iedereen die wil dat er een verandering in de software wordt aangebracht, wil dat immers om een bepaalde reden. De beweegredenen van uw bedrijf zijn niet minder legitiem dan die van ieder ander. Het is alleen zo dat het belang dat aan de doelen van uw bedrijf wordt gehecht, wordt bepaald door de status van de vertegenwoordigers daarvan in het project en niet door de omvang, het budget of het ondernemingsplan van het bedrijf.

5.1 SOORTEN BETROKKENHEID

Er zijn veel verschillende redenen waarom open source-projecten worden gefinancierd. De onderdelen op deze lijst sluiten elkaar niet uit. Vaak is de financiële steun van een project het gevolg van verschillende van deze redenen, of van alle.

De lasten delen

Afzonderlijke organisaties met min of meer dezelfde softwarebehoeften doen het werk vaak dubbel, hetzij doordat ze intern dezelfde (dus overbodige) code obtwikkelen, hetzij door soortgelijke gesloten producten in te kopen van commerciële leveranciers. Wanneer ze beseffen wat er aan de hand is, kunnen de organisaties hun krachten bundelen en een open source-project creëren (of zich daarbij aansluiten) dat past bij hun behoeften. De voordelen zijn duidelijk: de ontwikkelkosten worden gedeeld en iedereen heeft er baat bij. Hoewel dit scenario het meest logisch lijkt voor non-profitorganisaties, kan het ook strategisch zinvol zijn voor concurrenten met een winstoogmerk.

Voorbeelden: http://www.openadapter.org/, http://www.koha.org/

Uitbreiding van diensten

Wanneer een bedrijf diensten verkoopt die afhankelijk zijn van, of aantrekkelijker worden door, bepaalde open source-programma's, is het uiteraard in het belang van dat bedrijf om ervoor te zorgen dat die programma's actief worden onderhouden.

Voorbeeld: CollabNets ondersteuning van http://subversion.tigris.org/ (disclaimer: dat is mijn dagelijkse werk, maar het is ook een fraai voorbeeld van dit model).

Ondersteuning van de verkoop van hardware

De waarde van computers en computeronderdelen houdt direct verband met de hoeveelheid software die ervoor beschikbaar is. Hardwareleveranciers – niet alleen leveranciers van hele machines, maar ook makers van randapparatuur en microchips – hebben ontdekt dat het voor klanten van belang is dat zij beschikken over open source-software van goede kwaliteit die gebruikt kan worden op hun hardware.

Ondermijning van een concurrent

Soms ondersteunen bedrijven een bepaald open source-project als middel om het product van een concurrent te ondermijnen, dat zelf al dan niet open source is.

Het afsnoepen van marktaandeel van een concurrent is doorgaans niet de enige reden om betrokken te raken bij een open source-project, maar het kan een factor zijn.

Voorbeeld: http://www.openoffice.org/ (nee, dit is niet de enige reden dat OpenOffice bestaat, maar de software is, althans deels, een reactie op Microsoft Office).

Marketing

Het kan een goede marktstrategie zijn om uw bedrijf te koppelen aan een populaire open source-toepassing.

Dubbele licentie

Dubbele licentie is de praktijk van het aanbieden van software onder een traditionele octrooilicentie voor klanten die deze willen doorverkopen als onderdeel van een gepatenteerde toepassing van henzelf en tegelijkertijd onder een vrije licentie voor degenen die deze willen gebruiken onder open source-voorwaarden (zie het gedeelte 'Modellen voor dubbele licenties' in Hoofdstuk 9, *Licenties, auteursrechten en octrooien*). Als de open source-ontwikkelaarsgemeenschap actief is, kan de software profiteren van debugging en ontwikkeling op grote schaal, terwijl het bedrijf nog steeds inkomsten ontvangt uit de opbrengst van rechten om enkele fulltime programmeurs te ondersteunen.

Twee bekende voorbeelden zijn MySQL, makers van de gelijknamige databasesoftware, en Sleepycat, die distributies en ondersteuning voor de Berkeley Database biedt. Het is geen toeval dat dit beide databasebedrijven zijn. Databasesoftware wordt vaker in toepassingen geïntegreerd dan dat deze rechtstreeks aan de gebruiker wordt verkocht en is dus erg geschikt voor het model van dubbele licentie.

Donaties

Een op grote schaal gebruikt project ontvangt soms grote bedragen van zowel afzonderlijke personen als organisaties, gewoon via een online donatieknop of, soms, door middel van merchandising van koffiemokken, T-shirts, muismatjes enz. met het merk erop. Enige voorzichtigheid is geboden. Als uw project donaties accepteert, plan dan de besteding van het geld *voordat* het binnenkomt en vermeld de plannen op de website van het project. Discussies over de toewijzing van geld verlopen doorgaans veel soepeler wanneer ze worden gehouden voordat er daadwerkelijk geld is om uit te geven. Bovendien is het beter om in het geval van grote onenigheid daar achter te komen zolang alles nog theoretisch is.

Het bedrijfsmodel van de financier is niet de enige factor in de relatie tot een open source-gemeenschap. Ook de historische relatie tussen beide is van belang: is het bedrijf met het project begonnen, of sluit het zich aan bij een bestaand ontwikkelingsproject? In beide gevallen moet de financier geloofwaardigheid verdienen, maar het zal geen verbazing wekken dat er in het laatste geval iets meer moet worden verdiend. De organisatie moet duidelijke doelen hebben met betrekking tot het project. Probeert het bedrijf zijn leidende positie te behouden, of probeert het eenvoudigweg een stem in de gemeenschap te zijn om de richting van het project te leiden maar niet noodzakelijkerwijs te besturen? Of wil het slechts over een paar committers beschikken die bugs van klanten kunnen repareren en de wijzigingen zonder gedoe in de openbare distributie kunnen krijgen?

Houd deze vragen in gedachten wanneer u de richtlijnen hieronder leest. Deze gelden voor de toepassing van iedere soort betrokkenheid van de organisatie bij een open source-softwareproject, maar elk project is een menselijke omgeving en daarom zijn geen twee projecten precies gelijk. In zekere zin moet u altijd kijken hoe het loopt, maar als u deze principes volgt, is het waarschijnlijker dat dingen gaan lopen zoals u wilt.

5.2 AANSTELLING VOOR DE LANGE TERMIJN

Als u programmeurs bij een open source-project aanstuurt, zorg dan dat u hen daar lang genoeg houdt, zodat ze zowel technische als politieke ervaring opdoen: ten minste twee jaar. Geen enkel project, of dit nu een open source- of closed source-project is, is er natuurlijk bij gebaat als er te vaak van programmeurs wordt gewisseld. De noodzaak om telkens nieuwkomers aan te trekken die alles weer van de grond af moeten leren, werkt afschrikwekkend, ongeacht de omgeving. Maar de nadelen zijn nog groter voor een open source-project, omdat ontwikkelaars die vertrekken niet alleen hun kennis van de code met zich meenemen, maar tevens hun status in de gemeenschap en de informele contacten die ze daar zijn aangegaan.

De geloofwaardigheid die een ontwikkelaar heeft verworven, kan niet worden overgedragen. Om het meest logische voorbeeld te geven: een nieuw binnengekomen ontwikkelaar kan de commit access van een vertrekkende ontwikkelaar niet overnemen (zie het gedeelte 'Met geld kun je geen liefde kopen' verderop in dit hoofdstuk). Dus als de nieuwe ontwikkelaar niet reeds commit access heeft, moet hij patches indienen totdat hij deze krijgt. Maar commit access is slechts de best meetbare manifestatie van verloren invloed. Een lange-termijnontwikkelaar kent tevens alle oude argumenten die bij herhaling zijn besproken op de discussielijsten. Een nieuwe ontwikkelaar, die deze discussies niet kent, kan proberen het probleem opnieuw naar voren te brengen, wat leidt tot een verlies van geloofwaardigheid voor uw organisatie. De anderen kunnen zich dan afvragen: 'Onthouden ze dan niets?' Een nieuwe ontwikkelaar heeft ook geen politiek gevoel voor de persoonlijkheden van het project en kan niet zo snel of gemakkelijk invloed uitoefenen op de ontwikkelingsrichting als iemand die er al lang is.

Train nieuwkomers door middel van een programma van betrokkenheid onder toezicht. De nieuwe ontwikkelaar moet vanaf het allereerste begin rechtstreeks in contact staan met de openbare ontwikkelgemeenschap en beginnen met bug fixes en opschoontaken. Zo kan hij de basis van de code leren en een reputatie verwerven in de gemeenschap, maar geen langdurige ontwerpdiscussies aanzwengelen. Gedurende die tijd moeten er één of meer ervaren ontwikkelaars beschikbaar zijn om vragen te beantwoorden. Zij moeten elk bericht lezen dat de nieuwkomer op

de ontwikkelingslijsten plaatst, ook als deze in threads staan waaraan de ervaren ontwikkelaars normaliter geen aandacht besteden. Hierdoor kan de groep hopelijk zandbanken ontdekken voordat de nieuwkomer vastloopt. Afzonderlijke coaching en aanwijzingen achter de schermen kunnen ook veel helpen, vooral als de nieuwkomer niet gewend is aan grootschalige gelijktijdige beoordeling van zijn code door collega's.

Wanneer CollabNet een nieuwe ontwikkelaar inhuurt voor werk aan Subversion, gaan we samen om tafel zitten en kiezen we een aantal openstaande bugs waar de nieuwe persoon zijn tanden in kan zetten. We bespreken het technische kader van de oplossingen en benoemen dan ten minste één ervaren ontwikkelaar voor de (openbare) beoordeling van de patch die de nieuwe ontwikkelaar (eveneens openbaar) post. Doorgaans kijken we niet eens naar de patch voordat de hoofdontwikkelingslijst hiernaar kijkt, hoewel we dat wel zouden kunnen als daar reden voor was. Het belangrijkste is dat de nieuwe ontwikkelaar het proces van openbare beoordeling doorloopt, waarbij hij de basis van de code leert terwijl hij tegelijkertijd gewend raakt aan het krijgen van kritiek van wildvreemden. Maar we proberen de timing zodanig te coördineren dat onze eigen beoordeling vlak na het plaatsen van de patch komt. Op die manier ziet de lijst onze beoordeling als eerste, wat kan helpen de toon te zetten voor de andere beoordelingen. Het draagt tevens bij aan het idee dat deze nieuwe persoon serieus genomen moet worden: als anderen zien dat wij de tijd nemen om gedetailleerde beoordelingen te geven, met grondige uitleg en zo nodig verwijzingen naar de archieven, zullen ze beseffen dat het hier om een vorm van training gaat en dat dit waarschijnlijk duidt op een lange-termijninvestering. Dit kan er ook voor zorgen dat ze positiever naar die ontwikkelaar kijken, althans in zoverre dat ze wat extra tijd investeren in het beantwoorden van vragen en het beoordelen van patches.

5.3 KOM OVER ALS INDIVIDUEN, NIET ALS EEN BLOK

Uw ontwikkelaars moeten ernaar streven om in de publieke forums van het project te verschijnen als afzonderlijke deelnemers en niet als een monolithische bedrijfsgroep. Dit is niet omdat het fenomeen monolithische bedrijfsgroep negatieve gedachten oproept (nou ja, misschien ook wel, maar daar gaat dit boek niet over). Het is meer omdat afzonderlijke personen de enige entiteit zijn waar open sourceprojecten structureel mee om kunnen gaan. Een afzonderlijke bijdrager kan discussies voeren, patches indienen, geloofwaardigheid verwerven, een stem uitbrengen enz. Een bedrijf kan dat niet.

Doordat u zich op gedecentraliseerde wijze gedraagt, voorkomt u tevens dat centralisatie van de oppositie wordt bevorderd. Laat uw ontwikkelaars het met elkaar oneens zijn op de mailinglijsten. Stimuleer ze om elkaars code zo vaak en zo openbaar te beoordelen als ze dat met die van anderen zouden doen. Zorg ervoor dat ze niet altijd als een blok stemmen, omdat anderen anders het gevoel kunnen krijgen dat er, op basis van algemene principes, sprake zou moeten zijn van een georganiseerde inspanning om hen onder controle te houden.

Er is een verschil tussen echt gedecentraliseerd zijn en er alleen maar naar streven om gedecentraliseerd te lijken. Onder bepaalde omstandigheden kan het vrij nuttig zijn om uw ontwikkelaars gezamenlijk te laten optreden, en ze moeten erop voorbereid zijn zich om zo nodig achter de schermen te bundelen. Bij het doen van een voorstel kan het bijvoorbeeld helpen om verschillende mensen in een vroeg stadium hiermee te laten instemmen, zodat de indruk van een toenemende consensus wordt gewekt. Anderen zullen het idee krijgen dat het voorstel potentieel heeft en dat ze dat potentieel tegenwerken als ze er bezwaar tegen maken. Daardoor zullen mensen alleen bezwaar maken als ze er een goede reden voor hebben. Er is niets mis mee om instemming op deze wijze te organiseren, zolang bezwaren maar serieus worden genomen. Openbare manifestaties van onderhandse overeenkomsten zijn niet minder oprecht alleen omdat ze van tevoren zijn gecoördineerd. Ze zijn onschadelijk zolang ze maar niet worden gebruikt om op vooringenomen wijze tegenargumenten de nek om te draaien. Het doel ervan is louter om het type mensen af te remmen dat alleen maar bezwaar maakt omwille van het bezwaar maken. Zie het gedeelte 'Hoe makkelijker het onderwerp, des langer de discussie' in Hoofdstuk 6, *Communicatie,* voor meer hierover.

5.4 WEES OPEN OVER UW BEWEEGREDENEN

Wees zo open over de doelen van uw organisatie als mogelijk is, zonder bedrijfsgeheimen in gevaar te brengen. Als u wilt dat het project een bepaalde functie krijgt omdat uw klanten hier bijvoorbeeld om schreeuwen, zeg dit dan openlijk op de mailinglijst. Als de klanten anoniem willen blijven, wat soms gebeurt, vraag hen dan ten minste of ze mogen worden gebruikt als anonieme voorbeelden. Hoe meer de openbare ontwikkelgemeenschap weet over *waarom* u wilt wat u wilt, des te makkelijker ze aanvaarden wat u voorstelt.

Dit gaat in tegen het adagium – dat men zich zo makkelijk eigen maakt en maar zelden loslaat – dat kennis macht is en dat hoe meer anderen weten over uw doelen, des te meer macht ze over u hebben. Dat adagium geldt hier echter niet. Door de functie (of de bugfix, of wat dan ook) openbaar te promoten, *hebt* u uw kaarten al op tafel gelegd. De enige vraag is nu of u erin slaagt om de gemeenschap zo te sturen dat zij uw doel deelt. Als u alleen maar zegt dat u het wilt, maar geen concrete voorbeelden kunt geven over het waarom, is uw argument zwak en zullen mensen een verborgen agenda beginnen te vermoeden. Maar al een paar echte scenario's waarin wordt aangetoond waarom de voorgestelde functie van belang is, kunnen van grote invloed zijn op de discussie.

Om de reden hiervoor te begrijpen, moet u het alternatief overwegen. Maar al te vaak zijn discussies over nieuwe functies of nieuwe richtingen lang en vermoeiend. De argumenten die mensen naar voren brengen, zijn vaak beperkt tot 'Ik zelf wil X', of het zeer populaire 'In mijn jarenlange ervaring als softwareontwerper is X van groot belang gebleken voor gebruikers / nutteloze franje waar niemand op zit te wachten'. Zoals kan worden verwacht, worden deze discussies niet korter of gematigder door het gebrek aan echte gebruiksgegevens, maar kunnen ze daarentegen steeds verder afdrijven van het vastleggen van feitelijke gebruikservaringen. Zonder

een tegenkracht wordt het eindresultaat waarschijnlijk bepaald door degene die zich het beste kan uitdrukken, of door de meest volhardende of de meest ervaren persoon.

Als organisatie met de beschikking over veel klantgegevens kunt u zo'n tegenkracht bieden. U kunt een kanaal zijn voor informatie die anders geen middel heeft om de ontwikkelgemeenschap te bereiken. Het feit dat die informatie uw wensen ondersteunt, is helemaal niets om u voor te schamen. De meeste ontwikkelaars hebben zelf geen brede ervaring met hoe de software wordt gebruikt die zij schrijven. Elke ontwikkelaar gebruikt de software op zijn of haar eigen idiosyncratische manier. Wat betreft andere gebruikspatronen vertrouwt hij op intuïtie en giswerk en is zich hier, diep van binnen, van bewust. Door geloofwaardige gegevens te verstrekken over een groot aantal gebruikers geeft u de openbare ontwikkelgemeenschap een soort zuurstof. Zolang u het op de juiste wijze presenteert, zullen zij dit enthousiast verwelkomen en zal de zaak in de richting bewegen die u nastreeft.

De sleutel hiertoe is uiteraard juiste presentatie. Het is niet goed om erop te staan dat uw oplossing wordt geïmplementeerd alleen omdat u te maken hebt met een groot aantal gebruikers en omdat zij een bepaalde functie nodig hebben (of denken te hebben). In plaats daarvan richt u uw eerste posts op het probleem, in plaats van op één bepaalde oplossing. Beschrijf uitvoerig welke ervaringen uw klanten hebben, geef zoveel analyse als u beschikbaar hebt en zoveel redelijke oplossingen als u maar kunt bedenken. Als mensen beginnen te speculeren over de effectiviteit van verschillende oplossingen, kunt u blijven putten uit uw gegevens om datgene wat zij zeggen te ondersteunen of te weerleggen. U kunt de hele tijd al één bepaalde oplossing in gedachten hebben, maar leg hier in het begin niet een bijzondere nadruk op. Dit is geen bedrog, maar eenvoudigweg standaardgedrag van een 'eerlijke makelaar'. Uw ware doel is immers het oplossen van het probleem. Een oplossing is slechts een middel voor dat doel. Als de oplossing die uw voorkeur heeft inderdaad de beste is, zullen andere ontwikkelaars dat uiteindelijk zelf erkennen en gaan zij daar vervolgens uit eigen vrije wil achter staan. Dat is veel beter dan wanneer u hen koeioneert om deze te implementeren. (Er bestaat ook nog de mogelijkheid dat zij een betere oplossing bedenken.)

Dat wil niet zeggen dat u nooit een specifieke oplossing kan aanmoedigen. Maar u moet het geduld hebben om de analyse die u intern reeds hebt gemaakt, overgenomen te zien worden op de openbare ontwikkelingslijsten. Plaats geen bericht zoals: 'Ja, daar hebben we het hier al over gehad, maar dat werkt niet om redenen A, B en C. Als je er goed naar kijkt, is de enige manier om dit op te lossen …'. Het probleem is niet zo zeer dat dit arrogant klinkt, maar dat het de indruk wekt dat u *inmiddels al* de nodige (mensen zullen uitgaan van veel) analytische energie en aandacht achter gesloten deuren aan het probleem hebt gewijd. Dit wekt de indruk dat er inspanningen zijn geleverd en misschien wel besluiten zijn genomen waarvan het publiek niet op de hoogte is. Dat is een beproefd recept voor verontwaardiging.

U weet uiteraard hoeveel inspanning u intern aan het probleem hebt gewijd. Die kennis is, in zekere zin, een nadeel. Het plaatst uw ontwikkelaars in een iets andere mentale omgeving dan alle anderen op de mailinglijsten, waardoor hun vermogen

wordt beperkt om dingen te bekijken vanuit het standpunt van degenen die nog niet zo over het probleem hebben nagedacht. Hoe eerder u iedereen over dingen op dezelfde manier aan het denken kunt zetten als u dat doet, des te kleiner dit afstandseffect zal zijn. Deze logica geldt niet alleen voor afzonderlijke technische situaties, maar ook voor de bredere taak om uw doelen zo duidelijk mogelijk te maken. Het onbekende zorgt altijd voor meer instabiliteit dan het bekende. Als mensen begrijpen waarom u wilt wat u wilt, vinden ze het prettig om met u te praten, ook wanneer ze het er niet mee eens zijn. Als ze er niet achter kunnen komen waarom u bepaalde dingen doet, zullen ze, althans een deel van de tijd, uitgaan van het ergste.

U kunt natuurlijk niet alles publiceren en dat verwachten mensen ook niet. Alle organisaties hebben geheimen. Misschien hebben organisaties met een winstoogmerk er meer, maar ook non-profitorganisaties hebben ze. Als u moet pleiten voor een bepaalde koers, maar niets bekend kunt maken over het waarom, geef dan gewoon de beste argumenten die u met die beperking kunt geven en aanvaard het feit dat u niet zoveel invloed op de discussie hebt als u wilt. Dit is een van de compromissen die u sluit om over een ontwikkelgemeenschap te beschikken die niet op uw loonlijst staat.

5.5 MET GELD KUN JE GEEN LIEFDE KOPEN

Als u een betaalde ontwikkelaar bij een project bent, bepaal dan al vroeg richtlijnen over wat u met dat geld wel en niet kunt kopen. Dit betekent niet dat u tweemaal per dag berichten moet plaatsen op de mailinglijsten waarin u uw nobele en onomkoopbare aard benadrukt. Het betekent alleen dat u moet uitkijken naar mogelijkheden om de spanningen te verminderen die door geld *kunnen* ontstaan. U hoeft er niet vanaf het begin vanuit te gaan dat er spanningen zijn. U moet laten zien dat u zich ervan bewust bent dat deze zouden kunnen ontstaan.

Een mooi voorbeeld hiervan kwam naar voren bij het Subversion-project. Subversion werd in 2000 gestart door CollabNet, dat vanaf het begin de hoofdfinancier van het project is geweest en de salarissen van verschillende ontwikkelaars betaalde (disclaimer: ik ben een van hen). Vlak nadat het project was begonnen, namen we ter ondersteuning nog een ontwikkelaar in dienst, Mike Pilaton. Er was toen al begonnen met de codering. Hoewel Subversion zich nog steeds in een pril stadium bevond, beschikte het reeds een ontwikkelaarsgemeenschap met een reeks basisregels.

De komst van Mike leidde tot een interessante vraag. Subversion had al een beleid over hoe een nieuwe ontwikkelaar commit access krijgt. Eerst zet hij enkele patches op de ontwikkelingsmailinglijst. Nadat er genoeg patches langs zijn gekomen op grond waarvan de andere committers kunnen zien dat de nieuwe bijdrager weet wat hij doet, stelt iemand voor dat hij rechtstreeks mag committen (dat voorstel is privé, zoals beschreven in het gedeelte 'Committers'). Ervan uitgaande dat de committers hiermee instemmen, mailt een van hen de nieuwe ontwikkelaar en biedt hem rechtstreekse commit access tot de repository van het project.

CollabNet had Mike specifiek ingehuurd om aan Subversion te werken. Onder degenen die hem al kenden, bestond geen twijfel over zijn coderingsvaardigheden of zijn bereidheid om aan het project te werken. Bovendien hadden de vrijwillige ontwikkelaars een zeer goede verstandhouding met de medewerkers van CollabNet en zouden ze waarschijnlijk geen bezwaar hebben gemaakt als wij Mike op de dag dat hij werd aangesteld meteen commit access hadden gegeven. Maar we wisten dat we dan een precedent zouden scheppen. Als we Mike middels een fiat commit access hadden gegeven, hadden we gezegd dat CollabNet het recht had om projectrichtlijnen te negeren, eenvoudigweg omdat het bedrijf de hoofdfinancier is. Hoewel de schade hiervan niet noodzakelijkerwijs onmiddellijk duidelijk zou zijn, zou dit geleidelijk resulteren in het feit dat de onbetaalde ontwikkelaars het gevoel zouden krijgen dat hun rechten waren ontnomen: andere mensen moeten hun commit access verdienen en CollabNet koopt het gewoon.

Mike stemde er dus mee in om zijn tewerkstelling bij CollabNet net zo te beginnen als iedere andere vrijwillige ontwikkelaar: zonder commit access. Hij stuurde patches naar de openbare mailinglijst, waar ze door iedereen konden worden – en werden – beoordeeld. We zeiden tevens op de lijst dat we dit expres op deze manier deden, zodat het absoluut duidelijk was. Na een paar zeer actieve weken van Mike droeg iemand (ik weet niet meer of dit wel of niet een CollabNet-ontwikkelaar was) hem voor commit access voor en werd hij geaccepteerd, zoals wij al hadden verwacht.

Dit soort consistentie geeft je een geloofwaardigheid die niet voor geld te koop is. En geloofwaardigheid is een waardevolle munteenheid in technische discussies: het zorgt voor immuniteit tegen vragen achteraf over iemands beweegredenen. In de hitte van de discussie zoeken mensen soms naar niet-technische manieren om de strijd te winnen. De hoofdfinancier van het project is, vanwege zijn grote betrokkenheid en duidelijke bezorgdheid over de richtingen die het project op gaat, een groter doelwit dan anderen. Door alle projectrichtlijnen van meet af aan nauwgezet na te leven, maakt de financier zichzelf niet belangrijker dan de rest.

(Zie ook de blog van Danese Cooper op http://blogs.sun.com/roller/page/DaneseCooper/20040916 voor een soortgelijk verhaal over commit access. Cooper was toen de 'open source-diva' – volgens mij was dat haar officiële titel – van Sun Microsystem en in de blog beschrijft ze hoe de Tomcat-ontwikkelaarsgemeenschap Sun zover kreeg dat het zijn eigen ontwikkelaars hield aan dezelfde normen voor commit access als de ontwikkelaars die niet van Sun waren.)

De noodzaak dat de financiers zich aan dezelfde spelregels houden als alle anderen betekent dat het bestuursmodel van 'Vriendelijke dictators' (zie het gedeelte 'Vriendelijke dictators' in Hoofdstuk 4, *Sociale en politieke infrastructuur*) iets moeilijker te bewerkstelligen is als er sprake is van financiering, vooral als de dictator voor de hoofdfinancier werkt. Omdat een dictatuur weinig regels kent, is het voor de financier moeilijk te bewijzen dat hij zich houdt aan de normen van de gemeenschap, zelfs als dat wel het geval is. Het is beslist niet onmogelijk, maar je hebt er wel een projectleider voor nodig die dingen kan bekijken vanuit het standpunt van de externe ontwikkelaars en dat van de financier, en die dienovereenkomstig handelt. Ook dan is het waarschijnlijk een goed idee om een voorstel voor een niet-dictatoriaal bestuur achter de hand te hebben, dat naar voren kan worden gebracht op het moment dat er aanwijzingen zijn van wijdverbreide ontevredenheid in de gemeenschap.

5.6 AANBESTEDING

Met aanbesteed werk moet bij open source-softwareprojecten zorgvuldig worden omgesprongen. Idealiter wilt u dat het werk van een aannemer wordt aanvaard door de gemeenschap en wordt opgenomen in de openbare distributie. In theorie maakt het niet uit wie de aannemer is, zolang hij zijn werk maar goed doet en voldoet aan de projectrichtlijnen. Theorie en praktijk stemmen soms ook met elkaar overeen: een wildvreemde die met een goede patch komt *is* doorgaans in staat om deze in de software te krijgen. Het probleem is dat het erg moeilijk is om als echte wildvreemde een goede patch te produceren voor een niet-triviale verbetering of nieuwe functie. Deze moet eerst besproken worden met de rest van het project. De duur van die discussie kan niet exact worden voorspeld. Als de aannemer per uur wordt betaald, kan dat ertoe leiden dat u meer betaalt dan u verwachtte. Als hij een forfaitair bedrag krijgt, kan hij uiteindelijk meer werk doen dan hij zich kan veroorloven.

Er zijn twee manieren om dit te omzeilen. De manier die de voorkeur verdient, is om op basis van eerdere ervaringen een gefundeerde schatting te doen over de lengte van het discussieproces, om een foutmarge in te bouwen en om het contract daarop te baseren. Het helpt ook om het probleem onder te verdelen in zo veel mogelijk kleine, afzonderlijke porties om de voorspelbaarheid van elk portie te vergroten. De andere manier is om alleen aan te besteden voor levering van een patch en om de aanvaarding van de patch in het openbare project als een afzonderlijke kwestie te behandelen. Dan wordt het veel gemakkelijker om het contract op te stellen, maar blijft u wel zitten met de last om een particuliere patch te behouden zolang u van de software afhankelijk bent, of ten minste zolang u nodig hebt om die patch of equivalente functionaliteit in de hoofdlijn te krijgen. Uiteraard is het zo, zelfs met de voorkeursmanier, dat niet in het contract zelf kan worden bepaald dat de patch daadwerkelijk in de code wordt opgenomen, omdat dat zou betekenen dat je iets verkoopt wat niet te koop is. (Wat als de rest van het project onverwacht besluit om de functie niet te ondersteunen?) Het contract kan echter een *bonafide* inspanning vereisen om de verandering aanvaard te krijgen door de gemeenschap en dat deze in de repository wordt geplaatst als de gemeenschap daarmee instemt. Als het project bijvoorbeeld geschreven normen heeft voor codewijzigingen, kan het contract naar die normen verwijzen en specificeren dat het werk daaraan moet voldoen. In de praktijk werkt dit doorgaans zoals iedereen hoopt.

De beste tactiek voor een succesvolle aanbesteding is om een van de ontwikkelaars van het project – bij voorkeur een committer – in te huren als aannemer. Dit kan lijken op een vorm van invloed kopen; en dat is het eigenlijk ook. Maar het is niet zo corrupt als het lijkt. De invloed van een ontwikkelaar op het project hangt voornamelijk af van de kwaliteit van zijn code en zijn interactie met andere ontwikkelaars.

Het feit dat hij een contract heeft om bepaalde dingen te doen, verhoogt zijn status op geen enkele wijze en verlaagt deze evenmin, hoewel mensen wel nauwlettender naar hem kunnen gaan kijken. De meeste ontwikkelaars zetten hun langetermijnpositie in het project niet op het spel door een ongeschikte of breed afgekeurde nieuwe functie te ondersteunen. Een deel van wat u krijgt, of zou moeten krijgen, door het inhuren van zo'n aannemer is in feite advies over welke soorten veranderingen waarschijnlijk worden geaccepteerd door de gemeenschap. Er komt tevens een kleine verschuiving in de prioriteiten van het project. Omdat prioriteiten stellen slechts een kwestie is van wie tijd heeft om aan wat te werken, zorgt u ervoor dat u, wanneer u voor iemands tijd betaalt, hun werk een beetje hoger plaatst in de prioriteitenrangorde. Dit is een algemeen bekend feit bij ervaren open source-ontwikkelaars en ten minste een aantal van hen zal aandacht besteden aan het werk van de aannemer, eenvoudigweg omdat het erop lijkt dat het *wordt gedaan* en ze dus willen helpen om het goed te doen. Misschien schrijven ze geen code, maar ze zullen wel het ontwerp bespreken en de code beoordelen, wat beide erg nuttig kan zijn. Om al deze redenen kan de aannemer het beste iemand zijn die reeds bij het project betrokken is.

Dit roept meteen twee vragen op: Moeten contracten altijd onderhands zijn? En als ze dit niet zijn, moet u zich dan zorgen maken over het creëren van spanningen in de gemeenschap door het feit dat u met sommige ontwikkelaars een contract hebt gesloten en met andere niet?

Het is het beste om, indien mogelijk, open kaart te spelen over contracten. Anders kan het gedrag van de aannemer op anderen in de gemeenschap vreemd overkomen. Misschien geeft hij opeens onverklaarbaar veel prioriteit aan functies waarvoor hij in het verleden nooit interesse heeft getoond. Als mensen hem vragen waarom hij deze nu wel wil, hoe kan hij dan overtuigend antwoorden als hij niet mag spreken over het feit dat hij is ingehuurd om deze te schrijven?

Tegelijkertijd moeten u noch de aannemer zich gedragen alsof anderen uw regeling als iets belangrijks moeten zien. Te vaak heb ik aannemers over een ontwikkelingslijst heen zien walsen met een houding alsof hun berichten serieuzer moeten worden genomen, eenvoudigweg omdat zij betaald worden. Een dergelijke houding maakt de rest van het project duidelijk dat de aannemer het feit van het contract – in tegenstelling tot de code die *voortvloeit* uit het contract – het belangrijkste vindt. Maar vanuit het standpunt van de andere ontwikkelaars is alleen de code van belang. De aandacht moet te allen tijde gericht blijven op technische kwesties en niet op informatie over wie wie betaalt. Een van de ontwikkelaars in de Subversion-gemeenschap gaat bijvoorbeeld op een bijzonder fraaie manier om met aanbesteding. Tijdens de bespreking van zijn codewijzigingen in IRC merkt hij terloops op (vaak met een privéopmerking, een IRC *privmsg*, aan een van de andere committers) dat hij wordt betaald voor zijn werk aan deze specifieke bug of functie. Maar hij geeft ook consistent de indruk dat hij toch al graag aan die wijziging wilde werken en dat hij blij is dat het geld het hem mogelijk maakt om dat te doen. Hij kan al dan niet de identiteit van zijn klant onthullen, maar hij weidt in elk geval niet uit over het contract. Zijn opmerkingen daarover zijn slechts een extra in een verder technische discussie over hoe iets gedaan moet worden.

Dat voorbeeld laat nog een reden zien waarom het goed is om open kaart te spelen over contracten. Er kunnen meer organisaties zijn die contracten voor een bepaald open source-project sponsoren en als iedereen weet wat de anderen proberen te doen, kunnen ze hun krachten wellicht bundelen. In het bovenstaande geval is de grootste financier van het project (CollabNet) op geen enkele wijze betrokken bij deze stukwerkcontracten, maar in de wetenschap dat iemand anders bepaalde fixes van bugs sponsort, kan CollabNet zijn medewerkers inzetten voor andere bugs, wat leidt tot een grotere efficiëntie voor het hele project.

Zullen andere ontwikkelaars ontstemd zijn dat sommigen worden betaald om aan het project te werken? Doorgaans niet. Vooral wanneer degenen die betaald worden toch al gevestigde en gerespecteerde leden van de gemeenschap zijn. Niemand verwacht dat aanbestedingswerk gelijk wordt verdeeld onder alle committers. Mensen begrijpen het belang van langetermijnrelaties: de onzekerheden bij aanbesteding zijn zodanig dat je, wanneer je eenmaal iemand vindt met wie je betrouwbaar kunt werken, liever niet overstapt naar een andere persoon alleen maar omwille van de gelijkheid. Bekijk het op deze manier: de eerste keer dat u iemand inhuurt, komen er geen klachten omdat u overduidelijk *iemand* moest kiezen; het is niet uw schuld dat u niet iedereen in kunt huren. Later, wanneer u dezelfde persoon nogmaals inhuurt, is dat alleen maar gezond verstand: u kent hem al, de laatste keer ging het goed, dus waarom zou u onnodige risico's nemen? Daarom is het volkomen normaal om één of twee ingehuurde mensen in de gemeenschap te hebben, in plaats van het werk evenredig te verdelen.

Beoordeling en aanvaarding van wijzigingen

De gemeenschap is nog steeds belangrijk voor het welslagen van aanbestedingswerk. Hun betrokkenheid bij het ontwerp- en beoordelingsproces voor grote veranderingen kan geen overweging achteraf zijn. Deze moet worden gezien als deel van het werk en moet volledig worden aanvaard door de aannemer. Zie de beoordeling van de gemeenschap niet als een hindernis die overwonnen moet worden. Zie het als een gratis ontwerpraad en afdeling kwaliteitsgarantie. Het is een voordeel om agressief achtervolgd te worden en niet slechts te worden verdragen.

Casestudy: het protocol voor CVS-wachtwoordauthenticatie

In 1995 was ik één helft van een partnerschap dat ondersteuning en verbeteringen leverde aan CVS (het Concurrent Versions System; zie http://www.cvshome.org/). Mijn partner Jim en ik waren, informeel, op dat moment degenen die het onderhoud van CVS deden. Maar we dachten nooit goed na over onze relatie met de bestaande, veelal vrijwillige CVS-ontwikkelaarsgemeenschap. We gingen er gewoon vanuit dat zij patches zouden sturen en dat wij ze zouden toepassen; en zo werkte het ook eigenlijk wel.

Toentertijd kon CVS in een netwerk alleen worden gedaan via een inlogprogramma op afstand, zoals `rsh`. Hetzelfde wachtwoord gebruiken voor CVS-toegang en om in te loggen vormde een duidelijk beveiligingsrisico. Dit schrikte veel organisaties af om deze methode toe te passen. Een grote investeringsbank huurde ons in om een nieuw authenticatiemechanisme toe te voegen, zodat ze op een veilige manier CVS in een netwerk konden gebruiken vanuit hun kantoren op afstand.

Jim en ik namen de opdracht aan en gingen aan het werk om het nieuwe authenticatiesysteem te ontwerpen. Waar we mee kwamen, was vrij eenvoudig. De Verenigde Staten hadden toentertijd exportcontrole op cryptografische code, dus de klant begreep dat we geen zware authenticatie konden implementeren. Omdat we echter niet ervaren waren in het ontwerpen van dergelijke protocollen begingen we toch een paar blunders die een deskundige had weten te voorkomen. Deze fouten hadden gemakkelijk kunnen worden opgemerkt als we de tijd hadden genomen om een voorstel op te schrijven en dit ter beoordeling voor te leggen aan de andere ontwikkelaars. Maar dat hebben we nooit gedaan, omdat we er simpelweg niet aan gedacht hadden om de ontwikkelingslijst als een hulpmiddel te gebruiken. We wisten dat men waarschijnlijk sowieso zou accepteren wat wij committen en, omdat we niet wisten wat we niet wisten, namen we niet de moeite om het werk op een zichtbare manier te doen, bijvoorbeeld door vaak patches te posten, kleine, gemakkelijk te behappen commits aan een bepaalde branch te doen enz. Het authenticatieprotocol dat daaruit voortvloeide, was niet erg goed en was uiteraard, toen het eenmaal gevestigd was, moeilijk te verbeteren vanwege compatibiliteitskwesties.

De kern van het probleem was geen gebrek aan ervaring. We hadden makkelijk te weten kunnen komen wat we hadden moeten weten. Het probleem was onze houding ten opzichte van de vrijwillige ontwikkelaarsgemeenschap. Wij beschouwden aanvaarding van veranderingen als een hindernis die overwonnen moest worden en niet als een proces waarmee de kwaliteit van de veranderingen verbeterd kon worden. Omdat we ervan uitgingen dat vrijwel alles wat we deden, aanvaard zou worden (zoals gebeurde), deden we niet echt ons best om anderen erbij te betrekken.

Als u een aannemer kiest, wilt u voor het werk natuurlijk iemand met de juiste technische vaardigheden en ervaring. Maar het is ook van belang om iemand te kiezen met een staat van dienst op het gebied van constructieve interactie met de andere ontwikkelaars in de gemeenschap. Op die manier krijgt u meer dan slechts één persoon. U krijgt een agent die kan putten uit een netwerk van ervaring, zodat u er zeker van bent dat het werk op een solide en onderhoudsvriendelijke manier wordt gedaan.

5.7 FINANCIERING VAN NIET-PROGRAMMERINGS-ACTIVITEITEN

Programmering is slechts een deel van het werk dat wordt gedaan bij een open source-project. Vanuit het standpunt van de vrijwilligers van het project is dit het meest zichtbare en aantrekkelijke deel. Dat betekent jammer genoeg dat andere activiteiten, zoals documentatie, formeel testen enz., soms worden verwaarloosd, althans in vergelijking met de hoeveelheid aandacht die daar bij gepatenteerde software vaak aan wordt besteed. Bedrijven kunnen dit soms compenseren door een deel van hun interne softwareontwikkelingsinfrastructuur in te zetten voor open source-projecten.

De sleutel om dit succesvol te doen is om een vertaalslag te maken tussen de interne processen in het bedrijf en die van de openbare ontwikkelaarsgemeenschap. Deze vertaalslag kost moeite. Vaak sluiten beide processen niet goed op elkaar aan en kunnen de verschillen alleen overbrugd worden door menselijk ingrijpen. Het bedrijf kan bijvoorbeeld gebruik maken van een andere bug tracker dan het openbare project. Ook al gebruiken ze dezelfde opsporingssoftware, dan nog zullen de gegevens die daarin zijn opgeslagen zeer verschillend zijn, omdat de behoeften voor bugopsporing van een bedrijf sterk verschillen van die van een open source-softwaregemeenschap. Een stuk informatie dat in één bug tracker begint, moet mogelijk worden weergegeven in de andere, waarbij vertrouwelijke delen worden verwijderd of juist worden toegevoegd.

De onderstaande onderdelen gaan over het bouwen van dergelijke bruggen en het onderhoud ervan. Het eindresultaat moet zijn dat het open source-project soepeler verloopt, dat de gemeenschap ziet welke investering in middelen het bedrijf heeft gepleegd en toch niet het idee heeft dat het bedrijf op ongepaste wijze de richting manipuleert ten gunste van zijn eigen doelstellingen.

Kwaliteitsgarantie (oftewel professioneel testen)

Bij de ontwikkeling van gepatenteerde software is het gebruikelijk dat teams van mensen zich alleen richten op kwaliteitsgarantie: opsporen van bugs, testen van prestatie en schaalbaarheid, controle van interface en documentatie enz. Als regel worden deze activiteiten niet zo sterk nagestreefd door de vrijwillige gemeenschap van een open source-softwareproject. Dit komt deels omdat het moeilijk is om vrijwilligers te vinden voor onaantrekkelijk werk zoals testen, deels omdat mensen de neiging hebben ervan uit te gaan dat een grote gebruikersgemeenschap het project een goede testdekking geeft en, in het geval van het testen van prestaties en schaalbaarheid, omdat vrijwilligers vaak toch geen toegang hebben tot de benodigde hardware.

De veronderstelling dat het hebben van veel gebruikers gelijk staat aan het hebben van veel testers is niet geheel ongegrond. Het heeft zeker weinig zin om testers aan te stellen voor basisfunctionaliteiten in gemeenschappelijke omgevingen Bugs worden daar bij de normale gang van zaken snel gevonden door gebruikers. Maar omdat gebruikers alleen maar proberen om werk gedaan te krijgen, gaan ze niet bewust op verkenning in niet in kaart gebrachte randgevallen in de functionaliteit van het programma en vinden ze bepaalde categorieën bugs waarschijnlijk niet. En als ze een bug ontdekken die makkelijk te omzeilen is, implementeren ze bovendien vaak zonder het te zeggen de programmaomleiding zonder de moeite te nemen om de fout te melden. Wat veel verraderlijker is, is dat de gebruikspatronen van uw klanten (de mensen die de motor zijn van *uw* belang in de software) statistisch grote verschillen kunnen vertonen met die van de ‘gemiddelde gebruiker op straat’.

Een professioneel testteam kan dit soort bugs aan het licht brengen en kan dit net zo makkelijk doen bij open source-software als bij gepatenteerde software. De uitdaging is om de resultaten van het testteam op een nuttige wijze terug te koppelen naar het publiek. Interne testafdelingen hebben doorgaans hun eigen manier om testresultaten te rapporteren, waarbij bedrijfsspecifiek jargon wordt gebruikt of gespecialiseerde kennis over bepaalde klanten en hun datasets vereist is. Deze rapporten zijn niet geschikt voor de openbare bug tracker, zowel vanwege hun vorm als vanwege vertrouwelijkheidskwesties. Zelfs als de interne foutopsporingssoftware

van uw bedrijf dezelfde is als degene die wordt gebruikt door het openbare project, moet het management misschien bedrijfsspecifieke opmerkingen en metadatawijzigingen in de problemen maken (bijvoorbeeld om de interne prioriteit van een kwestie hoger te maken of om de oplossing daarvan te plannen voor een bepaalde klant). Doorgaans zijn dergelijke notities vertrouwelijk. Soms krijgt zelfs de klant ze niet te zien. Maar ook als ze niet vertrouwelijk zijn, zijn ze niet van belang voor het openbare project en moet het publiek er daarom niet door worden afgeleid.

Maar het basale bugrapport zelf is *wel* van belang voor het publiek. Een foutenrapportage van uw testafdeling is op bepaalde manieren namelijk waardevoller dan eentje die wordt ontvangen van gebruikers in extenso, omdat de testafdeling op zoek gaat naar dingen die andere gebruikers niet zoeken. Aangezien u die bepaalde foutrapportage waarschijnlijk niet uit een andere bron zult krijgen, wilt u deze beslist behouden en beschikbaar stellen voor het openbare project.

Hiertoe kan de afdeling kwaliteitsgarantie problemen rechtstreeks archiveren in de openbare issue tracker, als ze dat gemakkelijk vinden, of kan een tussenpersoon (meestal een van de ontwikkelaars) de interne rapporten van de testafdeling 'vertalen' in nieuwe problemen in de openbare tracker. Vertaling betekent eenvoudigweg het beschrijven van de bug op een manier die geen verwijzing bevat naar klantspecifieke informatie (het reproductierecept kan gebruik maken van klantgegevens, uiteraard mits de klant hiervoor toestemming geeft).

Het verdient de voorkeur om de afdeling kwaliteitsgarantie problemen rechtstreeks te laten archiveren in de openbare tracker. Dat zorg er op een meer directe manier voor dat het publiek waardering krijgt voor de betrokkenheid van uw bedrijf bij het project: nuttige bugrapporten vergroten de geloofwaardigheid van uw organisatie, net zoals iedere technische bijdrage dat zou doen. Het geeft ontwikkelaars tevens een rechtstreekse communicatielijn met het testteam. Als het interne team kwaliteitsgarantie bijvoorbeeld toezicht houdt op de openbare foutopsporing, kan een ontwikkelaar een fix committen voor een bug op het gebied van schaalbaarheid (waarvoor de ontwikkelaar misschien niet de middelen heeft om dit zelf te testen) en daarna een notitie over het issue toevoegen waarin het team kwaliteitsgarantie wordt gevraagd of de reparatie het gewenste effect had. Verwacht enige weerstand van sommige ontwikkelaars. Programmeurs hebben de neiging om kwaliteitsgarantie op z'n best te zien als een noodzakelijk kwaad. Het team kwaliteitsgarantie kan dit gemakkelijk overwinnen door grote bugs te vinden en begrijpelijke rapporten in te dienen. Aan de andere kant is het zo dat, als hun rapporten niet ten minste net zo goed zijn als die van de gebruikelijke gebruikersgemeenschap, het geen zin heeft om deze rechtstreeks in contact te laten staan met het ontwikkelingsteam.

Hoe dan ook, wanneer er een openbaar issue is, moet de oorspronkelijke interne issue eenvoudigweg naar de openbare issue verwijzen voor technische inhoud. Management en betaalde ontwikkelaars kunnen doorgaan met het annoteren van de interne issue met zo nodig bedrijfsspecifieke opmerkingen, maar kunnen de openbare issue gebruiken voor informatie die voor iedereen beschikbaar zou moeten zijn.

Wanneer u dit proces ingaat, moet u extra overhead verwachten. Het onderhouden van twee issues voor één bug is, uiteraard, meer werk dan het onderhouden van één issue. Het voordeel is dat veel meer programmeurs het rapport zien en kunnen bijdragen aan een oplossing.

Juridisch advies en bescherming

Bedrijven, al dan niet met winstoogmerk, zijn vrijwel de enige entiteiten die aandacht besteden aan complexe juridische kwesties bij open source-software. Afzonderlijke ontwikkelaars begrijpen vaak de nuances van verschillende open source-licenties, maar hebben doorgaans de tijd noch de middelen om wetgeving op het gebied van auteursrechten, handelsmerken en octrooien gedetailleerd te volgen. Als uw bedrijf een afdeling juridische zaken heeft, kan deze een project helpen door de auteursrechtstatus van de code door te lichten en de ontwikkelaars te helpen inzicht te krijgen in mogelijke problemen op het gebied van octrooien en handelsmerken. De exacte vorm die deze hulp kan krijgen, wordt besproken in Hoofdstuk 9, *Licenties, auteursrechten en octrooien*. Het belangrijkste is om ervoor te zorgen dat de communicatie tussen de afdeling juridische zaken en de ontwikkelaarsgemeenschap, als deze er al is, plaatsvindt in een sfeer van wederzijds inzicht in de zeer verschillende werelden waar de partijen vandaan komen. Soms praten deze twee groepen langs elkaar heen, waarbij elke groep uitgaat van gebiedsspecifieke kennis die de andere niet heeft. Een goede strategie is om een tussenpersoon te hebben (meestal een ontwikkelaar of anders een jurist met technische expertise) die zo lang het nodig is de boodschappen vertaalt.

Documentatie en bruikbaarheid

Documentatie en bruikbaarheid zijn beide beroemde zwakke plekken bij open source-projecten, hoewel ik denk, althans in het geval van documentatie, dat het verschil tussen vrije en gepatenteerde software vaak wordt overdreven. Het is echter empirisch vastgesteld dat veel open source-software eersteklasonderzoek naar documentatie en bruikbaarheid ontbeert.

Als uw organisatie deze gaten in een project wil helpen dichten, kan deze waarschijnlijk het beste mensen inhuren die *niet* de gebruikelijke ontwikkelaars van het project zijn, maar die in staat zijn om productief met de ontwikkelaars samen te werken. Het niet inhuren van de gebruikelijke ontwikkelaars is om twee redenen gunstig. Allereerst hoeft u geen ontwikkelingstijd te onttrekken aan het project en ten tweede zijn degenen die het dichtst bij de software staan toch vaak de verkeerde mensen om documentatie te schrijven of onderzoek te doen naar bruikbaarheid, omdat ze de software maar moeilijk kunnen bekijken door de ogen van een buitenstaander.

Degene die aan deze problemen werkt, moet echter wel met de ontwikkelaars communiceren. Zoek mensen die technisch genoeg zijn om met het programmeringsteam te praten, maar die niet zo deskundig zijn op het gebied van de software dat ze zich niet meer in kunnen leven in de normale gebruiker.

Een gebruiker op gemiddeld niveau is waarschijnlijk de juiste persoon om goede documentatie te schrijven. Na publicatie van de eerste uitgave van dit boek kreeg ik de volgende e-mail van een open source-ontwikkelaar genaamd Dirk Reiners:

Een opmerking bij Geld: Documentatie en bruikbaarheid: toen we wat geld hadden en besloten dat een beginnershandleiding het belangrijkste was wat we nodig hadden, huurden we een gebruiker op gemiddeld niveau in om deze te schrijven. Hij had recent genoeg de inleiding bij het systeem doorlopen om zich de problemen te herinneren, maar hij had deze al overwonnen en wist dus hoe hij ze moest beschrijven. Daardoor kon hij iets schrijven waar door de kernontwikkelaars slechts wat kleine aanpassingen in hoeden worden aangebracht met betrekking tot dingen die hij niet bij het juiste eind had, maar waar nog steeds alle 'voor de hand liggende' dingen instonden die ontwikkelaars zouden hebben overgeslagen.

In zijn geval ging het nog beter, omdat hij de opdracht had gekregen om een aantal andere mensen (studenten) bekend te maken met het systeem. Daardoor kon hij de ervaring van veel mensen combineren, wat gewoon een gelukkige samenloop van omstandigheden was en in de meeste gevallen waarschijnlijk niet voorkomt.

Hosting/bandbreedte bieden
Voor een project dat geen gebruik maakt van een van de gratis standaard-hostingsites (zie het gedeelte 'Canned hosting' in Hoofdstuk 3, *Technische infrastructuur*), kan het bieden van een server- en netwerkverbinding – en zeer belangrijk: hulp van systeembeheerders – veel hulp bieden. Ook als dit alles is wat uw organisatie voor het project doet, kan het een vrij doeltreffende manier zijn om goed pr-karma te verkrijgen, hoewel het geen invloed op de richting van het project met zich meebrengt.

U kunt waarschijnlijk een banner of een blijk van waardering op de homepage van het project verwachten, waarin uw bedrijf wordt bedankt voor het leveren van hosting. Als u de hosting zodanig opzet dat het webadres van het project onder uw domeinnaam staat, krijg u wat extra associatie via de URL. Dit zorgt ervoor dat de meeste gebruikers denken dat de software *iets* met uw bedrijf te maken heeft, ook al draagt u op geen enkele wijze bij aan de ontwikkeling. Het probleem is dat de ontwikkelaars zich eveneens bewust zijn van deze associatie en zich er niet prettig bij voelen om het project in uw domein te hebben, tenzij u meer bijdraagt dan alleen bandbreedte. Er zijn tegenwoordig immers veel plaatsen om te hosten. De gemeenschap kan uiteindelijk het idee hebben dat de geïmpliceerde misplaatsing van credit het gemak van hosting niet waard is en het project elders onderbrengen. Dus als u hosting wilt bieden, kunt u dat doen, maar plan dan om binnenkort nauwer betrokken te raken of wees op uw hoede ten aanzien van de hoeveelheid betrokkenheid u opeist.

5.8 MARKETING

Dat marketing effectief is, willen de meeste open source-ontwikkelaars waarschijnlijk niet graag toegeven. Een goede marketingcampagne *kan* ruchtbaarheid geven aan een open source-product, zelfs zodanig dat koppige programmeurs vaag positieve gedachten over de software krijgen om redenen waar ze hun vinger niet echt op kunnen leggen. Het is niet mijn taak om de dynamiek van de bewapeningswedloop van marketing in het algemeen te ontleden. Elk bedrijf dat betrokken is bij open source-software gaat uiteindelijk overwegen hoe het zichzelf, de software of zijn relatie tot de software gaat marketen. Het onderstaande advies gaat over het vermijden van algemene valkuilen bij een dergelijke inspanning. Zie ook het gedeelte 'Publiciteit' in Hoofdstuk 6, *Communicatie*.

Onthoud dat u in de gaten gehouden wordt
Om de vrijwillige ontwikkelaarsgemeenschap aan uw zijde te houden, is het van *groot* belang om niets te zeggen wat niet aantoonbaar waar is. Controleer alle beweringen zorgvuldig voordat u ze doet en geef het publiek de middelen om uw beweringen zelf te controleren. Onafhankelijke feitencontrole vormt een belangrijk deel van open source en dit geldt voor meer dan alleen de code.

Natuurlijk zal niemand bedrijven adviseren om oncontroleerbare beweringen te doen. Maar bij open source-activiteiten is er een ongebruikelijk groot aantal mensen met de expertise om beweringen te controleren; mensen die waarschijnlijk ook zeer uitgebreide internettoegang hebben en de juiste sociale contacten om hun bevindingen op schadelijke wijze te publiceren, als zij daarvoor kiezen. Wanneer Global Megacorp Chemical Industries een rivier vervuilt, is dat controleerbaar, maar alleen door getrainde wetenschappers die vervolgens kunnen worden weersproken door de wetenschappers van Global Megacorp, zodat het publiek zich afvraagt welk verhaal waar is. Aan de andere kant is uw gedrag in de open source-wereld niet alleen zichtbaar en vastgelegd, het is ook voor veel mensen gemakkelijk om dit onafhankelijk te controleren, hun eigen conclusies te trekken en deze conclusies te verspreiden via mond-tot-mondcommunicatie. Deze communicatienetwerken bestaan al. Ze zijn de essentie van hoe open source werkt en kunnen worden gebruikt om allerlei soorten informatie te verspreiden. Weerlegging is vaak moeilijk, zo niet onmogelijk, vooral als datgene wat mensen zeggen waar is.

Het is bijvoorbeeld in orde om te zeggen dat uw organisatie 'project X heeft gefinancierd' als u dat daadwerkelijk hebt gedaan. Maar noem uzelf niet de 'makers van X' als de meeste code is geschreven door buitenstaanders. Zeg omgekeerd niet dat u een sterk betrokken vrijwillige ontwikkelaarsgemeenschap hebt als iedereen uw repository kan bekijken en kan zien dat er weinig tot geen codewijzigingen afkomstig zijn van buiten uw organisatie.

Niet zo lang geleden zag ik een aankondiging van een zeer bekend computerbedrijf dat zei dat ze een belangrijk softwarepakket publiceerden onder een open sourcelicentie. Toen de aanvankelijke aankondiging uitkwam, keek ik naar hun inmiddels publieke versiebeheer-repository en zag dat deze maar drie revisies bevatte. Met andere woorden, ze hadden eerst de broncode geïmporteerd, maar sindsdien weinig gedaan. Dat is op zich niet verontrustend. Ze hadden dit immers alleen maar aangekondigd. Er was geen reden om onmiddellijk veel ontwikkelingsactiviteit te verwachten.

Iets later deden ze een andere mededeling. Dit is wat werd gezegd, waarbij de naam en het publicatienummer zijn vervangen door pseudoniemen:

Met genoegen kondigen wij aan dat, na grondig te zijn getest door de Singer-gemeenschap, Singer 5 voor Linux en Windows nu gereed is voor productiegebruik.

Uit nieuwsgierigheid over wat de gemeenschap had ontdekt tijdens het 'grondig testen', ging ik terug naar de repository om te kijken naar de recente change history. Het project zat nog steeds in revisie 3. Blijkbaar hadden ze geen *enkele* bug gevonden die het waard was om voor release te worden hersteld! Omdat ik dacht dat de resultaten van de gemeenschaptest elders moesten zijn opgeslagen, keek ik vervolgens naar de bug tracker. Er stonden precies zes issues open, waarvan er vier al enkele maanden openstonden.

Dit gaat natuurlijk ten koste van de geloofwaardigheid. Wanneer testers gedurende een bepaalde tijd zwoegen op een groot en complex stuk software vinden ze bugs. Zelfs als de fixes van die fouten niet worden opgenomen in de komende release, zou je toch nog enige versiebeheeractiviteiten verwachten als gevolg van het testproces, of ten minste een aantal nieuwe issues. Maar naar het schijnt, was er niets gebeurd tussen de aankondiging van de open source-licentie en de eerste open source-release.

Het punt is niet dat het bedrijf loog over de test door de gemeenschap. Ik heb geen idee of dit zo was. Maar ze waren zich niet bewust van hoeveel het *erop leek* dat ze logen. Aangezien de versiebeheer-repository noch de issue tracker enige indicatie gaf dat de aangevoerde grondige test had plaatsgevonden, had het bedrijf de aanspraak allereerst niet moeten maken, of had het een duidelijke link moeten verstrekken naar een tastbaar resultaat van die test ('We hebben 278 fouten gevonden; klik hier voor informatie'). Dit laatste had het iedereen mogelijk gemaakt om heel snel een idee te krijgen van de mate van gemeenschapactiviteit. In dit geval kostte het me een paar minuten om vast te stellen dat, wat deze gemeenschaptest ook inhield, deze in elk geval geen sporen had achtergelaten op een gebruikelijke plaatsen. Dat is niet veel moeite en ik weet zeker dat ik niet de enige ben die deze moeite heeft genomen.

Transparantie en controleerbaarheid zijn uiteraard eveneens een belangrijk onderdeel van geloofwaardigheid. Raadpleeg voor meer informatie het gedeelte 'Erkenning' in Hoofdstuk 8*, Het managen van vrijwilligers*.

Kraak concurrerende open source-producten niet af

Onthoud u van negatief commentaar over concurrerende open source-software. Het is geen probleem om met negatieve *feiten* te komen, oftewel gemakkelijk te bevestigen beweringen zoals je vaak ziet in goede vergelijkingstabellen. Maar negatieve typeringen van een minder strikte aard kunnen het beste worden vermeden. Hiervoor zijn twee redenen. Allereerst kunnen ze leiden tot heftige ruzies die afleiden van een productieve discussie. Ten tweede, en nog belangrijker, kunnen sommige vrijwillige ontwikkelaars van *uw* project tevens aan het concurrerende product blijken te werken. Dit is waarschijnlijker dan het op het eerste gezicht lijkt. De projecten bevinden zich al op hetzelfde terrein (daarom zijn ze concurrerend) en ontwikkelaars met expertise op dat gebied kunnen bijdragen leveren overal waar hun expertise kan worden toegepast. Ook als er geen directe ontwikkelaarsoverlapping is, is het waarschijnlijk dat de ontwikkelaars bij uw project op z'n minst ontwikkelaars bij gerelateerde projecten *kennen*. Hun vermogen om constructieve persoonlijke relaties te onderhouden, kan worden belemmerd door al te negatieve marketingboodschappen.

Het afkraken van concurrerende closed source-producten lijkt meer algemeen aanvaard te zijn in de open source-wereld, vooral wanneer die producten zijn gemaakt door Microsoft. Persoonlijk betreur ik deze tendens (hoewel er dus niets mis is met duidelijke vergelijkingen van feiten), niet zozeer omdat het onbeleefd is, maar ook omdat het voor een project gevaarlijk is om te gaan geloven in zijn eigen hype en daarbij voorbij te gaan aan de manieren waarop de concurrentie eigenlijk beter kan zijn. Over het algemeen geldt dat u uit moet kijken voor de gevolgen die marketinguitspraken kunnen hebben voor uw eigen ontwikkelaarsgemeenschap. Mensen kunnen het zo fijn vinden om te worden gesteund door marketinggeld dat ze niet langer objectief zijn over de ware sterke en zwakke punten van hun software. Het is normaal, en zelfs te verwachten, dat de ontwikkelaars van een bedrijf een bepaalde afstand van marketinguitspraken nemen, ook in openbare forums. Uiteraard moeten ze de marketingboodschap niet openlijk en rechtstreeks tegenspreken (tenzij deze feitelijk fout is, maar je hoopt dat zoiets al eerder is opgemerkt), maar ze kunnen er af en toe grapjes over maken als een manier om de rest van de ontwikkelaarsgemeenschap weer met beide benen op de grond te zetten.

6

COMMUNICATIE

De gave om duidelijk te schrijven zou wel eens de belangrijkste vaardigheid kunnen zijn voor iemand in een open source-omgeving. Op de lange duur speelt dit een grotere rol dan programmeertalent. Een uitstekende programmeur met waardeloze communicatieve vaardigheden kan maar één ding tegelijk. En zelfs daarvoor weet hij wellicht niet voldoende aandacht te trekken. Maar een waardeloze programmeur met uitstekende communicatieve vaardigheden kan vele anderen coördineren en overhalen verschillende dingen te doen en daarmee een aanzienlijke bijdrage leveren aan de richting en de voortgang van het project.

Er lijkt weinig verband te bestaan tussen de vaardigheid om goede code te schrijven en de vaardigheid om met je medemensen te communiceren. Er is wel een enig verband tussen het goed kunnen programmeren en technische kwesties goed kunnen omschrijven, maar het beschrijven van technische kwesties maakt slechts een zeer klein onderdeel uit van de communicatie binnen een project. Veel belangrijker is de vaardigheid je in te leven in de doelgroep, je eigen posts en commentaren te zien zoals anderen ze zien en ervoor te zorgen dat anderen hun eigen posts met dezelfde objectiviteit zien. Net zo belangrijk is kunnen herkennen wanneer een bepaald medium of communicatiemiddel niet langer goed werkt, misschien omdat het niet meer past bij de omvang van het project wanneer het aantal gebruikers groter wordt, en in dat geval de tijd nemen hier iets aan te doen.

Al deze punten liggen in theorie voor de hand, maar in de praktijk bemoeilijkt de verwarrende diversiteit van omgevingen waarin open source-software wordt ontwikkeld, wat betreft zowel doelgroepen als communicatiemechanismen, de situatie. Moet een bepaald idee in een post op de mailinglijst worden gezet, als een melding voor de bug tracker of als een opmerking in de code? En hoeveel kennis mag u, bij het beantwoorden van vragen op een publiek forum, aannemen dat de lezer heeft, uitgaande van het feit dat de ‘lezer’ niet alleen degene is die de vraag stelt maar ook iedereen die uw reactie leest? Hoe kan er constructief contact zijn tussen ontwikkelaars en gebruikers, zonder te worden overspoeld door verzoeken om functies, foutieve bugrapporten en gekwebbel? Hoe kunt u zien wanneer een medium zijn maximale capaciteit heeft bereikt en wat moet u daaraan doen?

Oplossingen voor deze problemen lossen vaak maar een deel ervan op, omdat iedere oplossing uiteindelijk achterhaald is als de omvang van het project toeneemt of de structuur verandert. Ze worden ook vaak ad hoc toegepast, omdat het geïmproviseerde reacties zijn op dynamische situaties. Alle deelnemers moeten zich bewust zijn van wanneer en hoe de communicatie vast kan lopen en een bijdrage leveren aan de oplossing. Mensen hierbij helpen is een belangrijk onderdeel van het managen van een open source-project. In de onderstaande gedeelten wordt besproken hoe u met uw eigen communicatie moet omgaan en hoe u ervoor kunt zorgen dat het onderhoud van de communicatiemechanismen voor iedereen binnen het project prioriteit heeft.[22]

6.1 U BENT WAT U SCHRIJFT

Denk hier eens over na: het enige dat mensen over u weten, weten ze door wat u schrijft of wat anderen over u schrijven. Misschien bent u een briljant, scherpzinnig en charismatisch persoon, maar als uw e-mails onsamenhangend en ongestructureerd zijn, zullen mensen denken dat u dat ook bent. Of misschien bent u juist wel een onsamenhangend en ongestructureerd persoon, maar dat hoeft niemand ooit te weten te komen als uw posts helder en informatief zijn.

Extra aandacht besteden aan uw schrijfstijl loont uiteindelijk zeer de moeite. Dit verhaal is afkomstig van Jim Blandy, een ervaren programmeur van open source-software.

In 1993 werkte ik voor de Free Software Foundation. We waren bezig met het testen van bètaversie 19 van GNU Emacs. We maakten ongeveer iedere week een bètarelease. Men probeerde die dan uit en stuurde ons bugrapporten toe. Er was één man die niemand van ons persoonlijk had ontmoet maar die geweldig werk verzette. Zijn bugrapporten waren altijd duidelijk en leidden ons recht op het probleem af. En als hij zelf met een fix kwam, klopte die meestal precies. Hij was echt super.

Voordat de FSF echter code kan gebruiken die door iemand anders is geschreven, moet er wat juridisch papierwerk worden gedaan waarmee mensen de auteursrechten van hun code overdragen aan de FSF. Code aannemen van een totale onbekende en zomaar gebruiken, zorgt gegarandeerd voor een juridische ramp.
Ik e-mailde onze vriend daarom de formulieren met de boodschap "Hier zijn wat papieren die we nodig hebben. U ondertekent dít formulier, vraagt uw werkgever dát formulier te ondertekenen en we kunnen uw fixes gaan gebruiken. Bij voorbaat dank."

Hij antwoordde: "Ik heb geen werkgever."

Dus zei ik "Ok, prima, laat uw universiteit het dan ondertekenen en stuur het terug."

Na een tijdje schreef hij terug en zei, "Nou ja, eh ... ... ik ben pas dertien en ik woon nog bij mijn ouders."

Omdat de jongen niet schreef als een dertienjarige dacht iedereen dat hij veel ouder was. Hieronder volgens een paar aanwijzingen voor uw schrijfstijl waarmee ook u een goede indruk kan maken.

Structuur en opmaak
Laat u niet verleiden alles op te schrijven als een sms-bericht. Schrijf in complete zinnen, met een hoofdletter aan het begin van iedere zin, en begin waar nodig een nieuwe paragraaf. Dit is in e-mails en andere schrijfsels zeer belangrijk. Bij IRC's of soortgelijke kortstondige forums is het meestal geen probleem om hoofdletters weg te laten, verkorte vormen van veelgebruikte uitdrukkingen te gebruiken enz. Neem deze gewoonte alleen niet over in formelere en langer durende forums. E-mails, documentatie, bugrapporten en andere documenten die bedoeld zijn voor permanent gebruik moeten een samenhangende en beschrijvende structuur hebben. Dat is niet omdat het per se goed is om willekeurige regels te volgen, maar omdat deze regels *niet* willekeurig zijn Ze zijn geëvolueerd tot hun huidige vorm omdat ze tekst leesbaarder maken. En dat is de reden dat u ze moet toepassen. Leesbaarheid is niet alleen belangrijk omdat het betekent dat mensen begrijpen wat u schrijft, maar ook omdat het u neerzet als iemand die de tijd neemt om duidelijk te communiceren; dat wil zeggen, iemand die de moeite waard is om naar te luisteren.

Met name ten aanzien van e-mails zijn er voor ontwikkelaars van open source bepaalde ongeschreven regels:

Stuur alleen e-mails met platte tekst, geen HTML, RichText of andere bestandsindelingen die niet leesbaar zijn voor e-mailprogramma's die alleen platte tekst kunnen lezen. Stel uw regels zo in dat ze ongeveer 72 karakters lang zijn. Kom niet boven de 80 karakters, dit is de *de facto* standaardbeeldschermbreedte (dat wil zeggen dat mensen wel bredere beeldschermen kunnen hebben maar geen smallere). Door uw tekst zo op te maken dat het iets *minder* is dan 80 karakters breed houdt u plek over voor een aantal extra aanhalingstekens die in antwoorden van anderen kunnen worden ingevoegd zonder dat uw tekst moet worden opgemaakt.

Gebruik echte regelterugloop. Sommige e-mailprogramma's gebruiken een soort onechte regelterugloop, waarbij tijdens het schrijven van een e-mail uw scherm regelafbrekingen laat zien die er in werkelijkheid niet zijn. Wanneer de e-mail wordt verzonden bevat deze niet de regelafbrekingen die u dacht dat hij had en zal de tekst op sommige beeldschermen anders teruglopen. Wanneer uw e-mailprogramma onechte regelafbreking gebruikt, zoek dan naar een instelling om de harde afbrekingen tijdens het schrijven zichtbaar te maken.

Als u schermoutput, stukjes code of andere opgemaakte tekst opneemt, laat dit dan duidelijk in de opmaak zien, zodat zelfs de luie lezer de grens tussen uw eigen schrijfsels en het aangehaalde materiaal makkelijk kan herkennen. Toen ik met dit boek begon, had ik nooit gedacht dit te moeten adviseren, maar ik heb recentelijk op een aantal mailinglijsten gezien dat mensen de tekst van verschillende bronnen door elkaar gooiden zonder duidelijk te maken welke tekst waarbij hoort. Het effect hiervan is erg frustrerend. Hun posts zijn hierdoor aanzienlijk moeilijker te begrijpen en die mensen komen daardoor eerlijk gezegd een beetje ongeorganiseerd over.

Wanneer u een e-mail van iemand anders citeert, voeg dan uw antwoord toe op de plek waar het bij hoort, zo nodig op verschillende plaatsen, en verwijder de gedeeltes van de e-mail die u niet gebruikt. Wanneer u een korte reactie schrijft die betrekking heeft op iemands gehele e-mail kunt u een *top-post* maken (d.w.z. dat u uw antwoord boven de aangehaalde tekst van de e-mail zet). In andere gevallen zou u het betreffende deel van de oorspronkelijke tekst eerst moeten aanhalen, gevolgd door uw reactie.

Denk goed na over de onderwerpregels van nieuwe e-mails. Het is de belangrijkste regel van uw e-mail, omdat het de anderen binnen het project gelegenheid geeft te beslissen of ze wel of niet verder willen lezen. Moderne e-mailprogramma's kunnen groepen bij elkaar behorende berichten in threads zetten, waarbij e-mails niet alleen aan de hand van een algemeen onderwerp kunnen worden gegroepeerd, maar ook aan de hand van andere koppen (die soms niet worden weergegeven). Logisch gevolg hiervan is dat u, wanneer een thread afdwaalt naar een nieuw onderwerp, het onderwerpveld zou kunnen (en moeten) aanpassen aan het nieuwe onderwerp. De integriteit van de thread blijft bewaard, als gevolg van de vorige koppen, maar het nieuwe onderwerp laat mensen die een overzicht van de thread bekijken, weten dat het onderwerp is veranderd. Dat is ook de reden dat u, wanneer u een nieuw onderwerp wilt beginnen, een nieuwe e-mail moet maken en niet moet antwoorden op een bestaande e-mail en het onderwerp veranderen. Uw e-mail zou anders nog steeds worden ondergebracht in dezelfde thread als waarop u antwoordt, waardoor mensen ten onrechte kunnen denken dat hij over hetzelfde onderwerp gaat. Ook hier is het nadeel niet alleen verspilling van hun tijd, maar ook de kleine beschadiging van uw reputatie als iemand die goed is in het gebruik van communicatiemiddelen.

Inhoud

Goed opgemaakte e-mails trekken lezers aan, maar de inhoud houdt ze vast. Natuurlijk zijn er geen vaste regels die goede inhoud garanderen, maar er zijn wel een paar uitgangspunten die ertoe bijdragen.

Maak het uw lezers gemakkelijk. Er gaat ontzettend veel informatie om in een actief open source-project en van de lezers kan niet worden verwacht dat ze overal van op de hoogte zijn. Sterker nog, er kan zelfs niet van hen worden verwacht dat ze weten hoe ze ervan op de hoogte moeten blijven. Waar mogelijk moet uw post informatie bevatten in een vorm die het meest geschikt is voor de lezers. Als u twee extra minuten moet spenderen aan het opsporen van de URL van een bepaalde thread in het archief van de mailinglijst om uw lezers deze moeite te besparen, dan is het de moeite waard. Wanneer u vijf of tien minuten extra kwijt bent aan het samenvatten van de conclusies tot dan toe van een complexe thread, om mensen daarmee een context te geven voor uw e-mail, doe dat dan. U moet het zo zien: hoe succesvoller een project, des te gunstiger de verhouding tussen het aantal lezers ten opzichte van het aantal schrijvers op de forums. Wanneer iedere post wordt gezien door *n* personen, dan gaat het nut van het besteden van extra moeite aan het besparen van tijd voor deze mensen ook omhoog wanneer *n* omhoog gaat. Ook zullen mensen die zien dat u zichzelf deze taak oplegt zelf ook hun best hiervoor doen in hun eigen berichten. In het ideale geval is het resultaat een stijging van de algehele efficiëntie van het project: wanneer er gekozen kan worden tussen *n* mensen die moeite moeten doen, of één persoon, dan is de laatste het voordeligst voor het project.

Verval niet in overdrijving. Overdrijven in online posts is een klassieke bewapeningswedloop. Een persoon meldt bijvoorbeeld een bug en hij is bang dat de ontwikkelaars er te weinig aandacht aan zullen besteden. Daarom omschrijft hij de bug als een ernstig probleem dat het hele project schaadt en waardoor hij, en al zijn vrienden/collega's/familieleden, de software niet productief kunnen gebruiken, terwijl het in feite slechts om een kleine ergerlijkheid gaat. Overdrijving wordt echter niet alleen door gebruikers gebezigd, ook programmeurs hebben er een handje van tijdens technische discussies, met name wanneer de onenigheid betrekking heeft op smaak en niet op correcte code:

"Als we dat doen wordt de code compleet onleesbaar. Het onderhoud ervan wordt een nachtmerrie, in tegenstelling tot het voorstel van J. Random ..."

Hetzelfde gevoel kan zelfs *sterker* worden overgebracht met minder scherp taalgebruik:

"Het werkt wel, maar volgens mij is het minder geschikt wat betreft leesbaarheid en onderhoudsgemak. Het voorstel van J. Random voorkomt dit probleem omdat het ..."

U zult niet helemaal zonder overdrijving kunnen en over het algemeen hoeft u dat ook niet te proberen. Vergeleken met andere vormen van miscommunicatie is overdrijving niet schadelijk voor het hele project, alleen voor de overdrijver. De ontvangers weten er echt wel mee om te gaan, alleen de zender verliest iedere keer weer een beetje van zijn geloofwaardigheid. Daarom zou u, ten behoeve van uw eigen invloed binnen het project, moeten proberen gematigd te blijven in uw uitspraken. Ook nemen mensen u dan serieus wanneer u *wel* een belangrijk punt naar voren brengt.

Kijk uw bericht twee keer na. Lees ieder bericht dat langer is dan een gemiddelde paragraaf nog eens van A tot Z door voordat u het verstuurt, en dan nog een keer. Dit is een bekend advies voor mensen die een schrijfcursus hebben gevolgd, maar voor online discussies is het extra belangrijk. Omdat in veel gevallen het schrijfproces van online berichten dikwijls wordt onderbroken (tijdens het schrijven moet u misschien andere e-mails nalezen, webpagina's bezoeken, een commando runnen om de output te genereren enz.) kunt u uw verhaallijn makkelijk uit het oog verliezen. Berichten die met veel onderbrekingen zijn geschreven en niet meer gecontroleerd zijn, zijn dikwijls als zodanig te herkennen, vaak tot grote ergernis van hun auteurs (dat hopen we tenminste). Neem de tijd om na te kijken wat u verstuurt. Hoe beter de structuur van uw posts is, des te meer ze zullen worden gelezen.

Toon

Na het schrijven van duizenden berichten zult u merken dat uw schrijfstijl steeds beknopter wordt. Dit lijkt normaal te zijn voor de meeste technische forums en daar is in principe niets mis mee. Een bepaalde mate van beknoptheid, die in normale sociale omgang niet acceptabel zou zijn, is nou eenmaal de norm voor hackers van

open source-software. Hieronder een reactie die ik ooit kreeg op een mailinglijst over open source-software voor contentmanagement, volledig geciteerd:

```
Kun je misschien wat meer vertellen over de problemen waar je
precies tegenaan liep, etc.?

Ook:

Welke versie van Slash gebruik je? Dit kon ik niet opmaken uit
je oorspronkelijke bericht.

Hoe heb je de source van apache/mod_perl precies gebouwd?

Heb je de Apache 2.0-patch die gepost is op slashcode.com al ge-
probeerd?

Shane
```

Dat is nog eens beknopt! Geen begroeting, behalve zijn naam geen afsluiting en het bericht bestaat uit niet meer dan een reeks vragen die zo compact mogelijk zijn gesteld. En de enige verklarende zin was ook nog kritiek op mijn oorspronkelijke bericht. Maar toch was ik blij met Shane's e-mail en het feit dat hij beknopt was, zag ik alleen als teken dat hij het druk had. Alleen al het feit dat hij vragen stelde, in plaats van mijn post te negeren, betekende dat hij bereid was tijd te besteden aan mijn probleem.

Maar zullen alle lezers positief reageren op deze manier van schrijven? Niet per definitie, dat hangt af van de persoon en de context. Wanneer iemand bijvoorbeeld net een fout heeft moeten toegeven - misschien heeft hij een bug geschreven - en u weet uit het verleden dat deze persoon soms onzeker is, dan kunt u nog steeds een korte reactie schrijven, maar zorg er ook voor dat u laat merken begrip te hebben voor zijn gevoelens. Het grootste deel van uw reacties kan een korte en technische analyse van de situatie zijn, zo kort en zakelijk als u maar wilt. Maar sluit uw bericht altijd af met iets waarmee u aangeeft dat u misschien wel kortaf maar niet onvriendelijk bent. Wanneer u bijvoorbeeld een hele waslijst met adviezen heeft gegeven over hoe iemand een bug moet herstellen, sluit dan af met "Succes, <uw naam>" om aan te geven dat u hem het beste toewenst en niet boos bent. Een strategisch geplaatste smiley of andere emoticon kan vaak ook genoeg zijn om uw gesprekspartner gerust te stellen.

Het lijkt misschien vreemd om evenveel aandacht te besteden aan iemands gevoelens als aan wat ze inhoudelijk zeggen, maar gevoelens zijn nou eenmaal van invloed op de productiviteit. Gevoelens zijn ook om andere redenen belangrijk, maar zelfs als we ons beperken tot puur pragmatische redenen zullen we merken dat ongelukkige mensen slechtere software schrijven, en ook nog eens minder. Door de beperkingen van de meeste elektronische media hebben we meestal geen duidelijk beeld van hoe een persoon zich voelt. U zult dat zo goed mogelijk moeten inschatten op basis van a) hoe de meeste mensen zich in een dergelijke situatie zouden voelen en b) wat u over deze persoon weet uit voorgaande gesprekken. Sommige mensen hebben de voorkeur voor een meer afstandelijke benadering en gaan met iedereen om op basis van de indruk die ze maken. Het idee daarachter is dat als een deelnemer zijn gevoelens niet direct aangeeft iemand anders niet het recht heeft hem wel op die manier te behandelen. Ik ben het om een aantal redenen niet met deze benadering eens. Eén ervan is dat mensen zich in het dagelijks leven ook niet zo gedragen. Dus waarom zouden ze dat online wel doen? Een tweede reden is dat mensen op publieke forums - waar de meeste gesprekken plaatsvinden - nog minder geneigd zijn hun gevoelens te tonen dan in hun privéleven. Of beter gezegd, mensen zijn vaak wel bereid hun gevoelens ten opzichte van anderen te uiten, zoals dankbaarheid of verontwaardiging, maar niet hun innerlijke gevoelens, zoals onzekerheid of trots. Toch werken de meeste mensen beter wanneer ze weten dat anderen zich bewust zijn van hun gemoedstoestand. Door aandacht te besteden aan kleine aanwijzingen kunt u meestal wel een juiste inschatting maken. Daardoor kunt u mensen beter motiveren om betrokken te blijven dan anders het geval zou zijn.

Ik bedoel natuurlijk niet dat u altijd de rol van therapeut van de hele groep op zich moet nemen door iedereen steeds te helpen met zichzelf in het reine te komen. Maar door voldoende aandacht te besteden aan patronen in het gedrag van mensen kunt u inzicht krijgen in hun persoonlijkheid, zelfs wanneer u ze nooit persoonlijk ontmoet. En door zich bewust te zijn van de toon van uw schrijfsels kunt u verrassend veel invloed hebben op hoe anderen zich voelen, wat uiteindelijk alleen maar goed kan zijn voor het project.

Onbeleefd gedrag herkennen

Een van de belangrijkste kenmerken van de open source-cultuur is duidelijke ideeën over wat wel en niet wordt gezien als onbeleefd taalgebruik. Hoewel de onderstaande discussies niet uniek zijn voor de ontwikkeling van open source-software en zelfs niet van software in het algemeen - ze zullen iedereen die werkt in de vakgebieden wiskunde, de exacte wetenschappen of techniek zelfs bekend voorkomen - is open source-software, met zijn soms onduidelijke grenzen en constante toestroom van nieuwkomers, een omgeving waar mensen vaak tegen normen aanlopen waarmee ze bekend zijn.

Laten we beginnen met de dingen die *niet* onbeleefd zijn:

Technische kritiek, zelfs als dit direct en met weinig tact wordt gebracht, is niet onbeleefd. Het kan zelfs als een compliment worden beschouwd: de kritiek impliceert dat het lijdend voorwerp ervan serieus moet worden genomen en de moeite waard is tijd aan te besteden. Dat wil zeggen dat hoe groter de kans zou zijn dat iemands post gewoon wordt genegeerd, des te groter het compliment is wanneer iemand de tijd neemt kritiek te leveren, tenzij de kritiek natuurlijk vervalt in een *persoonlijke* aanval of een andere vorm van overduidelijke onbeleefdheid.

Botte vragen zonder enige omlijsting, zoals Shane's vraag in de hiervoor geciteerde e-mail, zijn evenmin onbeleefd. Vragen die in een andere context misschien koel, gekunsteld of zelfs onecht kunnen overkomen, zijn vaak serieus bedoeld en hebben geen andere bedoeling dan zo snel mogelijk informatie te krijgen. De beroemde

vraag van een technische dienst "Zit de stekker van uw computer in het stopcontact?" is hier een klassiek voorbeeld van. De vraagsteller moet echt weten of de stekker er in zit en was het al na een paar dagen dit werk te hebben gedaan zat om zijn vraag steeds weer in te leiden met beleefdheden ("Neemt u mij niet kwalijk, ik wil u graag een paar eenvoudige vragen stellen om een aantal mogelijkheden uit te kunnen sluiten. Sommige daarvan lijken misschien wel erg simpel, maar ik vraag u geduld met me te hebben ..."). Op dit punt neemt hij niet meer de moeite zijn vraag mooi in te kleden en vraagt direct op de man af: "Zit de stekker in het stopcontact?" Vergelijkbare vragen worden ook op mailinglijsten van open source-softwareprojecten steeds weer gesteld. De bedoeling is niet om de ontvanger te beledigen, maar gewoon om snel de meest voor de hand liggende (en misschien wel meest voorkomende) verklaringen uit te kunnen sluiten. Ontvangers die dit begrijpen en overeenkomstig reageren, worden gewaardeerd om hun ruimdenkende instelling zonder hier verder woorden aan vuil te maken. Ontvangers die hier slecht op reageren, hoeven hier echter nou ook niet voor op hun kop te krijgen. Het is gewoon een botsing tussen twee culturen waar niemand schuld aan heeft. Leg op een vriendelijke manier uit dat u met uw vraag of kritiek geen bijbedoelingen had. Het was alleen bedoeld om op een zo efficiënt mogelijk manier informatie te krijgen of te geven.

Wat is dan wel onbeleefd?

Om dezelfde reden dat gedetailleerde technische kritiek een vorm van complimenteren kan zijn, kan het niet geven van kwalitatieve kritiek een vorm van belediging zijn. Ik bedoel hier niet het eenvoudigweg negeren van iemands werk, voorstel, codewijziging, gerapporteerde issue of wat dan ook. Tenzij u vooraf hebt beloofd om een gedetailleerde reactie te geven, is het over het algemeen geen probleem wanneer u helemaal niet reageert. Mensen zullen er vanuit gaan dat u geen tijd had om te reageren. Maar als u *wel* reageert, raffel het dan niet af. Neem de tijd om alles goed te analyseren, geef waar mogelijk concrete voorbeelden, ga in de archieven graven om gerelateerde posts uit het verleden te vinden enz. Als u hiervoor geen tijd hebt maar wel een korte reactie moet geven, wees dan open over deze tekortkoming ("Ik denk dat dit onderwerp al eerder aan bod is geweest, maar helaas heb ik geen tijd gehad om ernaar te zoeken, sorry"). Het belangrijkste dat onderkend moet worden, is het bestaan van een cultuur. Dat kan door erin mee te gaan of door openlijk toe te geven een keer de fout in te zijn gegaan. Wat er ook gebeurt, de norm wordt hierdoor versterkt. Maar als u niet aan de norm voldoet, terwijl u niet toegeeft *waarom* niet, komt het over alsof het onderwerp (en de mensen die eraan meedoen) niet de moeite waard was om tijd aan te besteden. U kunt beter later merken dat uw tijd kostbaar is door bondig te zijn dan lui.

Er zijn natuurlijk nog vele andere vormen van onbeleefd gedrag, maar de meeste daarvan zijn niet specifiek voor softwareontwikkeling en kunnen met wat gezond verstand worden voorkomen. Zie ook het gedeelte 'Grofheden in de kiem smoren' in Hoofdstuk 2*, Aan de gang*, wanneer u dat nog niet hebt gedaan.

Het gezicht

Er is een gedeelte van het menselijk brein dat speciaal is gereserveerd voor het herkennen van gezichten. Dit staat ook wel bekend als het 'fusiforme gezichtsherkenningsgebied' en de eigenschappen hiervan zijn voor het grootste deel aangeboren, niet aangeleerd. Het blijkt dat het herkennen van mensen afzonderlijk zo essentieel is om te overleven, dat we daar in de loop van de evolutie speciale 'hardware' voor hebben ontwikkeld.

Samenwerking via internet is daarom psychologisch gezien ongewoon, omdat er nauw wordt samengewerkt tussen mensen die elkaar bijna nooit leren kennen via de meest natuurlijke en intuïtieve methodes: gezichtsherkenning op de eerste plaats, maar ook stemgeluid, houding enz. Probeer, om dit te compenseren, om overal een consistente *schermnaam* te gebruiken. U kunt hiervoor het eerste deel van uw e-mailadres (het deel voor het @-karakter), uw IRC-gebruikersnaam, uw committernaam, uw gebruikersnaam als issue tracker enz. gebruiken. Deze naam is uw online 'gezicht': een korte herkenbare reeks karakters die wat betreft de bedoeling ervan overeenkomsten heeft met uw echte gezicht, hoewel deze helaas niet dezelfde ingebouwde hardware in onze hersenen stimuleert als een echt gezicht.

Uw schermnaam kan bijvoorbeeld een intuïtieve lettercombinatie zijn uit uw eigen naam. Die van mij is bijvoorbeeld 'kfogel'. In sommige situaties gaat deze sowieso vergezeld van uw volledige naam, bijvoorbeeld in e-mailvelden:

```
Van: "Karl Fogel" <kfogel@whateverdomain.com>
```

In feite zijn er twee dingen te zien in dit voorbeeld. Zoals eerder al gezegd komt de schermnaam op een intuïtieve manier overeen met de werkelijke naam. Maar de echte naam is ook *echt*. Dat wil zeggen dat het niet zomaar een bedachte benaming is zoals:

```
Van: "Wonder Hacker" <wonderhacker@whateverdomain.com>
```

Paul Steiner publiceerde op 5 juli 1993 een beroemd geworden cartoon in *The New Yorker*, waarin een hond te zien is die is ingelogd op een computerterminal en een andere hond samenzweerderig vertelt: "Op internet ziet niemand dat je een hond bent." Deze gedachtegang is waarschijnlijk de reden achter de vele zelfverheerlijkende, hip bedoelde online identiteiten die mensen zichzelf toekennen. Alsof een naam als 'Wonder Hacker' er werkelijk voor zorgt dat mensen geloven dat deze persoon ook een geweldige hacker *is*. Maar één feit blijft: ook als niemand weet dat u een hond bent, blijft u een hond. Een fantastische online identiteit heeft nog nooit indruk gemaakt op een lezer. Mensen denken in plaats daarvan dat het bij u meer om uw imago dan om de inhoud gaat, of dat u gewoon onzeker bent. Gebruik uw echte naam voor al uw communicatie. Als u om wat voor reden dan ook anoniem wilt blijven, bedenk dan een naam die heel normaal klinkt en gebruik deze consequent.

Naast een consistent online gezicht zijn er een paar dingen die het gezicht nog aantrekkelijker kunnen maken. Als u een officiële titel hebt, zoals 'doctor', 'professor' of 'directeur', loop er dan niet mee te koop. Vermeld hem niet eens, behalve wanneer het direct relevant is voor de discussie. In het hackerswereldje in het alge-

meen, en in het bijzonder binnen de vrijesoftwarecultuur, wordt het pronken met titels gezien als arrogantie en een teken van onzekerheid. Het is geen probleem als uw titel vermeld staat in de standaarde-mailhandtekening onderaan ieder bericht dat u verstuurt. Gebruik deze alleen nooit om uw gelijk te halen in een discussie; dit heeft gegarandeerd een averechtse uitwerking. U wilt dat mensen u respecteren als persoon, niet om uw titel.

En nu we het toch over e-mailhandtekeningen hebben: houd ze klein en smaakvol, of nog beter, gebruik ze helemaal niet. Voorkom eveneens lange juridische disclaimers aan het einde van iedere e-mail, met name wanneer daar opvattingen in staan die niet passen bij deelname aan een open source-softwareproject. De volgende klassieker uit het genre staat aan het eind van iedere post van een bepaalde gebruiker van een publieke mailinglijst waarvan ik ook lid ben:

```
IMPORTANT NOTICE

If you have received this e-mail in error or wish to read our
e-mail disclaimer statement and monitoring policy, please refer
to the statement below or contact the sender.

This communication is from Deloitte & Touche LLP.  Deloitte &
Touche LLP is a limited liability partnership registered in
England and Wales with registered number OC303675.  A list of
members' names is available for inspection at Stonecutter Court,
1 Stonecutter Street, London EC4A 4TR, United Kingdom, the firm's
principal place of business and registered office.  Deloitte &
Touche LLP is authorised and regulated by the Financial Services
Authority.

This communication and any attachments contain information
which is confidential and may also be privileged.  It is for the
exclusive use of the intended recipient(s).  If you are not the
intended recipient(s) please note that any form of disclosure,
distribution, copying or use of this communication or the
information in it or in any attachments is strictly prohibited
and may be unlawful.  If you have received this communication in
error, please return it with the title "received in error" to
IT.SECURITY.UK@deloitte.co.uk then delete the email and destroy
any copies of it.

E-mail communications cannot be guaranteed to be secure or
error free, as information could be intercepted, corrupted,
amended, lost, destroyed, arrive late or incomplete, or contain
viruses.  We do not accept liability for any such matters or
their consequences.  Anyone who communicates with us by e-mail
is taken to accept the risks in doing so.

When addressed to our clients, any opinions or advice contained
in this e-mail and any attachments are subject to the terms and
conditions expressed in the governing Deloitte & Touche LLP
client engagement letter.

Opinions, conclusions and other information in this e-mail and
any attachments which do not relate to the official business of
the firm are neither given nor endorsed by it.
```

Voor iemand die zich zo nu en dan laat zien om een vraag te stellen, lijkt deze gigantische disclaimer misschien wat onnozel, maar het kan waarschijnlijk niet echt kwaad. Als deze persoon echter actief wil participeren in het project kan zo'n blok tekst een sluipende impact hebben. Hij zendt minstens twee mogelijk destructieve signalen uit: ten eerste beheerst deze persoon zijn technische middelen niet volledig (hij zit vast in een bedrijfse-mailsysteem dat irritante boodschappen toevoegt aan het eind van iedere e-mail en hij weet niet hoe hij dat moet omzeilen) en ten tweede heeft hij weinig steun vanuit zijn organisatie voor zijn activiteiten op het gebied van open source-software. De organisatie heeft hem duidelijk niet rechtstreeks verboden om berichten te plaatsen op publieke mailinglijsten, maar zorgt er wel voor dat zijn posts er beslist onvriendelijk uitzien, alsof het risico op het bekendmaken van vertrouwelijke informatie alle andere prioriteiten overtreft.

Als u voor een organisatie werkt die erop staat dat zulke tekstblokken aan alle uitgaande berichten worden toegevoegd, overweeg dan om een gratis e-mailaccount speciaal voor het project te nemen, bijvoorbeeld bij gmail.google.com, www.hotmail.com of www.yahoo.com.

6.2 DE GEBRUIKELIJKE VALKUILEN OMZEILEN

Plaats geen nutteloze posts

Een veel voorkomende valkuil bij deelname aan projecten is denken dat u op alles moet reageren. Dat hoeft niet. Allereerst lopen er meestal meerdere threads tegelijk, die u niet allemaal kunt volgen, in ieder geval niet als het project de eerste fase van enkele maanden achter zich heeft. Ten tweede is er op veel dingen die mensen zeggen op the threads die u wel volgt helemaal geen reactie nodig. Met name ontwikkelingsforums worden gedomineerd door drie soorten berichten:

1. berichten met triviale voorstellen;
2. berichten waarin bijval of verschil van mening wordt geuit op iets dat eerder is gezegd; en
3. samenvattende berichten.

Op geen van deze berichten hoeft in principe te worden gereageerd, met name als u er op basis van de voorgaande berichten in de thread redelijk zeker van kunt zijn dat iemand anders gaat zeggen wat u ook zou zeggen. (Wees niet bang dat u met z'n allen in een vicieuze wachtcirkel terecht komt omdat iedereen dezelfde tactiek gebruikt. Er is altijd wel *iemand* die zich in het strijdgewoel stort.) Wanneer u reageert, moet u daar een bepaald doel mee hebben. Vraag uzelf eerst af: weet ik wat

ik wil bereiken? En daarna: wordt dit niet ook bereikt als ik niet reageer?

Twee goede redenen om uw stem in een thread te laten horen zijn a) als u een fout ziet in een voorstel en vermoedt dat u de enige bent die dit gezien heeft en b) als u ziet dat er sprake is van miscommunicatie tussen anderen en u weet dat u dit kunt oplossen door een verklarende post. Het is over het algemeen ook geen probleem om alleen iemand te bedanken voor iets, of "Ik ook!" te zeggen, omdat een lezer direct kan zien dat er op zo'n post geen reactie of verdere actie vereist is en de geestelijke inspanning die vereist is voor de post zodoende volledig eindigt wanneer de lezer de laatste regel van de e-mail bereikt heeft. Maar zelfs dan moet u goed nadenken voordat u iets zegt. Het is altijd beter dat mensen hopen op meer posts van u dan op minder. (Zie de tweede helft van Bijlage C*, Waarom moet ik me druk maken over de kleur van het fietsenhok?* voor meer ideeën over hoe u zich kunt gedragen op een drukke mailinglijst.)

Productieve versus niet-productieve threads

Op een drukke mailinglijst staat u twee dingen te doen. Het eerste is, uiteraard, dat u uit moet zien te vinden waar u aandacht aan gaat besteden en wat u kunt negeren. Het tweede is dat u geen ruis *veroorzaakt*: u wilt niet alleen dat uw posts een hoog opmerkingsgehalte hebben, u wilt ook dat uw posts *andere* mensen stimuleren om posts in te sturen met hetzelfde opmerkingsgehalte, of anders om helemaal geen posts in te sturen.

Om te zien hoe u dat kunt bereiken, kijken we naar de context. Wat zijn kenmerken van een niet-productieve thread?

- Argumenten die eerder al zijn geuit, worden herhaald alsof degene die ze post, denkt dat niemand ze de eerste keer heeft opgemerkt.
- Er wordt steeds meer overdreven en steeds meer mensen raken betrokken, terwijl de belangen steeds kleiner worden.
- Het grootste deel van de reacties komt van mensen die over het algemeen weinig tot niets doen, terwijl mensen die wel wat doen meestal weinig laten horen.
- Veel ideeën worden besproken zonder dat er ooit een duidelijk voorstel wordt gedaan. (Natuurlijk begint ieder interessant idee met een grove notie. De belangrijke vraag is welke richting het vanaf dat moment op gaat. Lijkt het erop dat de thread dit inzicht omzet in iets concreters of mondt het uit in nog meer subinzichten, neveninzichten en discussies over de zin van leven?)

Een thread is niet per definitie tijdverspilling als deze niet vanaf het eerste begin productief is. Hij kan over een erg belangrijk onderwerp gaan (in welk geval het feit dat het nergens naartoe gaat nog meer reden tot bezorgdheid is).

Een thread in een productieve richting sturen zonder opdringerig te zijn, is een kunst op zich. Het werkt niet als u mensen aanspoort geen tijd te verspillen aan trivialiteiten of hun te vragen om alleen een bericht te posten als ze een constructieve inbreng hebben. U kunt dit natuurlijk wel denken, maar als u dit hardop zegt, kunt u mensen beledigen. In plaats daarvan geeft u suggesties over welke kant de thread op zou moeten gaan, door mensen een richting te geven, een route die leidt naar de resultaten waarop u zit te wachten, maar wel zonder dat het klinkt alsof u mensen de les leest over hoe ze zich moeten gedragen. Het verschil tussen de twee zit 'm vooral in de toon. Dit is bijvoorbeeld slecht:

Deze discussie leidt tot niets. Kunnen we dit onderwerp alsjeblieft laten vallen totdat iemand een patch heeft om één van de voorstellen te implementeren? Het heeft geen zin om steeds maar rondjes te blijven draaien en hetzelfde te herhalen. Code zegt meer dan duizend woorden, mensen.

En dit is dan een goed voorbeeld:

Er zijn verschillende voorstellen in omloop in deze thread, maar in geen ervan zijn alle details uitgewerkt, in ieder geval niet genoeg voor een duidelijke stemming voor of tegen. Ook wordt er de laatste tijd niets nieuws meer ingebracht; we herhalen alleen maar wat er eerder ook al is gezegd. Op dit moment zou ik zeggen dat het het beste zou zijn wanneer toekomstige posts óf een volledige specificatie geven van het voorstel óf een patch. Dan hebben we tenminste iets concreets om mee te werken (bijv. het eens worden over de specificatie of de patch toepassen en testen).

Vergelijk de tweede benadering eens met de eerste. De tweede maakt geen onderscheid tussen u en de anderen en beschuldigt anderen er niet van dat zij de discussie in een cirkel laten lopen. Er wordt over 'wij' gesproken, wat erg belangrijk is, of u nou wel of niet eerder hebt deelgenomen aan de thread, omdat het iedereen eraan herinnert dat zelfs degenen die tot nu toe niet veel van zich hebben laten horen belang hebben bij de uitkomst van de thread. Het beschrijft waarom de thread geen enkele kant opgaat, maar zegt dit zonder negatieve bijklank of oordeel; het benoemt alleen zakelijk enkele feiten. En wat het belangrijkst is, het laat een positieve manier van werken zien, waardoor mensen niet het gevoel hebben dat de discussie gesloten is (een restrictie die alleen maar opstandigheid kan uitlokken) maar dat er een manier wordt voorgesteld om de conversatie naar een meer productief niveau te tillen. Dat is een norm waaraan mensen van nature graag willen voldoen.

U hoeft er niet altijd voor te zorgen dat de thread constructiever wordt. Soms wilt u alleen maar dat hij vanzelf verdwijnt. Het doel van uw post is dan de thread de ene of de andere kant op te sturen. Wanneer u uit het verloop van de thread kunt opmaken dat niemand werkelijk de stappen zal nemen die u voorstelt, dan wordt door uw post de thread afgesloten zonder dat die indruk wordt gewekt. Natuurlijk bestaat er geen waterdichte manier om een thread te sluiten, maar zelfs als die er wel was, zou u die niet willen gebruiken. Deelnemers echter vragen om voor duidelijke vooruitgang te zorgen of anders te stoppen met posten, is absoluut gerechtvaardigd, mits het op diplomatieke manier gebeurt. Pas alleen op dat u threads niet te vroeg afbreekt. Een beetje speculatief gebabbel kan productief zijn, al naar gelang van het onderwerp, en als u dan vraagt hiermee te stoppen, kan het creatieve proces in de kiem worden gesmoord en zorgt u er tegelijkertijd voor dat u ongeduldig overkomt.

Verwacht niet dat een thread van het ene op het andere moment stopt. Er volgen

waarschijnlijk nog een paar posts na die van u, of omdat e-mails elkaar kruisen of omdat men het laatste woord wil hebben. Daarover hoeft u zich geen zorgen te maken. Ook hoeft u geen nieuw bericht te posten. Laat de thread vanzelf doodbloeden, of niet doodbloeden, want dat kan ook het geval zijn. U heeft nooit volledige controle over andere mensen, maar aan de andere kant kunt u er wel vanuit gaan dat u in vele threads statistisch gezien een aanzienlijke invloed hebt.

Hoe makkelijker het onderwerp, des te langer de discussie

Hoewel discussies binnen ieder onderwerp van de rode draad kunnen gaan afwijken, is de kans hierop groter wanneer de technische complexiteit van het onderwerp minder wordt. Tenslotte kunnen minder deelnemers volgen wat er gebeurt als een onderwerp technisch erg moeilijk is. Degenen die dat wel kunnen, zijn hoogstwaarschijnlijk de meest ervaren ontwikkelaars, die al duizenden keren dergelijke discussies hebben gevoerd en weten welk soort gedrag leidt tot de consensus waarmee iedereen kan leven.

Daarom is het het moeilijkst om consensus bereiken over technische vragen die makkelijk te begrijpen zijn en waarover iedereen een mening heeft en over 'softe' onderwerpen, zoals organisatie, publiciteit, financiering enz. Mensen kunnen eindeloos in dergelijke discussies blijven hangen, omdat men niet gekwalificeerd hoeft te zijn om mee te mogen praten, omdat er (zelfs achteraf) geen duidelijke manier is om vast te stellen of een beslissing goed of fout is en omdat het soms een goede tactiek kan zijn de reacties van andere deelnemers in de discussie af te wachten.

Het principe dat de hoeveelheid discussie omgekeerd evenredig is aan de complexiteit van het onderwerp is al lang bekend en staat informeel ook bekend als het *fietsenhokeffect*. Poul-Henning Kamp gaf hiervoor in een inmiddels beroemde post aan BSD-ontwikkelaars een verklaring:

Het is een lang verhaal, of eigenlijk een heel oud verhaal, en eigenlijk toch een kort verhaal. C. Northcote Parkinson schreef in het begin van de jaren 60 van de vorige eeuw een boek met de titel 'De wet van Parkinson', met veel inzichten in de dynamiek van management.

[...]

In het specifieke voorbeeld van het fietsenhok is de andere essentiële component een kerncentrale. Dat geeft denk ik een goede datering van dit boek.

Parkinson laat zien hoe u in de raad van bestuur goedkeuring kunt krijgen voor de bouw van een kerncentrale van miljoenen of zelfs miljarden dollars, maar dat u in eindeloze discussies verzeild raakt als u een fietsenhok wilt neerzetten.

Parkinson legt uit dat dit komt doordat een kerncentrale zo enorm groot, duur en gecompliceerd is dat mensen er met hun hoofd niet bij kunnen. In plaats van te proberen het toch te begrijpen, vertrouwen ze op de aanname dat iemand anders de informatie in een vorig stadium wel gecontroleerd zal hebben. Richard P. Feynmann geeft in zijn boeken een paar interessante en zeer treffende voorbeelden met betrekking tot Los Alamos.

Aan de andere kant hebben we het fietsenhok. Iedereen kan in een weekend een fietsenhok bouwen en dan nog genoeg tijd over houden om op zondag naar Studio Sport te kijken. Het maakt niet uit hoe goed voorbereid u bent en hoe redelijk uw voorstel is, iemand zal altijd de kans grijpen om te laten zien dat hij zijn werk doet, dat hij goed oplet, dat hij *er is*.

We noemen dit 'je geur achterlaten'. Het heeft te maken met persoonlijke trots en aanzien; je moet iets aan kunnen wijzen en zeggen "Kijk! Dat heb *ik* gedaan." Politici bijvoorbeeld hebben sterk die neiging, maar iedereen loopt kans ermee besmet te raken. Denk maar aan het achterlaten van voetstappen in nat cement.

(Zijn volledige post is de moeite van het lezen waard. Zie Bijlage C, *Waarom zou ik me druk maken over de kleur van het fietsenhok?* Of http://bikeshed.com.)

Iedereen die wel eens te maken heeft gehad met het nemen van groepsbeslissingen weet waar Kamp het over heeft. Het is over het algemeen echter onmogelijk om *iedereen* ervan te overtuigen dat ze het fietsenhok niet hoeven schilderen. Het beste dat u kunt bereiken, wanneer u het ziet gebeuren, is laten zien dat het fenomeen bestaat en de seniorontwikkelaars (de mensen die het meest in de melk te brokkelen hebben) hun verfkwasten zo snel mogelijk weg te laten leggen zodat zij in ieder geval niet bijdragen aan de ruis. Deze fietsenhokdiscussies zullen nooit helemaal verdwijnen, maar door mensen binnen het project over het fenomeen te informeren kunt u zorgen dat ze korter worden en minder vaak voorkomen.

Voorkom heilige oorlogen

Een *heilige oorlog* is een discussie, vaak - maar niet altijd - over een relatief onbelangrijke kwestie, die niet kan worden opgelost op basis van argumenten, maar waarbij mensen zo emotioneel betrokken zijn, dat ze blijven discussiëren in de hoop gelijk te krijgen. Een heilige oorlog is niet hetzelfde als een fietsenhokdiscussie. Mensen die fietsenhokken verven, hebben meestal snel hun mening klaar (omdat ze die hebben) maar zijn er niet per se emotioneel bij betrokken en kunnen ook andere tegenstrijdige argumenten aandragen om aan te geven dat ze alle gezichtspunten van de kwestie begrijpen. In een heilige oorlog is begrip voor andere argumenten echter een teken van zwakheid. In een heilige oorlog is iedereen het erover eens dat er maar één antwoord juist is; men verschilt echter van mening over welk antwoord dat is.

Zodra een heilige oorlog is begonnen kan deze meestal niet naar ieders tevredenheid worden opgelost. U doet er geen goed aan door te zeggen dat er sprake is van een heilige oorlog terwijl die volop aan de gang is. Dat weet iedereen namelijk al. Helaas is een algemeen kenmerk van heilige oorlogen dat er geen overeenstemming bestaat over de vraag of het geschilpunt sowieso wel met discussiëren kan worden opgelost. Van een afstand bezien is duidelijk dat geen van beide kanten de mening van de ander kan veranderen. Maar de deelnemers zelf denken dat de andere partij gewoon wat traag van begrip is en niet helder denkt, en dat hij wel overstag gaat als hij maar genoeg met argumenten wordt bestookt. Ik zeg *niet* dat

er nooit iemand gelijk heeft in een heilige oorlog. Soms is dat wel het geval. Bij de heilige oorlogen waaraan ik deel heb genomen, was ik altijd degene die gelijk had, uiteraard. Maar dat maakt helemaal niet uit, er bestaat namelijk geen formule voor om overtuigend aan te tonen dat één van beide partijen gelijk heeft.

Een algemene, maar weinig voldoening biedende manier om een heilige oorlog op te lossen is door te zeggen “We hebben al veel meer tijd en energie aan de discussie besteed dan het onderwerp waard is! Kunnen we het niet gewoon laten voor wat het is?" Deze oplossing kent twee problemen. Het eerste is dat er al tijd en energie besteed zijn en dat die nooit meer herwonnen kunnen worden. De enige vraag die overblijft, is hoe veel energie er *nog meer* in moet worden gestoken. Wanneer sommige mensen geloven dat een kleine beetje extra discussie de kwestie kan oplossen, dan heeft het (voor hen) nog steeds zin om ermee door te gaan.

Het andere probleem is dat wanneer wordt gevraagd om de kwestie te laten vallen dit vaak gelijkstaat aan het tot winnaar uitroepen van de partij die voor de status quo staat, omdat er niets verandert. En in sommige gevallen is het behouden van de status quo ook niet acceptabel: iedereen is het erover eens dat er een beslissing moet worden genomen en dat er tot actie moet worden overgegaan. De kwestie laten vallen is voor iedereen een nog slechtere optie dan een eigen argument moeten opgeven. Maar omdat dit dilemma voor iedereen in gelijke mate geldt, is het nog steeds mogelijk om eindeloos door te blijven discussiëren over wat er moet gebeuren.

Dus hoe moet u met heilige oorlogen omgaan?

Het eerste antwoord op deze vraag is: de zaken zo opzetten, dat ze niet voorkomen. Dit is niet zo’n hopeloze taak als het lijkt.

Op bepaalde standaard heilige oorlogen kunt u anticiperen: ze gaan vaker over de programmeertaal, licenties (zie het gedeelte ‘De GPL en compatibiliteit van licenties’ in Hoofdstuk 9, *Licenties, auteursrechten en patenten*), doordraven over ‘Reply to’ (zie het gedeelte ‘Het grote ‘Reply to’-debat’ in Hoofdstuk 3, T*echnische infrastructuur*) en nog een paar van zulke onderwerpen. Meestal heeft ieder project zo wel een heilig oorlogje of twee. Ervaren ontwikkelaars raken hier snel genoeg aan gewend. De technieken voor het stoppen van heilige oorlogen, of in ieder geval voor het beperken van de schade ervan, zijn overal ongeveer gelijk. Zelfs als u zeker weet dat u gelijk hebt, probeer dan in ieder geval *een beetje* sympathie en begrip te tonen voor de argumenten van de tegenpartij. Vaak is het grootste probleem van een heilige oorlog dat beide zijden een zo hoog mogelijke muur hebben opgetrokken en duidelijk hebben gemaakt dat iedere afwijkende mening je reinste onzin is. Daardoor is overgave of van gedachten veranderen psychologisch ondraaglijk geworden, Het betekent niet alleen toegeven dat je ongelijk had, maar dat je iets *zeker* wist en ongelijk had. De manier om dit aanvaardbaar te maken voor de andere partij is door zelf aan te geven onzeker te zijn, juist door te laten zien dat u hun argumenten begrijpt en ze logisch vindt, misschien zelfs overtuigend. Maak een gebaar dat ruimte biedt voor een soortgelijk gebaar van de andere kant. Gewoonlijk verbetert de situatie dan wel. De kans dat u het technische resultaat krijgt dat u wilde, wordt er niet groter of kleiner van, maar u kunt in ieder geval onnodige gevolgschade aan het moreel van het project voorkomen.

Wanneer een heilige oorlog niet kan worden voorkomen, beslis dan meteen aan het begin hoeveel het u eigenlijk kan schelen en wees bereid openlijk toe te geven. Wanneer u dat doet, kunt u zeggen dat u zich terugtrekt omdat u het geen heilige oorlog waard vindt. Toon echter geen bitterheid en gebruik de gelegenheid *niet* om nog een laatste hatelijke opmerking te maken over de argumenten van de tegenpartij. Opgeven is alleen effectief als het met stijl wordt gedaan.

Heilige oorlogen over de programmeertaal zijn een geval apart, omdat ze vaak erg technisch van aard zijn, terwijl er toch veel mensen zijn die zich gekwalificeerd genoeg voelen om er aan deel te nemen. Ook staan er veel belangen op het spel, omdat het resultaat van de oorlog bepaalt in welke taal een groot deel van de code van het project wordt geschreven. De beste oplossing hiervoor is om de taal in een vroeg stadium te kiezen, met behulp van invloedrijke ontwikkelaars die er vanaf het begin bij zijn betrokken en de keuze verdedigen door te zeggen dat het de taal is waar iedereen zich prettig bij voelt, *niet* omdat hij beter is dan andere talen die in plaats daarvan ook hadden kunnen worden gebruikt. Laat een discussie nooit ontaarden in een academische vergelijking van programmeertalen (dit lijkt extra vaak te gebeuren wanneer iemand met Perl op de proppen komt, om wat voor reden dan ook). Dit is een gesloten hoofdstuk waarbij u gewoon moet weigeren betrokken te raken.

Voor een meer historische achtergrond over heilige oorlogen, zie http://catb.org/~esr/jargon/html/H/holy-wars.html en het artikel van Danny Cohen waarmee de term populair werd http://www.ietf.org/rfc/ien/ien137.txt.

Het effect van de ‘luidruchtige minderheid’

Tijdens discussies op mailinglijsten gebeurt het maar al te vaak dat een kleine minderheid, door de lijst met talrijke en lange e-mails te bestoken, de indruk wekt dat een groot deel van de lijst het ergens niet mee eens is. Het lijkt een beetje op obstructie in het politieke debat, behalve dat de illusie van een wijdverbreide afwijkende mening nog sterker is omdat die wordt geuit in een willekeurig aantal afzonderlijke posts en de meeste mensen niet de moeite nemen bij te houden wie wat heeft gezegd en wanneer. Men krijgt echter instinctief de indruk dat het onderwerp erg controversieel is en wil wachten totdat de storm is geluwd.

De beste manier om dit effect tegen te gaan is door een en ander zeer duidelijk uit te leggen en met bewijs te komen voor hoe klein het werkelijke aantal opposanten is vergeleken met het aantal mensen dat het er wel mee eens is. Om het verschil tussen voor- en tegenstanders nog groter te maken kunt u privé navraag doen bij mensen die meestal hun mond houden maar waarvan u denkt dat ze het met de meerderheid eens zijn. Zeg nooit iets waarmee u kunt suggereren dat de opposanten doelbewust proberen de boel op te blazen. De kans is groot dat dit helemaal niet het geval is, en zelfs als dat wel het geval zou zijn, heeft het geen enkel strategisch voordeel dit hardop te zeggen. Het enige dat u hoeft doen, is de aantallen naast elkaar te zetten zodat mensen zich realiseren dat hun intuïtie over de situatie niet overeenkomt met de werkelijkheid.

Dit advies is niet van toepassing op kwesties met duidelijke voor- en tegenargumenten. Het is van toepassing op discussies waar veel drukte over wordt gemaakt, maar waarvan niet duidelijk is of de meesten wel geloven dat de kwestie een reëel probleem is. Na een poosje kunt u, wanneer u het erover eens bent dat het niet de moeite waard is om actie over de kwestie te nemen en u kunt zien dat deze niet veel garen spint (ook al worden er wel veel e-mails gegenereerd) publiekelijk opmerken dat er niet voldoende garen gesponnen wordt. Als er sprake was van het 'luidruchtige minderheid'-effect zal uw post op de mensen overkomen als een frisse wind door de gelederen. De indruk die de meeste mensen van de discussie tot dan toe hebben, zal weinig verheffend zijn: "Hm, het lijkt echt wel alsof er iets belangrijks aan de hand is, want er zijn ontzettend veel posts, maar ik zie geen duidelijke vooruitgang." Door uit te leggen hoe de vorm van de discussie ervoor zorgde dat het turbulenter leek dan het in feite was, krijgen mensen een nieuw kader waardoor ze hun inzichten over wat er zich afspeelde kunnen herzien.

6.3 MOEILIJKE MENSEN

Moeilijke mensen zijn binnen elektronische platforms net zo moeilijk in de omgang als daarbuiten. Met 'moeilijk' bedoel ik niet 'onbeleefd'. Onbeleefde mensen zijn irritant, maar ze zijn niet per definitie moeilijk. Ik heb eerder in dit boek al besproken hoe u met hen kunt omgaan. Reageer meteen de eerste keer op de onbeleefdheden en negeer ze daarna of behandel ze net als ieder ander dat doet. Wanneer ze onbeleefd blijven, maken ze zich meestal niet populair bij de rest. Daardoor is de kans groot dat ze weinig invloed krijgen op het project. Het is eigenlijk een probleem dat zichzelf oplost.

De echt moeilijke gevallen zijn mensen die niet openlijk onbeleefd zijn maar die de projectprocessen manipuleren of saboteren waardoor andere mensen tijd en energie verliezen zonder dat het project daar baat bij heeft. Dergelijke mensen zoeken vaak naar mazen in de procedures van het project om zichzelf meer invloed te geven dan ze anders misschien zouden hebben. Dit is veel verraderlijker dan pure onbeleefdheid, omdat het gedrag en de daaruit voortvloeiende schade geen van beide duidelijk zichtbaar zijn voor de toevallige voorbijganger. Een klassiek voorbeeld is de filibuster, iemand die een politiek debat vertraagt door eindeloze redevoeringen, waarbij iemand (die natuurlijk altijd zo verstandig overkomt als maar mogelijk is) blijft zeggen dat de besproken kwestie nog niet toe is aan een oplossing en steeds meer mogelijke oplossingen of nieuwe gezichtspunten voor oude oplossingen blijft aandragen, terwijl hij in feite aanvoelt dat het project op het punt staat om consensus te bereiken over de kwestie of om het ter stemming voor te leggen en hij het niet eens is met de waarschijnlijke uitkomst. Een ander voorbeeld is wanneer er een debat plaatsvindt dat maar niet tot consensus lijkt te kunnen komen, maar waarbij de groep in ieder geval probeert de punten van onenigheid te verduidelijken en een samenvatting te maken waaraan iedereen kan refereren. De dwarsligger, die weet dat de samenvatting kan leiden tot een oplossing waar hij het niet mee eens is, probeert vaak zelfs de samenvatting te vertragen, door halsstarrig de vraag wat er wel en niet in moet gecompliceerder te maken, door het niet eens te zijn met redelijke suggesties of door onverwachte nieuwe aspecten in te brengen.

Omgaan met moeilijke mensen

Om dit soort gedrag tegen te kunnen gaan, is het nuttig om de mentaliteit van dergelijke mensen te begrijpen. Mensen doen over het algemeen niet bewust moeilijk. Niemand wordt 's morgens wakker en zegt tegen zichzelf: "Vandaag zal ik eens op cynische wijze proberen de procedures te manipuleren en me als irritante dwarsligger gaan gedragen." In plaats daarvan ligt aan dergelijk gedrag vaak een semiparanoïde gevoel ten grondslag, dat iemand van de interacties en de beslissingen binnen de groep wordt uitgesloten. De persoon heeft het gevoel dat hij niet serieus wordt genomen of (in meer ernstige gevallen) dat er bijna sprake is van een samenzwering tegen hem, dat de andere projectleden hebben besloten een exclusief clubje te vormen waarvan hij geen deel uitmaakt. Voor hem rechtvaardigt dit de keuze om de regels letterlijk op te vatten en de procedures van het project te manipuleren, om er zo voor te zorgen dat iedereen hem *wel* serieus neemt. In extreme gevallen gelooft de persoon dat hij een eenzame strijd voert om het project te redden.

Het ligt in de aard van dergelijke bedreigingen van binnenuit, dat niet iedereen het verschijnsel tegelijkertijd in de gaten heeft en sommigen het helemaal niet zien, totdat ze met de harde feiten worden geconfronteerd. Dit betekent dat het erg moeilijk kan zijn om dit fenomeen te neutraliseren. Het is niet voldoende als u zichzelf ervan overtuigt dat dit gebeurt. U moet voldoende bewijsmateriaal verzamelen om ook anderen hiervan te overtuigen en u moet dit bewijsmateriaal op doordachte wijze verspreiden.

Omdat het zo moeilijk is om dit probleem tegen te gaan, is het vaak beter om het eerst een poosje te tolereren. Zie het als een parasitaire, maar niet ernstige ziekte. Als deze het project niet te zeer ondermijnt, is het niet erg als de infectie blijft bestaan. En medicijnen kunnen negatieve bijwerkingen hebben. Wanneer het probleem echter onverantwoord veel schade gaat aanrichten, is het tijd om tot actie over te gaan. Begin met het verzamelen van aantekeningen van wat u ziet gebeuren. Zorg ervoor dat u ook referenties naar openbare archieven opneemt. Dit is één van de redenen waarom er een archief wordt bijgehouden van het project, dus kunt u het dan ook maar beter gebruiken. Zodra u sterk staat, kunt u privégesprekken voeren met andere deelnemers aan het project. Vertel ze niet direct wat u hebt geconstateerd, maar vraag ze eerst wat zij hebben gezien. Dit is waarschijnlijk uw laatste kans op ongecensureerde feedback over hoe anderen het gedrag van de onruststoker zien. Zodra u begint er openlijk over te praten, zullen de meningen zich polariseren en zal niemand zich nog kunnen herinneren hoe ze voorheen over de zaak dachten.

Als uit de privégesprekken blijkt dat minstens een paar anderen het probleem ook zien, is het tijd iets te ondernemen. Vanaf dat moment moet u *heel* voorzichtig te werk gaan. Het is voor dit soort mensen namelijk zeer eenvoudig om het te laten voorkomen alsof u ten onrechte op ze afgeeft. Wat u ook doet, beschuldig iemand er nooit van willens en wetens de procedures van het project te misbruiken, paranoide te zijn of, in algemene zin, van al het andere waarvan u sterke vermoedens hebt. Uw strategie zou moeten zijn om altijd verstandiger en meer gericht op het algemene belang van het project over te komen dan de ander, met als doel het gedrag van die persoon te veranderen of hem het project permanent te laten verlaten. Afhan-

kelijk van de andere ontwikkelaars en uw relatie met hen kan het handig zijn eerst privé medestanders te verzamelen. Het effect hiervan kan echter ook averechts zijn. Het veroorzaakt achter de schermen namelijk alleen maar onrust wanneer mensen denken dat u een misplaatste roddelcampagne bent gestart.

Vergeet niet dat, hoewel de andere persoon degene is die destructief gedrag vertoont, *u* dan degene bent die destructief overkomt wanneer u uw publieke beschuldigingen niet hard kunt maken. Zorg ervoor dat u voldoende voorbeelden heeft waarmee u kunt laten zien wat u bedoelt en zeg het zo beleefd mogelijk, maar wel direct. Misschien bent u niet in staat de betreffende persoon te overtuigen, maar het is voldoende als u alle anderen weet te overtuigen.

Casestudy

Ik herinner me uit mijn ruim tien jaar ervaring met open source-softwareprojecten slechts één situatie waarbij het zo uit de hand liep dat we iemand moesten vragen helemaal te stoppen met het sturen van posts. Zoals zo vaak was deze meneer niet onbeleefd en wilde hij oprecht behulpzaam zijn. Hij wist alleen niet wanneer hij wel en wanneer hij niet moest posten. Onze mailinglijsten waren publiekelijk toegankelijk en hij postte zo veel en stelde zo veel vragen over zo veel verschillende onderwerpen dat het een probleem werd binnen de gemeenschap. We hadden al eens geprobeerd hem op een nette manier te vragen wat meer vooronderzoek te doen voordat hij een post verstuurde, maar dat had geen effect.

De strategie die uiteindelijk werkte, was een perfect voorbeeld van hoe u sterk kunt staan op basis van neutrale en kwantitatieve informatie. Eén van onze ontwikkelaars had wat speurwerk gedaan in de archieven en stuurde het volgende bericht privé naar enkele andere ontwikkelaars. De boosdoener (de derde naam op de lijst, hier 'J. Random') had nog weinig met het project van doen gehad en had geen code of documentatie geschreven. Toch was hij de op twee na meest actieve poster op de mailinglijst:

```
Van: "Brian W. Fitzpatrick" <fitz@collab.net>
Aan: [... ontvangerslijst weggelaten i.v.m. anonimiteit ...]
Onderwerp: De bodemloze energieput van Subversion
Datum: Woensdag 12 Nov 2003 23:37:47 -0600

De afgelopen 25 dagen zijn de onderstaande 6 personen de mensen
met de meeste posts naar de svn [dev|users]:

    294  kfogel@collab.net
    236  "C. Michael Pilato" <cmpilato@collab.net>
    220  "J. Random" <jrandom@problematic-poster.com>
    176  Branko Čibej <brane@xbc.nu>
    130  Philip Martin <philip@codematters.co.uk>
    126  Ben Collins-Sussman <sussman@collab.net>

Ik zou zeggen dat vijf van deze mensen een bijdrage leveren aan
de release van Subversion 1.0 in de nabije toekomst.

Ik zou ook zeggen dat één van deze personen constant tijd en
energie in beslag neemt van de andere 5, om maar niet te spreken
van de andere deelnemers van de lijst, waardoor hij (zij het on-
bedoeld) de ontwikkeling van Subversion vertraagt.  Ik heb geen
grondige analyse gedaan, maar als ik het archief van Subversion
bekijk zie ik dat iedere mail van deze persoon minimaal één keer
wordt beantwoord door minstens 2 van de andere 5 personen uit de
bovenstaande lijst.

Ik denk dat er radicaal moet worden ingegrepen, zelfs als dit
betekent dat we de bovengenoemde persoon wegjagen.  Van genuan-
ceerde opmerkingen en vriendelijkheid is al gebleken dat ze geen
effect hebben.

dev@subversion is een mailinglijst ter ondersteuning van de ont-
wikkeling van een versiebeheersysteem, geen groepstherapie.

-Fitz, in een poging zich door drie dagen opgestapelde svn-mail
heen te werken
```

Hoewel dit op het eerste gezicht misschien niet zo lijkt, was het gedrag van J. Random een klassiek geval van het misbruiken van de projectprocedures. Hij deed niet iets waar je onmiddellijk bezwaar tegen zou maken, zoals een stemming proberen te saboteren, maar hij maakte misbruik van het mailinglijstbeleid, dat was gebaseerd of zelfrestrictie van de leden ten aanzien van hun posts. We hadden het aan ieders beoordelingsvermogen overgelaten wanneer men postte en over welke onderwerpen. Daardoor hadden we geen procedures om op terug te vallen toen er sprake was van iemand die niet beschikte over een dergelijk beoordelingsvermogen of weigerde dit te gebruiken. Er waren geen regels waarvan we konden zeggen dat de man ze aan het overtreden was, maar iedereen wist dat zijn veelvuldige posts een serieus probleem begonnen te worden.

De strategie van Fitz was, achteraf gezien, meesterlijk. Hij verzamelde kwantitatief belastend bewijsmateriaal en verspreide dit vervolgens op discrete wijze, door het eerst naar een paar mensen te sturen wiens steun belangrijk zou zijn bij eventuele drastische maatregelen. Ze kwamen overeen dat er in ieder geval iets moest worden ondernomen en uiteindelijk hebben we J. Random opgebeld, het probleem direct aan hem voorgelegd en hem gevraagd te stoppen met posten. Hij heeft nooit werkelijk begrepen waarom dit was. Als hij in staat zou zijn geweest dit te begrijpen, was hij waarschijnlijk nooit begonnen met zijn obstructiegedrag. Maar hij ging ermee akkoord om niet meer te posten en de mailinglijst werd weer werkbaar. Deze strategie werkte deels misschien door de impliciete dreiging dat we zijn posts zouden kunnen weigeren met behulp van moderatorsoftware die normaal gesproken wordt gebruikt om spam tegen te houden (zie het gedeelte 'Spam voorkomen' in Hoofdstuk 3, *Technische infrastructuur*). Maar de belangrijkste reden dat we deze optie achter de hand hadden, was dat Fitz de noodzakelijke steun had ingewonnen van de sleutelfiguren binnen het project.

6.4 OMGAAN MET GROEI

De prijs van succes is hoog in het wereldje van open source-software. Zodra uw software populairder wordt, neemt het aantal mensen dat opduikt op zoek naar informatie drastisch toe, terwijl het aantal mensen dat in staat is informatie te verstrekken veel langzamer groeit. Maar zelfs wanneer deze verhouding wel gelijk zou zijn, bestaat er een fundamenteel uitbreidbaarheidsprobleem door de manier waarop de meeste open source-projecten met communicatie omgaan. Laten we bijvoorbeeld eens naar mailinglijsten kijken. De meeste projecten hebben mailinglijsten voor algemene vragen van gebruikers. Soms heet de lijst 'users', 'discuss', 'help' of iets dergelijks. Maar wat de naam ook is, het doel van de lijst is altijd hetzelfde: een omgeving bieden waar mensen vragen krijgen op hun antwoorden, terwijl anderen toekijken en (vermoedelijk) kennis in zich opnemen door deze vragen en antwoorden te observeren.

Deze mailinglijsten werken prima tot een paar duizend gebruikers en/of een paar honderd posts per dag. Maar ongeveer als dat aantal bereikt is, begint het systeem te wankelen, omdat iedere deelnemer iedere post te zien krijgt. Wanneer het aantal posts groter wordt dan het aantal dat iedere lezer per dag aankan, dan wordt de lijst een belasting voor de leden. Stelt u zich eens voor dat Microsoft een dergelijke mailinglijst zou hebben voor Windows XP. Windows XP heeft honderden miljoenen gebruikers. Zelfs als ééntiende of zelfs maar één procent daarvan elk etmaal een vraag zou stellen, dan zou deze hypothetische lijst honderdduizenden posts per dag krijgen! Zo'n lijst is natuurlijk nooit levensvatbaar, gewoon omdat niemand er lid van blijft. Dit probleem beperkt zich niet alleen tot mailinglijsten. Hetzelfde geldt voor IRC-kanalen, online discussieforums en ieder ander systeem waarbij een groep de vragen van individuen te zien krijgt. De implicaties daarvan zijn schokkend: het gebruikelijke open source-model van immense parallelle ondersteuning laat zich gewoonweg niet opschalen naar het niveau dat nodig is om de wereld te veroveren.

Er is geen explosie te zien wanneer een forum zijn breekpunt bereikt. Zichtbaar is alleen het negatieve feedbackeffect: mensen schrijven zich uit van de mailinglijst, gaan weg bij het IRC-kanaal of stoppen met het stellen van vragen omdat ze merken dat ze tussen alle andere posts niet worden gehoord. Als steeds meer mensen deze uiterst rationele beslissing nemen, lijkt het alsof de activiteit van het forum gemakkelijk bestuurbaar blijft. Maar het blijft bestuurbaar omdat de verstandige (of in ieder geval ervaren) mensen ergens anders op zoek zijn gegaan naar informatie, terwijl de onervaren mensen achterblijven en blijven posten. Met andere woorden, een neveneffect van het blijven gebruiken van niet-uitbreidbare communicatiemodellen wanneer het project groeit, is dat de gemiddelde kwaliteit van zowel de vragen als de antwoorden omlaag gaat. Daardoor lijkt het alsof nieuwe gebruikers dommer zijn dan voorheen, terwijl dat waarschijnlijk in werkelijkheid niet het geval is. Het belangrijkste is dat het rendement van het gebruik van dergelijke overvolle forums achteruit gaat, waardoor het heel logisch is dat mensen met ervaring eerst ergens anders op zoek gaan naar antwoorden. Het aanpassen van de communicatiemechanismen aan de groei van het project vereist dan ook twee aan elkaar gerelateerde strategieën:

1. Herkennen wanneer bepaalde delen van een forum *niet* te lijden hebben onder de ongebreidelde groei, zelfs wanneer dat voor het forum in zijn geheel wel het geval is, en deze delen afsplitsen naar nieuwe, meer gespecialiseerde forums (d.w.z., gooi het kind niet met het badwater weg).
2. Ervoor zorgen dat er veel geautomatiseerde informatiebronnen beschikbaar zijn en dat deze goed georganiseerd, up-to-date en gemakkelijk te vinden zijn.

Strategie (1) is meestal niet zo moeilijk. De meeste projecten beginnen met één algemeen forum: een algemene discussielijst, waarop iedereen terecht kan met ideeën voor functies, ontwerpvragen en problemen met de code. Iedereen die bij het project betrokken is, is ook lid van de lijst. Na een poosje wordt het meestal duidelijk dat de lijst zich heeft ontwikkeld in een aantal sublijsten met afzonderlijke onderwerpen. Sommige threads gaan duidelijk over ontwikkeling en ontwerp, anderen zijn gebruikersvragen als "Hoe moet ik X doen?" en misschien is er een derde groep onderwerpen rond het afhandelen van bugrapporten en verbeteringsverzoeken enz. Ieder individu kan natuurlijk deelnemen in vele verschillende threads, maar het belangrijkste is dat er niet veel overlap is tussen de verschillende type threads. Ze kunnen worden opgesplitst in afzonderlijke lijsten zonder dat er schadelijke wrijving kan ontstaan, omdat threads zelden over de grenzen van de onderwerpen heen gaan.

In feite is deze opsplitsing een proces met twee fasen. U creëert de nieuwe lijst (of IRC-kanaal, of wat dan ook) en blijft vervolgens mensen net zolang zachtjes aan hun hoofd zeuren totdat ze de nieuwe forums daadwerkelijk correct *gebruiken*. Deze fase kan weken in beslag nemen, maar uiteindelijk zal bij de mensen het kwartje wel vallen. Het enige wat u hoeft doen, is de afzender altijd vertellen wanneer een post naar de verkeerde lijst is gestuurd, en dat ook zichtbaar te doen, zodat andere mensen worden aangemoedigd om te helpen met het routeren. Het kan ook handig zijn om een webpagina te hebben met een richtlijn voor alle beschikbare lijsten. In uw reacties kunt u naar deze webpagina verwijzen waarbij, als extra bonus, de ontvanger ook nog iets kan leren over het zoeken naar richtlijnen voordat hij een post verstuurt.

Strategie (2) is een doorlopend proces, dat tijdens de gehele projectcyclus doorgaat en waar veel deelnemers bij zijn betrokken. Dit is natuurlijk voor een deel een kwestie van het hebben van up-to-date documentatie (zie het gedeelte 'Documentatie' in Hoofdstuk 2, *Aan de slag*) en mensen daar ook naar te verwijzen. Maar er komt nog veel meer bij kijken. De volgende gedeelten geven een gedetailleerde beschrijving van deze strategie.

Opvallend gebruik van archieven

Normaal gesproken wordt alle communicatie (behalve sommige IRC-discussies) gearchiveerd. De archieven zijn voor iedereen toegankelijk, kunnen worden doorzocht en hebben referentiële stabiliteit: dat wil zeggen dat wanneer informatie wordt opgeslagen op een bepaald adres, het altijd op dat adres beschikbaar blijft.

Gebruik deze archieven zo veel mogelijk en zo opvallend mogelijk. Zelfs wanneer

u het antwoord op een vraag uit uw hoofd weet, als u denkt dat er een referentie in de archieven te vinden is dat het antwoord bevat, neem dan de tijd om dit op te zoeken en te laten zien. Iedere keer dat u dit op een voor iedereen zichtbare manier doet, zullen er weer enkele mensen ontdekken dat de archieven bestaan en dat het zoeken in de archieven productieve resultaten kan opleveren. Tevens versterkt u de heersende norm over het dupliceren van informatie wanneer u refereert aan het archief in plaats van het advies te herschrijven. Waarom zouden de antwoorden op twee verschillende plaatsen beschikbaar moeten zijn? Als het aantal plaatsen waarop het antwoord kan worden gevonden tot een minimum wordt beperkt, is de kans groter dat mensen die het hebben gevonden zullen onthouden waar ze naar moeten zoeken om het opnieuw te vinden. Goedgeplaatste referenties dragen ook bij aan de kwaliteit van de zoekresultaten in het algemeen, omdat ze de positie van de bron in de rangorde van internetzoekmachines versterken.

Er zijn echter momenten waarop het dupliceren van informatie wel zinvol is. Neem bijvoorbeeld een antwoord in het archief, dat niet van u is, waarin staat:

```
Het lijkt erop dat er met uw Scanley-indexen 'gefrobd' is.  Om
dit terug te draaien moet u de volgende stappen volgen:

1. Zet de Scanley-server uit.
2. Start het programma 'defrobnicate' van Scanley.
3. Start de server weer op.
```

Vervolgens ziet u maanden later een andere post waarin iemand aangeeft dat zijn indexen zijn gefrobd. U zoekt in de archieven en vindt het bovenstaande antwoord, maar u realiseert zich dat er enkele stappen ontbreken (misschien per ongeluk of misschien omdat de software is veranderd sinds deze post werd geschreven). De klassieke manier om dit op te lossen is het posten van nieuwe, completere instructies en expliciet aangeven dat de oude post verouderd is door het te vermelden:

```
Het lijkt erop dat er met uw Scanley-indexen 'gefrobd' is.  In
juli zijn we dit probleem al eerder tegengekomen en J. Random
poste een oplossing op http://blahblahblah/blah.  Onderstaand
een completere beschrijving over hoe u uw indexen kunt "ontfrob-
ben", gebaseerd op de instructies van J. Random maar met enkele
uitbreidingen:

1. Zet de Scanley-server uit.
2. Log in als de gebruiker waaronder de Scanley-server normaal
   gesproken actief is.
3. Start als deze gebruiker het programma 'defrobnicate' voor de
   indexen.
4. Start Scanley handmatig op om te zien of de indexen nu wel
   werken.
5. Start de server weer op.
```


(In het ideale geval zou het mogelijk moeten zijn om een notitie toe te voegen aan een oude post, met de vermelding dat er nieuwe informatie beschikbaar is en te verwijzen naar de nieuwe post. Ik ken echter geen archiveringssoftware die het mogelijk maakt om een vermelding 'verouderd door' toe te voegen, misschien omdat het een beetje lastig is dit op zo'n manier te implementeren dat het de integriteit van het archief als woordelijk naslagwerk niet aantast. Dit is nog een extra reden waarom het maken van een speciale webpagina met antwoorden op algemene vragen een goed idee is.)

Archieven worden waarschijnlijk het meest afgezocht naar antwoorden op technische vragen, maar het belang ervan voor het project gaat veel verder. De formele richtlijnen van een project kunnen worden gezien als de officiële regels, de archieven als de gewoonteregels: een overzicht van alle genomen beslissingen en hoe tot deze beslissingen werd gekomen. In iedere terugkerende discussie wordt tegenwoordig als min of meer verplicht gezien om te beginnen met een zoektocht door de archieven. Hierdoor kunt u de discussie beginnen met een samenvatting van de huidige stand van zaken, anticiperen op tegenwerpingen, tegenbewijs verzamelen en misschien invalshoeken ontdekken waaraan u nog niet had gedacht. De andere deelnemers zullen ook van u *verwachten* dat u het archief hebt nagezocht. Zelfs als voorgaande discussies nergens toe geleid hebben, moet u er toch naar verwijzen wanneer u het onderwerp opnieuw aansnijdt, zodat mensen zelf kunnen zien dat a) de discussie nergens toe leidde en b) u uw huiswerk hebt gedaan en u daarom waarschijnlijk nu iets zegt dat nog niet eerder is gezegd.

Behandel alle bronnen als archieven

Alle bovenstaande adviezen hebben betrekking op meer dan alleen de archieven van de mailinglijsten. Het zou een organisatorisch principe van het project moeten zijn om bepaalde stukken informatie beschikbaar te hebben op stabiele en makkelijk vindbare adressen. Laten we de FAQ van het project als voorbeeld nemen.

Hoe gebruiken mensen een FAQ?

1. Ze willen erin zoeken naar specifieke woorden of uitdrukkingen.
2. Ze willen er doorheen kunnen bladeren, om informatie te verzamelen zonder dat ze specifiek op zoek zijn naar bepaalde antwoorden.
3. Ze verwachten dat zoekmachines zoals Google op de hoogte zijn van de inhoud van de FAQ's, zodat zoekacties kunnen resulteren in informatie hieruit.
4 Ze willen anderen direct naar bepaalde items in de FAQ kunnen verwijzen.
5. Ze willen nieuw materiaal kunnen toevoegen aan de FAQ, maar merk op dat dit veel minder vaak gebeurt dan dat er naar antwoorden wordt gezocht. Er wordt veel meer in FAQ's gelezen dan geschreven.

Punt 1 impliceert dat de FAQ beschikbaar moet zijn in tekstformaat. De punten 2 en 3 impliceren dat de FAQ beschikbaar moet zijn als HTML-pagina, waarbij punt 2 bovendien impliceert dat de HTML moet zijn ontworpen voor leesbaarheid (d.w.z. dat u de uitstraling ervan onder controle moet houden) en een inhoudsopgave moet bevatten. Punt 4 betekent dat aan ieder afzonderlijk item in de FAQ een HTML-*'named anchor'* moet worden toegekend, een tag die mensen in staat stelt een be-

paalde plek op de pagina te bereiken. Punt 5 houdt in dat het bronbestand van de FAQ op een praktische manier beschikbaar moet zijn (zie het gedeelte 'Houd alles onder versie' in Hoofdstuk 3, *Technische infrastructuur*), in een formaat dat makkelijk kan worden aangepast.

Named Anchors en ID-attributen
Er zijn twee manieren waarop een browser naar een specifieke plek op een webpagina kan springen: named Anchors en ID-attributen.

Een *named anchor* is een normaal HTML-ankerelement (`<a>...</a>`), maar dan met een 'naam'-attribuut:

```
<a name="mylabel">...</a>
```

Recentere versies van HTML ondersteunen een generiek *id-attribuut*, dat aan ieder HTML-element kan worden toegekend, niet alleen aan `<a>`. Bijvoorbeeld:

```
<p id="mylabel">...</p>
```

Named anchors en id-attributen worden op dezelfde manier gebruikt. Iemand voegt een hekje en het label toe aan een URL zodat de browser direct naar de plek op de pagina springt:

```
http://myproject.example.com/faq.html#mylabel
```

Zo goed als alle browsers ondersteunen named anchors, de meeste moderne browsers ondersteunen ook ID-attributen. Om er zeker van te zijn dat het voor iedereen goed werkt, zou ik aanraden óf alleen named anchors te gebruiken, óf named anchors *en* ID-attributen samen (met natuurlijk hetzelfde label for beide). Named anchors sluiten zichzelf niet. Zelfs als het element geen tekst bevat, moet u het toch in de dubbelzijdige vorm schrijven:

```
<a name="mylabel"></a>
```

... hoewel er natuurlijk normaal gesproken wel tekst in staat, zoals een titel of een sectie.

Of u nu een named anchor gebruikt of een ID-attribuut of beide, houd in gedachten dat het label niet zichtbaar is voor iemand die de plek op de pagina bekijkt zonder een label te gebruiken. Deze persoon wil echter misschien de label van een bepaalde plek wel graag weten, zodat hij bijvoorbeeld de URL van een antwoord in de FAQ naar een vriend kan mailen. Om hem daarbij te helpen voegt u een *title-attribuut* toe aan hetzelfde element waar u het attribuut 'name' en/of 'id' aan hebt toegevoegd, bijvoorbeeld:

```
<a name="mylabel" title="#mylabel">...</a>
```

Wanneer de muiswijzer over de tekst gaat in een element waaraan een titel is toegewezen, laten de meeste browsers een klein boxje zien met de titel. Ik voeg meestal ook een hekje toe, om de gebruiker eraan te herinneren dat hij dit aan het eind van de URL moet zetten om de volgende keer direct naar deze plaats te springen.
Het op deze manier opmaken van de FAQ is slechts één voorbeeld van hoe u een informatiebron toonbaar kunt maken. Dezelfde eigenschappen (directe doorzoekbaarheid, beschikbaarheid voor de grote zoekmachines op internet, kunnen browsen, referentiële stabiliteit en (indien van toepassing) de mogelijkheid voor het doorvoeren van wijzigingen) zijn ook van toepassing op de andere webpagina's, het broncodeschema, de bug tracker enz. Gelukkig wordt het belang van deze eigenschappen al lang door de meeste archiveringssoftware voor mailinglijsten onderkend, zodat de meeste mailinglijsten al over deze eigenschappen beschikken. Voor andere formaten moet degene die de informatie onderhoudt misschien wat meer moeite doen (in Hoofdstuk 8, *Het managen van vrijwilligers* wordt besproken hoe u deze onderhoudstaak over verschillende vrijwilligers kunt verdelen).

Gewoontes voor codificeren
Wanneer het project groeit en complexer wordt, wordt ook de hoeveelheid informatie die iedere nieuwe deelnemer in zich op moet nemen groter. De mensen die al lange tijd bij het project betrokken zijn, konden de gewoontes van het project leren en bedenken tijdens het ontstaan ervan. Ze zijn zich er vaak niet van bewust hoe groot de hoeveelheid gewoontes na verloop van tijd is geworden en kunnen zich verbazen over hoeveel fouten nieuwkomers lijken te maken. Natuurlijk komt dit niet doordat nieuwkomers van een mindere kwaliteit zijn dan voorheen, ze moeten alleen meer moeite doen om zich te voegen in de projectcultuur dan nieuwkomers in het verleden.

De gewoontes die binnen een project worden opgebouwd, hebben minstens evenveel betrekking op hoe er moet worden gecommuniceerd en hoe informatie moet worden opgeslagen als op coderingsnormen en andere technische details. We hebben al gekeken naar de beide soorten normen, in het gedeelte 'Ontwikkelaarsdocumentatie' in Hoofdstuk 2, *Het begin* en het gedeelte 'Alles opschrijven' in Hoofdstuk 4, *Sociale en politieke infrastructuur*, waar ook voorbeelden worden gegeven. Deze sectie gaat over hoe deze richtlijnen tijdens de loop van het project up-to-date moeten worden gehouden, in het bijzonder de richtlijnen over het managen van de communicatie, omdat deze het meest veranderen wanneer de omvang en de complexiteit van een project toenemen.

Let er ten eerste op of er bij de dingen waarover mensen in de war raken patronen te herkennen zijn. Als u steeds weer dezelfde situaties ziet opduiken, met name bij nieuwe deelnemers, is de kans groot dat er een richtlijn moet worden gedocumenteerd die dat nog niet is. Laat u zich ten tweede niet ontmoedigen als u hetzelfde steeds opnieuw moet zeggen en zorg er ook voor dat u niet ontmoedigd *klinkt* doordat u een en ander steeds moet herhalen. U en de andere veteranen van het project zullen zichzelf regelmatig moeten herhalen. Dat is een onvermijdelijk neveneffect van de komst van nieuwe deelnemers.

Iedere webpagina, ieder bericht op een mailinglijst en ieder IRC-kanaal moet worden beschouwd als een reclamemogelijkheid, niet voor commerciële advertenties

maar voor aankondigingen over de eigen informatiebronnen van het project. Waarvoor u deze ruimte gebruikt hangt af van de eigenschappen van de mensen die het waarschijnlijk zullen lezen. Een IRC-kanaal voor vragen van gebruikers wordt bijvoorbeeld waarschijnlijk gebruikt door mensen die nog niet eerder met het project te maken hebben gehad. Vaak zijn dat mensen die de software net hebben geïnstalleerd en een vraag hebben waar ze graag direct antwoord op willen (want als de vraag had kunnen wachten, hadden ze die ook naar een mailinglijst kunnen sturen, wat waarschijnlijk minder tijd had gekost maar waarbij ze langer op antwoord hadden gewacht). Men is vaak niet permanent aanwezig op een IRC-kanaal. Iemand meldt zich aan, stelt zijn vraag en vertrekt weer.

Daarom moet het onderwerp via dit kanaal zijn gericht op mensen die *nu* technische antwoorden zoeken op technische vragen, in tegenstelling tot mensen die bijvoorbeeld voor langere tijd bij het project betrokken kunnen raken en voor wie richtlijnen over de omgangsvormen binnen de gemeenschap misschien handiger zijn. Dit is een voorbeeld van hoe een heel druk IRC-kanaal hiermee omgaat (vergelijk dit met het eerder genoemde voorbeeld in het gedeelte 'IRC-/ realtime-chatsystemen' in Hoofdstuk 3, *Technische infrastructuur*):

```
You are now talking on #linuxhelp

Topic for #linuxhelp is Please READ
http://www.catb.org/~esr/faqs/smart-questions.html &&
http://www.tldp.org/docs.html#howto BEFORE asking questions |
Channel rules are at http://www.nerdfest.org/lh_rules.html |
Please consult http://kerneltrap.org/node/view/799 before as-
king about upgrading to a 2.6.x kernel | memory read possible:
http://tinyurl.com/4s6mc -> update to 2.6.8.1 or 2.4.27 | hash
algo disaster: http://tinyurl.com/6w8rf | reiser4 out
```

Bij mailinglijsten is de 'advertentieruimte' een kleine voettekst die aan ieder bericht wordt toegevoegd. De meeste projecten plaatsen hier instructies over aan- en afmelden en eventueel ook een verwijzing naar de homepagina of de FAQ van het project. U denkt natuurlijk dat iedereen die bij de lijst is aangemeld deze dingen al weet, en dat is waarschijnlijk ook het geval, maar niet alleen subscribers lezen deze mailinglijstberichten. Er kunnen op vele plaatsen links te vinden zijn naar een gearchiveerd bericht en sommige berichten kunnen uiteindelijk zo algemeen bekend worden dat er meer mensen buiten de lijst zijn die het lezen dan van de lijst zelf.

Opnieuw vormgeven kan een groot verschil maken. Binnen het Subversion-project hadden we bijvoorbeeld maar weinig succes met de bugfiltertechniek zoals beschreven in het gedeelte 'Het voorfilteren van de bug tracker' in Hoofdstuk 3, *Technische infrastructuur*. Er werden nog steeds veel onterechte bugrapporten geplaatst door onervaren mensen en iedere keer dat dit gebeurde, moest degene die het rapport had ingediend op precies dezelfde manier worden geïnstrueerd als de vijfhonderd mensen vóór hem. Toen uiteindelijk één van de ontwikkelaars er niet meer tegen kon en uit zijn slof schoot tegen een arme gebruiker die de richtlijnen voor de issue tracker niet goed had gelezen, besloot een andere ontwikkelaar dat het welletjes

was. Hij stelde voor dat we de homepagina van de issue tracker opnieuw zouden vormgeven zodat het belangrijkste deel, de uitdrukkelijke opdracht een bug eerst op de mailinglijst of een IRC-kanaal te bespreken voor deze te melden, duidelijk zichtbaar zou zijn in grote rode letters op een gele achtergrond, centraal geplaatst boven alle andere elementen op de pagina. Dat deden we (het resultaat is te zien op http://subversion.tigris.org/project_issues.html) en het gevolg was een duidelijke daling van het aantal onterechte bugrapporten. Natuurlijk komen ze nog steeds binnen en dat zal altijd wel zo blijven, maar het aantal is beduidend minder, zelfs met een toenemend aantal gebruikers. Het gevolg is niet alleen dat de bugdatabase minder rommel bevat, maar ook dat de mensen die op gemelde issues reageren minder snel geïrriteerd raken en waarschijnlijk vriendelijker blijven in hun reacties op de nu minder voorkomende onterechte rapporten. Dit is een verbetering voor zowel het imago van het project als de geestelijke gezondheid van de vrijwilligers.

De les voor ons was dat het alleen opschrijven van de richtlijnen niet voldoende was. We moesten het ook ergens plaatsen waar het zichtbaar zou zijn voor de mensen die ze het meest nodig zouden hebben, en ze zo vormgeven dat de status ervan als introductieboodschap direct duidelijk is voor mensen die niet bekend zijn met het project.

Statische webpagina's zijn niet de enige plaats waar de gewoontes van het project kunnen worden geadverteerd. Een zekere mate van toezicht (in de vorm van vriendelijke geheugensteuntjes, u hoeft niet direct strenge straffen uit te delen) is ook vereist. Iedere review door collega-programmeurs, zelfs commit-reviews zoals beschreven in het gedeelte 'De code nakijken op verdachte onderdelen' in Hoofdstuk 2, *Aan de slag*, zou een onderdeel moeten bevatten waarin wordt aangegeven of wel of niet aan de normen van het project is voldaan, met name wat betreft de communicatie.

Nog een voorbeeld uit het Subversion-project. We hadden de gewoonte om 'r12908' gebruiken voor 'revisie 12908 in de versiebeheerrepository.' De klein geschreven 'r' als voorvoegsel is makkelijk te typen en omdat hij half zo hoog is als de cijfers wordt het een makkelijk te herkennen blokje tekst in combinatie met cijfers. Natuurlijk betekent het instellen van een gewoonte niet dat iedereen deze vanaf het begin consequent zal toepassen. Wanneer er dus een commit-mail binnenkomt met een logbericht als dit ...

```
------------------------------------------------------------------
r12908 | qsimon | 2005-02-02 14:15:06 -0600 (Wed, 02 Feb 2005) |
4 lines

Patch from J. Random Contributor <jrcontrib@gmail.com>

* trunk/contrib/client-side/psvn/psvn.el:
  Fixed some typos from revision 12828.
------------------------------------------------------------------
```

... moet de review van de commit een onderdeel bevatten met "Zou je trouwens

‘r12828’ en niet ‘revisie 12828’ willen gebruiken wanneer je refereert aan wijzigingen uit het verleden? Bedankt.” Dit is geen geleerddoenerij. Het is belangrijk, zowel voor de automatische ontleedbaarheid als de leesbaarheid.

Door het algemene principe aan te hangen dat er officieel aanvaarde referentiemethoden zijn voor gangbare stukjes informatie en dat deze referentiemethoden overal consistent moeten worden toegepast, kan het project in feite bepaalde normen exporteren. Deze normen stellen mensen in staat tools te schrijven waarmee de berichten van het project op een meer bruikbare manier worden gepresenteerd. Een revisie in de vorm ‘r12828’ kan bijvoorbeeld worden getransformeerd in een directe link naar het zoeksysteem van de repository. Dit is veel moeilijker wanneer de revisie wordt omschreven met ‘revisie 12828’, zowel omdat deze vorm kan worden opgesplitst tussen twee regels, als omdat het minder opvallend is (het woord ‘revisie’ komt vaker losstaand voor en ook groepen cijfers kunnen vaker voorkomen, terwijl de combinatie ‘r12828’ alleen een revisienummer kan zijn). Hetzelfde is van toepassing op issue-nummers, FAQ-items (hint: gebruik een URL met een named anchor, zoals beschreven in Named Anchors en ID-attributen) enz.

Zelfs voor elementen waarvoor geen duidelijke korte en algemeen aanvaarde vorm is, moeten mensen worden aangemoedigd belangrijke stukjes informatie steeds op dezelfde manier weer te geven. Als er bijvoorbeeld wordt gerefereerd aan een bericht van de mailinglijst, noem dan niet alleen de afzender en het onderwerp, maar ook de URL in het archief *en* de bericht-ID. Dit laatste geeft mensen die hun eigen kopie van de mailinglijst bewaren (mensen bewaren soms offline kopieën, bijvoorbeeld voor gebruik op een laptop onderweg) de kans het juiste bericht ondubbelzinnig terug te vinden, zelfs als ze geen toegang hebben tot de archieven. De afzender en het onderwerp zijn niet genoeg, omdat dezelfde persoon meerdere berichten kan versturen binnen dezelfde thread, zelfs op dezelfde dag.

Hoe groter een project wordt, des te belangrijker deze vorm van consistentie wordt. Consistentie houdt in dat waar mensen ook kijken, ze zien dat dezelfde patronen worden aangehouden, zodat ze deze patronen zelf ook kunnen aanhouden. Hierdoor neemt ook het aantal vragen dat ze moeten stellen af. Een miljoen lezers hebben is niet moeilijker dan één lezer. Problemen met de schaalbaarheid ontstaan pas wanneer een bepaald percentage van deze lezers vragen gaat stellen. Wanneer een project groeit, moet het dit percentage daarom verlagen door de beschikbaarheid en de toegankelijkheid van informatie te verbeteren, zodat de kans dat een willekeurig persoon vindt wat hij zoekt zonder te hoeven vragen groter wordt.

6.5 GEEN DISCUSSIES IN DE BUG TRACKER

Bij ieder project dat actief gebruik maakt van de bug tracker bestaat het gevaar dat de tracker zelf een discussieforum wordt, ook al zijn mailinglijsten daarvoor echt geschikter. Meestal begint het onschuldig. Iemand maakt een aantekening bij een issue met bijvoorbeeld een voorgestelde oplossing of een gedeeltelijk patch. Iemand anders ziet dit en realiseert zich dat de oplossing ook weer problemen met zich meebrengt. Hij voegt nog een opmerking toe, waarin hij op deze problemen wijst.

De eerste persoon reageert, weer door iets aan het issue toe te voegen enz.

Het probleem hiervan is ten eerste dat de bug tracker een nogal lompe omgeving is voor een discussie en ten tweede dat andere mensen hier misschien geen aandacht aan besteden. Uiteindelijk verwachten ze dat ontwikkelingsdiscussies op de ontwikkelingsmailinglijst worden gevoerd, en daar kijken ze dan ook. Misschien zijn ze helemaal geen subscriber van de lijst met issuewijzigingen, en zelfs als ze dat wel zijn volgen ze die misschien niet op de voet.

Maar waar ging het in het proces nou precies mis? Was dat op het moment dat de eerste persoon zijn oplossing aan het issue koppelde? Had hij deze in plaats daarvan op de mailinglijst moeten posten? Of was het de tweede persoon die reageerde bij het issue in plaats van op de lijst?

Hierop valt geen duidelijk antwoord te geven. Er is echter een algemeen principe. Als u gegevens toevoegt aan een issue kunt u dat in de tracker doen, maar als u een *discussie* begint, doe dat dan op de mailinglijst. Misschien kunt u niet altijd een duidelijk onderscheid maken, maar gebruik uw eigen beoordelingsvermogen. Als u bijvoorbeeld een patch toevoegt die een mogelijk controversiële oplossing bevat, kunt u verwachten dat mensen er vragen over hebben. Dus zelfs als u normaal gesproken een patch aan het issue zou toevoegen (ervan uitgaande dat u de wijziging niet direct kunt doorvoeren) zou u er in dit geval voor kunnen kiezen een bericht naar de mailinglijst te sturen. In ieder geval is er na verloop van tijd altijd wel iemand kan zien dat er niet meer alleen sprake is van het toevoegen van gegevens maar van een feitelijke discussie. In het voorbeeld aan het begin van deze sectie zou dat de tweede persoon zijn, die zou kunnen voorspellen dat er een discussie ontstaat op het moment dat hij zich realiseerde dat er problemen waren met de patch, en dat dit via een ander medium zou moeten gebeuren.

Om een analogie uit de wiskunde te gebruiken, wanneer informatie er uitziet alsof het snel convergeert, dan kan het in de bug tracker worden geplaatst. Wanneer het er divergerend uitziet is een mailinglijst of IRC-kanaal een betere plek.

Dit betekent niet dat er nooit enige uitwisseling van informatie kan plaatsvinden in de bug tracker. Het vragen om meer informatie over de reproductiemethode aan degene die het oorspronkelijke bericht heeft geplaatst, is bijvoorbeeld een zeer convergerend proces. Het is niet waarschijnlijk dat de reactie van die persoon nieuwe issues zal oproepen. Het zal alleen de reeds geboden informatie verder uitwerken. Het is niet nodig de mailinglijst met dit proces lastig te vallen. U kunt dit op alle mogelijke manieren afhandelen met een aantal reacties in de tracker. Hetzelfde is het geval als u er tamelijk zeker van bent dat een bug onjuist is gerapporteerd (d.w.z. dat het geen bug is). Dan kunt u dit gewoon direct binnen het issue melden. Ook het melden van een klein probleem met een voorgestelde oplossing is prima, zolang dit probleem niet de hele oplossing onderuit haalt.

Aan de andere kant, als u filosofische kwesties oprakelt over de invloed van de bug of de correcte werking van de software kunt u er redelijk zeker van zijn dat andere ontwikkelaars ook willen meepraten. Het is erg waarschijnlijk dat de discussie eerst

divergeert voordat hij weer convergeert. Daarom moet deze op de mailinglijst worden gevoerd.

Maak altijd een link vanuit het issue naar de thread in de mailinglijst als u ervoor kiest een bericht op de mailinglijst te plaatsen. Het blijft belangrijk voor iemand die het issue volgt dat hij ook de discussie kan zien, zelfs wanneer het issue zelf niet het forum is voor de discussie. Dit lijkt misschien wat omslachtig voor degene die de thread begint, maar binnen open source bestaat in principe een cultuur waarbij de schrijver verantwoordelijk is: het is veel belangrijker om dingen makkelijker te maken voor de tientallen of honderden mensen die over de bug lezen dan voor de vier of vijf mensen die erover schrijven.

Het is prima om belangrijke conclusies en samenvattingen van de discussie op de lijst naar het issue te kopiëren, als dit het voor de lezers makkelijker maakt. De meest gebruikte manier van werken is een discussie te starten op de lijst, een link naar de thread bij het issue te plaatsen en dan, wanneer de discussie afgelopen is, de uiteindelijke samenvatting in het issue te plakken (samen met een link naar het bericht waar deze samenvatting uitkomt), zodat iemand die door het issue heen bladert de conclusie snel kan zien zonder nog ergens anders naartoe te hoeven klikken. Merk op dat het gebruikelijke probleem van gegevensduplicaten hier niet bestaat omdat zowel de archieven als de commentaren bij het issue meestal statische, niet te wijzigen gegevens bevat.

6.6 PUBLICITEIT

Binnen open source-software is de grens tussen puur interne discussies en pr-verklaringen erg vaag. Dit komt voor een deel doordat de doelgroep altijd slecht gedefinieerd is: omdat de meeste of alle berichten publiekelijk toegankelijk zijn, heeft het project niet de volledige controle over de indruk die de buitenwereld ervan krijgt. Iemand, bijvoorbeeld een editor van slashdot.org, kan de aandacht van miljoenen lezers naar een bepaalde post trekken waarvan niemand ooit had verwacht dat deze buiten het project gelezen zou worden. Dit is een feit waar alle open source-projecten mee moeten leren leven, hoewel het risico in de praktijk nogal klein is. Over het algemeen zullen de aankondigingen waarvan het project het liefst wil dat ze worden gepubliceerd ook daadwerkelijk worden gepubliceerd, ervan uitgaande dat u de juiste mechanismen gebruikt om iets met nieuwswaarde aan de buitenwereld kenbaar te maken.

Voor belangrijke aankondigingen zijn er vier of vijf algemene distributiekanalen, waarop de aankondigingen zo gelijktijdig mogelijk gedaan zouden moeten worden:

1. De homepagina van uw project wordt waarschijnlijk door meer buitenstaanders gezien dan welk ander deel van het project dan ook. Wanneer u een echt belangrijke aankondiging heeft, plaats daar dan een samenvattende tekst. Deze tekst moet een zeer korte samenvatting zijn die doorlinkt naar het persbericht (zie onder) voor meer informatie.

2. Tegelijkertijd zou u op uw website een gedeelte met ‘Nieuws’ of ‘Persberichten’ moeten hebben, waar de aankondigingen gedetailleerd kunnen worden uitgewerkt. Een deel van het doel van een persbericht is een enkel, algemeen aanvaard ‘aankondigingsobject’ te bieden waar andere websites naartoe kunnen verwijzen. Zorg er dus voor dat het ook op zo’n manier is gestructureerd: óf als één webpagina per persbericht, als een afzonderlijk blogbericht of in een andere vorm waar naartoe kan worden gelinkt en waarbij het bericht afzonderlijk van andere persberichten te lezen is.

3. Wanneer uw project een RSS-feed heeft (zie het gedeelte ‘RSS-feeds’), zorg er dan voor dat de aankondiging ook daarop terecht komt. Dit kan automatisch gebeuren wanneer u het persbericht opstelt, afhankelijk van de instellingen van uw website.

4. Als uw aankondiging betrekking heeft op een nieuwe softwarerelease, wijzig dan ook de gegevens over uw project op http://freshmeat.net/ (zie het gedeelte ‘Aankondigen’ over hoe uw uw project daar moet registreren). Iedere keer dat u uw gegevens bij Freshmeat wijzigt, worden uw gegevens getoond op de wijzigingenlijst van Freshmeat voor die dag. De wijzigingenlijst wordt niet alleen op Freshmeat zelf geüpdatet, maar ook op verschillende webportalen (waaronder http://slashdot.org) die zeer goed in de gaten worden gehouden door hele hordes mensen. Freshmeat biedt dezelfde informatie ook via een RSS-feed, zodat mensen die niet zijn aangemeld bij de RSS-feed van uw project het bericht nog altijd kunnen zien via die van Freshmeat.

5. Stuur een e-mail naar de mailinglijst met aankondigingen van uw project. De naam van de lijst moet ‘announce’ zijn, dus, `announce@yourprojectdomain.org`, omdat dat inmiddels algemeen gebruik is geworden. En uit het handvest van de lijst moet duidelijk worden dat het een lijst met weinig verkeer is, gereserveerd voor belangrijke aankondigingen van het project. De meeste aankondigingen zullen betrekking hebben op nieuwe softwarereleases, maar soms ook op andere dingen, zoals een evenement voor fondsenwerving, beveiligingskwetsbaarheden (zie het gedeelte ‘Beveiligingskwetsbaarheden aankondigen’ later in dit hoofdstuk) of belangrijke wijzigingen in de leiding van het project kunnen hier worden gepost. Omdat er weinig verkeer is en alleen voor belangrijke dingen, heeft de `announce`-lijst meestal het hoogste aantal deelnemers van alle mailinglijsten van het project (dit betekent natuurlijk dat u het niet voor verkeerde doeleinden mag gebruiken. Denk dus goed na voordat u een bericht post). Om te voorkomen dat iedereen aankondigingen gaat sturen of, erger nog, er spam op de lijst terechtkomt, moet de `announce`-lijst altijd worden gemodereerd.

Probeer de aankondigingen op al deze plaatsen tegelijk te doen, of zo dicht bij elkaar als maar mogelijk is. Mensen kunnen in de war raken wanneer ze een aankondiging op de mailinglijst zien maar hier niets van terugzien op de homepagina van het project of bij de persberichten. Als u de verschillende wijzigingen (e-mails, wijzigingen van de website, etc.) in een rij plaatst en ze allemaal tegelijk verstuurt, kunt u de periode waarin de verschillende media niet consistent zijn zo kort mogelijk houden.

Voor minder belangrijke gebeurtenissen kunt u sommige of alle media weglaten. De gebeurtenis wordt door de buitenwereld toch wel opgemerkt in verhouding tot de importantie ervan. Terwijl een nieuwe softwarerelease bijvoorbeeld een belangrijke gebeurtenis is, kan alleen het noemen van een datum voor de volgende release wel enige nieuwswaarde hebben, maar is het bij lange na niet zo belangrijk als de release zelf. Wanneer een releasedatum is vastgesteld, is het de moeite waard dit feit naar de dagelijkse mailinglijst te versturen (niet naar de aankondigingenlijst) en de planning of de projectstatus op de website te updaten, maar dat is alles.

Het kan wel gebeuren dat u de datum ziet opduiken in discussies op andere plaatsen op internet, wanneer er mensen zijn die in het project geïnteresseerd zijn. Mensen die alleen maar meelezen op uw mailinglijsten en niet bijdragen, zijn niet per definitie net zo stil op andere plaatsen. Mond-tot-mondreclame zorgt voor breedschalige distributie. Daar kunt u van uitgaan en u kunt zelfs kleine aankondigingen op zo'n manier opstellen dat ze accuraat worden doorgegeven. Met name posts waarvan u verwacht dat ze zullen worden geciteerd, moeten een onderdeel bevatten dat duidelijk bedoeld is om te worden geciteerd, alsof u een formeel persbericht schrijft. Bijvoorbeeld:

hier een update van de voortgang: we zijn van plan medio augstus 2005 versie 2.0 van Scanley uit te brengen. U kunt altijd kijken op http://www.scanley.org/status.html voor updates. De belangrijkste nieuwe functie is een zoekfunctie voor letterlijke uitdrukkingen .

Andere nieuwe functies zijn: ... Er zullen ook diverse bugs worden opgelost, waaronder: ...

De eerste paragraaf is kort, geeft de twee belangrijkste stukjes informatie (de releasedatum en de belangrijkste nieuwe functie) en een URL voor aanvullende informatie. Als deze paragraaf het enige is dat iemand anders te zien krijgt, doet u het aardig goed. De rest van de e-mail kan verloren gaan zonder dat het belangrijkste deel van de inhoud hieronder te lijden heeft. Natuurlijk zullen sommige mensen een link maken naar de hele e-mail, maar evenzovaak zal slechts een klein deel van het bericht worden aangehaald. Gezien het feit dat dit laatste mogelijk is, kunt u het de ander beter makkelijk maken en uiteindelijk nog een beetje invloed houden op wat er wordt geciteerd.

Aankondigen van beveiligingskwetsbaarheden

Het omgaan met beveiligingskwetsbaarheden wijkt af van de manier waarop met andere soorten bugrapporten wordt omgegaan. Voor open source-software is open en transparant zijn een bijna heilig credo. Iedere stap van het standaardproces waarmee met bugs wordt omgegaan, is zichtbaar voor iedereen die het wil zien: de ontvangst van het eerste rapport, de daaruit volgende discussie en uiteindelijk de fix.

Bugs in de beveiliging zijn anders. Ze kunnen de gegevens van gebruikers en mogelijk hun gehele computer in gevaar brengen. Als dit probleem openlijk zou worden besproken, wordt het bestaan ervan aan de hele wereld bekendgemaakt, dus ook aan iedereen die misbruik zou willen maken van de bug. Zelfs het committen van een fix is een doeltreffende manier om anderen op de hoogte te brengen van het bestaan van de bug (er zijn potentiële aanvallers die de commit-logs van publieke projecten in de gaten houden en zoeken naar wijzigingen die wijzen op beveiligingsproblemen in de code voorafgaande aan de wijziging). De meeste open source-projecten gaan te werk met ongeveer dezelfde reeks stappen om met dit conflict tussen openheid en geheimhouding om te gaan, gebaseerd op de volgende basisrichtlijnen:

1. Praat niet publiekelijk over de bug totdat er een fix beschikbaar is. Stel de fix dan beschikbaar op exact hetzelfde moment dat u de bug bekendmaakt.

2. Kom zo snel mogelijk met de fix, met name als iemand van buiten het project de bug heeft gemeld, omdat u weet dat er minstens één persoon buiten het project is die deze kwetsbaarheid kan uitbuiten.

In de praktijk leiden deze principes tot een redelijk gestandaardiseerd stappenplan, dat in de onderstaande secties wordt behandeld.

Het rapport ontvangen

Natuurlijk moet een project in staat zijn van iedereen rapporten over beveiligingsbugs te ontvangen. Het normale adres voor het rapporteren van bugs is hier echter niet geschikt voor, omdat iedereen die kan zien. Zorg daarom voor een afzonderlijke mailinglijst voor het ontvangen van rapporten voor beveiligingsbugs. De archieven van deze mailinglijst mogen niet publiekelijk toegankelijk zijn en de deelname moet onder strenge controle staan. Alleen betrouwbare ontwikkelaars die al lang bij het project zijn betrokken, mogen deelnemen aan de lijst. Als u een meer formele omschrijving nodig hebt van 'betrouwbaar', kunt u het hebben over 'iedereen die al twee jaar of meer commit access heeft' of iets in die trant, om vriendjespolitiek te vermijden. Dit is de groep die de bugs in de beveiliging zal afhandelen.

In het ideale geval is de beveiligingslijst niet beveiligd tegen spam en niet gemodereerd, omdat u niet wilt dat een belangrijk rapport wordt uitgefilterd of vertraagd, alleen omdat geen van de moderatoren online is tijdens het weekend. Wanneer u wel automatische spambeveiligingssoftware gebruikt, probeer het dan met hoge tolerantie te configureren. Het is beter een paar spamberichten binnen te krijgen dan een rapport te missen. Om te zorgen dat de lijst werkt, moet u het adres ervan natuurlijk bekendmaken, maar gezien het feit dat deze niet gemodereerd wordt en op zijn hoogst slechts minimaal tegen spam is beveiligd, moet u proberen het adres altijd zo te transformeren dat het niet zichtbaar is, zoals beschreven in het gedeelte 'Adressen verbergen in archieven' in Hoofdstuk 3, *Technische infrastructuur*. Gelukkig hoeft het verbergen van het adres niet te betekenen dat het niet leesbaar is. Zie ook http://subversion.tigris.org/security.html en bekijk bijvoorbeeld de HTML-broncode.

De fix in stilte ontwikkelen

Wat moet de beveiligingslijst doen wanneer deze een rapport ontvangt? Het eerste dat er gedaan moet worden, is het evalueren van de ernst en de urgentie van het probleem:

1. Hoe ernstig is de kwetsbaarheid? Geeft het een kwaadwillende aanvaller gelegenheid de computer van iemand anders die de software gebruikt over te nemen? Of is het probleem alleen dat er informatie uitlekt over de omvang van enkele van hun bestanden?

2. Hoe makkelijk kan de kwetsbaarheid worden uitgebuit? Kan een aanval worden omgezet in een script of is er kennis nodig over de omstandigheden, gefundeerde gissingen en puur geluk?

3. *Wie* heeft het probleem gerapporteerd? Het antwoord op deze vraag verandert natuurlijk niets aan de aard van de kwetsbaarheid, maar het geeft u een idee over hoeveel mensen er van af kunnen weten. Als het rapport komt van één van de eigen ontwikkelaars van het project kunt u een beetje opgelucht ademhalen (maar alleen een klein beetje), omdat u er op kunt vertrouwen dat deze niemand anders over het probleem heeft verteld. Aan de andere kant, als de e-mail van `anonymous14@globalhackerz.net` kwam, dan kunt u beter zo snel mogelijk reageren. Deze persoon heeft u een gunst bewezen door u erover te informeren, maar u hebt geen idee hoeveel mensen hij erover heeft verteld of hoelang hij zal wachten voordat hij de kwetsbaarheid gaat uitbuiten op in gebruik zijnde systemen.

Merk op dat het verschil waar we het hier over hebben vaak een heel dun lijntje is tussen *urgent* en *extreem urgent*. Zelfs als het rapport van een bekende en goedwillende bron afkomstig is, dan nog kunnen er anderen zijn op internet die de bug lang geleden al hebben ontdekt en dit alleen niet hebben gerapporteerd. Het probleem is alleen niet urgent als de bug de beveiliging niet ernstig in gevaar brengt.

Het voorbeeld "`anonymous14@globalhackerz.net`" is trouwens niet grappig bedoeld. U kunt werkelijk bugrapporten krijgen van mensen die hun identiteit geheim houden en van wie u uit hun woorden en gedrag niet goed kunt opmaken of ze wel of niet aan uw kant staan. Het maakt in feite niet uit: als ze het gat in de beveiliging aan u rapporteren, zullen ze het gevoel hebben dat ze een goede daad hebben gedaan en moet u dienovereenkomstig reageren. Bedank de persoon voor het rapport, geef hem een datum waarop u uiterlijk van plan bent een fix te publiceren en houd hem op de hoogte. Soms geeft de rapporteur *u* een datum. Dat is dus een impliciet dreigement om de bug op een bepaalde datum te publiceren, of u er nu klaar voor bent of niet. Dit kan overkomen als een machtsspelletje van een kwelgeest, maar de kans is groot dat het een preventieve actie is, volgend op een teleurstelling over niet reagerende softwareproducenten die een beveiligingsrapport niet serieus genoeg namen. Maar hoe het ook zit, u kunt het zich niet veroorloven deze persoon een veeg uit de pan te geven. Als de bug ernstig is, heeft hij uiteindelijk de kennis in huis om bij uw gebruikers grote problemen te veroorzaken. Behandel dergelijke rapporteurs goed en hoop dat ze u ook goed zullen behandelen.

Een andere veel voorkomende rapporteur over beveiligingsbugs is de beveiligingsprofessional, iemand die zijn geld verdient met het controleren van code en op de hoogte is van de laatste nieuwtjes over softwarekwetsbaarheden. Deze mensen hebben vaak ervaring met beide kanten van het verhaal. Ze hebben zowel rapporten verzonden als ontvangen, en waarschijnlijk vaker dan de meeste ontwikkelaars in uw project. Ook zij geven vaak een deadline voor het oplossen van de kwetsbaarheid voordat ze de bug bekend zullen maken. De deadline kan onderhandelbaar zijn, maar dat is aan de rapporteur. Beveiligingsprofessionals zijn het er over eens dat deadlines ongeveer de enige betrouwbare manier zijn om ervoor te zorgen dat organisaties direct reageren op beveiligingsproblemen. Behandel de deadline dus niet als onbeleefd. Het gaat om een reeds lang bestaande traditie waar een goede redenen voor zijn.

Zodra u weet hoe ernstig en urgent het probleem is, kunt u aan de oplossing gaan werken. Soms moet een compromis worden gesloten tussen een elegante en een snelle oplossing. Daarom moet u het eens worden over de urgentie voordat u van start gaat. Beperk de discussies over de fix uiteraard tot de leden van de beveiligingslijst en de oorspronkelijke rapporteur (als hij op de hoogte gehouden wil worden) en andere ontwikkelaars die er om technische redenen bij betrokken moeten worden.

Voeg de fix niet toe aan de repository. Houd de patch-vorm aan tot het moment van publicatie Als u een commit invoert kan iemand zelfs uit een onschuldig ogend logbericht begrijpen waar de verandering over gaat. U weet maar nooit wie er meekijkt in uw database en wie er geïnteresseerd kan zijn. Het uitzetten van de commit-e-mails helpt ook niet. Ten eerste is een hiaat in de reeks commit-mails op zich al verdacht, en bovendien blijven de gegevens toch nog beschikbaar in de repository. Ontwikkel de hele fix in de vorm van een patch en bewaar de patch op een veilige plek, misschien in een afzonderlijke en persoonlijke database die alleen bekend is bij mensen die al op de hoogte waren van de bug. (Als u een gedecentraliseerd versiebeheersysteem zoals Arch of SVK gebruikt, kunt u het werk doen onder volledige versiebeheer en hoeft u de database alleen maar ontoegankelijk te houden voor buitenstaanders.)

CAN/CVE-nummers

Mogelijk hebt u een *CAN-nummer* of een *CVE-nummer* gezien dat geassocieerd wordt met beveiligingsproblemen. Deze nummers kunnen er bijvoorbeeld zo uit zien: 'CAN-2004-0397" of "CVE-2002-0092'.

Beide soorten nummers staan voor dezelfde soort entiteit: een post op de lijst met 'Common Vulnaribilities and Exposures (algemene kwetsbaarheden en blootstellingen)' die wordt bijgehouden op http://cve.mitre.org/. De bedoeling van de lijst is het bieden van gestandaardiseerde namen voor alle bekende beveiligingsproblemen, zodat iedereen een unieke en algemeen aanvaarde term kan gebruiken bij de bespreking ervan, en een centrale plek waar men terecht kan voor informatie. Het enige verschil tussen een 'CAN'-nummer en een 'CVE'-nummer is dat de eerste staat voor een mogelijk entry die nog niet is goedgekeurd voor opname op de officiële lijst bij het CVE-bestuur. Het tweede nummer staat voor een goedgekeurde entry. Beide soorten entry's zijn echter zichtbaar voor het publiek, en een entry-nummer verandert niet als het wordt goedgekeurd. In dat geval wordt het voorvoegsel 'CAN' gewoon vervangen door 'CVE'.

Een CAN/CVE-entry bevat zelf geen volledige beschrijving van de bug en hoe u zichzelf ertegen kunt beschermen. In plaats daarvan geeft het een korte samenvatting en een lijst met referenties naar externe informatiebronnen (zoals mailinglijstarchieven) waar mensen kunnen kijken voor meer gedetailleerde informatie. Het werkelijk doel van http://cve.mitre.org/ is een goed georganiseerde plek te bieden waar iedere kwetsbaarheid een naam kan krijgen en een duidelijke aanwijzing naar meer informatie. Zie http://cve.mitre.org/cgi-bin/cvename.cgi?name=2002-0092 voor een voorbeeld van een entry. Merk op dat de referenties heel erg beknopt kunnen zijn en de informatiebronnen vermeld zijn in de vorm van cryptische afkortingen. Een verklaring van de afkortingen kan worden gevonden op http://cve.mitre.org/data/refs/refkey.html.

Als uw kwetsbaarheid voldoet aan de CVE-criteria kunt u een CAN-nummer krijgen. De procedure daarvoor is met opzet moeilijk gemaakt. Het komt er in feite op neer dat u iemand moet kennen, of iemand moet kennen die weer iemand kent. Dat is niet zo gek als het misschien lijkt. Om te voorkomen dat het CVE-bestuur wordt overladen met onechte of slecht geschreven voorstellen, accepteren ze alleen voorstellen van mensen die ze kennen en vertrouwen. Om er dus voor te zorgen dat uw kwetsbaarheid wordt opgenomen, moet u dus een lijntje van kennissen zien te vinden tussen uw project en het CVE-bestuur. Doe eens navraag bij uw ontwikkelaars. Eén van hen kent vast wel iemand die óf zelf een CAN-proces heeft doorlopen óf die iemand kent die dat heeft gedaan enz. Het grote voordeel hiervan is ook dat ergens in de keten iemand kan zitten die genoeg van de materie afweet om u te vertellen dat a) het probleem niet wordt aangemerkt als kwetsbaarheid of blootstelling volgens de MITRE-criteria en het dus geen zin heeft een voorstel in te dienen of b) de kwetsbaarheid al een CAN- of CVE-nummer heeft. Dit laatste kan het geval zijn wanneer een bug al eerder is gepubliceerd op een andere beveiligingslijst, bijvoorbeeld op http://www.cert.org/ of op de BugTraq-mailinglijst op http://www.securityfocus.com/. (Als dit is gebeurd zonder dat uw project hierover gehoord heeft, moet u zich wel zorgen gaan maken over wat er allemaal nog meer aan de hand kan zijn waar u niets vanaf weet.)

Als u wel een CAN/CVE-nummer kunt krijgen, wilt u dat normaal gesproken zo vroeg mogelijk in het proces van onderzoek naar de bug hebben, zodat alle toekomstige communicatie naar dat nummer kan verwijzen. CAN-entry's zijn geblokkeerd tot de datum van publicatie. De entry is zichtbaar als lege tijdelijke aanduiding (zodat u de naam niet kwijtraakt), maar laat geen informatie zien over de kwetsbaarheid tot het moment dat u de bug en de fix bekendmaakt.

Meer informatie over de CAN/CVE-procedure kunt u vinden op http://cve.mitre.org/about/candidates.html. Een bijzonder heldere uiteenzetting over het gebruik van CAN/CVE-nummers in een project kunt u vinden op http://www.debian.org/security/cve-compatibility.

Vooraankondiging

Zodra uw beveiligingsteam (dat zijn de ontwikkelaars op de beveiligingslijst of degenen die u bij het team hebt gehaald om een bepaald probleem op te lossen) een fix klaar heeft, moet u beslissen hoe u deze verspreidt.

Als u de fix gewoon aan uw repository toevoegt of deze op een andere manier aan de hele wereld bekendmaakt, dwingt u iedereen die uw software gebruikt om onmiddellijk te upgraden, anders lopen ze het gevaar te worden gehackt. Daarom kan het soms wenselijk zijn bij bepaalde belangrijke gebruikers een *vooraankondiging* te doen. Dit is met name van belang voor client/serversoftware, in het geval van bekende servers die een aantrekkelijk slachtoffer kunnen zijn voor aanvallers. De beheerders van deze servers hebben graag een dag of twee extra voor hun upgrade, zodat ze al beschermd zijn op het moment dat het probleem wereldkundig wordt gemaakt.

Een vooraankondiging betekent dat er e-mails worden verzonden naar deze beheerders voorafgaande aan de bekendmakingsdatum, waarin hun wordt verteld over de kwetsbaarheid en de oplossing daarvoor. U zou alleen een vooraankondiging moeten sturen naar mensen bij wie u er op kunt vertrouwen dat zij discreet met de informatie om zullen gaan. Dat wil zeggen dat de ontvanger van de vooraankondiging aan twee voorwaarden moet voldoen: de ontvanger moet een grote, belangrijke server beheren waarvoor een bedreiging een serieus probleem zou zijn *en* de ontvanger moet erom bekendstaan dat hij zijn mond vóór de bekendmakingsdatum niet voorbijpraat over het beveiligingsprobleem.

Stuur iedere vooraankondiging afzonderlijk (één per keer) naar iedere ontvanger. Stuur het bericht *niet* naar alle ontvangers in één keer, omdat ze elkaars namen dan kunnen zien, wat betekent dat u in principe alle ontvangers vertelt dat iedere *andere* ontvanger een gat in de beveiliging van zijn server zou kunnen hebben. Versturen via een blinde bcc is ook geen goed idee, omdat sommige beheerders hun inboxen beschermen met spamfilters die met bcc verstuurde e-mails blokkeren of een lagere prioriteit toekennen, omdat er tegenwoordig veel spam met bcc wordt verstuurd.

Hier is een eenvoudig voorbeeld van een vooraankondigingsmail:

```
Van: Uw naam
Aan: admin@large-famous-server.com
Reply-to: Uw naam (niet het adres van de beveiligingslijst) Onderwerp: Vertrouwelijke aankondiging kwetsbaarheid Scanley.

Deze e-mail is een vertrouwelijke vooraankondiging van een beveiligingswaarschuwing in de Scanley-server.

Stuur dit bericht of delen ervan *niet door* naar anderen. De publieke bekendmaking is pas op 19 mei en we willen de informatie graag tot die datum geheim houden.

Wij sturen u deze e-mail omdat u (volgens onze informatie) een Scanley-server heeft en dat u de patch zou willen installeren voordat het gat in de beveiliging op 19 mei wordt bekendgemaakt.
```

```
Referenties:
============
CAN-2004-1771: Scanley stack overflow in queries

Kwetsbaarheid:
==============
Er kunnen willekeurige commando's vanaf de server worden gestart
wanneer de locale van de server niet correct is geconfigureerd en
de client een misvormde query verstuurt.

Ernst:
======
Zeer ernstig, er kan willekeurige code worden uitgevoerd op de
server.

Tijdelijke oplossing:
====================
Stel de instelling voor 'natural-language-processing' op 'off'
in scanley.conf, deze kwetsbaarheid wordt hiermee opgeheven.

Patch:
======
De onderstaande patch heeft betrekking op Scanley 3.0, 3.1 en
3.2.

Er komt een nieuwe publieke release uit (Scanley 3.2.1) op of
kort voor 19 mei, zodat deze beschikbaar is op hetzelfde moment
dat deze kwetsbaarheid bekend wordt gemaakt.  U kunt de patch nu
uitvoeren of wachten op de publieke release.  Het enige verschil
tussen 3.2 en 3.2.1 is deze patch.

[...voeg de patch hier in...]
```

Als u een CAN-nummer heeft, voeg dit dan aan de vooraankondiging toe (zoals hierboven getoond), zelfs als de informatie nog steeds geblokkeerd is en de MITRE-pagina niet zichtbaar zal zijn. Door het CAN-nummer in te voegen weet de ontvanger zeker dat de bug waarover de vooraankondiging heeft ontvangen dezelfde is als de bug waarover hij later via de publieke kanalen te horen krijgt, zodat hij zich geen zorgen hoeft maken of hij nog meer actie moet ondernemen, wat precies de bedoeling is van CAN/CVE-nummers.

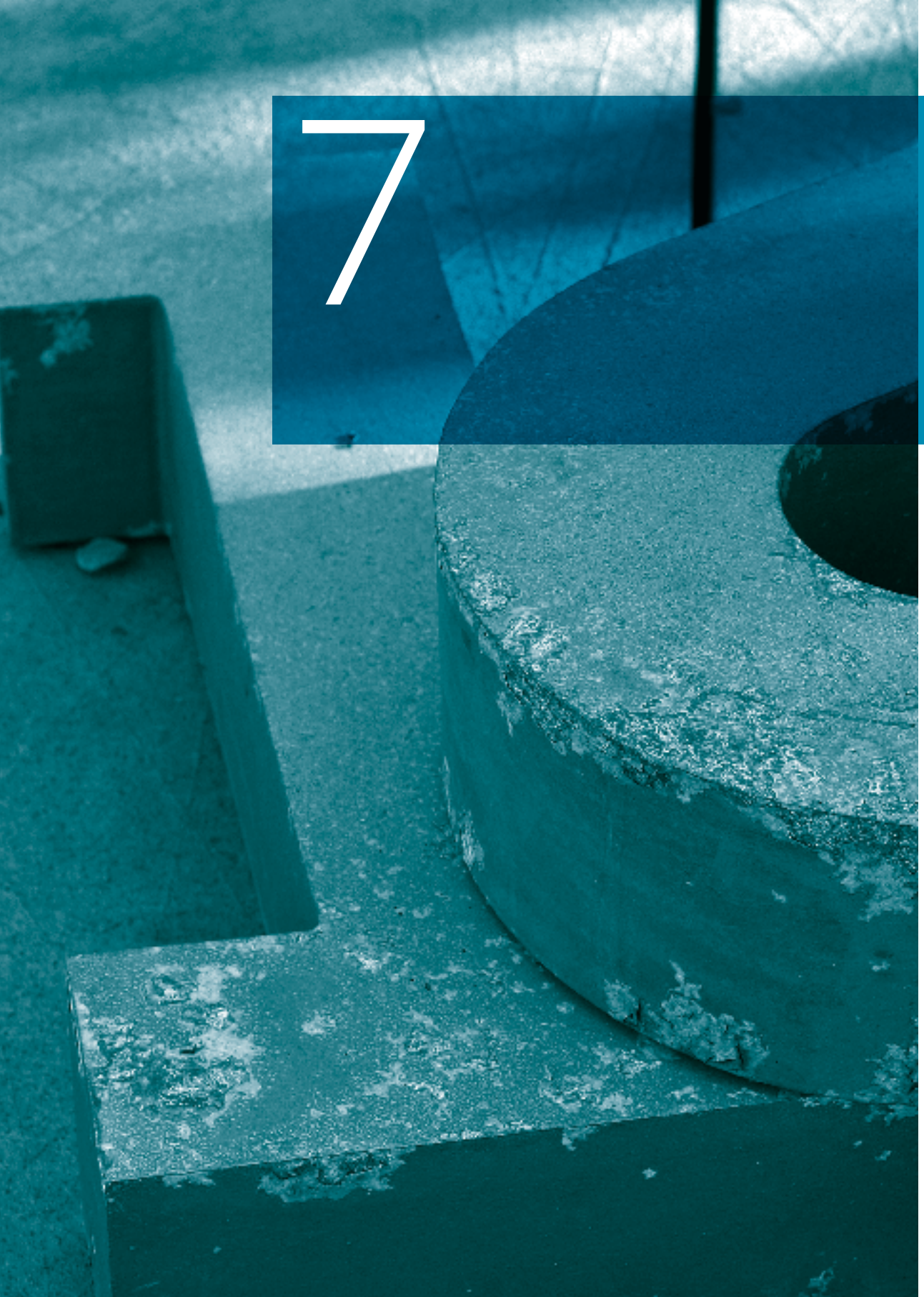
De fix publiceren

De laatste stap in het oplossen van een beveiligingsbug is het distribueren van de fix. U zou het probleem in één uitgebreide aankondiging moeten beschrijven, eventueel het CAN/CVE-nummer geven, aangeven hoe de bug tijdelijk kan worden omzeild en hoe deze permanent moet worden opgelost. Meestal betekent 'oplossen' het upgraden naar een nieuwe versie van de software, hoewel het soms ook het uitvoeren van een patch kan inhouden, met name wanneer de software normaal gesproken al in source-vorm loopt. Als u wel een nieuwe release uitbrengt, moet deze alleen met deze patch afwijken van de vorige release. Daardoor kunnen meer behoudende beheerders upgraden zonder zich zorgen te hoeven maken over wat er nog meer verandert. Ze hoeven zich ook niet druk te maken over toekomstige upgrades, omdat de beveiligingspatch per definitie in alle toekomstige releases is opgenomen. (De details van releaseprocedures worden verder besproken in het gedeelte 'Beveiligingsreleases' in Hoofdstuk 7, *Het maken van downloadpakketten, releases en dagelijkse ontwikkeling*.)

Of u de publieke fix nu wel of niet uitbrengt in een nieuwe release, doe de aankondiging met ongeveer dezelfde prioriteit als waarmee u een nieuwe release zou uitbrengen: stuur een e-mail naar de `announce`-lijst van het project, maak een nieuw persbericht, update de entry bij Freshmeat enz. Hoewel u het bestaan van een beveiligingsbug nooit mag marginaliseren om de reputatie van het project niet te schaden, kunt u de toon en het belang van een beveiligingsaankondiging wel zo treffen dat deze overeenkomt met de feitelijke ernst van het probleem. Als het gat in de beveiliging slechts een minimaal probleem is voor het lekken van informatie en geen groot probleem waarbij de hele computer van de gebruiker kan worden overgenomen, dan hoeft er ook niet veel ophef over te worden gemaakt. U kunt zelfs beslissen de `announce`-lijst hier helemaal niet mee lastig te vallen. Als het project namelijk steeds vals alarm slaat, kunnen gebruikers uiteindelijk gaan denken dat de software minder veilig is dan werkelijk het geval is en zullen ze u misschien ook niet meer geloven wanneer er wel een groot probleem wordt aangekondigd. Zie http://cve.mitre.org/about/terminology.html voor een goede inleiding over het inschatten van de ernst van een probleem.

Als u niet weet hoe u met een beveiligingsprobleem om moet gaan, kunt u over het algemeen het beste met iemand praten die er ervaring mee heeft. Het beoordelen en oplossen van kwetsbaarheden is in grote lijnen een vaardigheid die u kunt aanleren en u kunt de eerste paar keer makkelijk fouten maken.

22| Er is interessant wetenschappelijk onderzoek gedaan over dit onderwerp. Zie bijvoorbeeld *Group Awareness in Distributed Software Development* van Gutwin, Penner en Schneider. Dit rapport was een tijdje online beschikbaar, vervolgens een poosje niet, en nu weer wel op http://www.st.cs.uni-sb.de/edu/empirical-se/2006/PDFs/gutwin04.pdf. U kunt deze link dus proberen, maar misschien moet u een zoekmachine gebruiken als de locatie weer veranderd is.



7

DOWNLOADPAKKETTEN MAKEN, RELEASES EN DAGELIJKSE ONTWIKKELING

Dit hoofdstuk gaat over hoe open source-softwareprojecten downloadpakketten maken en de software uitbrengen en over de manier waarop algemene ontwikkelingspatronen rond deze doelstellingen moeten worden georganiseerd.

Een belangrijk verschil tussen open source-projecten en propriëtaire projecten is het ontbreken van een gecentraliseerde controle over het ontwikkelingsteam. Dit verschil is met name van belang bij het voorbereiden van een nieuwe release. Een bedrijf kan het gehele team van ontwikkelaars vragen zich te concentreren op de komende release en nieuwe toekomstige ontwikkelingen en het verhelpen van niet-kritische bugs op een laag pitje te zetten totdat de release is uitgebracht. Groepen vrijwilligers zijn niet zo homogeen. Mensen werken om uiteenlopende redenen aan een project en mensen die niet geïnteresseerd zijn in een bepaalde release willen liever doorgaan met hun normale ontwikkelingswerk terwijl de release wordt voorbereid en uitgebracht. Omdat de ontwikkeling niet wordt onderbroken, duren releaseprocessen van open source-projecten meestal langer, maar ontwrichten het proces minder dan commerciële releaseprocessen. Het lijkt een beetje op een reparatie aan een snelweg. Er zijn twee manieren om een weg te repareren: je kunt de weg volledig afsluiten, zodat de werklui zich volledig en met de volle capaciteit kunnen inzetten totdat het probleem is opgelost, of je kunt aan een paar rijbanen tegelijk werken terwijl de andere voor verkeer open blijven. De eerste manier is het meest efficiënt *voor de werklui*, maar niet voor de weggebruikers. De weg wordt volledig afgesloten totdat de klus is geklaard. De tweede manier kost veel meer tijd en moeite van de werklui (ze moeten met minder mensen en minder apparatuur werken, op een beperkte ruimte en met mensen die met vlaggen het verkeer moeten waarschuwen enz.), maar de weg blijft bruikbaar, hoewel niet volledig.

Open source-projecten werken meestal op de tweede manier. In feite bevindt het project, wanneer er sprake is van volledig ontwikkelde software met verschillende releaselijnen die tegelijkertijd worden onderhouden, zich permanent in een staat van kleinschalige wegwerkzaamheden. Er zijn altijd wel een paar rijbanen afgesloten en de groep ontwikkelaars tolereert aanhoudende maar wel minimale ongemakken, zodat de releases volgens een regulier programma kunnen worden uitgebracht.

Het model dat dit mogelijk maakt, zorgt ervoor dat er meer dan één release tegelijk kan worden uitgebracht. Het parallel laten lopen van taken die niet van elkaar afhankelijk zijn, is natuurlijk in geen geval uniek voor de ontwikkeling van open source-software, maar wel een principe dat open source-projecten op een geheel eigen wijze implementeren. Ze kunnen het zich niet veroorloven de werklui én het normale verkeer te veel te hinderen, maar ze kunnen het zich ook niet veroorloven mensen speciaal toe te wijzen aan het zwaaien met vlaggetjes om het verkeer te waarschuwen. Daarom moeten ze terugvallen op processen met een laag en constant niveau van administratieve rompslomp, in plaats van één met pieken en dalen. Vrijwilligers zijn over het algemeen bereid te werken met een kleine maar consistente hoeveelheid ongemak, waarvan de voorspelbaarheid hun de mogelijkheid geeft om in hun eigen tempo te werken zonder dat ze zich zorgen hoeven maken dat hun agenda botst met wat er binnen het project gebeurt. Als het project echter afhankelijk zou zijn van een overkoepelend programma waarbij sommige activiteiten andere activiteiten uitsluiten, zou het gevolg zijn dat veel ontwikkelaars een groot deel van de tijd niets kunnen doen. Dat is niet alleen weinig efficiënt maar ook nog eens saai en daarom riskant, omdat de kans groot is dat een verveelde ontwikkelaar straks een ex-ontwikkelaar is.

Releasewerk is vaak de meest in het oog springende niet-ontwikkelingstaak die parallel aan het ontwikkelen plaatsvindt. De in de volgende gedeelten beschreven methodes zijn dan ook grotendeels gericht op het mogelijk maken van releases. Merk echter op dat ze ook van toepassing zijn op andere taken die naast elkaar kunnen lopen, zoals vertalingen en internationalisering, breedschalige API-veranderingen die stapsgewijs door het hele codebestand worden doorgevoerd enz.

7.1 RELEASENUMMERS

Voordat we het gaan hebben over hoe een release moet worden gemaakt, kijken we eerst naar hoe we releases een naam moeten geven. Daarvoor moeten we weten wat releases in de praktijk voor gebruikers betekenen. Een release betekent dat:
oude bugs zijn opgelost. Dit is waarschijnlijk het enige waar gebruikers zeker van kunnen zijn bij iedere release;

- er nieuwe bugs zijn bijgekomen. Ook dit is meestal zeker, behalve soms in het geval van beveiligingsreleases en andere eenmalige releases (zie het gedeelte ‘Beveiligingsreleases’ verderop in dit hoofdstuk);
- er nieuwe functies zijn toegevoegd;
- er nieuwe configuratieopties kunnen zijn toegevoegd of de betekenis van oude opties iets kunnen zijn veranderd. De installatieprocedure kan ook iets zijn veranderd ten opzichte van de laatste release, hoewel iedereen altijd hoopt dat dit niet het geval is;
- er veranderingen kunnen zijn doorgevoerd waardoor nieuwe versies niet meer compatibel zijn met oudere versies, zodat gegevensformaten die worden gebruikt door oudere versies van de software niet langer gebruikt kunnen worden zonder één of andere vorm van (mogelijk handmatige) conversie.

Zoals u kunt zien zijn het niet alleen goede dingen die veranderen. Daarom benaderen ervaren gebruikers nieuwe releases altijd met enige terughoudendheid, met name wanneer de software al uitontwikkeld is en in grote lijnen deed wat zij wilden (of dachten dat zij wilden). Zelfs de komst van nieuwe functies kan een twijfelachtig voordeel zijn, omdat de software onverwachte dingen kan gaan doen.

De bedoeling van releasenummers is daarom tweeledig. Uiteraard laten de nummers ondubbelzinnig de volgorde van de releases zien. Dat wil zeggen dat men aan twee releasenummers direct kan zien welke de meest recente is. Ze kunnen echter ook in een zo compact mogelijke vorm de mate en aard van de veranderingen in een release aangeven.

En dat allemaal met één nummer? Nou ja, min of meer. De discussie over de methodiek van de releasenummering is zo'n beetje de moeder van alle triviale discussies (zie het gedeelte ‘Hoe makkelijker het onderwerp, des te langer de discussie’ in Hoofdstuk 6, *Communicatie*) en het valt dan ook niet te verwachten dat er in de nabije toekomst één complete, mondiale norm zal komen. Er bestaan echter enkele goede methodes, alsmede één principe waar iedereen het over eens is: *wees consistent*. Kies een nummeringssysteem, documenteer het en wijk er niet van af. Uw gebruikers zullen u er dankbaar voor zijn.

Componenten van releasenummers

Dit gedeelte beschrijft in detail de formele conventies van releasenummering. Hiervoor is zeer weinig voorkennis nodig. Het is vooral als referentiemateriaal bedoeld. Als u reeds op de hoogte bent van deze afspraken, kunt u dit gedeelte overslaan.

Releasenummers zijn groepen getallen gescheiden door punten:

Scanley 2.3
Singer 5.11.4
... enz. De punten zijn *geen* decimale scheidingstekens, ze zijn alleen bedoeld als separator. ‘5.3.9’ wordt opgevolgd door ‘5.3.10’. Een paar projecten hebben het soms iets anders aangepakt. Het beroemdste voorbeeld daarvan is de Linux-kernel met de nummering ‘0.95’, ‘0.96’ ... ‘0.99’ die uiteindelijke leidde naar Linux 1.0. De algemene afspraak dat de punten geen decimale scheidingstekens zijn, is inmiddels echter stevig verankerd en moet als norm worden beschouwd. Er is geen limiet aan het aantal componenten (groepjes cijfers die geen punt bevatten), maar de meeste projecten gebruiken er niet meer dan drie of vier. De redenen daarvoor worden later duidelijk.

Naast de numerieke componenten voegen projecten soms ook een beschrijvend label toe, zoals ‘alfa’ of ‘bèta’ (zie alfa en bèta), bijvoorbeeld:

Scanley 2.3.0 (Alpha)
Singer 5.11.4 (Beta)

Een alfa- of bèta-achtervoegsel betekent dat deze release *voorafgaat* aan een toekomstige release met hetzelfde nummer, maar dan zonder het achtervoegsel.

‘2.3.0 (alfa)’ leidt dus uiteindelijk naar ‘2.3.0’. Om verschillende van deze kandidaat-releases achter elkaar te kunnen vrijgeven, kunnen deze achtervoegsels zelf zijn voorzien van meta-achtervoegsels. Hier is bijvoorbeeld een reeks releases in de volgorde waarmee ze beschikbaar zouden worden gesteld aan het publiek:

Scanley 2.3.0 (Alpha 1)
Scanley 2.3.0 (Alpha 2)
Scanley 2.3.0 (Beta 1)
Scanley 2.3.0 (Beta 2)
Scanley 2.3.0 (Beta 3)
Scanley 2.3.0

Merk op dat wanneer er een ‘alfa’-achtervoegsel is, Scanley ‘2.3’ wordt geschreven als ‘2.3.0’. De twee nummers zijn gelijkwaardig. Componenten met alleen nullen kunnen altijd worden weggelaten om de naam korter te maken. Wanneer er echter een achtervoegsel is, is bondigheid sowieso niet van het grootste belang en kan men net zo goed kiezen voor volledigheid.

Andere achtervoegsels die semiregulier zijn, zijn onder andere ‘Stable’, ‘Unstable’, ‘Development’ en ‘RC’ (voor ‘Release Candidate’). De meest gebruikte zijn nog steeds ‘alfa’ en ‘bèta’, met ‘RC’ vlak daarachter als goede derde. NB: ‘RC’ heeft altijd een meta-achtervoegsel. Dat wil zeggen dat u nooit release ‘Scanley 2.3.0 (RC)’ uitgeeft, maar wel ‘Scanley 2.3.0 (RC 1)’, gevolgd door RC 2 enz.

Deze drie labels, ‘alfa’, ‘bèta’ en ‘RC’, zijn inmiddels vrij algemeen bekend en ik adviseer geen andere te gebruiken, zelfs wanneer andere achtervoegsel op het eerste gezicht betere opties lijken omdat het gewone woorden zijn en geen jargon. Mensen die releasesoftware installeren zijn echter reeds bekend met deze grote drie en er is geen reden dingen nodeloos anders te doen dan anderen.

Hoewel de punten in releasenummers geen decimale punten zijn, geven ze wel het belang aan van een cijfer door de plaats waar het staat. Alle releases ‘0.X.Y’ gaan vooraf aan ‘1.0’ (wat uiteraard gelijk staat aan ‘1.0.0’). ‘3.14.158’ gaat direct vooraf aan ‘3.14.159’ en gaat ook vooraf, maar niet direct, aan ‘3.14.160’, evenals ‘3.15. watdanook’ enz.

Een consistent beleid ten aanzien van releasenummers geeft de gebruiker de kans aan de hand van twee releasenummers van de software de belangrijkste verschillende tussen de twee releases te herkennen. In een doorsneesysteem met drie componenten is de eerste component het *hoofdnummer*, het tweede het *subnummer* en het derde het *micronummer*. Release ‘2.10.17’ is bijvoorbeeld de zeventiende microrelease uit de tiende subreleasereeks binnen de tweede hoofdreleaseserie. De woorden ‘reeks’ en ‘serie’ worden hier informeel gebruikt, maar ze betekenen wel wat men verwacht dat ze betekenen. Een hoofdserie bestaat simpelweg uit alle releases die hetzelfde hoofdnummer hebben en een subserie (of subreeks) bestaat uit alle releases die hetzelfde sub- *en* hoofdnummer hebben. Dat wil zeggen dat ‘2.4.0’ en ‘3.4.1’ niet in dezelfde subreeks zitten, hoewel het subnummer voor beide wel een ‘4’ is. Aan de andere kant zitten ‘2.4.0’ en ‘2.4.2’ wel in dezelfde subreeks, hoewel ze elkaar niet direct opvolgden indien ‘2.4.1’ tussen beide in is uitgebracht.

De betekenis van deze cijfers is precies wat u ervan zou verwachten: een stijging in het hoofdnummer betekent dat er een belangrijke verandering is, een stijging in het subnummer geeft aan dat het om een kleine verandering gaat en een verhoging van het micronummer staat voor onbeduidende veranderingen. Sommige projecten voegen hier nog een vierde component aan toe, dat meestal het *patchnummer* wordt genoemd, voor bijzonder minutieuze controle over de verschillen tussen de releases (verwarrend genoeg gebruiken andere projecten het woord ‘patch’ als synoniem voor ‘micro’ in een systeem met drie componenten). Er zijn ook projecten die de laatste component gebruiken als *build number*, dat iedere keer wordt verhoogd wanneer de software wordt opgebouwd *(built)* en dat geen enkele andere verandering aangeeft dan de betreffende build. Dit helpt het project ieder bugrapport te koppelen aan een specifieke build en is waarschijnlijk het meest bruikbaar wanneer er binaire pakketten worden gebruikt als standaarddistributiemethode.

Hoewel er zeer veel verschillende afspraken zijn over hoeveel componenten er moeten worden gebruikt en wat de componenten betekenen, zijn de verschillen daartussen minimaal. U hebt dus een klein beetje speelruimte, maar niet veel. De volgende twee gedeelten behandelen enkele van de meest gebruikte systemen.

De simpele methode

De meeste projecten hebben regels over welke soort veranderingen zijn toegestaan in een release wanneer alleen het micronummer wordt verhoogd, andere regels voor het subnummer en weer andere voor het hoofdnummer. Er is nog geen vaste norm voor deze regels, maar ik behandel hier een methode die met succes door verschillende projecten is gebruikt. U kunt deze methode gewoon overnemen voor uw eigen project. Zelfs als u dat niet doet, is het nog steeds een goed voorbeeld van het soort informatie dat releasenummers zouden moeten uitdrukken. Deze methode is overgenomen van het nummersysteem dat wordt gebruikt door het APR-project; zie http://apr.apache.org/versioning.html.

1. Veranderingen in alleen het micronummer (dat wil zeggen veranderingen binnen dezelfde subreeks) moeten zowel vooruit als achteruit compatibel zijn. Dat wil zeggen dat veranderingen alleen bugfixes zijn of zeer kleine verbeteringen aan bestaande functies. Nieuwe functies zouden niet in een microrelease mogen worden geïntroduceerd.

2. Veranderingen in het subnummer (dat wil dus zeggen binnen dezelfde hoofdlijn) moeten wel achteruit compatibel zijn, maar hoeven niet vooruit compatibel te zijn. Het is normaal nieuwe functies te introduceren binnen een subrelease, maar normaal gesproken niet te veel tegelijk.

3. Veranderingen in het hoofdnummer geven de grenzen van compatibiliteit aan. Een nieuwe hoofdrelease hoeft zowel vooruit als achteruit niet compatibel te zijn. Van een hoofdrelease wordt verwacht dat deze nieuwe functies heeft. Hij kan zelfs een geheel nieuwe set functies hebben.

Wat *achteruit compatibel* en *vooruit compatibel* precies betekenen hangt af van wat uw software doet, maar binnen de context zijn de termen vaak niet geschikt voor al te brede interpretatie. Als uw project bijvoorbeeld een client/servertoepassing is, dan betekent 'achteruit compatibel' dat het upgraden naar 2.6.0 er niet toe leidt dat clients met 2.5.4 functionaliteit verliezen of zich anders gaan gedragen dan voorheen (natuurlijk met uitzondering van opgeloste bugs). Aan de andere kant kan een upgrade naar één van deze clients naar 2.6.0, samen met de server, *nieuwe* functionaliteit beschikbaar stellen aan die client, functionaliteit die clients met 2.5.4 niet kunnen gebruiken. Als dat gebeurt is de upgrade *niet* 'vooruit compatibel': het is duidelijk dat u de client niet kunt downgraden naar 2.5.4 en toch alle functionaliteit behouden die het ook met 2.6.0 had, omdat sommige functionaliteiten nieuw waren in 2.6.0.

Dat is de reden waarom microreleases in principe alleen bedoeld zijn voor bugfixes. Ze moeten compatibel blijven in beide richtingen. Als u upgradet van 2.5.3 naar 2.5.4, vervolgens van gedachten verandert en weer teruggaat naar 2.5.3, dan mag geen functionaliteit verloren gaan. Natuurlijk komen de bugs die in 2.5.4 waren gerepareerd terug na de downgrade, maar u raakt geen functies kwijt, behalve in de gevallen waarin de teruggezette bugs het gebruik van bestaande functies verhinderen.

Client/serverprotocollen zijn slechts één van de vele mogelijke compatibiliteitsgebieden. Een andere is gegevensformaten. Schrijft de software de gegevens weg op een permanent opslagmedium? Als dat het geval is, dan moeten de formaten die het kan lezen en schrijven overeenkomen met de compatibiliteitsrichtlijnen zoals verwacht kan worden uit het releasenummersysteem. Versie 2.6.0 moet in staat kunnen zijn om bestanden te lezen die geschreven zijn met 2.5.4, maar kan het format stilletjes upgraden naar iets dat 2.5.4 niet kan lezen, omdat de mogelijkheid tot downgraden niet vereist is binnen de grenzen van een subnummer. Als uw project codebibliotheken distribueert voor gebruik in andere programma's, dan vormen de API's ook een compatibiliteitsgebied. U moet er voor zorgen dat de compatibiliteitsregels voor de bron- en de binaire code zodanig zijn uitgewerkt dat de geïnformeerde gebruiker zich nooit hoeft af te vragen of het wel of niet veilig is om te upgraden. Hij kan dit direct aan de nummers zien.

In dit systeem krijgt u nooit de kans op een nieuwe start totdat u het hoofdnummer verhoogt. Dit kan vaak heel onhandig zijn. Er kunnen functies zijn die u zou willen toevoegen of protocollen die u opnieuw zou willen vormgeven, maar u kunt dit niet doen omdat u de compatibiliteit moet waarborgen. Er is hiervoor geen eenduidige oplossing, behalve dat u zou moeten proberen de dingen vanaf het eerste begin op een uitbreidbare manier te ontwerpen. (Over dit onderwerp zou een heel boek geschreven kunnen worden, hetgeen absoluut buiten het bestek van dit boek valt.) Het publiceren van beleid over releasecompatibiliteit, en zich daar ook aan houden, is een onmisbaar onderdeel van het distribueren van software. Eén onaangename verrassing kan voldoende zijn om veel gebruikers weg te jagen. De hierboven beschreven methode is voor een deel goed omdat hij nogal wijdverbreid is, maar ook omdat hij makkelijk uit te leggen en te onthouden is, zelfs voor degenen die er niet vertrouwd mee zijn.

Algemeen wordt aanvaard dat deze regels niet gelden voor releases voorafgaande aan de 1.0-release, hoewel uw releasebeleid dat expliciet zou moeten vermelden, al is het alleen maar voor de duidelijkheid. Een project dat zich nog in de eerste ontwikkelingsfase bevindt, kan de releases 0.1, 0.2, 0.3 enz. op rij uitbrengen, tot het moment dat het klaar is voor release 1.0, en de verschillen tussen deze releases kunnen willekeurig groot zijn. Micronummers voor releases voorafgaande aan de 1.0-release zijn optioneel. Afhankelijk van de aard van uw project en de verschillen tussen de releases kan het wel of niet handig voor u zijn de nummers 0.1.0, 0.1.1 enz. te gebruiken. De afspraken over de releases voorafgaande aan release 1.0 zijn niet erg strikt, voornamelijk omdat mensen begrijpen dat strikte regels voor compatibiliteit in de beginfase de ontwikkeling te veel zou hinderen en omdat early adopters *('eerste gebruikers')* sowieso nogal vergevingsgezind zijn.

Vergeet niet dat al deze regels alleen van toepassing zijn op het systeem met drie componenten. Uw project kan met evenzoveel gemak een ander systeem met drie componenten bedenken, of beslissen dat een dergelijke fijne indeling helemaal niet nodig is en een systeem met twee componenten gebruiken. Het belangrijkste is dat u hier in een vroeg stadium over beslist, nauwkeurig publiceert wat de componenten inhouden en u zich er aan houdt.

De even-/onevenmethode

Sommige projecten gebruiken de pariteit van de subcomponent om de stabiliteit van de software aan te geven: even betekent stabiel, oneven betekent instabiel. Dit is alleen van toepassing op het subnummer, niet op de hoofd- en de micronummers. Een verhoging van het micronummer houdt nog steeds een bugfix in (geen nieuwe functies) en een verhoging in het hoofdnummer betekent een grote verandering, nieuwe feature enz.

Het voordeel van dit even-/onevensysteem, dat onder andere werd gebruikt bij het Linux kernel-project, is dat het een mogelijkheid biedt nieuwe functionaliteit uit te brengen om te worden getest, zonder dat andere gebruikers mogelijk met instabiele code moeten werken. Mensen kunnen aan de cijfers al zien dat ze '2.4.21' met een gerust hart kunnen installeren op hun operationele webserver, maar dat ze '2.5.1' beter kunnen bewaren voor experimenten op hun werkstation thuis. Het ontwikkelingsteam behandelt de bugrapporten van instabiele (dus met oneven subnummers) series en als alles na een aantal microreleases wat stabieler is geworden binnen de serie verhogen ze het subnummer (waarmee ze het dus even maken). Ze zetten het micronummer terug op '0' en releasen een vermoedelijk stabiel pakket.

Dit systeem houdt vast aan, of conflicteert in ieder geval niet met, de eerder genoemde richtlijnen voor compatibiliteit. Het voegt alleen wat extra informatie toe aan het subnummer. Dit betekent alleen dat het subnummer ongeveer twee keer zo vaak moet worden verhoogd als anders het geval zou zijn geweest, maar dat kan niet zo veel kwaad. Het systeem met even/oneven subnummers is waarschijnlijk het meest geschikt voor projecten met lange releasecycli en projecten, die door hun aard een groot aantal conservatieve gebruikers hebben die meer belang hechten aan stabiliteit dan aan nieuwe functies. Het is echter niet de enige manier om nieuwe functionaliteiten in de praktijk te testen. Het gedeelte 'Een release stabiliseren'

verderop in dit hoofdstuk beschrijft een andere, misschien vaker gebruikte methode voor het releasen van mogelijk instabiele code bij het publiek. De release wordt van een markering voorzien, waaraan mensen direct kunnen zien of er risico's dan wel voordelen aan zitten.

7.2 RELEASE-BRANCHES

Vanuit het oogpunt van een ontwikkelaar bevindt een open source-softwareproject zich permanent in de releasefase. Ontwikkelaars gebruiken eigenlijk altijd de meest recente code, omdat ze bugs willen opsporen en omdat ze het project van dichtbij genoeg volgen om van mogelijke instabiele gebieden binnen de functies af te blijven. Ze updaten hun versie van de software vaak iedere dag, soms meerdere keren per dag, en wanneer ze een verandering controleren, kunnen ze met redelijke zekerheid stellen dat alle andere ontwikkelaars deze ook binnen 24 uur zullen hebben.

Hoe kan het project dan een formele release uitbrengen? Moet het gewoon een momentopname maken van de software, er een pakketje van maken en het aan de wereld overhandigen als, bijvoorbeeld, versie '3.5.0'? Ons gezond verstand zegt ons dat het zo niet moet. Allereerst is er mogelijk geen enkel moment waarop de hele ontwikkelings-tree schoon is en klaar voor release. Er zijn mogelijk tal van nieuw begonnen functies in diverse stadia van voltooiing. Iemand heeft misschien een grote verandering ingediend om een bug te repareren, maar de verandering is wellicht controversieel en staat ter discussie op het moment dat de momentopname wordt gemaakt. Als dat het geval is, werkt het niet om de momentopname gewoon uit te stellen totdat de discussie is afgelopen. Immers, intussen kan een andere discussie zijn begonnen die daar helemaal los van staat, zodat u ook moet wachten tot *die discussie* weer is afgesloten. Er bestaat geen garantie dat dit proces ooit zal stoppen.

In ieder geval zal het gebruiken van momentopnames van de gehele softwaretree voor releases de lopende ontwikkelingswerkzaamheden in de weg staan, zelfs als de tree kan worden omgezet naar een vorm die geschikt is voor release. Stel dat deze momentopname versie '3.5.0' wordt. De volgende momentopname zou in dat geval '3.5.1' zijn en moet vooral fixes bevatten voor bugs in release 3.5.0. Maar beide zijn momentopnamen van dezelfde tree. Wat moeten de ontwikkelaars dan doen in de tijd tussen de twee releases? Ze kunnen geen nieuwe functies toevoegen: de richtlijnen voor compatibiliteit verbieden dat. Maar niet iedereen loopt warm voor het repareren van de bugs in release 3.5.0. Sommige mensen zullen nieuwe functies hebben die ze proberen af te ronden. Zij raken waarschijnlijk geïrriteerd als ze worden gedwongen te kiezen tussen op hun achterste zitten en niks doen of werken aan dingen waar ze niet in geïnteresseerd zijn, alleen omdat het releaseproces van het project van de ontwikkelingstree verwacht dat deze onnatuurlijk rustig blijft.

De oplossing voor deze problemen is om altijd een *release-branch* te gebruiken. Een release-branch is een aftakking in het versiebeheersysteem (zie *branch*), waarbinnen de code die bedoeld is voor deze release kan worden afgezonderd van de hoofdontwikkelingslijn. Het concept van release-branches is absoluut niet specifiek voor open source-software. Veel commerciële ontwikkelingsorganisaties gebruiken ze ook. In commerciële omgevingen worden release-branches echter soms beschouwd als luxe, een soort formele 'best practice' waar men onder druk van een belangrijke deadline ook best zonder kan omdat het hele team de koppen bij elkaar moet steken om de hoofdlijn te stabiliseren.

Release-branches kunnen echter voor open source-projecten als noodzakelijk worden beschouwd. Ik heb projecten gezien waar niet met release-branches werd gewerkt, maar het gevolg was altijd dat sommige ontwikkelaars met de armen over elkaar zaten terwijl anderen, meestal een minderheid, aan het werk waren om de release de deur uit te krijgen. Dit heeft over het algemeen verschillende negatieve gevolgen. Ten eerste gaat de vaart uit het ontwikkelingsproces. Ten tweede is de kwaliteit van de release onnodig slechter, omdat er maar een paar mensen aan werkten en ze zich haastten om het af te krijgen zodat de rest ook weer aan het werk zou kunnen. Ten derde zorgt het voor een psychologische splitsing in het ontwikkelingsteam, omdat er een situatie ontstaat waarbij de verschillende soorten werkzaamheden elkaar onnodig in de weg lopen. De ontwikkelaars die niks te doen hebben, zouden waarschijnlijk maar al te graag *iets* van hun energie in de release-branch steken, zolang ze daar zelf een keuze in hebben overeenkomstig hun eigen agenda en interesses. Zonder branch kunnen ze alleen maar kiezen voor "Participeer ik vandaag in het project of niet?" in plaats van "Werk ik vandaag aan de release of aan die nieuwe feature die ik heb ontwikkeld voor de hoofdlijn?"

De werking van release-branches

De exacte techniek voor het creëren van een release-branch hangt natuurlijk af van uw versiebeheersysteem, maar het algemene concept is gelijk voor de meeste systemen. Een branch begint meestal vanuit een andere branch of vanuit de trunk ('stam'). De trunk is vanouds de plek waar de hoofdlijn van de ontwikkeling plaats vindt, niet gehinderd door de beperkingen van een release. De eerste release-branch, die leidt naar release '1.0', begint vanuit de trunk. In CVS zou de branch-opdracht er ongeveer zo uit zien:

```
$ cd trunk-working-copy
$ cvs tag -b RELEASE_1_0_X
```

en in Subversion zo:

```
$ svn copy http://.../repos/trunk http://.../repos/branches/1,0.x
```

(Deze voorbeelden gaan uit van een releasenummersysteem met drie componenten. Ik kan hier niet de exacte opdrachten voor alle versiebeheersystemen laten zien, maar ik geef voorbeelden voor CVS en Subversion in de hoop dat de corresponderende opdrachten in de andere systemen uit deze twee kunnen worden afgeleid.)

Merk op dat we een branch '1.0.x' (met een feitelijke 'x') in plaats van '1.0.0' hebben gecreëerd. De reden hiervoor is dat dezelfde sublijn, d.w.z. dezelfde branch, zal worden gebruikt voor alle microreleases binnen deze lijn. Het feitelijke proces van het stabiliseren van een branch voor de release wordt behandeld in het gedeelte 'Een release stabiliseren' verderop in dit hoofdstuk. We houden ons hier alleen bezig

met de interactie tussen het versiebeheersysteem en het releaseproces. Wanneer de release-branch gestabiliseerd en gereed is, is het tijd een momentopname van de branch te labelen:

```
$ cd RELEASE_1_0_X-working-copy
$ cvs tag RELEASE_1_0_0
```

of

```
$ svn copy http://.../repos/branches/1.0.x http://.../repos/
tags/1.0.0
```

Dit label staat nu voor de exacte status van de source tree van het project in release 1.0.0 (dit kan handig zijn wanneer iemand ooit een oude versie nodig heeft nadat de distributiepakketten en binaries zijn weggehaald). De volgende microrelease wordt op dezelfde manier binnen de 1.0.x-branch gemaakt en wanneer deze klaar is wordt een tag gemaakt voor 1.0.1. Vervolgens wordt deze weer opgepoetst voor 1.0.2 enz. Wanneer het tijd is om over een 1.1.x-release te gaan nadenken, maakt u een nieuwe branch vanaf de trunk:

```
$ cd trunk-working-copy
$ cvs tag -b RELEASE_1_1_X
```

of

```
$ svn copy http://.../repos/trunk http://.../repos/branches/1.1.x
```

Het onderhoud voor zowel 1.0.x en 1.1.x kan parallel worden gedaan en releases kunnen onafhankelijk van elkaar vanuit beide lijnen worden uitgebracht. Het is zelfs niet ongewoon dat bijna tegelijkertijd releases worden vrijgegeven uit twee verschillende lijnen. De oudere serie wordt aanbevolen voor meer conservatieve websitebeheerders, die de grote sprong naar (bijvoorbeeld) 1.1 niet willen maken zonder zorgvuldig te zijn voorbereid. Intussen kunnen de meer avontuurlijke types de meest recente release van de hoogste lijn gebruiken, zodat ze zeker weten dat ze de nieuwste functies hebben, ondanks het risico van instabiliteit.

Dit is natuurlijk niet de enige strategie voor het werken met release-branches. In sommige situaties hoeft dit niet eens de beste strategie te zijn, hoewel het wel prima heeft gewerkt voor projecten waarbij ik betrokken ben geweest. U kunt iedere strategie gebruiken die lijkt te werken, maar onthoud de belangrijkste punten: het doel van een release-branch is het releasewerk te isoleren van de veranderingen in de dagelijkse ontwikkeling en het project een fysieke entiteit te geven waaromheen u het releaseproces kunt organiseren. Dit proces wordt in het volgende gedeelte verder uitgewerkt.

7.3 EEN RELEASE STABILISEREN

Stabilisatie is het proces waarbij een release-branch zo wordt bewerkt dat deze klaar is om te worden uitgebracht. Dat wil zeggen dat moet worden besloten welke veranderingen in de release worden opgenomen (en welke niet) en dat de inhoud van de branch dienovereenkomstig moet worden vormgegeven.

Dat ene woord 'besloten' kan een heleboel mogelijke problemen inhouden. De haast op het laatste moment nog functies toe te voegen is een bekend fenomeen bij cooperatieve softwareprojecten. Zodra ontwikkelaars weten dat een nieuwe release nabij is, raffelen ze het afronden van de bestaande veranderingen af om de boot niet te missen. Dat is natuurlijk precies het tegenovergestelde van wat u bij het uitbrengen van een release zou willen. Het zou veel beter zijn wanneer mensen rustig aan functies kunnen werken en zich niet druk hoeven maken of hun veranderingen in deze release worden opgenomen of in de volgende. Hoe meer veranderingen iemand op het laatste nippertje nog in de release probeert te proppen, des te instabieler de code wordt en des te meer nieuwe bugs er (meestal) worden gecreëerd.

De meeste software-engineers zijn het in theorie eens over de globale criteria ten aanzien van welke veranderingen in een releaselijn moeten worden toegelaten tijdens het stabilisatieproces. Uiteraard horen fixes voor ernstige bugs te worden opgenomen, met name bugs die niet op een andere manier kunnen worden omzeild. Updates van de documentatie zijn prima, evenals fixes in de foutmeldingen (behalve als ze deel uitmaken van de interface en stabiel moeten blijven). Veel projecten staan ook bepaalde niet-centrale veranderingen met weinig risico's toe tijdens de stabilisatie en beschikken zelfs over formele richtlijnen om de risico's te meten. Maar het proces kan nog zo geformaliseerd zijn, er is altijd menselijk beoordelingsvermogen nodig. Er zullen altijd situaties zijn waarbij het project gewoon moet beslissen of een bepaalde verandering wel of niet in de release wordt opgenomen. Het risico bestaat dat iedere persoon zijn eigen favoriete veranderingen in de release wil terugzien. Dat betekent dat er veel mensen zijn die veranderingen willen toestaan en maar weinig die ze willen tegenhouden.

Het proces van releasestabilisatie draait dus voornamelijk om het creëren van een mechanisme om 'nee' te zeggen. De oplossing voor open source-projecten in het bijzonder is een manier te vinden om 'nee' te zeggen zonder dat er te veel mensen op hun ziel worden getrapt of ontwikkelaars teleurgesteld zijn, maar die ook niet tot gevolg heeft dat veranderingen die het verdienen te worden opgenomen, uitgesloten blijven. Er zijn verschillende manieren om dit te doen. Het is vrij eenvoudig een systeem te ontwikkelen waarmee aan deze criteria wordt voldaan zodra het team aan deze criteria voldoende belang heeft toegekend. Ik geef hier een beschrijving van de meest populaire systemen aan de beide uitersten van het spectrum, maar laat uw project zich vooral niet ontmoedigen om creatief te zijn. Er zijn vele andere oplossingen mogelijk. Dit zijn er twee waarvan ik heb gezien dat ze in de praktijk goed werkten.

De eigenaar van de release als dictator

De groep komt overeen één persoon aan te wijzen als *eigenaar van de release*.

Deze persoon heeft het laatste woord over welke veranderingen in de release worden opgenomen. Natuurlijk is het normaal en ook te verwachten dat er over wordt gediscussieerd en geargumenteerd, maar uiteindelijk moet de groep de release-eigenaar voldoende autoriteit geven om de laatste beslissing te kunnen nemen. Om dit systeem te laten werken is het noodzakelijk iemand te kiezen met de technische competentie om alle veranderingen te begrijpen en de sociale status en handigheid in de omgang met mensen om de discussies te sturen zonder op al te veel tenen te trappen.

Het is gebruikelijk dat een eigenaar van een release zegt: "Ik denk niet dat er iets mis is met deze verandering, maar we hebben nog niet genoeg tijd gehad om het te testen. Daarom zou het nog niet in deze release moeten worden opgenomen." Het helpt veel als de release-eigenaar brede technische kennis heeft over het project en redenen kan geven waarom een verandering mogelijk destabiliserend zal werken (bijvoorbeeld door interacties met andere delen van de software of problemen met de portabiliteit). Mensen zullen soms verwachten dat dergelijke beslissingen worden beargumenteerd, of ze zeggen dat een verandering niet zo risicovol is als het lijkt. Dergelijke discussies hoeven niet confronterend te zijn, zolang de release-eigenaar in staat is om de argumenten objectief te beoordelen en niet meteen de hakken in het zand zet.

De release-eigenaar hoeft overigens niet perse dezelfde persoon hoeft te zijn als de projectleider (in die gevallen waar er sprake is van een projectleider, zie het gedeelte 'Vriendelijke dictators' in Hoofdstuk 4, *Sociale en politieke infrastructuur*). In feite kan het zelfs goed zijn om *niet* dezelfde persoon te nemen. De vaardigheden die een goede ontwikkelingsleider dient te hebben, zijn niet noodzakelijkerwijs dezelfde als die van een goede release-eigenaar. Bij een belangrijk aspect als het releaseproces kan het verstandig zijn iemand te nemen die tegenwicht kan bieden aan de inzichten van de projectleider.

Vergelijk de rol van de release-eigenaar met de minder dictatoriale rol zoals beschreven in het gedeelte 'Releasemanager' verderop in dit hoofdstuk.

Stemmen over veranderingen

Diametraal tegenovergesteld aan dictatorschap van de release-eigenaar staat natuurlijk democratie: ontwikkelaars kunnen stemmen over welke veranderingen in de release worden opgenomen. Omdat de belangrijkste functie van releasestabilisatie echter is om veranderingen *uit te sluiten*, is het belangrijk om het stemsysteem zo op te zetten dat er positieve actie van meerdere ontwikkelaars nodig is om een verandering in de release opgenomen te krijgen. Om een verandering op te nemen moet wel meer nodig zijn dan een gewone meerderheid van stemmen (zie het gedeelte 'Wie mag er stemmen?' in Hoofdstuk 4, *Sociale en politieke infrastructuur*). Anders zou één stem voor en geen enkele stem tegen een bepaalde verandering voldoende zijn om deze in de release op te nemen. Dat zou kunnen leiden tot de ongewenste situatie dat iedere ontwikkelaar voor zijn eigen veranderingen stemt en niet tegen andermans veranderingen durft te stemmen uit angst voor wraak. Om dat te voorkomen moet het systeem zo worden opgezet dat subgroepen van ontwikkelaars moeten samenwerken om een bepaalde verandering in de release te krijgen.

Dit betekent niet alleen dat meer mensen een verandering moeten reviewen, het zorgt er tevens voor dat een ontwikkelaar minder snel zal aarzelen om tegen een verandering te stemmen, omdat hij weet dat niemand van de voorstemmers zijn tegenstem als een persoonlijke belediging zal voelen. Hoe groter het aantal betrokkenen is, des te meer de discussie gaat draaien om de verandering en niet om het individu.

Het systeem dat we bij het Subversion-project gebruiken, lijkt een goede balans te hebben gevonden. Daarom kan ik dit hier van harte aanbevelen. Om een verandering aan de release-branch te kunnen toevoegen, moeten minimaal drie ontwikkelaars voor hebben gestemd en mag niemand tegen hebben gestemd. Eén enkele 'nee'-stem is genoeg om te voorkomen dat een verandering wordt opgenomen, dat wil zeggen dat een 'nee' binnen de context van releases gelijk staat aan een veto (zie het gedeelte 'Veto's'). Natuurlijk moet een dergelijk 'nee' beargumenteerd worden. En in theorie kan een veto worden genegeerd als maar genoeg mensen vinden dat het onredelijk is en daarom een bijzondere stemming daarover afdwingen. Dit is in de praktijk echter nog nooit gebeurd en ik verwacht ook niet dat het ooit zal gebeuren. Mensen zijn sowieso conservatief als het op releases aankomt en als iemand dermate overtuigd is van zijn zaak, dat hij een veto uitspreekt over de opname van een verandering in de release, dan is daar meestal een goede reden voor.

Omdat de releaseprocedure met opzet enigszins conservatief is, zijn de argumenten die voor veto's worden gegeven soms meer procedureel dan technisch van aard. Iemand kan bijvoorbeeld vinden dat een verandering goed is geschreven en dat de kans klein is dat deze voor nieuwe bugs zal zorgen, maar toch tegen opname in een microrelease stemmen gewoon omdat de verandering te groot is, misschien omdat het nieuwe functies toevoegt of omdat het op subtiele wijze voorbijgaat aan de richtlijnen voor compatibiliteit. Ik heb het ook wel eens meegemaakt dat ontwikkelaars hun veto gebruikten omdat ze instinctief aanvoelden dat de verandering nog meer getest zou moeten worden, hoewel ze tijdens de inspectie geen bugs konden ontdekken. Men mopperde hier wel over, maar het veto was al uitgesproken en de verandering werd niet in de release opgenomen (ik kan me echter niet meer herinneren of er later nog bugs zijn gevonden).

Het managen van coöperatieve releasestabilisatie

Als uw project ervoor kiest om over veranderingen te stemmen, dan is het van het allergrootste belang dat het fysieke mechanisme van het sturen van de stembiljetten en het stemmen zo gemakkelijk mogelijk is. Hoewel er genoeg open source-stemsoftware beschikbaar is, is het in de praktijk het makkelijkst om gewoon een tekstbestand te maken in de releasebranch met de naam `STATUS` of `STEMMEN` of iets dergelijks. In dit bestand worden de voorgestelde veranderingen (iedere ontwikkelaar mag een verandering voorstellen voor opname in de release) weergegeven, samen met alle stemmen voor en tegen, plus opmerkingen. (Een verandering voorstellen betekent trouwens niet per definitie dat erover gestemd wordt, hoewel de twee wel vaak samen gaan.) Een entry in zo'n bestand kan er als volgt uitzien:

```
* r2401 (issue #49)
  Voorkomen dat een client/server-handshake twee keer plaatsvindt.
```

```
Argument:
  voorkomt extra turnaround-tijd van het netwerk; kleine ver-
  andering en makkelijk te controleren.
Opmerkingen:
  dit is besproken in http://.../mailing-lists/message-7777.
  html en andere berichten binnen die thread.
Stemmen:
  +1: jsmith, kimf
  -1: tmartin (verbreekt de compatibiliteit met sommige -1.0
      servers, toegegeven, deze servers bevatten nogal wat
      bugs, maar waarom incompatibel zijn als dat niet hoeft?)
```

In dit geval kreeg de verandering twee positieve stemmen, maar werd deze weggestemd door tmartin, die de reden gaf voor zijn veto in een extra opmerking. Het exacte formaat van de entry doet er niet toe. Wat uw project hier ook over afspreekt is prima. Misschien moet de uitleg van tmartin voor zijn veto bijvoorbeeld geplaatst worden onder 'Opmerkingen:'-sectie, of misschien moet de beschrijving van de verandering een kop met 'Opmerkingen:' krijgen om bij de andere secties te passen. Het belangrijkste is dat alle benodigde informatie voor het beoordelen van de verandering bereikbaar is en dat het mechanisme voor het indienen van stemmen zo licht mogelijk is. Er wordt aan de voorgestelde verandering gerefereerd met het revisienummer in de database (in dit geval een enkelvoudige revisie, r2401, hoewel een voorgestelde verandering net zo goed uit meerdere revisies kan bestaan). De revisie wordt verondersteld te refereren aan een verandering aan de trunk. Als de verandering al in de release-branch zou zijn opgenomen, zou het niet nodig zijn erover te stemmen. Als uw versiebeheersysteem geen duidelijke syntaxis heeft voor referenties aan afzonderlijke veranderingen, dan moet het project er een opzetten. Om het stemmen werkbaar te maken, moet iedere verandering waarover gestemd moet worden ondubbelzinnig identificeerbaar zijn.

Degenen die een verandering voorstellen of erover stemmen, moeten ervoor zorgen dat deze zuiver en alleen betrekking heeft op de release-branch, dat wil zeggen zonder conflicten van toepassing is (zie *conflict*). Als er sprake is van conflicten, dan moet de entry of verwijzen naar een aangepaste patch die eenduidig van toepassing is, of naar een tijdelijke branch die een aangepaste versie van de verandering bevat, bijvoorbeeld:

```
* r13222, r13223, r13232
  Herschrijven libsvn_fs_fs's auto-merge algoritme
  Argument:
    onacceptabele prestatie (>50 minuten voor een kleine commit)
    in een repository met 300.000 revisies
  Branch:
    1.1.x-r13222@13517
  Stemmen:
    +1: epg, ghudson
```

Dit voorbeeld is direct uit de praktijk overgenomen. Het komt uit het `STATUS`-bestand voor het releaseproces van Subversion 1.1.4. U ziet hoe het de oorspronkelijke revisies gebruikt als handvaten voor de verandering, hoewel er ook een branch is met een versie van de verandering die is aangepast voor het conflict (de branch combineert ook de drie trunk-revisies in één, r13517, om het makkelijker te maken de verandering in de release in te voegen, als daar goedkeuring voor komt). De oorspronkelijke revisies worden ook gegeven, omdat deze nog steeds het makkelijkst te beoordelen zijn aangezien ze de oorspronkelijke logberichten bevatten. De tijdelijke branch beschikt niet over deze logberichten, om duplicatie van informatie te voorkomen (zie het gedeelte 'Enkelvoudigheid van informatie' in Hoofdstuk 3, *Technische Infrastructuur*) moet het logbericht van de branch voor r13517 alleen 'Aanpassen r13222, r13223 en r13232 voor backport naar 1.1.x-branch' zijn. Alle overige informatie over de veranderingen kan worden opgespoord bij de oorspronkelijke revisies.

Releasemanager

Het feitelijke samenvoegingsproces (zie *merge (ook wel port geheten)*) van goedgekeurde veranderingen in de release-branch kan door iedere ontwikkelaar worden gedaan. Er hoeft niet één persoon te worden aangewezen wiens taak het is de veranderingen samen te voegen. Als er veel veranderingen zijn, is het beter de taak over meerdere mensen te verdelen.

Hoewel zowel het stemmen als het samenvoegen echter op een gedecentraliseerde manier gebeurt, zijn er in de praktijk meestal één of twee mensen die het releaseproces aansturen. Deze krijgt soms de formele titel *releasemanager*, maar dat is heel iets anders dan een release-eigenaar (zie het gedeelte 'De eigenaar van de release als dictator' eerder in dit hoofdstuk) die het laatste woord heeft over de veranderingen. Releasemanagers houden in de gaten over hoeveel veranderingen er op dat moment gestemd wordt, hoeveel er zijn goedgekeurd, hoeveel er waarschijnlijk goedgekeurd gaan worden enz. Als ze bang zijn dat belangrijke veranderingen niet genoeg aandacht krijgen en door een gebrek aan stemmen buiten de release zouden vallen, kunnen ze andere ontwikkelaars voorzichtig aan hun jasje trekken om de verandering te beoordelen en erover te stemmen. Wanneer een cluster van veranderingen is goedgekeurd, zullen mensen vaak zelf de taak op zich nemen om deze in de release-branch te voegen. Het is prima als anderen deze taak aan hen overlaten, zolang iedereen wel begrijpt dat ze niet verplicht zijn al het werk te doen, behalve wanneer ze zich daartoe expliciet hebben verplicht. Wanneer de tijd is aangebroken om de release uit te brengen (zie het gedeelte 'Testen en uitbrengen' verderop in dit hoofdstuk) kan de releasemanager ook de zorg op zich nemen voor de logistiek van het maken van de definitieve releasepakketten, het verzamelen van digitale handtekeningen, het uploaden van de pakketten en de publieke aankondiging.

7.4 HET MAKEN VAN DOWNLOADPAKKETTEN

De meest geaccepteerde vorm voor de distributie van open source-software is als broncode. Dit geldt onafhankelijk van het feit of de software normaal gesproken in de vorm van de broncode draait (d.w.z. dat het kan worden geïnterpreteerd, zoals

Perl, Python, PHP enz.) of dat hij eerst gecompileerd moet worden (zoals C, C++, Java enz.). Met gecompileerde software zullen de meeste gebruikers de broncode waarschijnlijk niet zelf compileren, maar in plaats daarvan de installatie doen met een kant-en-klaar binair pakket (zie het gedeelte 'Binaire pakketten' verderop in dit hoofdstuk). Deze binaire pakketten worden echter nog steeds onttrokken uit distributie van de masterbron. Het doel van het broncodepakket is de release ondubbelzinnig te definiëren. Als het project 'Scanley 2.5.0' distribueert, betekent dit in feite 'de tree van broncodebestanden die, wanneer gecompileerd (wanneer nodig) en geïnstalleerd, Scanley 2.5.0 voortbrengt.'

Er zijn nogal strenge normen voor hoe een broncoderelease eruit zou moeten zien. Zo nu en dan zijn er wel afwijkingen te zien van deze norm, maar dit zijn de uitzonderingen en niet de regel. Tenzij er een zwaarwegende reden is om dit niet te doen, zou ook uw project deze norm moeten hanteren.

Format
De broncode moet worden verzonden in de standaardformaten voor het transporteren van directory trees. Voor Unix en op Unix lijkende besturingssystemen wordt over het algemeen het TAR-format gebruikt, gecomprimeerd met **compress**, **gzip**, **bzip** of **bzip2**. Voor MS Windows is de standaardmethode voor het distribueren van directory trees het *zip*-format. Dit comprimeert de tree ook, zodat het niet meer nodig is om het archief te comprimeren nadat u het hebt gemaakt.

TAR-bestanden
TAR staat voor 'Tape ARchive', omdat het tar-format een directory tree weergeeft als een lineaire gegevensstroom. Daardoor is het ideaal voor het opslaan van directory trees op tapes. Dezelfde eigenschap maakt dit ook de norm voor het distribueren van directory trees in de vorm van een enkelvoudig bestand. Het maken van gecomprimeerde tar-bestanden (ook wel *tarballs* genoemd) is nogal eenvoudig. Op sommige systemen kan de **tar-**opdracht zelf een gecomprimeerd archief produceren; op andere systemen wordt een afzonderlijk comprimeerprogramma gebruikt.

Naam en lay-out
De naam van het pakket moet bestaan uit de naam van de software, plus het releasenummer, plus de format-extensies die bij het archieftype horen. Scanley 2.5.0 bijvoorbeeld, verpakt voor Unix met behulp van GNU Zip (gzip)-compressie, zou er zo uit zien:

scanley-2.5.0.tar.gz

of voor Windows met zip-compressie:

scanley-2.5.0.zip

Beide archieven zouden, wanneer ze uitgepakt worden, een enkele nieuwe directory tree moeten creëren met de naam `scanley-2.5.0` in de huidige directory. Onder de nieuwe directory moet de broncode zijn ingedeeld in een lay-out die gereed is voor compilatie (wanneer compilatie nodig is) en installatie. Op het hoogste niveau van de directory tree moet er een `README`-bestand zijn opgenomen waarin wordt uitgelegd wat de software doet en welke release het is, met verwijzingen naar andere informatiebronnen zoals de website van het project, andere interessante bestanden enz. Bij deze andere bestanden moet een `INSTALL`-bestand zitten, een zusje van het `README`-bestand, waarin voor alle besturingssystemen die het ondersteunt instructies worden gegeven over hoe de software moet worden gebouwd en geïnstalleerd. Zoals eerder gezegd in het gedeelte 'Hoe u een licentie toepast op uw software' in Hoofdstuk 2, *Aan de slag*, moet er ook een bestand zijn met de naam `COPYING` of `LICENSE`, waarin de distributievoorwaarden van de software zijn beschreven.

Ook moet er een `CHANGES`-bestand zijn opgenomen (dat soms ook wel `NEWS` wordt genoemd), waarin wordt uitgelegd wat er nieuw is aan deze release. Het `CHANGES`-bestand is een totaaloverzicht van alle veranderingen van alle releases, in omgekeerd chronologische volgorde, zodat de lijst voor deze release bovenaan in het bestand staat. Het maken van deze lijst wordt meestal als laatste gedaan bij het stabiliseren van een release-branch. Sommige projecten schrijven deze lijst stukje bij beetje tijdens de ontwikkeling, anderen hebben er de voorkeur voor alles voor het laatst op te sparen zodat één persoon deze kan schrijven, op basis van informatie uit de logbestanden van het versiebeheer. De lijst ziet er ongeveer zo uit:

```
Versie 2.5.0
(20 december 2004, van /branches/2.5.x)
http://svn.scanley.org/repos/svn/tags/2.5.0/

Nieuwe functies, verbeteringen:
   * Regular expression queries toegevoegd (issue #53)
   * toegevoegde ondersteuning voor UTF-8- en UTF-16-documenten
   * Documentatie vertaald in het Pools, Russisch en Malagasi
   * ...

Bugfixes:
   * Reindexing bug gerepareerd (issue #945)
   * Enkele querybugs gerepareerd (issues #815, #1007, #1008)
   * ...
```

De lijst kan zo lang zijn als nodig is, maar u hoeft niet te veel moeite te doen door iedere kleine bugfix en functieuitbreiding te vermelden. Het doel is louter om gebruikers een overzicht te geven van wat ze erop vooruit zouden gaan als ze upgraden naar de nieuwe release. In feite wordt de lijst met veranderingen gewoonlijk opgenomen in de aankondigingsmail (zie het gedeelte 'Testen en uitbrengen' verderop in dit hoofdstuk), dus schrijf het met deze doelgroep in het achterhoofd.

CHANGES versus ChangeLog
Vanouds wordt een bestand met de naam *ChangeLog* gebruikt om een overzicht te geven van alle veranderingen die ooit in het project zijn doorgevoerd; dat wil zeggen iedere revisie die is gecommit in het versiebeheersysteem. Er zijn verschillende bestandsindelingen mogelijk voor ChangeLog-bestanden. De details van deze for-

mats zijn hier niet belangrijk, omdat ze allemaal dezelfde informatie bevatten: de datum van de verandering, de auteur en een korte samenvatting (of alleen het logbericht van de verandering).

Een `CHANGES`-bestand is iets anders. Ook dit is een lijst met de veranderingen, maar alleen die veranderingen die belangrijk worden geacht voor een bepaalde doelgroep en vaak met metagegevens zoals de exacte datum en de auteur verwijderd. Om verwarring te voorkomen zou u de termen niet door elkaar moeten gebruiken. Sommige gebruiken 'NEWS' in plaats van 'CHANGES'. Hoewel dit mogelijke verwarring met 'ChangeLog' voorkomt, is het een enigszins verkeerde benaming, omdat het CHANGES-bestand informatie bevat over alle releases en dus ook veel oud nieuws bevat naast het nieuwe nieuws bovenaan de lijst.

ChangeLog-bestanden verdwijnen zo langzamerhand sowieso. Ze waren nuttig in de tijd dat CVS het enige beschikbare versiebeheersysteem was, omdat gegevens over veranderingen niet zo makkelijk uit CVS gehaald konden worden. Bij de meer recente versiebeheersystemen kunnen de gegevens die voorheen werden opgeslagen in ChangeLog op ieder moment worden opgevraagd uit de versiebeheerdatabase, waardoor het zinloos wordt een statisch bestand bij te houden met deze gegevens. In feite is dit zelfs nog slechter dan zinloos, omdat het ChangeLog-bestand alleen de logberichten dupliceert die al in de repository zijn opgeslagen.

De feitelijke lay-out van de broncode binnen de tree moet dezelfde zijn, of in ieder geval zo goed mogelijk lijken op de lay-out van de broncode die te zien zou zijn wanneer het project direct wordt bekeken in de repository van het versiebeheersysteem. Meestal zijn er een paar verschillen, bijvoorbeeld omdat het pakket enkele bestanden bevat die zijn gegenereerd voor de configuratie en de compilatie (zie het gedeelte 'Compilatie en installatie' verderop in dit hoofdstuk), of omdat er externe software in is opgenomen die niet door het project wordt onderhouden, maar die wel vereist is en waarvan de kans klein is dat gebruikers die al hebben. Maar zelfs wanneer de gedistribueerde tree exact overeenkomt met een ontwikkelingstree in de repository van het versiebeheersysteem, zou de distributie zelf geen werkende kopie moeten zijn (zie *werkende kopie*). De release wordt geacht een statisch referentiepunt te zijn, een afzonderlijke, niet wijzigbare configuratie van de bronbestanden. Als het een werkende kopie zou zijn, bestaat het gevaar dat de gebruiker het updatet en achteraf denkt dat hij nog steeds de release heeft, terwijl hij in feite iets anders heeft.

Vergeet niet dat het pakket hetzelfde is, ongeacht de manier van verpakken. De release, dat wil zeggen de exacte entiteit waaraan wordt gerefereerd als iemand het over 'Scanley 2.5.0' heeft, is de tree die wordt gecreëerd door het uitpakken van een zip-bestand of tarball. Het project kan dus de onderstaande bestanden aanbieden om te downloaden ...

scanley-2.5.0.tar.bz2
scanley-2.5.0.tar.gz
scanley-2.5.0.zip

... maar de brontree die wordt aangemaakt door ze uit te pakken moet dezelfde zijn. Die bron-tree vormt de distributie. De vorm waarin deze wordt gedownload is alleen een kwestie van gemak. Bepaalde onbeduidende verschillen tussen bronpakketten kunnen wel worden toegelaten: in het Windows-pakket moeten de regels in tekstbestanden bijvoorbeeld eindigen met CRLF (Carriage Return en Line Feed), terwijl Unix-pakketten alleen een LF gebruiken. De trees kunnen ook worden opgezet met kleine verschillen tussen de bronpakketten voor de verschillende besturingssystemen, indien er voor de compilatie bij deze besturingssystemen verschillende lay-outvormen worden gebruikt. Dit zijn achter allemaal onbeduidende omzettingen. De basis van de bronbestanden moet dezelfde zijn voor alle pakketten van een bepaalde release.

Wel of geen hoofdletters

Wanneer mensen verwijzen naar de naam van een project gebruiken ze daarvoor meestal een hoofdletter, zoals voor een eigennaam gebruikelijk is. Ze gebruiken ook hoofdletters voor acroniemen: 'MySQL 5.0', 'Scanley 2.5.0' enz. Of dit gebruik van hoofdletters ook wordt toegepast in de naam van het pakket is aan het project om te beslissen. Zowel `Scanley-2.5.0.tar.gz` als `scanley-2.5.0.tar.gz` zijn prima (hoewel ik zelf de voorkeur heb voor de tweede, omdat ik mensen zo weinig mogelijk wil dwingen de shift-toets te gebruiken. Veel projecten gebruiken echter wel pakketten met hoofdletters). Het belangrijkste is dat de directory die wordt aangemaakt door het uitpakken van de tarball hoofdletters op dezelfde manier toepast. Er mogen geen verrassingen zijn: de gebruiker moet exact kunnen voorspellen wat de naam van de directory zal zijn die wordt aangemaakt wanneer hij een distributie uitpakt.

Voorreleases

Bij het versturen van een voorrelease of een kandidaat-release is het achtervoegsel een echt deel van het releasenummer. Neem dit dus op in de naam van het pakket. De gesorteerde volgorde van alfa- en bèta-releases zoals eerder genoemd in het gedeelte 'Componenten van releasenummers' zou bijvoorbeeld resulteren in pakketnamen als:

scanley-2.3.0-alpha1.tar.gz
scanley-2.3.0-alpha2.tar.gz
scanley-2.3.0-beta1.tar.gz
scanley-2.3.0-beta2.tar.gz
scanley-2.3.0-beta3.tar.gz
scanley-2.3.0.tar.gz

Het eerste bestand wordt uitgepakt naar een directory met de naam `scanley-2.3.0-alpha1`, de tweede naar `scanley-2.3.0-alpha2` enz.

Compilatie en installatie

Voor software die vanaf de broncode moet worden gecompileerd of geïnstalleerd is er meestal een standaardprocedure die ervaren gebruikers verwachten te kunnen volgen. Voor programma's geschreven in C, C++ of bepaalde andere gecompileerde talen is de norm onder op Unix lijkende systemen bijvoorbeeld dat de gebruiker het

volgende moet intypen:

```
$ ./configure
$ make
# make install
```

De eerste opdracht detecteert automatisch zo veel mogelijk informatie over de omgeving en bereidt deze voor op het build-proces. De tweede opdracht bouwt de software op de juiste plaats (maar installeert deze niet) en de laatste opdracht installeert deze op het systeem. De eerste twee opdrachten worden uitgevoerd als normale gebruiker, de derde als root. Voor meer informatie over het opzetten van dit systeem kunt u het uitstekende boek van Vaughan, Elliston, Tromey en Taylor, *GNU Autoconf, Automake, and Libtool* raadplegen. Het is in gedrukte vorm uitgegeven door New Riders en de inhoud ervan is ook gratis online beschikbaar via http://sources.redhat.com/autobook/.

Dit is niet de enige norm, maar het is wel één van de meest gebruikte. Het build-systeem Ant (http://ant.apache.org/) wordt steeds populairder, met name voor projecten die in Java zijn geschreven. Het heeft eigen standaardprocedures voor bouwen en installeren. Ook adviseren bepaalde programmeertalen, zoals Perl en Python, dezelfde methode te gebruiken voor de meeste programma's die met deze taal zijn geschreven (Perl-modules gebruiken bijvoorbeeld de opdracht **perl Makefile.pl**). Als u niet precies weet welke normen er van toepassing zijn op uw project, vraag dat dan aan een ervaren ontwikkelaar. U kunt er gevoeglijk vanuit gaan dat er in ieder geval *sommige* normen van toepassing zijn, zelfs als u in eerste instantie niet weet welke.

Wat de juiste normen voor uw project ook mogen zijn, wijk daar niet van af tenzij u echt geen andere keuze hebt. Het doorlopen van standaardinstallatieprocedures wordt door veel systeembeheerders tegenwoordig bijna instinctmatig gedaan. Als ze bekende termen zien in het `INSTALL`-bestand van uw project hebben ze er direct al vertrouwen in dat uw project zich algemeen bewust is van de gebruiken en dat de kans groot is dat ook andere dingen kloppen. Zoals besproken in het gedeelte 'Downloads' in Hoofdstuk 2, *Aan de slag* stelt het hebben van een standaard bouwprocedure ook potentiële ontwikkelaars tevreden.

Voor Windows zijn de normen voor bouwen en installeren wat minder vast. Voor projecten waarvoor compilatie nodig is, lijkt het inmiddels gewoonte te zijn om een tree zo aan te bieden dat deze past in het werkruimte-/projectmodel van de standaard-Windows-ontwikkelomgevingen (Developer Studio, Visual Studio, VS.NET, MSVC++ enz.). Afhankelijk van de aard van uw software kan het mogelijk zijn een op Unix lijkende build-optie aan te bieden voor Windows via de Cygwin-omgeving (http://www.cygwin.com/). En natuurlijk zou u, als u een taal of programmeerkader gebruikt dat zijn eigen build- en installatieconventies heeft (bijv. Perl of Python) de standaardmethode moeten gebruiken voor dat kader, of het nu voor Windows, Unix, Mac OS X of een ander besturingssysteem is.

Wees bereid veel extra energie te spenderen om uw project aan te passen aan de van toepassing zijnde build- of installatienormen. Bouwen en installeren is een eerste kennismaking met de software. Het is niet erg als het daarna moeilijker wordt, als dat niet anders kan, maar het zou jammer zijn als de eerste kennismaking tussen de gebruiker of de ontwikkelaar en de software uit onverwachte stappen bestaat.

Binaire pakketten

Hoewel de formele release een broncodepakket is, zullen de meeste gebruikers deze installeren vanaf binaire pakketten, die worden aangemaakt door het softwaredistributiemechanisme van hun besturingssysteem, of eigenhandig kunnen worden verkregen vanaf de website van het project of een externe partij. 'Binair' betekent in dit geval niet per definitie 'gecompileerd'. Het houdt alleen een voorgeconfigureerde vorm in van het pakket waarmee de gebruiker het op zijn computer kan installeren zonder de gebruikelijke build- en installatieprocedures voor de broncode te hoeven doorlopen. Voor RedHat GNU/Linux is dit het RPM-systeem, voor Debian GNU/Linux is dit het APT (`.deb`)-systeem, voor MS Windows zijn het over het algemeen `.MSI`-bestanden of zelf-installerende `.exe`-bestanden.

Of deze binaire pakketten nu worden samengesteld door mensen die direct bij het project zijn betrokken of door derden op afstand, gebruikers zullen ze wel gaan *behandelen* als equivalent van de officiële releases van het project en zullen issues in de bug tracker van het project invoeren die zijn gebaseerd op het gedrag van de binaire pakketten. Daarom is het in het belang van het project om makers van pakketten heldere richtlijnen te geven en nauw met hen samen te werken om erop toe te zien dat wat zij produceren de software eerlijk en nauwkeurig weergeeft.

Het belangrijkste dat pakketmakers moeten weten, is dat ze hun binaire pakketten altijd moeten baseren op een officiële release van de broncode. Soms laten makers van pakketten zich verleiden om een latere vorm van de code uit de repository te halen, of bepaalde veranderingen op te nemen die pas zijn toegevoegd nadat de release is uitgebracht, om gebruikers bepaalde bugfixes of andere verbeteringen te kunnen bieden. De maker van de pakketten denkt dat hij de gebruikers een plezier doet door ze recentere code te geven, terwijl deze manier van werken in feite voor grote verwarring kan zorgen. Projecten zijn erop voorbereid rapporten te ontvangen over bugs die worden gevonden in uitgebrachte releases en in de recente code van de trunk en de hoofd-branch (dat wil zeggen, gevonden door mensen die met opzet niet-uitontwikkelde code gebruiken). Als er van die mensen een bugrapport komt, zal degene die het rapport beantwoordt vaak wel in staat zijn te bevestigen dat de aanwezigheid van de bug bekend is voor die momentopname, en misschien ook dat het sindsdien is gerepareerd en dat de gebruiker moet upgraden of wachten op de volgende release. Als het om een voorheen onbekende bug gaat, maakt het weten van de exacte release het veel eenvoudiger om de bug te reproduceren en te categoriseren in de bug tracker.

Projecten zijn echter niet voorbereid op bugrapporten die gebaseerd zijn op niet-gespecificeerde tussentijdse of hybride versies. Het kan moeilijk zijn deze bugs te reproduceren. Ze kunnen ook het gevolg zijn van onverwachte interacties tussen de geïsoleerde veranderingen die uit een latere ontwikkeling zijn gehaald en die onjuist gedrag van de software veroorzaken waar de projectontwikkelaars niet op

aangekeken zouden mogen worden. Ik heb het zien gebeuren dat verbijsterende hoeveelheden tijd werden verspild omdat een bug *afwezig* was, terwijl deze er wel had moeten zijn. Iemand gebruikte namelijk een enigszins opgelapte versie, gebaseerd op (maar niet identiek aan) een officiële release, en toen de genoemde bug niet tevoorschijn kwam, moest iedereen blijven spitten om uit te zoeken waarom.

En toch zullen er soms situaties zijn waarbij een pakketmaker erop staat dat modificaties aan de release van de broncode doorgevoerd worden. Makers van pakketten zouden moeten worden aangemoedigd om dit met de ontwikkelaars van het project te bespreken en hun plannen voor te leggen. Misschien krijgen ze er toestemming voor, misschien niet. Maar zelfs in dat laatste geval hebben ze het project in ieder geval op de hoogte gesteld van hun bedoelingen, zodat het project kan uitkijken naar ongebruikelijke bugrapporten. De ontwikkelaars kunnen hierop reageren door een disclaimer op de website van het project te plaatsen en de maker van het pakket te vragen hetzelfde te doen op een daarvoor geschikte plaats, zodat gebruikers van het binaire pakket weten dat wat ze krijgen niet precies hetzelfde is als wat het project officieel heeft uitgebracht. Het is in zulke situaties helemaal niet nodig om vijandig te doen, hoewel dat helaas wel vaak gebeurt. Het probleem is dat de doelstellingen van pakketmakers vaak enigszins afwijken van die van ontwikkelaars. Pakketmakers willen voornamelijk een optimaal kant-en-klaarpakket voor hun gebruikers. Natuurlijk willen de ontwikkelaars dat ook, maar zij moeten er ook zeker van zijn dat ze weten welke softwareversies er beschikbaar zijn, zodat ze coherente bugrapporten kunnen ontvangen en garanties kunnen geven ten aanzien van de compatibiliteit. Soms is er een conflict tussen de verschillende doelen. Wanneer dat het geval is, moet u niet vergeten dat het project geen controle heeft over de makers van de pakketten en dat verplichtingen wederzijds zijn. Het klopt dat het project de pakketmakers een plezier doet door de software te produceren. Maar de pakketmakers doen het project ook een plezier door de weinig aantrekkelijk taak op zich te nemen om de software voor een breder publiek beschikbaar te maken, vaak in orde van grootte. Het is geen probleem als u het niet met de pakketmakers eens bent, maar jaag ze niet tegen u in het harnas. Probeer gewoon de best mogelijke oplossing te vinden.

7.5 TESTEN EN UITBRENGEN

Zodra de bron-tarball is gemaakt van de gestabiliseerde release-branch begint het publieke deel van het releaseproces. Maar voordat de tarball aan iedereen beschikbaar wordt gesteld, moet deze worden getest en goedgekeurd door een minimumaantal ontwikkelaars, meestal drie of meer. Deze goedkeuring bestaat uit meer dan alleen het controleren van de release op overduidelijke gebreken. In het ideale geval zullen de ontwikkelaars de tarball downloaden, deze op een schoon systeem bouwen en installeren, en een regressietestpakket erop loslaten (zie het gedeelte 'Geautomatiseerd testen') in Hoofdstuk 8, *Managen van vrijwilligers* en enkele handmatige tests. Ervan uitgaande dat de tarball deze tests goed doorloopt en dat deze ook voldoet aan andere checklistcriteria voor releases die het project heeft, ondertekenen de ontwikkelaars hem vervolgens digitaal met GnuPG (http://www.gnupg.org/), PGP (http://www.pgpi.org/), of een andere programma dat PGP-compatibele handtekeningen kan produceren.

In de meeste projecten gebruiken de ontwikkelaars hun eigen persoonlijke digitale handtekening, in plaats van een gezamenlijke projectsleutel, en er mogen zo veel ontwikkelaars ondertekenen als maar willen. Er is dus alleen een minimumaantal, geen maximumaantal. Hoe meer ontwikkelaars het ondertekenen, des te uitgebreider de release is getest en des te groter de kans dat een gebruiker die zich van de veiligheid van de software bewust is, kan zien hoe digitaal betrouwbaar de tarball is.

De release (dat wil zeggen alle tarballs, zip-bestanden en welke andere formaten er dan ook worden gedistribueerd) moet, zodra deze is goedgekeurd, op de downloadsectie van de website worden geplaatst, voorzien van de digitale handtekeningen en MD5/SHA1-checksums (zie http://en.wikipedia.org/wiki/Cryptographic_hash_function). Er bestaan hiervoor verschillende normen. Eén manier is om ieder releasepakket te voorzien van een bestand met de bijbehorende digitale handtekeningen en een bestand met de checksum. Als een van de uitgebrachte pakketten bijvoorbeeld `scanley-2.5.0.tar.gz` is, zet dan in dezelfde directory een bestand `scanley-2.5.0.tar.gz.asc` met de digitale handtekening voor de tarball, een ander bestand `scanley-2.5.0.tar.gz.md5` met de MD5-checksum en eventueel een derde bestand, `scanley-2.5.0.tar.gz.sha1`, met de SHA1-checksum. Een andere manier om te laten zien dat de release gecontroleerd is, is om alle handtekeningen voor een uitgebracht pakket samen te voegen in één enkel bestand, `scanley-2.5.0.sigs`. Hetzelfde kan met de checksums worden gedaan.
Het maakt in feite niet zo veel uit hoe u het doet. Maak een eenvoudig systeem, beschrijf dit duidelijk en gebruik het consistent voor alle releases. Het doel van deze handtekeningen en checksums is om de gebruikers een middel in handel te geven waarmee ze kunnen verifiëren dat er met de kopie die ze hebben ontvangen niet geknoeid is. Gebruikers zullen deze code op hun computer laten draaien. Als ermee geknoeid is, kan een aanvaller ineens toegang hebben tot al hun gegevens. Zie het gedeelte 'Beveiligingsreleases' verderop in dit hoofdstuk voor meer informatie over paranoia.

Kandidaat-releases
Voor belangrijke releases met veel veranderingen geven veel projecten er de voorkeur aan eerst een *kandidaat-release* uit te brengen, `d.w.z. scanley-2.5.0-beta1` voorafgaande aan `scanley-2.5.0`. Het doel van een kandidaat-release is dat deze uitvoerig kan worden getest voordat deze een officiële release wordt. Als er problemen worden gevonden, worden ze gerepareerd op de release-branch en wordt er een nieuwe kandidaat-release uitgerold (`scanley-2.5.0-beta2`). Deze cyclus wordt doorlopen totdat er geen onacceptabele bugs meer worden gevonden. Op dat moment wordt de kandidaat-release een officiële release. Dat wil zeggen dat het enige verschil tussen de laatste kandidaat-release en de feitelijke release is dat het achtervoegsel uit het versienummer wordt gehaald.

Wat betreft de meeste andere aspecten moet een kandidaat-release hetzelfde worden behandeld als een feitelijke release. De achtervoegsels *alfa*, *bèta* of *rc* is voldoende om conservatieve gebruikers te waarschuwen om te wachten op de echte release. Daarnaast moeten de aankondigingsmails voor de kandidaat-releases natuurlijk vermelden dat het doel van de release is om feedback te verzamelen. Buiten

deze verschillen om moeten de kandidaat-releases evenveel aandacht krijgen als gewone releases. Uiteindelijk wilt u dat mensen de kandidaat-release gaan gebruiken, omdat actief gebruik de beste manier is om bugs te ontdekken en ook omdat u niet zeker weet welke kandidaat-release uiteindelijk de officiële release zal worden.

Releases aankondigen

Het aankondigen van een release is gelijk aan het aankondigen van een andere gebeurtenis en moet de procedures volgen zoals beschreven in het gedeelte 'Publiciteit' in *Hoofdstuk 6, Communicatie*. Er zijn echter een paar specifieke dingen die u voor een release moet doen.

Telkens wanneer u de URL vrijgeeft naar de tarball van de release die gedownload kan worden, moet u ervoor zorgen dat u de MD5/SHA1-checksums en links naar de digitale handtekeningen vermeldt. Omdat de aankondiging op meerdere forums wordt geplaatst (mailinglijsten, nieuwspagina's enz.) kunnen gebruikers de checksums ook vanaf meerdere locaties krijgen, wat zelfs de gebruikers die zich het meest druk maken om de beveiliging extra kan geruststellen dat er niet met de checksums zelf is geknoeid. Het meerdere malen geven van de link naar de bestanden met de digitale handtekeningen maakt deze handtekeningen niet veiliger, maar het stelt mensen wel gerust (met name de mensen die het project niet van dichtbij volgen) dat het project de beveiliging serieus neemt.

Zorg ervoor dat u in de aankondigingsmail en op de nieuwspagina's die meer dan de minimale informatie over de release bevatten ook het betreffende deel van het CHANGES-bestand opneemt, zodat mensen kunnen zien wat ze aan een upgrade zouden kunnen hebben. Dit is net zo belangrijk voor kandidaat-releases als voor de uiteindelijke release. De aanwezigheid van bugfixes en nieuwe functies is belangrijk om mensen over te halen om de kandidaat-release uit te proberen.

Vergeet aan het eind ook niet om het team van ontwikkelaars, de testers en alle mensen die de tijd hebben genomen goede bugrapporten in te dienen te bedanken. Noem echter geen mensen bij naam, behalve wanneer er iemand is die in zijn eentje verantwoordelijk is voor een enorm deel van het werk, waarvan de betekenis door iedereen binnen het project wordt herkend. Ga gematigd om met complimentjes om te voorkomen dat ze aan waarde verliezen (zie het gedeelte 'Bedankjes' in Hoofdstuk 8, *Managen van vrijwilligers*).

7.6 HET ONDERHOUDEN VAN VERSCHILLENDE RELEASELIJNEN

De meeste volwassen projecten onderhouden meerdere releaselijnen, parallel aan elkaar. Nadat 1.0.0 bijvoorbeeld is uitgebracht, gaat die lijn verder met de microreleases (met bugfixes) 1.0.1, 1.0.2 enz., tot het moment dat het project expliciet besluit de lijn te beëindigen. Het uitbrengen van 1.1.0 alleen is overigens niet genoeg om de 1.0.x-lijn te beëindigen. Sommige gebruikers maken er bijvoorbeeld een gewoonte van nooit te upgraden naar de eerste release van een nieuwe sub- of hoofdlijn. Ze wachten eerst tot anderen de bugs uit 1.1.0 hebben gehaald en wachten op 1.1.1. Dit is niet per definitie egoïstisch (vergeet niet dat ze het ook zonder de bugfixes en de nieuwe functies moeten doen), ze willen alleen om wat voor reden dan ook voorzichtig zijn met upgrades. Dus als het project ontdekt dat er een grote bug zit in 1.0.3, vlak voordat release 1.1.0 wordt uitgebracht, het zou een beetje veel gevraagd zijn de bugfix alleen in 1.1.0 op te nemen en alle oude 1.0.x-gebruikers te vertellen dat ze moeten upgraden. Waarom dan niet zowel 1.1.0 als 1.0.4 uitbrengen om iedereen tevreden te houden?

Nadat de lijn 1.1.x een eind op weg is, kunt u het einde van 1.0.x aankondigen. Dit moet officieel worden aangekondigd. De aankondiging kan op zichzelf worden gedaan, of het besluit kan worden vermeld als onderdeel van de aankondiging van release 1.1.x. Hoe u het ook doet, gebruikers moeten weten dat een oude lijn wordt afgebouwd, zodat ze kunnen beslissen over een upgrade.

Sommige projecten geven een bepaald tijdsbestek aan waarin ondersteuning voor de voorgaande release wordt toegezegd. Binnen een open source-project betekent 'ondersteuning' het accepteren van bugrapporten voor die lijn en onderhoudsreleases maken wanneer er belangrijke bugs zijn gevonden. Andere projecten geven geen exacte periode aan, maar houden de binnenkomende bugrapporten in de gaten om in te kunnen schatten hoeveel mensen de oude lijn nog gebruiken. Wanneer het percentage onder een bepaalde waarde komt, wordt het einde van de lijn aangekondigd en stopt de ondersteuning.

Zorg er bij iedere release voor dat u een *doelversie* of *doelmijlpaal* in de bug tracker hebt opgenomen, zodat mensen die bugs indienen dat voor de juiste release kunnen doen. Vergeet niet ook een doelversie op te nemen met de naam 'ontwikkeling' of 'laatste' voor de meest recente ontwikkelingscode, omdat sommige mensen, en niet alleen de actieve ontwikkelaars, vaak vooruit lopen op de officiële releases.

Beveiligingsreleases

De meeste bijzonderheden over het omgaan met beveiligingsbugs zijn behandeld in het gedeelte 'Beveiligingskwetsbaarheden aankondigen' in *Hoofdstuk 6, Communicatie*, maar er zijn nog speciale bijzonderheden voor beveiligingsreleases die hier moeten worden besproken.

Een *beveiligingsrelease* is een release die uitsluitend wordt uitgebracht om een beveiligingskwetsbaarheid op te lossen. De code die de bug oplost, kan niet worden gepubliceerd totdat de release beschikbaar is, wat betekent dat de fixes niet alleen niet in de repository kunnen worden opgenomen tot de datum van de release, maar ook dat de release niet publiek kan worden getest voordat hij de deur uitgaat. Natuurlijk kunnen de ontwikkelaars de fix onderling onderzoeken en de release zelf testen, maar uitgebreid testen in de praktijk is niet mogelijk.

Omdat de release niet voldoende kan worden getest, moet een beveiligingsrelease altijd bestaan uit een bestaande release plus de fixes voor de beveiligingsbug, en *geen andere veranderingen*. Dit is omdat hoe meer veranderingen u opneemt die niet zijn getest, des te groter de kans is dat één daarvan een nieuwe bug zal veroorzaken, misschien zelfs wel een beveiligingsbug! Dit conservatisme is ook bedoeld voor beheerders die de beveiligingsfix wel moeten implementeren, maar wiens be-

leid het is geen andere veranderingen op hetzelfde moment te implementeren.

Het maken van een beveiligingsrelease betekent soms ook dat u misschien wat mensen moet misleiden. Het project heeft bijvoorbeeld al een poosje gewerkt aan release 1.1.3, waarbij een aantal bugfixes voor 1.1.2 al zijn aangekondigd. Op dat moment komt er een beveiligingsrapport binnen. Uiteraard kunnen de ontwikkelaars niet over het probleem praten totdat de fix beschikbaar is gesteld. Tot dat moment moeten ze in het openbaar blijven praten alsof 1.1.3 zo zal blijven als gepland. Maar wanneer 1.1.3 werkelijk uitkomt, zal het enige verschil met 1.1.2 de beveiligingsfix zijn. Alle andere fixes zijn uitgesteld tot 1.1.4 (die dan natuurlijk *ook* de beveiligingsfix bevat, net als alle toekomstige releases).

U kunt een extra component toevoegen aan een bestaande release om aan te geven dat het alleen een verandering in de beveiliging bevat. Mensen zouden bijvoorbeeld aan de hand van de cijfers moeten kunnen zeggen dat 1.1.2.1 een beveiligingsrelease is voor 1.1.2 en dat iedere release met een 'hoger' nummer (d.w.z. 1.1.3, 1.2.0 enz.) dezelfde beveiligingsfixes bevat. Voor mensen die er verstand van hebben, bevat dit systeem veel informatie. Aan de andere kant kan het voor mensen die het project van dichtbij volgen een beetje verwarrend zijn wanneer er meestal een releasenummer met drie componenten is met zo nu en dan een vierde component, naar het lijkt willekeurig. De meeste projecten die ik onder ogen heb gehad, kiezen voor consistentie en gebruiken gewoon het volgende geplande nummer voor de beveiligingsrelease, ook als dat betekent dat alle geplande releases een plaats moeten opschuiven.

7.7 RELEASES EN DAGELIJKSE ONTWIKKELING

Het onderhouden van parallelle releases heeft implicaties voor de dagelijks ontwikkeling. Een bepaalde regel, die sowieso wordt aanbevolen, wordt met name in dit geval een verplichting: zorg ervoor dat iedere commit één enkele logische verandering is en voeg veranderingen die niet met elkaar te maken hebben nooit samen in dezelfde commit. Als een verandering te groot is of te verstorend kan zijn, verdeel het dan over meerdere commits, waarbij iedere commit een duidelijk omlijnde subgroep is van de totale verandering en niets bevat dat niks met de grootschalige verandering te maken heeft.

Dit is een voorbeeld van een niet goed doordachte commit:

```
------------------------------------------------------------------
r6228 | jrandom | 2004-06-30 22:13:07 -0500 (Wed, 30 Jun 2004) |
8 lines

Fix issue #1729: de indexer moet de gebruiker vriendelijk waar-
schuwen wanneer een bestand wordt gewijzigd tijdens het indexeren.

* ui/repl.py
  (ChangingFile): nieuwe uitzonderingsklasse.
  (DoIndex): omgaan met nieuwe uitzondering.

* indexer/index.py
  (FollowStream: nieuwe uitzondering creëren wanneer bestand
  wordt gewijzigd tijdens indexeren.
  (BuildDir: enkele zaken die hier niets mee te maken hebben,
  zoals het verwijderen van enkele overbodige opmerkingen, op-
  nieuw formatteren van code en de foutcontrole tijdens het aan-
  maken van een directory repareren.

Andere poetsacties die hier niets mee te maken hebben:

* www/index.html: enkele typefoutjes herstellen, volgende re-
  leasedatum vaststellen.
------------------------------------------------------------------
```

Het probleem hiervan wordt duidelijk op het moment dat iemand de reparatie van de foutcontrole in `BuildDir` over wil brengen naar een branch voor een aanstaande onderhoudsrelease. Degene die dit doet, wil de andere veranderingen niet overbrengen. De fix voor issue #1729 is bijvoorbeeld misschien niet goedgekeurd voor de onderhouds-branch en de verbeteringen van `index.html` zijn gewoon niet relevant hier. Hij kan de verandering in `BuildDir` alleen niet ophalen via de samenvoegfunctie van het versiebeheersysteem, omdat in het versiebeheersysteem is opgenomen dat de verandering logisch gekoppeld is aan de andere veranderingen die hier geen relatie mee hebben. In feite wordt het probleem al duidelijk vóór de samenvoeging. De verandering vermelden om over te stemmen is al een probleem. In plaats van alleen een revisienummer te geven, moet degene die het voorstel maakt een speciale patch of veranderings-branch maken om het voorgestelde deel van de commit te isoleren. Voor anderen is dit nogal een klus om doorheen te worstelen, en dat allemaal omdat de oorspronkelijke committer niet de moeite heeft genomen om de veranderingen op te delen in logische groepen.

In feite had deze commit moeten worden opgesplitst in *vier* verschillende commits: één om issue #1729 te repareren, een tweede om overbodige opmerkingen te verwijderen en de code te formatteren in `BuildDir`, een derde om de foutcontrole in `BuildDir` te repareren en een vierde om `index.html` op te kalefateren. De derde commit zou dan worden gebruikt als voorstel voor de branch van de onderhoudsrelease.

Natuurlijk is het stabiliseren van de release niet de enige reden waarom iedere commit één logische verandering zou moeten bevatten. Psychologisch gezien is een semantisch homogene commit makkelijker te evalueren en makkelijker terug te draaien indien nodig (in sommige versiecontrolesystemen is reversie sowieso een bijzonder type samenvoeging). Een beetje zelfdiscipline van iedereen op voorhand kan het project in een later stadium veel hoofdbrekens besparen.

Het plannen van releases

Eén gebied waarop open source-projecten altijd al hebben afgeweken van proprietaire projecten is bij het plannen van de release. De deadlines van propriëtaire projecten zijn vaak veel strakker. Soms is dat omdat klanten is beloofd dat er op een bepaalde datum een upgrade beschikbaar zal zijn, omdat de nieuwe release gecoördineerd moet worden met andere activiteiten of marketingacties, of omdat de kapitaalverstrekkers die erin hebben geïnvesteerd resultaten willen zien voordat ze meer geld in het project stoppen. Open source-projecten werden tot voor kort echter gekenmerkt door amateurisme, in de meest letterlijke betekenis van het woord. Ze werden geschreven uit liefde voor het onderwerp, niet uit commerciële overwegingen. Niemand voelde de behoefte om alle functies uit te brengen voordat ze echt klaar waren, en waarom zouden ze ook? Niemands baan zou erdoor op de tocht kunnen komen te staan.

Tegenwoordig worden veel open source-projecten gefinancierd door bedrijven, waardoor ze meer en meer onder invloed komen te staan van de deadlinegerichte bedrijfscultuur. Dit is in veel gevallen positief, maar het kan ook prioriteitsconflicten veroorzaken tussen de ontwikkelaars die ervoor worden betaald en de ontwikkelaars die het werk vrijwillig doen. Deze conflicten steken vaak de kop op rond het issue van wanneer en hoe releases moeten worden gepland. De betaalde ontwikkelaars, die onder druk staan, willen natuurlijk het liefst gewoon een datum prikken voor de release waaraan iedereen zich dan moet aanpassen. De vrijwilligers kunnen een hele andere agenda hebben, misschien functies die ze willen afmaken, of tests die ze eerst willen doen en waarop naar hun mening de release moet wachten.

Natuurlijk is er geen andere oplossing voor dit probleem dan praten en compromissen sluiten. Maar u kunt de frequentie en de ernst van de problemen minimaliseren door het *bestaan* van een bepaalde release los te koppelen van de datum waarop hij de deur uit moet. Probeer dus de discussie in de richting te sturen van welke release het project op de korte en middellange termijn zal uitbrengen en welke functies er in zullen zitten, zonder direct al over een datum te beginnen, behalve voor een ruwe planning met hele grote marges[23]. Door de op te nemen functies al in een vroeg stadium vast te leggen wordt de discussie rond iedere afzonderlijke release minder complex, waardoor de voorspelbaarheid verbetert. Dit creëert ook een soort passief vooroordeel ten opzichte van mensen die een release willen uitbreiden door nieuwe functies of andere complicaties toe te voegen. Als de inhoud van een release goed is gedefinieerd, moet degene die een uitbreiding daarop voorstelt met goede argumenten komen voor die uitbreiding, zelfs als de datum voor de release nog helemaal niet is vastgelegd.

Dumas Malone schreef een uit meerdere delen bestaande biografie van Thomas Jefferson, *Jefferson and His Time*, waarin hij vertelt hoe Jefferson de eerste bijeenkomst leidde waarin de organisatie van de toekomstige universiteit van Virginia moest worden vastgesteld. De universiteit was in de eerste plaats Jeffersons idee, maar (wat overal en altijd het geval is, niet alleen in open source-projecten) er waren al snel anderen bij gekomen, allemaal met hun eigen belangen en agenda's. Op het moment dat ze bij elkaar kwamen voor de eerste vergadering om alles door te spreken, zorgde Jefferson ervoor dat hij zeer nauwkeurig voorbereide bouwtekeningen, gedetailleerde budgetten voor de bouw en operationele kosten, een voorstel voor een onderwijsplan en de namen van de specifieke faculteiten die hij wilde overbrengen vanuit Europa bij zich had. Niemand anders in de zaal was zo goed voorbereid als Jefferson en de groep had in feite niet veel keus dan te schikken naar zijn visie. Uiteindelijk werd de universiteit min of meer overeenkomstig Jeffersons plannen opgericht. Waarschijnlijk wist Jefferson heel goed dat de bouw uiteindelijk veel meer zou gaan kosten dan was gebudgetteerd en dat veel van zijn ideeën om verschillende redenen helemaal niet zouden werken. Zijn plannen waren strategisch.

Hij kwam opdagen bij een bijeenkomst met zoveel materiaal, dat de anderen niets anders konden dan veranderingen voorstellen op zijn materiaal, zodat de algehele vorm, en daarmee de planning van het project, ruwweg zou zij zoals hij het wilde. Bij open softwareprojecten is er geen afzonderlijke 'bijeenkomst', maar in plaats daarvan een reeks kleine voorstellen die voornamelijk via de issue tracker worden ingediend. Maar als u enige kredietwaardigheid hebt opgebouwd, en verschillende functies, verbeteringen en bugs toe gaat wijzen aan releases in de issue tracker overeenkomstig een vooraf uitgebrachte algemene planning, zullen mensen in de meeste gevallen met u meegaan. Zodra u de dingen ongeveer hebt georganiseerd zoals u het wilt, zullen de discussies over de feitelijke *releasedatum* veel soepeler verlopen.

Het is natuurlijk wel erg belangrijk dat u geen enkele individuele beslissing presenteert als een voldongen feit. Nodig mensen in de opmerkingen bij iedere toewijzing van een issue aan een specifieke toekomstige release uit tot discussie en afwijkende opvattingen en wees werkelijk bereid waar mogelijk uw mening te veranderen. Probeer niet de baas te spelen alleen maar om de baas te kunnen spelen. Hoe meer anderen zijn betrokken bij de planning van het releaseproces (zie het gedeelte 'Zowel de managementtaken als de technische taken delen met anderen' in Hoofdstuk 8, *Het managen van vrijwilligers*), des te makkelijker het zal zijn hen over te halen mee te gaan in uw prioriteiten voor issues die voor u belangrijk zijn.

Een andere manier om de spanning rond het plannen van releases te verminderen, is zeer regelmatig nieuwe releases uitbrengen. Wanneer er veel tijd zit tussen verschillende releases is het belang van iedere afzonderlijke release voor mensen veel groter. Ze zijn nog veel meer teleurgesteld als hun code niet wordt opgenomen in een release, omdat ze weten hoe lang het kan duren voordat ze een volgende kans krijgen. Afhankelijk van de complexiteit van het releaseproces en de aard van het project is een periode van drie tot zes maanden een goede interval tussen releases, hoewel onderhoudslijnen wat vaker microreleases kunnen uitbrengen, als daar behoefte aan is.

23| Voor een alternatieve benadering kunt u het proefschrift van Martin Michlmayr Ph.D. nalezen: *Quality Improvement in Volunteer Free and Open Source Software Projects: Exploring the Impact of Release Management* (http://www.cyrius.com/publications/michlmayr-phd.html). Dit proefschrift gaat over tijdgestuurde releaseprocessen, in tegenstelling tot releases op basis van functies, binnen grote open source-softwareprojecten. Michlmayr heeft ook voor Google een lezing gehouden over dit onderwerp, dat beschikbaar is via Google Video op http://video.google.com/videoplay?docid=-5503858974016723264.

8

HET MANAGEN VAN VRIJWILLIGERS

Om ervoor te zorgen dat mensen het eens worden over wat het project nodig heeft en dat ze samenwerken om dit voor elkaar te krijgen, is een prettige werkomgeving en het ontbreken van verstoorde onderlinge relaties alleen niet voldoende. Het project heeft iemand nodig - of meerdere personen - die alle bij het project betrokken mensen bewust aanstuurt. Het managen van vrijwilligers is misschien geen technische vaardigheid zoals programmeren, maar het is wel een vaardigheid in de zin dat hij kan worden verbeterd door studie en oefening.

Dit hoofdstuk kan worden gezien als een grabbelton vol losse technieken voor het managen van vrijwilligers. Het is misschien nog wel meer dan de voorgaande hoofdstukken gebaseerd op het Subversion-project, deels omdat ik aan dat project aan het werk was tijdens het schrijven van dit boek en ik alle informatiebronnen direct bij de hand had, en deels omdat zelfkritiek gemakkelijker wordt geaccepteerd dan kritiek op anderen. Ik heb echter ook bij diverse andere projecten gezien hoe goed het is om de onderstaande aanbevelingen op te volgen (en wat de consequenties zijn als u dit niet doet). Wanneer het uit politiek standpunt haalbaar is voorbeelden te geven uit deze andere projecten zal ik dat zeker doen.

Nu we het toch over politiek hebben, kunnen we dit vaak verfoeide woord direct eens onder de loep nemen. Veel techneuten zien politiek gedrag als iets waar anderen zich schuldig aan maken. "*Ik* pleit alleen voor wat het beste is voor het project, maar *hij* maakt bezwaar op politieke gronden." Ik geloof dat dit negatieve vooroordeel over politiek (of wat wordt beschouwd als politiek) met name sterk is bij technische mensen, omdat zij ervan overtuigd zijn dat sommige oplossingen objectief gezien beter zijn dan andere. Wanneer iemand zich dus opstelt op een manier die ingegeven lijkt te zijn door externe beweegredenen, bijvoorbeeld het in stand houden van de eigen invloed, het verminderen van de invloed van een ander, directe koehandel of iemand niet op zijn tenen willen trappen, kunnen andere deelnemers aan het project geïrriteerd raken. Helaas is dit voor henzelf slechts zelden reden om zich niet op dezelfde manier te gedragen wanneer hun eigen belangen op het spel staan.

Als u 'politiek' een vies woord vindt en hoopt dat uw project hiervan verschoond

zal blijven, moet ik u helaas teleurstellen. Politiek is onvermijdelijk wanneer mensen samen moeten werken en daarvoor gemeenschappelijke middelen ter beschikking hebben. Het is absoluut normaal dat bij het nemen van beslissingen de vraag hoe een bepaalde actie effect kan hebben op de eigen invloed op het project in de toekomst één van de overwegingen is. Uiteindelijk moet, als u vertrouwen hebt in uw eigen oordeel en vaardigheden - wat bij de meeste programmeurs het geval is - het mogelijke verlies van invloed worden gezien als een technisch effect. Dezelfde gedachtegang is van toepassing op ander gedrag dat op het eerste gezicht 'pure' politiek lijkt te zijn. In feite bestaat zoiets als pure politiek helemaal niet. Ook in het echte leven geldt dat juist omdat acties meerdere consequenties hebben, mensen zich bewust worden van het politieke spel. Politiek is, uiteindelijk, alleen de bevestiging van het feit dat *alle* gevolgen van beslissingen moeten worden overwogen. Als een bepaalde beslissing leidt tot een resultaat dat voor de meeste deelnemers bevredigend is, maar dat veranderingen met zich meebrengt voor de machtsrelaties binnen het project en sleutelfiguren geïsoleerd raken, is dit laatste zeker net zo belangrijk als het eerste. Dit negeren, is niet intellectueel, maar eerder kortzichtig.

Vergeet bij het lezen van het onderstaande advies, maar ook tijdens het werken aan uw project, dus niet dat *niemand* boven het politieke spel verheven is. Laten lijken alsof dat wel zo, is louter een politieke strategie op zich, en soms een heel bruikbare, maar het is nooit de realiteit. Politiek is gewoon wat er gebeurt wanneer mensen het met elkaar oneens zijn, en projecten die succesvol zijn, ontwikkelen politieke mechanismes om op constructieve wijze met deze meningsverschillen om te gaan.

8.1 HET MEESTE UIT UW VRIJWILLIGERS HALEN

Waarom doen vrijwilligers mee aan open source-softwareprojecten?[24]

Wanneer u hen ernaar vraagt, zeggen de meesten dat ze het doen omdat ze goede software willen maken of persoonlijk betrokken willen zijn bij het repareren van bugs waarvan ze zelf last hebben. Dit argument is echter meestal niet het hele verhaal. Kunt u zich voorstellen dat een vrijwilliger bij het project betrokken blijft, zelfs wanneer niemand ooit zijn waardering uitspreekt voor wat hij heeft gedaan of überhaupt naar hem luistert? Natuurlijk niet. Het is logisch dat mensen tijd spenderen aan open source-software om redenen die verdergaan dan de abstracte wens om goede code te produceren. Inzicht in de werkelijke beweegredenen van vrijwilligers kan u helpen om de dingen zo te organiseren dat u ze kunt aantrekken en vasthouden. De wens om goede software te maken kan één van die beweegredenen zijn, samen met de uitdaging en de educatieve waarde van het werken aan lastige problemen. Mensen hebben echter ook een ingebouwde behoefte om met andere mensen samen te werken en respect te geven en te verdienen door gezamenlijke activiteiten. Groepen die samenwerken moeten gedragsnormen ontwikkelen, op zo'n manier dat status wordt verkregen en behouden door middel van de activiteiten die bijdragen aan het realiseren van de doelen van die groep.

Deze normen ontstaan niet altijd vanzelf. Bij sommige projecten kunnen mensen het gevoel hebben dat ze status krijgen door vaak en breedsprakig te posten. Ervaren open source-ontwikkelaars kunnen hiervan waarschijnlijk zo enkele voorbeelden uit de mouw schudden. Mensen hebben dat gevoel niet toevallig, maar omdat ze worden beloond met respect voor hun lange en ingewikkelde argumenten, of deze nu wel of niet positief bijdragen aan het project. Hieronder geef ik u een aantal technieken om een sfeer te creëren waarbij acties die tot status leiden, tevens constructieve acties zijn.

Delegeren

Delegeren is niet alleen een manier om de werkdruk over meerdere mensen te verdelen, het is ook een politiek en sociaal instrument. Denk eens aan alle mogelijke gevolgen, wanneer u iemand vraagt iets te doen. Het meest voor de hand liggende gevolg is dat u iets niet meer hoeft doen omdat hij dat doet. Een ander gevolg is echter dat hij zich bewust wordt van het feit dat u hem een bepaalde taak toevertrouwt. Indien u het verzoek via een publiek forum hebt gedaan, weet hij bovendien dat anderen uit de groep zich eveneens bewust zijn van uw vertrouwen in hem. Hij kan ook enige druk voelen om de taak op zich te nemen. Dat betekent dat u het op zo'n manier moet vragen dat hij de taak zonder gezichtsverlies kan weigeren als hij deze werkelijk niet wil doen. Als de taak coördinatie met anderen binnen het project met zich meebrengt, komt uw verzoek er in feite op neer dat u hem vraagt meer bij het project betrokken te raken, samenwerkingsverbanden te creëren die anders niet zouden worden gecreëerd en misschien zelfs autoriteit te worden voor een bepaald subgebied van het project. De extra verantwoordelijkheid kan hem afschrikken, maar kan er anderzijds voor zorgen dat hij ook op andere manieren meer bij het project betrokken raakt omdat hij zich over het geheel genomen meer betrokken voelt bij het project.

Vanwege al deze mogelijke gevolgen is het vaak zinvol om iemand te vragen iets te doen, ook al weet u dat u het zelf sneller en beter zou kunnen. Natuurlijk is er soms ook een puur economisch argument voor. Soms zijn de kosten wanneer u het zelf doet te hoog en kunt u binnen hetzelfde tijdsbestek iets doen wat nog belangrijker is. Maar zelfs als dat argument geen rol speelt, kunt u nog steeds iemand anders willen vragen om de taak uit te voeren, omdat u die persoon op de lange termijn meer bij het project wilt betrekken. Het feit dat u in het begin meer tijd kwijt bent om een oogje in het zeil houden, speelt daarbij geen rol. Het omgekeerde is ook van toepassing. Als u zo nu en dan werk doet dat anderen niet willen doen of waarvoor ze geen tijd hebben, kweekt u daar goodwill en respect mee. Delegeren en substitutie zijn niet alleen bedoeld om bepaalde taken gedaan te krijgen, ze zijn ook bedoeld om mensen meer bij het project te betrekken.

Onderscheid maken tussen informeren en toewijzen (assign)

Soms kunt u best van een persoon verwachten dat hij een bepaalde taak op zich neemt. Als iemand bijvoorbeeld een bug in de code heeft geschreven of code inbrengt die overduidelijk niet voldoet aan de richtlijnen van het project, dan is het voldoende wanneer u het probleem aankaart en er daarna van uitgaat dat deze persoon het probleem oplost. Er zijn echter ook situaties waarin u niet overduidelijk het recht hebt om bepaalde dingen te verwachten. De persoon kan doen wat u hem vraagt, maar misschien ook niet. Omdat niemand wil dat zijn werk als vanzelfsprekend wordt beschouwd, moet u voorzichtig omgaan met het verschil tussen deze twee situaties en uw verzoek dienovereenkomstig formuleren.

Wanneer mensen wordt gevraagd iets te doen op een manier die impliceert dat u denkt dat het overduidelijk hun verantwoordelijkheid is, terwijl ze daar zelf anders over denken, zorgt dat bijna altijd voor irritatie. Het toewijzen van binnenkomende issues is bijvoorbeeld een bijzonder vruchtbare grond voor dergelijke ergernissen. De deelnemers aan een project weten meestal heel goed wie expert is op welk gebied, dus als er een bugrapport binnenkomt, zijn er vaak één of twee mensen van wie iedereen weet dat zij het probleem waarschijnlijk snel kunnen oplossen. Als u het issue echter aan één van deze mensen toewijst zonder hen daarom te vragen, kunnen ze het gevoel krijgen voor het blok te worden gesteld. Ze voelen de druk van de verwachtingen van anderen, maar kunnen ook het gevoel krijgen dat ze worden gestraft voor hun expertise. Uiteindelijk is het repareren van bugs de beste manier om expertise op te doen, dus misschien zou iemand anders deze issue toegewezen moeten krijgen! (Issue trackers die issues automatisch toewijzen aan bepaalde personen op basis van informatie in de bugrapporten, roepen minder weerstand op. Iedereen weet namelijk dat het issue via een geautomatiseerd proces is toegewezen en dat er derhalve geen sprake is van druk door direct toedoen van mensen.)

Hoewel het soms goed kan zijn om de taken zo gelijkmatig mogelijk te verdelen, zijn er ook momenten waarop u de persoon die de bug het snelst kan repareren, wilt aanmoedigen om dit daadwerkelijk te doen. Omdat u geen eindeloze briefwisselingen kunt voeren voor iedere toewijzing van taken ("Zou jij bereid zijn naar deze bug te kijken?" "Ja." "OK, dan wijs ik het issue aan jou toe." "OK."), kunt u de opdracht ook toewijzen in de vorm van een verzoek, zonder druk uit te oefenen. Bijna alle issue trackers bieden de mogelijkheid om een opmerking in te voegen bij de toewijzing van een issue. Die opmerking kan er als volgt uitzien:

"Ik wijs dit aan jou toe, jrandom, omdat jij deze code het best kent. Aarzel echter niet om dit terug te geven als je er geen tijd voor hebt. (En laat me weten als je dergelijke verzoeken in de toekomst liever niet meer ontvangt.)"

Dit maakt een duidelijk onderscheid tussen het *verzoek* voor de opdracht en de *acceptatie* van de opdracht door de ontvanger. Het publiek is in dit geval niet alleen de persoon aan wie u de opdracht toewijst. Iedereen luistert namelijk mee. De hele groep ziet een openlijke bevestiging van de expertise van degene die de opdracht krijgt aangeboden, maar de boodschap is ook duidelijk dat het deze persoon vrijstaat om de verantwoordelijkheid te weigeren.

Follow-up geven aan delegeren

Wanneer u iemand hebt gevraagd iets te doen, onthoud dan dat u dit hebt gedaan en geef hier follow-up aan. De meeste verzoeken worden via publieke forums gedaan, meestal in een vorm als "Kun je zorgen voor X? Laat ons weten of je dit kunt doen. Het is geen probleem als het niet kan, we willen het alleen graag horen." Het is mogelijk dat u geen reactie hierop krijgt. Als u wel een reactie krijgt en deze negatief is, is de cirkel rond. U moet een andere manier zien te vinden om X op te lossen. Als de reactie positief is, houdt de voortgang van het issue dan in de gaten en geef commentaar op de voortgang of het gebrek eraan (mensen presteren beter als ze weten dat iemand anders waardeert wat ze doen). Als er na een paar dagen nog geen reactie is, vraag het dan opnieuw, of post een nieuw bericht dat u geen reactie hebt gekregen en op zoek bent naar iemand anders die het kan doen. U kunt het ook zelf doen, maar zorg er wel voor dat u aangeeft geen reactie te hebben ontvangen op uw eerste verzoek.

Het doel van het publiekelijk opmerken dat er geen reactie is gekomen, is niet om de persoon de vernederen. U moet uw opmerking dan ook zo formuleren dat deze niet zo over kan komen. Het doel is louter te laten zien dat u in de gaten houdt wat u hebt gevraagd en dat de reacties die u krijgt uw aandacht hebben. De kans dat mensen een volgende keer ja zeggen wordt hierdoor groter, omdat ze zien (misschien zelfs onbewust) dat u aandacht hebt voor het werk dat zij doen, gezien het feit dat een minder zichtbaar feit als het niet reageren op een bericht al door u wordt opgemerkt.

Zien waar mensen in geïnteresseerd zijn

Wat mensen ook blij maakt, is als ze merken dat anderen rekening houden met hun belangen. Algemeen geldt dat hoe meer aspecten van iemands persoonlijkheid u opmerkt en onhoudt, des te prettiger hij zich voelt en des te meer hij zal willen werken met groepen waarvan u deel uitmaakt.

Er was binnen het Subversion-project bijvoorbeeld een groot verschil tussen mensen die tot een definitieve 1.0-release wilden komen (wat uiteindelijk ook gebeurde) en mensen die voornamelijk nieuwe functies wilden toevoegen en aan interessante problemen wilden werken, maar voor wie het niet veel uitmaakte wanneer 1.0 uit zou komen. Geen van beide uitgangspunten is beter dan het andere. Het gaat hierbij gewoon om twee verschillende soorten ontwikkelaars en beide soorten doen veel werk voor het project. Maar we leerden wel snel dat het belangrijk is er *niet* vanuit te gaan dat het enthousiasme voor het uitbrengen van 1.0 door iedereen wordt gedeeld. Elektronische media kunnen heel bedrieglijk zijn. U kunt een sfeer proeven van het gezamenlijk aan één doel werken, terwijl dit in feite alleen het geval is bij de mensen met wie u hebt gepraat. Anderen kunnen namelijk geheel andere prioriteiten hebben.

Hoe meer u zich bewust bent van wat mensen van het project verwachten, des te effectiever u dingen van hen kunt verwachten. Alleen al laten zien dat u begrijpt wat ze willen, zonder daar een verzoek aan te verbinden, is nuttig. Het geeft deze persoon het gevoel niet zomaar een onderdeel te zijn van een ondefinieerbare massa.

Complimenten en kritiek

Complimenten en kritiek zijn niet elkaars tegenovergestelde; ze lijken in veel dingen juist op elkaar. Beide zijn voornamelijk vormen van aandacht, die veel effectiever zijn wanneer ze concreet zijn dan wanneer ze algemeen worden gegeven. Beide dienen te worden toegepast met een concreet doel voor ogen. Beide kunnen hun waarde verliezen. Als u iemand te veel of te vaak complimentjes geeft, verliezen ze hun waarde. Hetzelfde geldt voor kritiek, alhoewel kritiek in de praktijk vaak een reactie op iets is en daarom iets minder snel zijn waarde verliest.

Een belangrijk kenmerk van de technische cultuur is dat gedetailleerde en kalme kritiek vaak wordt ervaren als een vorm van compliment (zoals besproken in het

gedeelte ‘Onbeleefd gedrag herkennen’ in Hoofdstuk 6, *Communicatie*), omdat het impliceert dat het werk van de bekritiseerde persoon de moeite van het analyseren waard is. Om dit gevoel daadwerkelijk te kunnen geven, moet echter wel aan beide voorwaarden (*gedetailleerd* en *kalm*) zijn voldaan. Als iemand bijvoorbeeld een slordige verandering aan de code maakt, is het nutteloos (en in feite schadelijk) om hierop te reageren met alleen “dat was slordig.” Slordigheid is uiteindelijk een kenmerk van een *persoon*, niet van zijn werk, en het is belangrijk dat uw reacties gericht zijn op het werk. Het is veel effectiever om alle dingen die er mis zijn aan de verandering met tact en zonder kwade bijbedoelingen te beschrijven. Als dit de derde of vierde keer is dat dezelfde persoon onzorgvuldige veranderingen doorvoert, is het gepast om aan het einde van uw kritiek, opnieuw zonder boos te worden, duidelijk te maken dat er een patroon zichtbaar is.

Als iemand zijn veranderingen niet verbetert op grond van de kritiek is de oplossing niet het geven van nog meer of sterkere kritiek. De oplossing is dan dat de groep deze persoon niet langer op een positie laat zitten waar hij niet geschikt voor is, maar wel op zo’n manier dat hij zo min mogelijk gekwetst wordt (zie het gedeelte ‘Transities’ verderop in dit hoofdstuk voor voorbeelden). Dit komt echter zeer zelden voor. De meeste mensen kunnen goed omgaan met kritiek die specifiek en gedetailleerd is en duidelijke (zelfs onuitgesproken) verwachtingen voor verbetering bevat.

Met complimenten kunt u natuurlijk niemand kwetsen. Dat betekent echter niet dat u er minder zorgvuldig mee om hoeft te gaan dan met kritiek. Een compliment is een hulpmiddel. Voordat u er een geeft, zou u zichzelf af moeten vragen *waarom* u dat wil doen. Normaal gesproken is het geen goed idee mensen complimentjes te geven voor iets dat ze altijd al doen of voor acties die normaal zijn, of verwacht kunnen worden omdat iemand deel uitmaakt van de groep. Als u daaraan gaat beginnen, is het einde zoek: moet u *iedereen* complimenteren voor het doen van de dingen die van hen worden verwacht? Als u namelijk bepaalde mensen overslaat zullen ze zich afvragen waarom. Het is veel beter zuinig om te gaan met complimenten en dankwoordjes, en ze te gebruiken als reactie op ongebruikelijke of onverwachte inspanningen, met de intentie om meer van dergelijke inspanningen aan te moedigen. Als een deelnemer permanent lijkt te zijn overgestapt naar een hoger productiviteitsniveau, pas de drempel voor uw complimentjes dan dienovereenkomstig aan. Herhaalde complimenten voor normaal gedrag verliezen na verloop van tijd sowieso hun betekenis. In plaats daarvan zou die persoon aan moeten voelen dat het hogere productiviteitsniveau nu beschouwd wordt als normaal en voor de hand liggend, en dat alleen werk dat daarboven uitstijgt bijzondere aandacht verdient.

Dit wil natuurlijk niet zeggen dat de bijdragen van deze persoon niet hoeven worden gewaardeerd. Vergeet echter niet dat als het project correct is opgezet, alles wat deze persoon doet zichtbaar is. De hele groep ziet dus wat hij allemaal doet en hij weet dat de hele groep dat ziet. Er zijn ook andere manieren om uw waardering voor iemands werk te laten blijken behalve een complimentje. U kunt, tijdens het bespreken van een bepaald onderwerp, terloops laten vallen dat hij veel werk heeft verzet op een bepaald gebied en dus de expert is. U kunt hem om advies vragen over een bepaald deel van de code, of, en dat is misschien wel het meest effectief, u kunt op een opvallende manier gebruik maken van wat hij heeft gedaan, zodat hij ziet dat anderen vertrouwen hebben in zijn werk. Het is waarschijnlijk niet nodig om deze dingen op een berekenende manier te doen. Mensen die regelmatig een belangrijke bijdrage leveren aan een project weten dit zelf en nemen vanaf het begin al een invloedrijke positie in. Er zijn meestal geen expliciete stappen nodig om hiervoor te zorgen, behalve als u het gevoel hebt dat iemand, om wat voor reden dan ook, niet voldoende wordt gewaardeerd.

Territoriumdrang voorkomen

Wees op uw hoede voor deelnemers die proberen de exclusieve eigenaar van bepaalde delen van het project te worden en die al het werk voor die delen zelf te lijken willen doen, in die mate dat ze op agressieve wijze werkzaamheden overnemen waar anderen aan beginnen. Dergelijk gedrag kan in het begin positief lijken. Op het eerste gezicht lijkt het namelijk alsof deze persoon meer verantwoordelijkheid neemt en verhoogde activiteit laat zien op een bepaald vlak. Op de lange duur is dit echter destructief. Wanneer mensen het gevoel hebben dat een bepaald gebied ‘verboden toegang’ is, dan blijven ze daar ook weg. Dit leidt tot minder reviews voor dat gebied en grotere kwetsbaarheid, omdat de eenzame ontwikkelaar een zwakke schakel in de ketting wordt. Erger nog, het verstoort de coöperatieve en egalitaire sfeer van het project. Het principe dat iedere ontwikkelaar mag helpen bij iedere taak op ieder moment mag niet worden opgegeven. Natuurlijk werken dingen in de praktijk vaak wat anders. Mensen hebben nou eenmaal gebieden waarop ze meer of minder invloed hebben. En mensen die ergens minder verstand van hebben, respecteren de experts voor bepaalde delen van het project. Het belangrijkste hierbij is echter dat dit vrijwillig is. Informele autoriteit wordt toegekend op basis van competentie en bewezen inzicht, maar het mag nooit actief worden *toegeëigend*. Zelfs als de persoon die de autoriteit wil zijn echt zeer competent is, is het nog steeds essentieel dat hij op dit vlak de informele autoriteit is op basis van de consensus van de groep, en dat deze autoriteit er nooit toe kan leiden dat anderen worden uitgesloten van dat gebied.

Natuurlijk is het weigeren of aanpassen van iemands werk om technische redenen een heel ander verhaal. Daarbij is de doorslaggevende factor de inhoud van de werkzaamheden, niet wie toevallig op dat moment het werk controleert. Het kan best gebeuren dat dezelfde persoon het meeste controlewerk doet voor een bepaald gebied, maar zolang hij nooit probeert te voorkomen dat iemand anders dit werk ook doet, is er waarschijnlijk niets aan de hand.

Om territoriumdrang - zelfs de eerste tekenen ervan - tegen te gaan, hebben veel projecten de stap genomen om de namen van auteurs of de persoon die is toegewezen voor het onderhoud uit de bronbestanden te schrappen. Ik ben het van harte eens met deze aanpak. We doen dit ook binnen het Subversion-project en het is min of meer officieel beleid bij de Apache Software Foundation. ASF-deelnemer Sander Striker legt het als volgt uit:

Bij de Apache Software foundation ontmoedigen we het gebruik van auteur-tags in de broncode. Er zijn meerdere redenen hiervoor, naast de juridische consequenties. De essentie van coöperatieve ontwikkeling is als groep werken aan projecten en

als groep zorg dragen voor dat project. Iemand bepaalde resultaten toeschrijven is prima. Dat moet ook worden gedaan, maar wel op een manier die geen oneigenlijke rechten toekent, zelfs niet impliciet. Er is geen duidelijke grens aan te geven wanneer er wel of geen auteur-tag moet worden toegevoegd. Voeg je je naam toe als je een opmerking wijzigt? Wanneer je een fix van één regel toevoegt? Verwijder je de auteur-tag van een ander als je de code herschrijft zodat het er voor 95% anders uit komt te zien? Wat doe je met mensen die aan ieder bestand sleutelen en er precies zoveel aan veranderen als voldoende is voor een auteur-tag, zodat hun naam overal bijstaat?

Er zijn betere manieren om mensen erkentelijk te zijn en wij geven er de voorkeur aan om die manieren te gebruiken. Vanuit technisch oogpunt zijn auteur-tags onnodig. Als je wilt weten wie een bepaald deel van de code heeft geschreven, kan het versiebeheersysteem uitkomst bieden. Auteur-tags hebben ook de neiging te verouderen. Wil je echt dat mensen persoonlijk contact met je opnemen over een stuk code dat je vijf jaar geleden hebt geschreven en waar je eerlijk gezegd niet meer aan herinnerd wilt worden?

De broncodebestanden van een softwareproject vormen de kern van de identiteit ervan. Hierin zou weerspiegeld moeten worden dat de ontwikkelaarsgemeenschap er in zijn geheel verantwoordelijk voor is en dat het niet kan worden onderverdeeld in kleine koninkrijkjes.

Mensen gebruiken soms als argument voor het gebruik van auteur-tags in de bronbestanden dat dit zichtbare erkentelijkheid biedt aan degenen die het meeste werk hebben verzet. Er kleven twee bezwaren aan deze redenering. Allereerst leiden de tags onvermijdelijk tot de ongemakkelijk vraag hoeveel werk iemand moet doen om zijn eigen naam ook op de lijst te krijgen. Ten tweede haalt het de begrippen erkentelijkheid en autoriteit door elkaar. In het verleden gedaan werk impliceert niet dat iemand ook eigenaar is van het onderdeel waar het werk is gedaan. Het is echter erg moeilijk, zo niet onmogelijk, om een dergelijke implicatie te vermijden als de individuele namen bovenaan de bronbestanden vermeld staan. In ieder geval kan auteursinformatie ook worden verkregen uit de logbestanden van het versiebeheersysteem en andere externe mechanismes zoals archieven van mailinglijsten, zodat er geen informatie verloren gaat als dit uit de bronbestanden zelf wordt geschrapt.[25]

Als het project besluit geen individuele namen op te nemen in de bronbestanden, zorg er dan voor niet te overdrijven. Veel projecten hebben bijvoorbeeld een `contrib`/-gebied, waar kleine tools en ondersteunende scripts worden bewaard, vaak geschreven door mensen die niet op andere wijze bij het project zijn betrokken. Het is prima wanneer dergelijke bestanden auteursnamen bevatten, omdat ze niet werkelijk worden onderhouden door het project als geheel. Aan de andere kant, als een ingebrachte tool wordt gebruikt door andere mensen van het project wilt u het misschien verplaatsen naar een minder geïsoleerde locatie en, als de oorspronkelijke auteur daarmee akkoord gaat, de naam van de auteur verwijderen zodat de code lijkt op alle andere door de gemeenschap onderhouden bronnen. Als de auteur daar een slecht gevoel bij krijgt, is een compromis ook acceptabel, bijvoorbeeld:

```
# indexclean.py: oude gegevens verwijderen uit de Scanley-index.
#
# Oorspronkelijke auteur: K. Maru
   <kobayashi@yetanotheremailservice.com>
# Nu onderhouden door: het Scanley-project
   <http://www.scanley.org/>
# en K. Maru.
#
# ...
```

Het is echter beter dergelijke compromissen waar mogelijk te vermijden. De meeste auteurs kunnen trouwens best worden overgehaald, omdat ze blij zijn als hun bijdrage een meer integraal deel gaat uitmaken van het project.

Het belangrijkste is dat er geen duidelijke lijn is tussen de kern van het project en de omringende delen. De belangrijkste broncodebestanden voor de software maken duidelijk deel uit van de kern en moeten worden beschouwd alsof ze worden onderhouden door de gemeenschap. Aan de andere kant kunnen ondersteunende hulpmiddelen of delen van de documentatie het werk zijn van individuen, die het voornamelijk alleen onderhouden, hoewel hun werk kan worden geassocieerd en misschien zelfs worden gedistribueerd met het project. Het is niet nodig om een algemene regel toe te passen op ieder bestand, zolang het principe dat door de gemeenschap onderhouden bronnen geen individueel eigendom kunnen worden maar blijft staan.

De automatiseringsratio

Probeer niet door mensen te laten doen wat ook door machines gedaan kan worden. Als vuistregel kan worden aangehouden dat om een doorsnee taak één keer uit te voeren een geautomatiseerde uitvoering minstens tienmaal de waarde heeft van de inspanning van een ontwikkelaar. Voor vaak voorkomende of zeer complexe taken kan deze ratio zelfs oplopen tot twintig of zelfs meer.

Beschouw uzelf als een ‘projectmanager’, in plaats van gewoon één van de ontwikkelaars; dat kan hier goed van pas komen. Soms zijn de afzonderlijke ontwikkelaars te veel betrokken bij de werkzaamheden op lagere niveaus om het grote geheel te kunnen overzien en te zien dat iedereen veel tijd verspilt aan het handmatig uitvoeren van taken die ook geautomatiseerd kunnen worden. En zelfs de mensen die zich dit wel realiseren, nemen vaak niet de tijd om dit probleem op te lossen. Omdat iedere afzonderlijke taak niet als een grote last wordt ervaren, ondervindt niemand er ooit genoeg last van om er iets aan te doen. Wat automatisering noodzakelijk maakt is het feit dat de kleine last wordt vermenigvuldigd met het aantal keren dat een ontwikkelaar ertegenaan loopt. En *dat* aantal wordt weer vermenigvuldigd met het aantal ontwikkelaars.

Ik gebruik de term ‘automatisering’ hier in de breedste zin van het woord. Het betekent niet alleen herhaalde acties met één of twee veranderende variabelen, maar iedere vorm van technische infrastructuur die bedoeld is om mensen te ondersteunen. De minimumnorm aan automatiseringstools die nodig zijn om een project van-

daag de dag te runnen, wordt beschreven in Hoofdstuk 3, *Technische infrastructuur*, maar ieder project kan zijn eigen specifieke problemen hebben. Een groep die samenwerkt aan de documentatie kan behoefte hebben aan een website waar altijd de laatste versie van de documenten te zien is. Omdat documentatie vaak wordt geschreven in een markup-taal zoals XML, kan er, soms zelfs erg complexe, compilatie nodig zijn voor het creëren van toonbare downloadbare documenten. Het opzetten van een website waarop deze compilatie automatisch wordt gedaan na iedere commit kan moeilijk en tijdrovend zijn, maar is de moeite waard, zelfs als het een dag of meer kost om alles op te zetten. Het totaalvoordeel van de beschikbaarheid van up-to-date pagina's op ieder moment is enorm, terwijl de nadelen van het *niet* hebben van een dergelijke tool maar een klein ongemak lijken op een gegeven moment voor een gegeven ontwikkelaar.

Het opzetten van zo'n website voorkomt niet alleen onnodig tijdverlies, maar ook de frustraties die ontstaan wanneer mensen fouten maken (wat onvermijdelijk zal gebeuren) bij het handmatig uitvoeren van complexe procedures. Het uitvoeren van meerdere van tevoren vastgestelde acties is precies waar computers voor bedoeld zijn. Gebruik uw mensen voor meer interessante taken.

Geautomatiseerd testen

Het draaien van geautomatiseerde tests is nuttig voor ieder softwareproject, maar in het bijzonder voor een open source-project omdat ze (en met name regressietests) ontwikkelaars de vrijheid geven code te wijzigen op plaatsen waar ze niet zo thuis in zijn. Op deze manier wordt verkennend ontwikkelen aangemoedigd. Omdat het handmatig opsporen van fouten in de code erg moeilijk is (iemand moet in feite gewoon maar raden waar hij mogelijk een breuk heeft veroorzaakt en met experimenten bewijzen dat dat niet het geval is) besparen geautomatiseerde opsporingsmethodes van deze breuken het project *erg veel* tijd. Het zorgt er tevens voor dat mensen zich minder druk maken over het herschrijven van grote lappen code, zodat het ook bijdraagt aan de onderhoudbaarheid van het project op de lange termijn.

Regressietests

Een *regressietest* is een test waarmee wordt gecontroleerd of eerder gerepareerde bugs niet opnieuw optreden. Het doel van regressietests is om de kans te verkleinen, dat veranderingen aan de code op andere plaatsen in de software onverwachte fouten veroorzaken. Wanneer een softwareproject steeds groter en complexer wordt, wordt ook de kans op dergelijke onverwachte bijwerkingen steeds groter. Goed codeontwerp kan de snelheid waarmee het aantal veranderingen toeneemt, verlagen maar het kan het probleem niet helemaal oplossen.

Als gevolg daarvan gebruiken veel projecten een *testpakket*. Dat is een afzonderlijk programma dat de software van het project aanroept op een manier waarvan bekend is dat het in het verleden specifieke bugs veroorzaakte. Als het testpakket erin slaagt één van deze bugs opnieuw tevoorschijn te laten komen, staat dit bekend onder de naam *regressie*, wat betekent dat iemands verandering onverwacht de reparatie van een oude bug ongedaan heeft gemaakt.

Zie ook http://nl.wikipedia.org/wiki/Regressietest.

Regressietests zijn geen wondermiddel. In de eerste plaats werken ze het best voor programma's met interfaces in een batch-stijl. Software die primair wordt toegepast via grafische gebruikersinterfaces is veel moeilijker met een programma aan te sturen. Een ander probleem is dat de opzet van een regressietestpakket zelf vaak erg complex kan zijn, met een eigen leercurve en onderhoudsproblemen. Het terugbrengen van de complexiteit ervan is één van de meest nuttige resultaten die u kunt boeken, hoewel dat een aanzienlijke hoeveelheid tijd kan kosten. Hoe makkelijker het is om nieuwe tests aan het pakket toe te voegen, des te meer ontwikkelaars dat ook zullen doen en des te minder bugs er in de release blijven zitten. Iedere inspanning om het schrijven van de tests makkelijker te maken, wordt tijdens de levensduur van het project meervoudig beloond.

Veel projecten hanteren als regel: '*Don't break the build*!' Dat betekent: commit geen verandering waardoor de software niet meer gecompileerd of gedraaid kan worden. Als u de persoon bent die de breuk heeft veroorzaakt, is dat vaak aanleiding voor wat gêne en plagerijtjes. Projecten met regressietestpakketten hanteren vaak een regel die hieruit voortvloeit: commit geen veranderingen waarop tests vastlopen. Dergelijke problemen zijn het makkelijkst op te sporen met het automatisch 's nachts draaien van het hele testpakket, waarbij het resultaat naar de ontwikkelingslijst of naar een speciale mailinglijst voor testresultaten wordt gemaild. Ook dit is een goed voorbeeld van een geautomatiseerd proces dat de moeite waard is.

De meeste vrijwillige ontwikkelaars zijn bereid extra tijd te spenderen aan het schrijven van regressietests, mits het testsysteem begrijpelijk is en makkelijk om mee te werken. Het combineren van veranderingen en tests wordt beschouwd als een verantwoordelijke keuze en is tevens een goede gelegenheid voor samenwerking. Vaak verdelen twee ontwikkelaars het werk voor een bugfix onder elkaar, waarbij de één de fix zelf schrijft en de ander de test. De tweede ontwikkelaar heeft uiteindelijk vaak meer te doen en omdat het schrijven van een test al minder bevredigend is dan de feitelijk fix van de bug, is het van groot belang dat het testpakket deze klus niet nog moeilijker maakt dan hij al is.

Sommige projecten gaan zelfs nog een stapje verder en vereisen dat *iedere* bugfix of nieuwe functie wordt vergezeld van nieuwe test. Of dit een goed idee is of niet hangt van vele factoren af: de aard van de software, de samenstelling van het ontwikkelingsteam en de moeilijkheidsgraad van het schrijven van nieuwe tests. Het CVS-project (http://www.cvshome.org/) hanteert deze regel al heel lang. In theorie is het een prima beleid, omdat CVS versiebeheersoftware is en daarom risico's vermijdt op het mogelijk blijvend beschadigen of verkeerd behandelen van gegevens van de gebruikers. Het probleem in de praktijk is dat het regressietestpakket van CVS bestaat uit één enkel gigantische shellscript (met de amusante naam `sanity.sh`), dat moeilijk te lezen is en moeilijk is aan te passen of uit te breiden. De moeilijkheid van het toevoegen van nieuwe tests, in combinatie met de vereiste dat patches worden voorzien van nieuwe tests, betekent dus in feite dat CVS het maken van patches ontmoedigt. Toen ik aan CVS werkte zag ik dat mensen begonnen aan patches voor de eigen code van CVS en die soms zelfs afmaakten, maar de moed opgaven zodra ze hoorden dat ze een nieuwe test aan `sanity.sh` moesten toevoegen.

Het is normaal dat het schrijven van een nieuwe regressietest meer tijd kost dan het repareren van de oorspronkelijke bug. CVS voerde dit fenomeen echter wel heel ver door. Mensen waren soms uren kwijt aan het ontwerpen van een correcte test en kregen dat dan niet voor elkaar, gewoon omdat er te veel onvoorspelbare moeilijkheden zitten in het wijzigen van een Bourne-shellscript van 35.000 regels. Zelfs ervaren CVS-ontwikkelaars mopperden vaak wanneer ze een nieuwe test moesten toevoegen.

Deze situatie was het gevolg van de tekortkoming van onze kant om rekening te houden met de automatiseringsratio. Het is waar dat omschakelen naar een testkader, of het nu een zelfgebouwd of een kant-en-klaar product is, veel inspanning zou hebben gekost.[26] Het niet nemen van deze moeite heeft het project echter in de loop der jaren veel meer gekost. Hoeveel bugfixes en nieuwe functies zijn nu *niet* in CVS beschikbaar, alleen door de obstakels van het onpraktische testpakket? Het is niet precies te zeggen, maar het aantal is beslist vele malen groter dan het aantal bugfixes of functies dat ontwikkelaars niet hebben kunnen implementeren omdat ze een nieuw testsysteem moesten ontwikkelen (of een kant-en-klaarsysteem moesten implementeren). De klus had slechts een afgebakende hoeveelheid tijd gekost, terwijl de nadelen van het gebruik van het bestaande testpakket voor altijd door blijven werken als niemand ingrijpt.

Het punt is niet dat het hebben van strenge voorwaarden voor het schrijven van tests een slechte zaak is, noch dat het schrijven van een testsysteem in de vorm van een Bourne-shell per definitie slecht is. Het zou prima kunnen werken, afhankelijk van de manier waarop het is ontworpen en wat er getest moet worden. Het punt is dat wanneer het testsysteem een groot obstakel gaat vormen voor de ontwikkeling er iets aan gedaan moet worden. Ditzelfde geldt voor ieder routineproces dat uitmondt in een obstakel of knelpunt.

Behandel iedere gebruiker als een mogelijke vrijwilliger

Iedere interactie met een gebruiker biedt de kans een nieuwe vrijwilliger te werven. Wanneer een gebruiker de tijd neemt om een post te plaatsen op één van de mailinglijsten van het project, of om een bugrapport in te dienen, dan geeft hij al aan dat de kans dat hij bij het project betrokken zal raken groter is dan bij andere gebruikers (van wie het project nooit iets zal horen). Ga in op deze kans. Als hij een bug meldt, bedank hem dan voor het rapport en vraag hem of hij wil proberen de bug te repareren. Als hij heeft geschreven dat er een belangrijke vraag ontbreekt in de FAQ of dat de documentatie van het programma ergens onvolledig is, geef dan openlijk het probleem toe (ervan uitgaande dat het werkelijk bestaat) en vraag of hij geïnteresseerd is om het ontbrekende deel zelf te schrijven. Natuurlijk zal de gebruiker in veel gevallen aarzelen. Maar het kost niet veel moeite om het te vragen en iedere keer dat u dat doet, herinnert u andere lezers op dat forum eraan dat iedereen bij het project betrokken kan zijn.

Beperk uzelf niet tot het werven van alleen nieuwe ontwikkelaars en documentatieschrijvers. Zelfs het trainen van mensen in het schrijven van bugrapporten is uiteindelijk de moeite meer dan waard, als u niet te veel tijd spendeert aan ieder individu en als zij daarna meer bugrapporten gaan indienen. De kans hierop is groter wanneer ze een opbouwende reactie hebben gekregen op hun eerste rapport. Een opbouwende reactie hoeft niet de fix voor die bug te zijn, hoewel dat wel altijd het meest ideaal zou zijn. Het kan ook een vraag om meer informatie zijn of alleen een bevestiging dat het gemelde gedrag van de software daadwerkelijk een bug is. Mensen willen graag dat er naar ze geluisterd wordt. Bovendien willen ze ook graag dat hun bugs worden gerepareerd. U kunt hun dit tweede misschien niet altijd op stel en sprong geven, maar u (of liever gezegd, het project) kan het eerste wel geven.

Een voortvloeisel hiervan is dat ontwikkelaars niet geïrriteerd moeten reageren op mensen die goedbedoelde maar vage bugrapporten indienen. Dit is één van mijn persoonlijke irritaties. Ik zie ontwikkelaars op diverse mailinglijsten van open source-projecten dit steeds weer doen en de schade die dit aanricht, is duidelijk. Een ongelukkige nieuweling dient een onbruikbaar rapport in:

"Hallo, ik krijg Scanley niet aan de praat. Iedere keer dat ik het opstart geeft het alleen foutmeldingen. Hebben anderen dit probleem ook?"

Eén van de ontwikkelaars, die dit soort rapporten al duizenden keren heeft gezien maar zich niet realiseert dat de nieuweling dat niet heeft, reageert zo:

"En wat worden wij geacht te doen met zo weinig informatie? Jemig. Geef dan ten minste wat meer informatie, zoals de versie van Scanley, je besturingssysteem en de foutmeldingen."

Deze ontwikkelaar is niet in staat zich in de gebruiker te verplaatsen en realiseert zich ook niet welk effect een dergelijke reactie zou kunnen hebben op de *andere* mensen die de discussie volgen. Natuurlijk weet een gebruiker, die geen ervaring heeft als programmeur en ook niet met het rapporteren van bugs, niet hoe hij een bugrapport moet indienen. Hoe moet u dan wel omgaan met dergelijke personen? Leer het ze! En doe dat op zo'n manier dat ze terugkomen om nog meer te leren.

"Vervelend dat het niet werkt. We hebben meer informatie nodig om uit te kunnen zoeken wat er aan de hand is. Wil je ons je versie van Scanley, je besturingssysteem en de exacte tekst van de foutmelding laten weten? Het beste dat je kunt doen is een transcriptie sturen waarin precies staat welke commando's je hebt uitgevoerd en welke output dit opleverde. Zie http://www.scanley.org/how_to_report_a_bug.html voor meer informatie."

Deze manier van reageren is veel effectiever voor het krijgen van informatie van de gebruiker, omdat de reactie is geschreven met de gebruiker in het achterhoofd. Allereerst drukt het antwoord begrip uit: *jij had een probleem en we leven met je mee*. (Dit is niet nodig voor iedere reactie op een bugrapport; dit hangt af van de ernst van het probleem en hoe aangeslagen de gebruiker erdoor lijkt te zijn.) Ten tweede vertelt u de gebruiker hoe hij een bugrapport moet indienen dat voldoende gedetailleerd is om bruikbaar te zijn, zonder hem te kleineren. Veel gebruikers realiseren zich bijvoorbeeld niet dat "vertel ons de foutmelding" betekent "vertel ons de exacte tekst van de foutmelding, zonder dingen weg te laten of samen te vatten."

De eerste keer dat u met een dergelijke gebruiker te maken krijgt, moet u hier heel duidelijk over zijn. Als laatste biedt deze reactie ook een verwijzing naar meer gedetailleerde en volledige instructies over het rapporteren van bugs. Als u zorgt voor succesvolle communicatie met de gebruiker zal hij vaak de tijd nemen dit document ook werkelijk te lezen en te doen wat erin staat. Dit betekent uiteraard dat u dit document van tevoren klaar moet hebben liggen. Er moeten duidelijke instructies in staan over het soort informatie dat uw ontwikkelingsteam nodig heeft in ieder bugrapport. In het ideale geval ontwikkelt ook dit document zich na verloop van tijd in reactie op dingen die mensen weglaten en onjuiste rapporten die gebruikers voor uw project indienen.

De instructies voor het rapporteren van bugs van het Subversion-project zijn een mooi standaardvoorbeeld hiervan (zie Bijlage D, *Voorbeeldinstructies voor het rapporteren van bugs*). Ze eindigen overigens met de vraag of de persoon een patch wil maken om de bug te repareren. De reden hiervoor is niet dat deze vraag zal leiden tot een groter aantal patches per bugrapport. De meeste gebruikers die in staat zijn bugs te repareren, weten al dat een patch altijd welkom is en hoeven daar niet om te worden gevraagd. De werkelijke bedoeling hiervan is om alle lezers, en met name de lezers die nieuw zijn bij het project of bij open source-software in het algemeen, eraan te herinneren dat het project afhankelijk is van vrijwillige bijdragen. In feite zijn de huidige ontwikkelaars van het project niet méér verantwoordelijk voor het repareren van de bug dan degene die hem heeft gerapporteerd. Dit is een belangrijk punt, waarvan veel nieuwe gebruikers zich niet bewust zijn. Zodra ze zich dit realiseren, is de kans dat ze helpen bij het realiseren van de fix groter, misschien niet door middel van het schrijven van code maar wel met een meer gedetailleerd reproductierecept of door aan te bieden fixes van anderen te testen. Het is de bedoeling om iedere gebruiker zich te laten realiseren dat er geen inherent verschil is tussen hemzelf en de mensen die aan het project werken, dat het alleen maar gaat om de vraag hoeveel tijd en moeite iemand investeert, niet om de vraag wie iemand is.

Het advies om niet geïrriteerd te reageren, geldt niet ten aanzien van onbeleefde gebruikers. Het gebeurt een enkele keer dat mensen bugrapporten of klachten indienen waarmee ze zich, ongeacht de inhoud, uiterst neerbuigend uitlaten over de tekortkomingen van het project. Vaak zijn dergelijke mensen afwisselend beledigend en vlijend, zoals deze persoon, die dit bericht plaatste op een mailinglijst van Subversion:

"Waarom zijn er na bijna 6 dagen nog steeds geen binaries gepost voor het windows-platform?!? Het is iedere keer hetzelfde liedje en erg frustrerend. Waarom zijn deze dingen niet geautomatiseerd, zodat ze direct beschikbaar zijn?? Wanneer je een "RC"-build post, lijkt het me dat je wilt dat gebruikers de build testen, maar toch geven jullie geen enkele manier om dit te doen. Waarom een inweekperiode inlassen als jullie geen middelen bieden om te testen??"

De eerste reactie op deze nogal opruiende post was verrassend beheerst. Mensen gaven aan dat het project beleid heeft gepubliceerd over het niet bieden van officiële binary's en zeiden, met uiteenlopende irritatieniveaus, dat hij zou moeten aanbieden deze zelf te maken als ze voor hem zo belangrijk zijn. Geloof het of niet, maar zijn volgende bericht begon hiermee:

"Allereerst wil ik graag zeggen dat Subversion geweldig is en ik waardeer de inspanningen van iedereen die erbij betrokken is ten zeerste." [...]

... maar vervolgens begon hij *opnieuw* het project af te katten omdat het geen binary's leverde. Hij bood nog steeds niet aan hier zelf iets aan te doen. Hierna vielen er ongeveer vijftig mensen vreselijk over hem heen, en ik kan niet zeggen dat ik het daarmee oneens was. Het 'nultolerantiebeleid' wat betreft onbeleefdheid, zoals beschreven in het gedeelte 'Grofheden in de kiem smoren' in Hoofdstuk 2, *Aan de slag* heeft betrekking op mensen met wie het project voortdurend contact heeft (of zou willen hebben). Als iemand echter vanaf het begin duidelijk maakt dat hij voortdurend gal zal gaan spuwen, heeft het geen zin hem het gevoel te geven dat hij welkom is.

Dergelijke situaties zijn gelukkig zeldzaam en zelfs merkbaar zeldzamer in projecten die moeite doen gebruikers vanaf het eerste contact op een constructieve en beleefde manier bij het project te betrekken.

8.2 ZOWEL MANAGEMENTTAKEN ALS TECHNISCHE TAKEN DELEN

Deel zowel de managementverplichtingen als de technische verplichtingen van het project met anderen. Naarmate het project complexer wordt, gaat er steeds meer tijd zitten in het managen van mensen en informatiestromen. Er is geen enkele reden om deze verantwoordelijkheid niet met anderen te delen. Het delen betekent ook niet per definitie dat er een hiërarchie moet zijn. In de praktijk is er eerder sprake van een netwerk van gelijken dan van een soort militaire gezagsstructuur.

Soms worden managementfuncties geformaliseerd, soms gebeurt dit ook spontaan. In het Subversion-project hebben we een patchmanager, een vertaalmanager, documentatiemanagers, (onofficiële) issuemanagers en een releasemanager. Voor sommige functies hebben we een beslissing genomen; anderen ontstonden vanzelf. Als het project nog verder groeit, verwacht ik dat er nog meer functies bij zullen komen. Verderop zullen we deze functies, en een aantal andere, nader bestuderen (behalve de releasemanager, die al behandeld is in het gedeelte 'Releasemanager' en het gedeelte 'De eigenaar van de release als dictator' eerder in dit hoofdstuk).

Wanneer u de functiebeschrijvingen leest, dan zult u merken dat voor geen van de functies exclusieve controle over het betreffende onderwerp vereist is. De issuemanager verbiedt anderen niet om veranderingen aan te brengen in de issuedatabase. De FAQ-manager staat er niet op dat hij de enige is die de FAQ's mag wijzigen enz. Deze functies draaien om verantwoordelijkheid zonder monopoly. Een belangrijk onderdeel van de functie van iedere manager is om te zien welke andere mensen op dat terrein werken en hen trainen de dingen te doen zoals de manager ze doet, zodat de inspanningen van alle mensen elkaar versterken en niet met elkaar in conflict

komen. Managers van bepaalde onderdelen zouden ook de procedures waarmee zij hun werk doen op papier moeten zetten, zodat iemand anders de draad direct kan oppakken wanneer iemand vertrekt.

Soms is er sprake van een conflict. Twee of meer mensen willen dezelfde functie. Er is geen juiste manier om hiermee om te gaan. U kunt voorstellen dat iedere kandidaat een voorstel schrijft (een 'sollicitatie') en alle committers te vragen om te stemmen welke de beste is. Dit is echter lastig en kan mogelijk tot ongemakkelijke situaties leiden. Persoonlijk zie ik meer in de techniek om de kandidaten gewoon te vragen de kwestie onderling op te lossen. Normaal gesproken lukt dat en zullen beiden eerder tevreden zijn met de uitkomst dan wanneer de beslissing van buitenaf aan hen opgedrongen wordt.

Patchmanager

In een open source-softwareproject dat veel patches binnenkrijgt, kan het bijhouden welke patches zijn ontvangen en wat erover is beslist een ware nachtmerrie zijn, vooral als dit niet centraal wordt gedaan. De meeste patches komen binnen op de ontwikkelingslijst van het project (hoewel sommige ook in de issue tracker of op een externe website op kunnen duiken) en er is een aantal verschillende routes die een patch kan afleggen nadat hij is binnengekomen.

Soms doet iemand een review van de patch, vindt een probleem en stuurt het terug naar de oorspronkelijke auteur om te worden opgelapt. Dit leidt meestal tot een zich herhalend proces (dat zichtbaar is op de mailinglijst) waarbij de oorspronkelijke auteur net zo lang gereviseerde versies van de patch post tot de reviewer niets meer kan vinden. Het is niet altijd makkelijk te zien wanneer dit proces afgerond is. Als de reviewer de patch commit is dit duidelijk het einde van de cyclus. Maar als hij dat niet doet, kan het zijn omdat hij daar geen tijd voor heeft, of zelf geen commit access heeft en hij niemand van de andere ontwikkelaars zo ver heeft kunnen krijgen om het voor hem te doen.

Een andere veel voorkomende reactie op een patch is een algemene discussie die niet noodzakelijkerwijs over de patch zelf gaat, maar over de vraag of het concept van de patch goed is. De patch kan bijvoorbeeld een bug repareren, maar het project repareert de bug liever op een andere manier, als onderdeel van de oplossing van een meer algemene groep problemen. Vaak is dit niet van tevoren bekend en is het de patch die leidt tot deze ontdekking.

Het komt ook wel eens voor dat een ingediende patch helemaal geen reactie krijgt. Meestal komt dit doordat geen enkele ontwikkelaar *op dat moment* tijd heeft de patch te evalueren, zodat iedereen hoopt dat iemand anders dat zal doen. Omdat er geen afgesproken limiet is voor de tijd dat iemand moet wachten totdat iemand anders het issue oppikt en er intussen altijd andere prioriteiten opduiken, kan het makkelijk gebeuren dat een patch er doorheen glipt zonder dat dat iemands bedoeling was. Het project kan op die manier een bruikbare patch missen. En er zijn nog meer nadelen. Het ontmoedigt de auteur, die tijd in de patch heeft gestoken, en het zorgt ervoor dat het lijkt alsof het project de touwtjes niet goed in handen heeft, vooral in de ogen van anderen die een patch zouden willen schrijven.

De taak van de patchmanager is om ervoor te zorgen dat patches 'er niet tussendoor glippen'. Dit wordt gedaan door iedere patch te volgen totdat hij een enigszins stabiele vorm heeft bereikt. De patchmanager houdt iedere thread in de mailinglijsten volgend op een geposte patch in de gaten. Als deze uitmondt in de commit van de patch hoeft hij niets te doen. Als hij terechtkomt in een vicieuze cirkel van evalueren/herzien, die uitmondt in een definitieve versie van de patch maar niet in een commit, dan dient hij een issue in die verwijst naar de definitieve versie en naar de thread op de mailinglijst die hierover ging, zodat er in ieder geval iets permanent is vastgelegd waar ontwikkelaars later op terug kunnen komen. Als de patch betrekking heeft op een bestaand issue, dan voorziet hij het issue van de betreffende informatie in plaats van een nieuw issue te openen.

Als een patch helemaal geen reacties krijgt, wacht de patchmanager een paar dagen en reageert dan door na te vragen of iemand van plan is de patch te evalueren. Meestal komt daar wel een reactie op. Een ontwikkelaar kan aangeven dat hij denkt dat de patch niet zou moeten worden toegepast en daarvoor redenen geven, of hij kan hem evalueren, in welk geval één van de hiervoor beschreven routes wordt gevolgd. Als er nog steeds geen reactie komt, kan de patchmanager er zelf voor kiezen al dan niet een issue voor de patch in te dienen. Degene die de patch oorspronkelijk heeft ingediend krijgt echter in iedere geval een reactie.

Het hebben van een patchmanager heeft het ontwikkelingsteam van Subversion veel tijd en geestelijke energie bespaard. Zonder aangewezen persoon om de verantwoordelijkheid hiervoor te nemen vraagt iedere ontwikkelaar zich constant af of iemand anders het wel oppikt als hij geen tijd heeft om op dat moment op de patch te reageren. "Moet ik proberen dit in de gaten te houden? Als anderen de patch echter om dezelfde reden in de gaten houden, doen we onnodig dubbel werk." De patchmanager zorgt ervoor dat men zich hierover geen zorgen hoeft maken. Iedere ontwikkelaar kan zelf beslissen wat te doen op het moment dat hij de patch voor het eerst onder ogen krijgt. Als hij wil reageren met een review kan hij dat doen; de patchmanager zal zijn reactie hierop aanpassen. Als hij de patch volledig wil negeren, is dat ook prima; de patchmanager zal er voor zorgen dat de patch niet wordt vergeten.

Omdat dit systeem alleen werkt als mensen er volledig op kunnen vertrouwen dat de patchmanager doet wat hij moet doen, moet de functie formeel worden toegekend. Bij Subversion hebben we op de mailinglijsten voor zowel ontwikkelaars als voor gebruikers een advertentie geplaatst voor een patchmanager. We kregen meerdere reacties en hebben degene aangewezen die het eerst reageerde. Toen deze persoon deze functie moest neerleggen (zie het gedeelte 'Transities' verderop in dit hoofdstuk), hebben we opnieuw dezelfde procedure gevolgd. We hebben nooit geprobeerd om één functie in meerdere mensen te verenigen vanwege de extra moeilijkheden met de communicatie tussen de verschillende personen, maar bij zeer grote aantallen patches kan het voor de hand liggen om met een team van patchmanagers te werken.

Vertaalmanager

Binnen softwareprojecten kan 'vertalen' betrekking hebben op twee verschillende

dingen. Het kan slaan op het vertalen van de documentatie van de software in andere talen of het vertalen van de software zelf. Dat wil zeggen dat het programma foutmeldingen en hulpberichten weergeeft in de gewenste taal van de gebruiker. Beide zijn complexe taken, maar zodra de juiste infrastructuur is opgezet, kunnen ze grotendeels worden losgekoppeld van de andere ontwikkelingsprocessen. Omdat de beide taken op sommige punten op elkaar lijken, kan het (afhankelijk van uw project) zin hebben beide door één persoon te laten verrichten. Soms is het echter beter om dit over twee managers te verdelen.

Binnen het Subversion-project hebben we één vertaalmanager die beide doet. Hij schrijft de vertaling uiteraard niet zelf. Hij kan een handje helpen met één of twee vertalingen, maar om ze allemaal te kunnen doen zou hij tien talen moeten spreken (twaalf als we ook de dialecten meetellen)! In plaats daarvan stuurt hij een team van vrijwillige vertalers aan. Hij helpt ze de zaken onderling te coördineren en hij coördineert zelf alles tussen de teams en de rest van het project.

Een van de redenen waarom de vertaalmanager nodig is, is het feit dat vertalers een ander slag volk zijn dan ontwikkelaars. Ze hebben soms weinig tot geen ervaring met het werken in een versiebeheersysteem of zelfs maar met het werken als onderdeel van een wijdverspreid team van vrijwilligers. Op andere vlakken zijn ze vaak de beste soort vrijwilliger die u zich kunt voorstellen: mensen met kennis op een specifiek vlak die de behoefte zagen en besloten te helpen. Ze zijn over het algemeen bereid om nieuwe dingen te leren en gaan enthousiast aan de slag. Het enige dat ze nodig hebben, is iemand die ze vertelt hoe ze het moeten doen. De vertaalmanager zorgt ervoor dat de vertalingen op zo'n manier worden gedaan dat ze het reguliere ontwikkelingswerk niet in de weg lopen. Hij fungeert ook als een soort vertegenwoordiger van de groep vertalers als geheel, wanneer ontwikkelaars geïnformeerd moeten worden over technische veranderingen die nodig zijn om de vertaalslag te ondersteunen.

De belangrijkste vaardigheden die vereist zijn voor de functie zijn dus diplomatiek van aard, niet technisch. Bij Subversion hebben we bijvoorbeeld het beleid dat er aan iedere vertaling minimaal twee mensen moeten werken, omdat de tekst anders niet kan worden nagekeken. Als er een nieuwe vrijwilliger opduikt die aanbiedt Subversion te vertalen, bijvoorbeeld naar het Malagasi, dan moet de vertaalmanager hem óf in contact brengen met iemand anders die zes maanden geleden een bericht heeft gepost dat hij wel een vertaling naar het Malagasi wil doen, óf de vrijwilliger vriendelijk vragen *een andere* Malagasi-vertaler te vinden om mee samen te werken. Zodra er genoeg mensen beschikbaar zijn, kent de manager hun de juiste commit access toe, informeert hij ze over de afspraken binnen het project (zoals over hoe een logbericht moet worden geschreven) en houdt hij een oogje in het zeil om te zien of ze zich aan deze afspraken houden.

Gesprekken tussen de vertaalmanager en de ontwikkelaars, of tussen de vertaalmanager en het vertaalteam worden meestal gevoerd in de oorspronkelijke projecttaal, dat wil zeggen de taal van waaruit alle vertalingen worden gedaan. Voor de meeste open source-softwareprojecten is dit het Engels, maar het maakt niet uit welke taal het is, zolang het project het hier maar over eens is. (Engels is echter waarschijnlijk de beste taal voor projecten die een brede internationale ontwikkelaarsgemeenschap willen aantrekken.)

Gesprekken *binnen* een bepaald vertaalteam worden echter meestal gevoerd in hun gemeenschappelijke taal en het is één van de taken van de vertaalmanager om hier een speciale mailinglijst voor in het leven te roepen. Op die manier kunnen de vertalers hun werk ongestoord bespreken, zonder de mensen op de hoofdmailinglijst, waarvan de meesten de te vertalen taal toch niet begrijpen, hiermee lastig te vallen.

Internationalisatie versus lokalisatie

Internationalisatie (*I18N*) en *lokalisatie* (*L10N*) hebben beide betrekking op het proces waarmee een programma wordt aangepast aan taalkundige en culturele omgevingen anders dan de omgeving waarvoor het oorspronkelijk was geschreven. De termen zijn vaak onderling verwisselbaar, maar in feite betekenen ze niet hetzelfde, zoals te lezen is op http://en.wikipedia.org/wiki/G11n:

het verschil tussen beide is subtiel maar belangrijk. Internationalisatie is de aanpassing van producten voor *mogelijk* gebruik overal ter wereld, terwijl lokalisatie het toevoegen van speciale functies is voor gebruik op een *specifieke* plek.

Als u uw software bijvoorbeeld zo aanpast dat deze zonder verlies werkt met Unicode-tekstcoderingen (http://en.wikipedia.org/wiki/Unicode) is dat internationalisatie, omdat het niet bedoeld is voor een bepaalde taal, maar meer om met tekst uit verschillende talen te kunnen werken. Aan de andere kant is de functie die ervoor zorgt dat alle foutmeldingen in het Sloveens worden weergegeven wanneer het detecteert dat het programma in een Sloveense omgeving draait lokalisatie.

De taak van de vertaalmanager heeft dus voornamelijk betrekking op lokalisatie, niet op internationalisatie.

Documentatiemanager

Het up-to-date houden van documentatie is een klus die nooit klaar is. Iedere nieuwe functie of verbetering die aan de code wordt toegevoegd, kan leiden tot een verandering in de documentatie. Ook zult u merken dat op het moment dat de documentatie van het project een bepaald afrondingsniveau bereikt veel van de ingezonden patches betrekking hebben op de documentatie, niet op de code. Dat komt doordat er veel meer mensen zijn die bugs in geschreven tekst kunnen repareren dan in code: alle gebruikers zijn lezers, maar slechts een enkeling is ook programmeur.

Patches voor de documentatie zijn meestal veel makkelijker te evalueren en door te voeren dan patches voor de code. Er hoeft weinig tot niet te worden getest en de kwaliteit van de verandering kan snel worden vastgesteld door deze alleen na te kijken. Omdat de kwantiteit hoog is, maar de omvang van de evaluatie redelijk gering, is de administratieve overhead in verhouding met het productieve werk hoger bij documentatiepatches dan bij codepatches. Bovendien is er voor de patches waarschijnlijk enige aanpassing nodig, om te zorgen voor een consistente toon in de documentatie.

In veel gevallen zullen patches andere patches overlappen of beïnvloeden en moeten ze met het oog daarop worden aangepast voordat ze worden doorgevoerd.

Gezien de eisen die worden gesteld aan patches in de documentatie en het feit dat de hele code permanent moet worden gemonitord opdat de documentatie up-to-date blijft, is het logisch dat één persoon, of een klein team, verantwoordelijk is voor deze taak. Zij kunnen bijhouden waar en hoe de documentatie achterloopt op de software en zij kunnen procedures in het leven roepen voor de geïntegreerde verwerking van grote aantallen patches.

Natuurlijk hoeft dit andere mensen binnen het project er niet van te weerhouden eveneens patches te maken voor de documentatie, met name kleine patches, wanneer daar gelegenheid voor is. Dezelfde patchmanager (zie het gedeelte 'Patchmanager' eerder in dit hoofdstuk) kan zowel de patches in de code als in de documentatie bijhouden en ze rapporteren wanneer het ontwikkelingsteam of het documentatieteam ze nodig heeft. (Als het totale aantal patches echter te groot wordt om door één persoon te worden bijgehouden, kan het een prima eerste stap zijn om afzonderlijke patchmanagers aan te wijzen voor de code en de documentatie.) Het belangrijkste doel van een documentatieteam is dat er mensen zijn die zichzelf verantwoordelijk voelen voor het organiseren en up-to-date en consistent houden van de documentatie. In de praktijk houdt dit in dat zij de documentatie goed kennen, de gehele code in de gaten houden, de veranderingen bijhouden die *anderen* doorvoeren in de documentatie, uitkijken naar binnenkomende documentatiepatches en al deze informatiebronnen gebruiken om al het nodige te doen om de documentatie gezond te houden.

Issuemanager

Het aantal issues in de bug tracker van een project groeit rechtevenredig met het aantal mensen dat de software gebruikt. Daarom moet u er vanuit blijven gaan dat, zelfs wanneer u bugs repareert en een steeds stabieler programma biedt, het aantal openstaande issues aanzienlijk groeit. Het aantal duplicaatissues zal ook stijgen, evenals het aantal incomplete of slecht beschreven issues.

Issuemanagers helpen bij het oplossen van deze problemen door in de gaten te houden wat er in de database gebeurt en er regelmatig doorheen te lopen op zoek naar specifieke problemen. Hun meest voorkomende activiteit is waarschijnlijk het herstellen van binnenkomende issues, of omdat degene die het issue heeft gerapporteerd niet alle velden correct heeft ingevuld, of omdat het issue een duplicaat is van een issue die al in de database stond. Uiteraard zal de issuemanager efficiënter worden in het vaststellen van gedupliceerde issues naarmate hij beter op de hoogte is van de bugdatabase van het project. Dit is één van de grootste voordelen van het hebben van een klein aantal bugdatabasespecialisten boven de situatie dat iedereen dit *ad hoc* probeert te doen. Als de groep dit op een gedecentraliseerde manier probeert te doen, bouwt niemand diepgaande expertise op over de inhoud van de database.

Issuemanagers kunnen ook helpen bij het in kaart brengen van de issues en de afzonderlijke ontwikkelaars. Als er veel bugrapporten binnenkomen, kan het gebeuren dat niet iedere ontwikkelaar de mailinglijst met meldingen van issues met evenveel aandacht leest. Als echter één persoon, die het ontwikkelingsteam goed kent, een oogje in het zeil houdt voor alle binnenkomende issues, dan kan hij zo nodig bepaalde ontwikkelaars discreet wijzen op specifieke bugs. Natuurlijk moet dit gebeuren met enige tact wat betreft de andere dingen die er bij de ontwikkeling gaande zijn, en wat betreft de wensen en het karakter van degene die hij aanschrijft. Het is daarom het beste als issuemanagers zelf ook ontwikkelaars zijn.

Afhankelijk van hoe uw project de issue tracker gebruikt, kunnen issuemanagers de database ook zo vormgeven dat de prioriteiten van het project duidelijk worden. Bij Subversion plannen we bijvoorbeeld issues in voor toekomstige releases, zodat wanneer iemand vraagt "Wanneer wordt bug X gerepareerd?" we kunnen zeggen "In de tweede release vanaf nu," zelfs als we geen exacte datum kunnen noemen. De releases worden in de issue tracker weergegeven als doelversie of mijlpaal, een veld dat beschikbaar is in IssueZilla.[27] In de regel bevat iedere release van Subversion één grote nieuwe functie en een lijst met specifieke bugfixes. We wijzen de juiste doelversie toe aan alle issue die gepland staan voor die release (inclusief de nieuwe functie, ook dat is een issue), zodat mensen door de bugdatabase kunnen bladeren aan de hand van de releaseplanning. De doelversies blijven echter statisch. Omdat er nieuwe bugs binnenkomen, moeten prioriteiten soms worden bijgesteld, zodat issues moeten worden verplaatst van de ene doelversie naar de andere, om iedere release beheersbaar te houden. Ook dit kan het beste worden gedaan door mensen die een overzicht hebben van wat er allemaal in de database zit en hoe de diverse issues verband houden met elkaar.

Een andere taak van issuemanagers is vaststellen wanneer issues achterhaald raken. Soms wordt een bug per ongeluk gerepareerd als onderdeel van een verandering in de software die hier niets mee te maken heeft. Of het project verandert van gedachten over de vraag of bepaald gedrag moet worden beschouwd als een bug. Het opsporen van achterhaalde bugs is niet makkelijk. De enige manier om dit te doen is door alle issues in de database systematisch na te lopen. Het nalopen van alle issues wordt echter na verloop van tijd steeds minder haalbaar, omdat het aantal issues steeds groter wordt. Vanaf een bepaald punt is de enige manier om de database gezond te houden, het toepassen van een verdeel-en-heersstrategie: categoriseer issues direct op het moment dat ze binnenkomen en breng ze onder de aandacht van de betreffende ontwikkelaar of het betreffende team. De ontvanger is vanaf dat moment verantwoordelijk voor de issue, totdat deze is opgelost of zo nodig geschrapt. Als de database deze omvang krijgt, wordt de issuemanager meer en meer een coördinator, die steeds minder tijd spendeert aan de issues zelf en steeds meer aan de zorg dat ze bij de juiste mensen terechtkomen.

FAQ-manager

FAQ-onderhoud is een verrassend moeilijk probleem. In tegenstelling tot de meeste andere documenten in het project, waarvan de inhoud vooruit kan worden gepland door de auteurs, is de FAQ een volledig reactief document (zie Bijhouden van veelgestelde vragen (FAQ)). Hoe groot dit document ook is, u weet nog steeds niet wat de volgende toevoeging zal zijn. En omdat alle delen beetje voor beetje worden toegevoegd, wordt het document als geheel snel onsamenhangend en ongeorga-

niseerd en kan het zelfs gedupliceerde of semigedupliceerde onderdelen bevatten. Zelfs als er geen sprake is van dergelijke zichtbare problemen zijn er vaak onopgemerkte afhankelijkheidsrelaties tussen onderdelen (links die gemaakt zouden moeten worden maar die er niet zijn) omdat de gerelateerde onderdelen een jaar na elkaar zijn toegevoegd.

De functie van de FAQ-manager is tweeledig. Ten eerste zorgt hij voor de algehele kwaliteit van de FAQ's door in ieder geval op de hoogte te blijven van de onderwerpen van alle vragen. Zo kan de juiste aanpassing worden gedaan wanneer mensen nieuwe onderdelen toevoegen die duplicaten zijn van of een relatie hebben met andere onderdelen. Ten tweede houdt hij de mailinglijst van het project en andere forums in de gaten met het oog op terugkerende problemen of vragen, om nieuwe FAQ's te schrijven op basis van de input. Deze tweede taak kan nogal gecompliceerd zijn. De persoon moet in staat zijn om een thread te volgen, de kernvraag erin te herkennen, een entry voor de FAQ voor te stellen, opmerkingen van anderen erin te integreren (omdat het onmogelijk is voor de FAQ-manager om deskundig te zijn op ieder onderwerp van de FAQ) en aan te voelen wanneer het proces is afgerond zodat het item kan worden toegevoegd.

De FAQ-manager wordt over het algemeen ook de expert voor de FAQ-vormgeving. Er komen ontzettend veel dingen kijken bij het in vorm houden van een FAQ (zie het gedeelte 'Behandel alle bronnen als archieven' in Hoofdstuk 6, *Communicatie*). Wanneer willekeurige mensen veranderingen aan gaan brengen in de FAQ's kan het gebeuren dat ze enkele van deze details over het hoofd zien. Dat is geen probleem, zolang de FAQ-manager de rommel maar achter ze opruimt.

Er is allerlei open source-software beschikbaar om te helpen bij het onderhoud van FAQ-documenten. Het is prima om deze software te gebruiken, zolang het de kwaliteit van de FAQ niet nadelig beïnvloedt. Wees echter op uw hoede voor overautomatisering. Sommige projecten proberen het gehele proces van FAQ-onderhoud te automatiseren, waarbij iedereen bijdragen kan leveren en veranderingen kan aanbrengen in de FAQ, net als bij wikipedia (zie het gedeelte 'Wiki's' in Hoofdstuk 3, *Technische infrastructuur*). Ik heb dit met name zien gebeuren met Faq-O-Matic (http://faqomatic.sourceforge.net/), hoewel wat ik heb gezien ook gewoon verkeerd gebruik van de software zou kunnen zijn en niet waar Faq-O-Matic oorspronkelijk voor bedoeld was. In ieder geval moet worden gesteld dat gedecentraliseerd onderhoud van de FAQ de werklast voor het project vermindert, maar ook de kwaliteit van de FAQ. Er is niet één persoon die overzicht houdt over de gehele FAQ, er is niemand die ziet wanneer een bepaald onderwerp moet worden bijgewerkt of helemaal overbodig wordt en niemand die de relaties tussen items in de gaten houdt. Dit resulteert in een FAQ die gebruikers niet biedt wat ze nodig hebben en hen in het ergste geval misleidt. Gebruik alle mogelijke hulpmiddelen die u nodig hebt om de FAQ van uw project te onderhouden, maar laat het gemak van dergelijk hulpmiddelen u niet verleiden om de kwaliteit van de FAQ in gevaar te brengen.

Zie voor beschrijvingen en evaluaties van open source-onderhoudstools voor FAQ het artikel van Sean Michael Kerner, *The FAQs on FAQs*, op http://osdir.com/Article1722.phtml.

8.3 TRANSITIES

Het kan gebeuren dat een vrijwilliger met permanente verantwoordelijkheid, zoals de patchmanager, de vertaalmanager enz., niet meer beschikbaar is voor de functie. Dat kan zijn omdat de functie arbeidsintensiever bleek te zijn dan hij had verwacht, of het gevolg zijn van externe factoren: huwelijk, een nieuwe baby, een nieuwe werkgever of wat dan ook.

Als een vrijwilliger de werklast hierdoor niet meer aankan, wordt dit meestal niet direct opgemerkt. Het bouwt zich in kleine stapjes op en er is geen herkenbaar moment waarop hij zich realiseert dat hij de verantwoordelijkheden van zijn functie niet langer kan bolwerken. In plaats daarvan hoort de rest van het project een poosje weinig van hem. Vervolgens laat hij ineens een vlaag van activiteit zien, alsof hij zich schuldig voelt dat hij het project zo lang heeft verwaarloosd en een nachtje doorhaalt om het in te halen. Daarna hoort u weer een poos niets van hem, waarna hij misschien weer een vlaag van activiteit laat zien. Het zal echter zelden voorkomen dat hij zich ongevraagd formeel terugtrekt. De vrijwilliger deed dit werk in zijn vrije tijd. Terugtreden betekent dus dat hij openlijk moet toegeven dat hij permanent minder vrije tijd heeft gekregen. Mensen doen dat over het algemeen niet graag.

Daarom is het aan u en de anderen binnen het project om te signaleren wat zich afspeelt (of liever gezegd, wat zich niet afspeelt) en de vrijwilliger te vragen wat er aan de hand is. Deze vraag moet vriendelijk en 100% zonder schuldvraag worden gesteld. Het is uw bedoeling om informatie te krijgen, niet om deze persoon een rotgevoel te geven. Over het algemeen zou deze vraag om informatie voor het hele project zichtbaar moeten worden gesteld, maar als u weet dat er sprake is van een bijzondere reden waardoor de vraag beter privé kan worden gesteld, is dat ook prima. De belangrijkste reden om dit publiekelijk te doen is dat als de vrijwilliger antwoordt dat hij het werk niet meer kan doen, u een context heeft voor uw *volgende* publieke post: een verzoek om een nieuwe vrijwilliger voor die functie.

Soms gebeurt het dat een vrijwilliger niet in staat is om de functie uit te oefenen die hij op zich heeft genomen, maar zich hiervan niet bewust is of het niet wil toegeven. Natuurlijk kan iedereen hier in het begin moeite mee hebben, vooral als de verantwoordelijkheid complex is. Als iemand echter niet de juiste persoon is voor de taak die hij op zich heeft genomen, zelfs nadat iedereen hem alle mogelijk hulp en suggesties heeft gegeven, dan is de enige oplossing dat hij zich terugtrekt en plaatsmaakt voor iemand anders. En als deze persoon dit zelf niet inziet, dan moet iemand hem dat vertellen. Volgens mij is er maar één manier om dit te doen, maar het is een proces dat uit meerdere stappen bestaat, waarbij iedere stap belangrijk is.

Allereerst moet u zeker weten dat u niet gek bent. Bespreek het probleem privé met anderen binnen het project om te zien of ze het met u eens zijn dat het zo ernstig is als u denkt dat het is. Zelfs als u hier al zeker van bent, is het doel hiervan ook dat u anderen laat weten dat u overweegt om de persoon te vragen zich terug te trekken. Normaal gesproken zal niemand hier bezwaar tegen maken. Ze zullen zelfs blij zijn dat u deze onprettige taak op zich neemt en zij het niet hoeven doen!

Vervolgens neemt u *privé* contact op met de vrijwilliger in kwestie en vertelt u hem, vriendelijk maar direct, dat u problemen heeft gesignaleerd. Wees concreet en geef zo veel mogelijk voorbeelden. Let erop dat u aangeeft hoe mensen hebben geprobeerd hem te helpen, maar dat het probleem er niet minder van is geworden. U kunt ervan uitgaan dat het schrijven van deze e-mail veel tijd in beslag neemt. Als u echter van plan bent om een kort bericht te schrijven waarin u niet onderbouwt wat u zegt, kunt u maar beter niets zeggen. Vertel dat u graag een andere vrijwilliger zou willen zoeken voor de functie, maar geef ook aan dat er vele andere manieren zijn waarop hij een bijdrage kan blijven leveren aan het project. Zeg in dit stadium niet dat u ook met anderen hierover hebt gesproken. Niemand hoort graag dat men achter zijn rug over hem heeft gepraat.

De zaken kunnen hierna op allerlei manieren verdergaan. De meest waarschijnlijke reactie is dat hij het met u eens is, of in ieder geval niet met u in discussie wil gaan, en dat hij bereid is terug te treden. Suggereer in dat geval dat hij dit zelf bekendmaakt, waarna u een post kunt plaatsen met de vraag om een vervanger.

Het kan ook gebeuren dat hij het met u eens is dat er een probleem is, maar om een beetje meer tijd vraagt (of, in het geval van functies met afgeronde taken zoals een releasemanager, om een laatste kans). Hoe u daarop reageert is aan u, maar wat u ook doet, ga hiermee niet akkoord alleen omdat u een dergelijk redelijk verzoek niet kunt weigeren. Dit vermindert het probleem niet, het verlengt het alleen maar. U heeft vaak een hele goede reden om dit verzoek af te slaan, namelijk dat hij al een heleboel kansen heeft gehad en dat de dingen daardoor zo ver zijn gekomen. Hier een voorbeeld van een e-mail die ik heb gestuurd aan iemand die de functie van releasemanager had maar die er niet echt geschikt voor was:

```
> Als je mij door iemand anders wilt vervangen, dan zal ik mijn
functie netjes overdragen aan iemand anders.  Ik heb echter één
verzoek, waarvan ik hoop dat het niet onredelijk is.  Ik zou het
graag nog voor één release willen proberen, om mezelf te kunnen
bewijzen.

Ik heb volledig begrip voor je wens (ik heb in hetzelfde schuit-
je gezeten!), maar in dit geval lijkt me zo'n "laatste kans"
geen goed idee.

Dit is niet de eerste of tweede release, het is de zesde of de
zevende ... En ik weet dat je zelf ook niet tevreden was met de
resultaten van al deze releases (omdat we het eerder hierover
hebben gehad).  In feite hebben we deze laatste kans dus al ge-
had.

Uiteindelijk moeten we één van de pogingen beschouwen als de
laatste ... Ik denk dat [deze laatste release] die laatste moet
zijn.
```

In het ergste geval is de vrijwilliger het openlijk met u oneens. In dat geval moet u zich erbij neerleggen dat de situatie onprettig wordt en toch doorzetten. U kunt op dat moment aangeven dat u ook met anderen hebt gesproken (maar zeg nog steeds niet wie, voordat u hun toestemming daarvoor heeft, omdat deze gesprekken vertrouwelijk waren) en dat u denkt dat het niet goed is voor het project om de dingen te laten zoals ze zijn. Wees vasthoudend, maar nooit dreigend. Houd in gedachten dat voor de meeste functies de overdracht pas een feit is wanneer iemand anders eraan begint, *niet* het moment waarop de vorige persoon ermee ophoudt. Als de discussie bijvoorbeeld gaat over de functie van issuemanager kunnen u en andere invloedrijke personen binnen het project een oproep doen voor een nieuwe issuemanager. Het is in feite helemaal niet nodig dat de persoon die het voorheen deed hier volledig mee stopt, zolang hij de inspanningen van de nieuwe vrijwilliger niet dwarsboomt (bewust of onbewust).

Dat kan een verleidelijke optie lijken. In plaats van de persoon te vragen om zich terug te trekken, biedt u hem hulp aan? Waarom niet werken met twee issuemanagers, of patchmanagers, of welke functie dan ook?

Hoewel dit in theorie misschien aantrekkelijk klinkt, is het over het algemeen geen goed idee. De managersfuncties werken goed (en zijn bruikbaar) doordat ze zijn gecentraliseerd. De dingen die gedecentraliseerd kunnen worden gedaan, worden dat vaak al gedaan.

De managersfunctie delen tussen twee mensen zorgt voor extra problemen met de communicatie tussen deze twee mensen en creëert bovendien de kans dat verantwoordelijkheden niet bij de juiste persoon terechtkomen (“Ik dacht dat jij de verbandtrommel zou meenemen!" "Ik? Nee, ik dacht dat jij de verbandtrommel zou meenemen!”). Natuurlijk zijn er uitzonderingen. Soms kunnen mensen erg goed samenwerken, of is de aard van de functie zodanig dat deze makkelijk over meerdere mensen kan worden verdeeld. Maar dit helpt waarschijnlijk niet veel als u ziet dat iemand zich door een functie heen moet worstelen waarvoor hij niet geschikt is. Als hij het probleem überhaupt al had erkend, dan had hij daarvoor al wel eerder hulp gezocht. Het is sowieso niet erg respectvol als u iemand tijd laat verspillen aan werk waar niemand wat aan heeft. Als u iemand vraagt zich terug te trekken, is privacy de belangrijkste factor. Geef hem de ruimte om een beslissing te nemen zonder dat hij het gevoel heeft dat anderen toekijken en afwachten. Ik heb ooit de fout gemaakt (wat achteraf gezien een overduidelijke fout was) de drie partijen tegelijk te mailen om de releasemanager van Subversion te vragen zich terug te trekken en de taak over te dragen aan twee andere vrijwilligers. Ik had privé al met de twee nieuwe mensen gesproken en ik wist dat ze bereid waren om de verantwoordelijkheid op zich te nemen. Ik dacht dus, nogal naïef en weinig fijnbesnaard, dat ik mezelf tijd en moeite kon besparen door één e-mail naar alle drie te sturen om de overdracht in gang te zetten. Ik ging ervan uit dat de huidige releasemanager zich volledig bewust was van de problemen en dat hij de redelijkheid van mijn argumenten onmiddellijk zou inzien.

Ik had het mis. De desbetreffende releasemanager voelde zich zeer beledigd, en terecht. Het is al erg genoeg wanneer je gevraagd wordt je functie aan iemand anders over te dragen. Als dit gedaan wordt *in aanwezigheid* van de mensen aan wie je je

functie moet overdragen, is dat echter onverdraaglijk. Zodra het tot me doordrong waarom hij zich beledigd voelde, heb ik mijn excuses aangeboden. Uiteindelijk heeft hij zich op een stijlvolle manier teruggetrokken, Hij is tot op heden bij het project betrokken. Alleen voelde zich hij aanvankelijk gekwetst. Ook voor de nieuwe vrijwilligers was dit niet de beste manier om met hun nieuwe functie te beginnen.

8.4 COMMITTERS

Omdat committers de enige mensen zijn binnen een open source-project die formeel een afzonderlijke groep vormen, verdienen ze hier extra aandacht. Committers zijn een onvermijdelijk concessie aan het maken van onderscheid binnen een systeem waarin voor het overige zo weinig mogelijk onderscheid wordt gemaakt. Maar 'onderscheid' heeft hier geen negatieve betekenis. De functie van committers is uiterst noodzakelijk en ik geloof niet dat een project goed kan functioneren zonder committers. Kwaliteitscontrole vereist nu eenmaal eh ..., controle. Er zijn altijd veel mensen die zichzelf competent genoeg achten om veranderingen in het programma aan te brengen. Slechts een klein aantal van hen is dat ook daadwerkelijk. Het project kan zich niet laten leiden door het eigen beoordelingsvermogen van mensen. Het moet normen opleggen en alleen commit access geven aan mensen die aan deze normen voldoen[28]. Aan de andere kant, als mensen die direct veranderingen kunnen doorvoeren, samenwerken met mensen die dat niet kunnen, zorgt dat voor een bepaalde machtsverhouding. Deze verhouding moet zo worden gemanaged dat dit het project niet schaadt.

In het gedeelte 'Wie mag er stemmen?' in Hoofdstuk 4, *Sociale en politieke infrastructuur* hebben we de dynamiek voor het aanwijzen van nieuwe committers reeds besproken. Hier kijken we alleen naar de normen op basis waarvan mogelijke nieuwe committers moeten worden beoordeeld en hoe dit proces moet worden gepresenteerd aan een grote gemeenschap.

Committers kiezen

Voor het Subversion-project kozen we de committers primair op basis van het principe van Hippocrates: *zorg er in de eerste plaats voor om geen schade aan te richten*. Ons belangrijkste criterium betreft niet de technische vaardigheden of zelfs maar kennis van de code, maar vooral dat de committer blijk geeft van een goed beoordelingsvermogen. Beoordelingsvermogen kan eenvoudigweg betekenen dat iemand weet wat hij zich niet op de hals moet halen. Misschien post een persoon alleen kleine patches, waarmee hij kleine problemen in de code repareert. Maar als die patches zonder probleem kunnen worden toegepast, geen bugs bevatten en in grote lijnen kloppen met het logbericht en de coderingsafspraken van het project, en als er voldoende patches zijn om uit te kunnen gaan van een patroon, dan zal een bestaande committer deze persoon meestal voordragen voor commit access. Als ten minste drie mensen ja zeggen en niemand bezwaarheeft , dan wordt het aanbod gedaan. Het klopt dat we niet zeker kunnen weten of de persoon in staat is complexe problemen voor alle onderdelen van de gehele code op te lossen, maar dat is geen probleem. De persoon heeft duidelijk gemaakt dat hij in staat is om zijn eigen vaardigheden goed in te schatten. Technische vaardigheden kunnen worden aangeleerd (en onderwezen), maar met beoordelingsvermogen ligt dat een stuk moeilijker. Daarom is dit het belangrijkste aspect, waarvan u zeker wilt zijn voordat u een persoon commit access geeft.

Wanneer de voordracht voor een nieuwe committer een discussie oproept, dan is dat meestal niet over zijn technische vaardigheden, maar eerder over het gedrag van de persoon op de mailinglijsten of op de IRC. Soms getuigt iemand van technische vaardigheden en de vaardigheid te werken binnen het kader van de formele richtlijnen van het project, maar is hij ook vaak op ruzie uit en niet erg coöperatief op publieke forums. Dit is een ernstige handicap. Als het er niet op lijkt dat deze persoon na verloop van tijd bijdraait, zelfs niet na een paar hints, dan voegen we hem niet toe aan ons committersbestand, hoe vaardig hij ook is. In een groep vrijwilligers zijn sociale vaardigheden, oftewel de vaardigheid in teamverband te kunnen spelen, net zo belangrijk als de puur technische vaardigheden. Omdat alles beheerd wordt met een versiebeheersysteem is de straf voor het opnemen van een committer die u niet had moeten opnemen niet zozeer een probleem in de code (die bij een review toch wel naar boven komt), maar dat het project uiteindelijk gedwongen kan zijn om de commit access van de persoon in te trekken, een actie die nooit plezierig en soms erg confronterend kan zijn.

Sommige projecten stellen als voorwaarde dat de potentiële committer een bepaald niveau van technische expertise en doorzettingsvermogen laat zien door het indienen van een aantal minder belangrijke patches. Dit houdt in dat het project niet alleen wil weten of de persoon geen schade berokkent aan het project, maar ook of hij een aanwinst zal zijn voor de code. Dit is prima, maar wees hier wel voorzichtig mee. Het hebben van commit access mag niet gelijk komen te staan met lidmaatschap van een exclusieve club. De vraag die iedereen in zijn achterhoofd moet houden, is: "Wat zorgt voor de beste resultaten voor de code?" en niet "Wordt de sociale status van committer zijn gedevalueerd door deze persoon toe te laten?" De bedoeling van commit access is niet het verhogen van de eigenwaarde van mensen, maar ervoor zorgen dat er goede veranderingen in de code worden doorgevoerd met een minimum aan gedoe. Als u honderd committers hebt waarvan er tien regelmatig grote veranderingen doorvoeren en negentig alleen een paar keer per jaar typefouten en kleine bugs herstellen, dan is dit nog altijd beter dan alleen de beschikking te hebben over de eerste tien committers.

Commit access intrekken

Het eerste dat er over het intrekken van commit access gezegd moet worden is: probeer in de eerste plaats niet in deze situatie terecht te komen. Afhankelijk van de persoon wiens toegang wordt ingetrokken en de reden daarvoor, kunnen de discussies rond deze actie voor zeer veel onenigheid zorgen. Zelfs als dit niet voor grote problemen zorgt, is het altijd nog een tijdrovende afleiding van het eigenlijke productieve werk.

Als u dit echter toch moet doen, dan moet de discussie privé worden gevoerd tussen dezelfde mensen die anders in de positie zouden zijn om te stemmen over het toewijzen van het soort commit access dat de betreffende persoon heeft. De persoon zelf moet niet bij de discussie worden betrokken. Dat is in tegenspraak met het

gangbare verbod op geheimzinnigheid, maar in dit geval noodzakelijk. Ten eerste zou anders niemand vrijuit kunnen spreken. Ten tweede zou u niet willen dat als de motie wordt verworpen, de persoon weet dat deze ter overweging op tafel lag. Dat zou namelijk vragen zou kunnen oproepen ("Wie stond er aan mijn kant? Wie heeft tegen me gestemd?") die leiden tot de ergste vorm van kliekvorming. In bepaalde zeldzame omstandigheden wil de groep de persoon laten weten dat er is gesproken over de vraag of zijn commit access moet worden ingetrokken, als waarschuwing, maar deze openheid moet een beslissing zijn van de groep als geheel. Niemand mag ooit op eigen initiatief informatie vrijgeven over de discussie en de stemming waarvan anderen hebben aangenomen dat deze geheim was.

Wanneer iemands commit access wordt ingetrokken, valt niet te vermijden dat dit publiekelijk bekend wordt (zie het gedeelte 'Voorkom geheimzinnigheid' verderop in dit hoofdstuk). Dus probeer zo tactvol mogelijk zijn in de manier waarop dit aan de buitenwereld bekend wordt gemaakt.

Gedeeltelijke commit access

Sommige projecten kennen gradaties in commit access. Er kunnen bijvoorbeeld contribuanten zijn wiens commit access hun vrije toegang geeft tot de documentatie, maar die geen toegang hebben tot de code zelf. Gebruikelijke gebieden voor gedeeltelijke commit access zijn documentatie, vertalingen, het koppelen van code aan andere programmeertalen, specificatiebestanden voor het maken van pakketten (bijv. RedHat RPM spec-bestanden enz.) en andere terreinen waarop een fout geen problemen geeft voor de kern van het project.

Omdat commit access niet alleen gaat over het doorvoeren van veranderingen, maar ook over het recht om te stemmen (zie het gedeelte 'Wie mag er stemmen?' in Hoofdstuk 4, *Sociale en politieke infrastructuur*) rijst natuurlijk de volgende vraag: Waarover kunnen mensen met gedeeltelijke commit access stemmen? Er is geen eenduidig antwoord op deze vraag. Het hangt af van de soorten gebieden voor gedeeltelijke commit access die uw project heeft. In Subversion hebben we dit nogal simpel gehouden. Een gedeeltelijke committer kan stemmen over onderwerpen die exclusief bij het domein van die committer behoren en over niets anders. Wat daarbij wel belangrijk is, is dat we een mechanisme hebben voor het uitbrengen van adviserende stemmen (dit komt erop neer dat de committer '+0' of '+1 (niet bindend)' aangeeft op het stembiljet, in plaats van alleen '+1'). Er is geen reden om mensen helemaal het zwijgen op te leggen alleen omdat hun stem formeel niet bindend is.

Volledige committers kunnen over alles stemmen, net zoals ze overal veranderingen kunnen doorvoeren, en alleen volledige committers stemmen over het toevoegen van nieuwe committers. In de praktijk wordt de bevoegdheid tot het toevoegen van gedeeltelijke committers meestal gedelegeerd. Iedere volledige committer kan een nieuwe gedeeltelijke committer 'sponsoren', en gedeeltelijke committers kunnen vaak in principe nieuwe committers kiezen voor hetzelfde gebied (dit is met name handig om het vertaalwerk soepel te laten verlopen).

Voor uw project moet u het misschien iets anders inrichten, afhankelijk van de aard van de werkzaamheden, maar hetzelfde algemene principe geldt voor alle projecten. Iedere committer zou moeten mogen stemmen over zaken die vallen binnen het kader van zijn commit access, en niet over zaken die daarbuiten vallen, en stemmen over procedurele vragen dient voorbehouden te zijn aan volledige committers, tenzij er reden is om de groep mensen met stemrecht te vergroten (dit wordt beslist door de volledige committers).

Over het toekennen van gedeeltelijke commit access: het is vaak het beste wanneer het versiebeheersysteem de gebieden voor gedeeltelijke commit access *niet* toekent, zelfs als dit wel mogelijk is. Zie het gedeelte 'Autorisatie' in Hoofdstuk 3, *Technische infrastructuur* voor de redenen hiervoor.

Slapende committers

Sommige projecten verwijderen automatisch de commit access van mensen als ze gedurende een bepaalde periode (bijvoorbeeld één jaar) geen commits hebben toegevoegd. Ik denkom twee redenen dat dit weinig zinvol is en soms zelfs contraproductief kan zijn.

Ten eerste kan het mensen ertoe brengen acceptabele maar onnodige veranderingen door te voeren, alleen maar om ervoor te zorgen dat hun commit access niet verloopt. Ten tweede is het eigenlijk nergens goed voor. Als het belangrijkste criterium voor het toekennen van commit access goed beoordelingsvermogen is, waarom dan aannemen dat iemands beoordelingsvermogen is afgenomen alleen omdat hij een poosje niet bij het project betrokken is geweest? Zelfs wanneer hij jaren uit het zicht verdwijnt, niet naar de code kijkt en de ontwikkelingsdiscussies niet volgt, *weet* hij dat wanneer hij terugkomt hij niet meer echt op de hoogte is en zal hij dienovereenkomstig handelen. U had voorheen vertrouwen in zijn beoordelingsvermogen. Waarom zou u daar niet altijd vertrouwen in hebben? Als middelbareschooldiploma's niet verlopen, dan hoort commit access dat al helemaal niet te doen.

Soms kan een committer zelf vragen om te worden verwijderd, of om expliciet te worden aangeduid als slapende committer in de lijst (zie het gedeelte 'Voorkom geheimzinnigheid' hieronder voor meer informatie over deze lijst). In dergelijke gevallen moet het project de wensen van deze persoon natuurlijk inwilligen.

Voorkom geheimzinnigheid

Hoewel de discussies rond het toevoegen van een bepaalde nieuwe committer vertrouwelijk moeten zijn, hoeven de regels en procedures zelf natuurlijk niet geheim te zijn. Het is juist beter deze te publiceren, zodat mensen zich realiseren dat de committers niet een soort mysterieus clubje vormen dat niet toegankelijk is voor gewone stervelingen, maar dat iedereen hier deel van kan uitmaken, gewoon door goede patches in te dienen en door te weten hoe je je binnen een ontwikkelaarsgemeenschap moet gedragen. In het Subversion-project staat deze informatie direct in het richtlijnendocument voor ontwikkelaars, omdat de mensen die het meest waarschijnlijk geïnteresseerd zijn in hoe commit access wordt toegekend de mensen zijn die overwegen code bij te dragen aan het project.

Publiceer naast de procedures ook de feitelijke *lijst* met committers. De meeste ge-

bruikte plaats hiervoor is een bestand met de naam `MAINTAINERS` of `COMMITTERS` in de bovenste laag van de broncode-tree van het project. Hierin moeten de volledige committers eerst worden vermeld, gevolgd door de diverse gebieden voor gedeeltelijke commit access en de leden voor ieder gebied. Van iedere persoon moeten de naam en het e-mailadres worden vermeld, hoewel het adres kan worden gecodeerd om spam te voorkomen (zie het gedeelte 'Adressen in archief verbergen' in Hoofdstuk 3, *Technische infrastructuur*) als de betreffende persoon dat graag wil.

Omdat het verschil tussen volledige en gedeeltelijke commit access overduidelijk en goed gedefinieerd is, is het correct om dit onderscheid ook op de lijst te maken. De lijst moet echter niet de informele verschillen bevatten die beslist voorkomen binnen een project, zoals wie in het bijzonder veel invloed heeft en op welke manier. Dit bestand bevat publieke informatie; het is geen bestand met dankbetuigingen. Geef een opsomming van de committers in alfabetische volgorde of in de volgorde waarin ze zijn toegevoegd.

8.5 ERKENNING

Erkenning is het belangrijkste betaalmiddel in de wereld van de open source-software. Wat mensen ook zeggen over hun motieven om aan een project deel te nemen, ik ken geen enkele ontwikkelaar die het werk met plezier doet en intussen anoniem blijft, of onder de naam van iemand anders bekend wordt. Hiervoor zijn goede redenen. Iemands reputatie binnen een project bepaalt hoeveel invloed die persoon heeft, en deelname aan een open source-project kan indirect ook geldelijke waarde hebben, omdat sommige werkgevers hier specifiek op letten bij de beoordeling van cv's. Er zijn ook minder concrete redenen, die misschien nog wel sterker gelden. Mensen willen bijvoorbeeld gewoon gewaardeerd worden en zijn altijd instinctief op zoek naar signalen dat hun werk door anderen wordt gezien. De belofte van erkenning is daarom één van de beste stimulansen die een project heeft. Wanneer mensen erkenning krijgen voor kleine bijdragen komen ze vaak terug voor meer.

Een van de belangrijkste kenmerken van coöperatieve ontwikkelingssoftware (zie Hoofdstuk 3, *Technische infrastructuur*) is dat het precies bijhoudt wie wat heeft gedaan en wanneer. Gebruik deze bestaande tools wanneer maar mogelijk is, om ervoor te zorgen dat erkenning op de juiste manier wordt toegekend, en wees concreet over de aard van de bijdrage. Schrijf in een logbericht niet alleen 'Dank aan J. Random <jrandom@example.com>' als u in plaats daarvan ook kunt schrijven 'Dank aan J. Random <jrandom@example.com> voor zijn bugrapport en zijn reproductierecept'.

In Subversion hebben we het informele maar consistente beleid om degene die een bug heeft gerapporteerd te bedanken in ofwel het geregistreerde issue-veld, ofwel in het logbericht van de commit waarmee de bug wordt gerepareerd als zo'n invulveld niet aanwezig is. Een snelle blik op commit-logberichten van Subversion tot aan commit-nummer 14525 laat zien dat ongeveer 10% van de commits iemand met naam en e-mailadres bedankt, meestal de persoon die de met de betreffende fix gerepareerde bug heeft gerapporteerd of geanalyseerd. Merk op dat dit een andere persoon is dan de ontwikkelaar die de feitelijke commit heeft gedaan. Diens naam wordt automatisch vastgelegd door het versiebeheersysteem. Van de circa tachtig volledige en gedeeltelijke committers die Subversion vandaag heeft, werden 55 bedankt in de commit-logberichten (meestal meerdere keren) voordat ze zelf committer werden. Dit bewijst natuurlijk niet dat bedankt worden van belang was voor hun aanhoudende betrokkenheid, maar het creëert in ieder geval een sfeer waarin mensen erop kunnen rekenen dat hun bijdragen de erkenning krijgen die ze verdienen.

Het is belangrijk om onderscheid te maken tussen routinematige bedankjes en bijzondere erkenning. Als een bepaald deel van de code of een bepaalde bijdrage die iemand heeft gemaakt wordt besproken, is het prima om deze mensen voor hun werk te bedanken. Door bijvoorbeeld te zeggen dat "Daniels recente veranderingen in de deltacode betekenen dat we functie X nu kunnen implementeren" kunnen mensen bepalen over welke veranderingen u het hebt en is Daniels werk tegelijkertijd erkend. Aan de andere kant heeft een post met daarin alleen een bedankje voor Daniel voor de veranderingen in de deltacode geen direct praktisch nut. Het voegt geen enkele informatie toe, omdat het versiebeheersysteem en andere mechanismes het feit dat hij deze veranderingen heeft doorgevoerd al hebben geregistreerd. Als u iedereen altijd voor alles bedankt, leidt dit alleen maar af en voegt uiteindelijk geen informatie toe. Het effect ervan wordt namelijk grotendeels bepaald door de mate waarin het uitsteekt boven het standaardniveau van positieve reacties die de hele tijd over een weer gaan. Dit betekent natuurlijk niet dat u nooit mensen moet bedanken. Zorg er alleen voor dat u dit zo doet dat het niet leidt tot inflatie van deze bedankjes. Het kan helpen de volgende richtlijnen aan te houden.

- Hoe kortstondiger een forum is, des te vrijer u zich zou moeten voelen om mensen te bedanken. Iemand terloops bedanken voor een bugfix tijdens een IRC-discussie is prima, evenals een opmerking in een e-mail die voornamelijk over andere onderwerpen gaat. Post daarentegen geen e-mail die alleen maar iemand bedankt, behalve als het over een echt ongebruikelijke prestatie gaat. Evenzo zou u ook de webpagina's van het project niet moeten volstoppen met uitingen van dankbaarheid. Als u daar eenmaal mee begint, is het nooit duidelijk wanneer u moet stoppen. Plaats *nooit* een dankwoord in de opmerkingen van de code. Dat leidt alleen maar af van waar deze opmerkingen primair voor bedoeld zijn, namelijk de lezer helpen de code te begrijpen.

- Hoe minder betrokken iemand is bij het project, des te gepaster het is hem te bedanken voor iets dat hij heeft gedaan. Dit lijkt onlogisch, maar het past bij de instelling dat u iemand bedankt wanneer hij meer doet dan u van hem had verwacht. Dit houdt in dat als u contribuanten regelmatig bedankt voor dingen die ze normaal gesproken al deden, u een lagere verwachting van hen uitspreekt dan zij van zichzelf hadden. Wat u ook had willen bereiken, dit in ieder geval niet!

  Er zijn enkele uitzonderingen op deze regel. Het is prima om iemand te bedanken voor het naar verwachting uitoefenen van zijn taak als die taak zo nu en dan tijdelijk extreem grote inspanningen met zich meebrengt. Een goed

voorbeeld hiervan is de releasemanager, die op volle snelheid komt rond het tijdstip van iedere release, maar de rest van de tijd niet veel te doen heeft (niet als releasemanager tenminste; hij kan ook een actieve ontwikkelaar zijn, maar dat is een ander verhaal).

- Net als kritiek en complimenten moet ook dankbaarheid concreet zijn. Bedank mensen niet omdat ze zo geweldig zijn, zelfs als dat het geval is. Bedank ze voor iets ongewoons dat ze hebben gedaan en vertel daarbij voor de bonuspunten ook wat er zo geweldig aan was.

In het algemeen is er altijd een spanningsveld tussen ervoor zorgen dat de individuele bijdragen van mensen worden gewaardeerd en laten zien dat het project een groepsinspanning is en geen optelsom van individuele successen. Probeert u zich altijd bewust te zijn van dit spanningsveld en kies zo veel mogelijk de optiek van het groepsgevoel, dan lopen de zaken niet uit de hand.

8.6 FORKS

In het de paragraaf 'Afsplitsbaarheid of forkability' in Hoofdstuk 4, *Sociale en politieke infrastructuur* zagen we dat de *mogelijkheid* van forks belangrijke gevolgen heeft voor de project besturing. Maar wat gebeurt er wanneer er werkelijk een fork ontstaat? Hoe moet u daarmee omgaan en welke gevolgen kunt u ervan verwachten? En aan de andere kant, wanneer moet *u* een fork *in gang zetten*?

De antwoorden op deze vragen hangen af van het soort fork. Sommige forks zijn het gevolg van vriendschappelijke maar onoverbrugbare meningsverschillen over de richting van het project. De meeste zijn echter toe te wijzen aan zowel technische onenigheid als persoonlijke conflicten. Het is natuurlijk niet altijd mogelijk om het verschil tussen beide te zien, omdat technische argumenten ook persoonlijke aspecten kunnen bevatten. Wat alle forks met elkaar gemeen hebben, is dat de ene groep ontwikkelaars (of soms zelfs maar één ontwikkelaar) besluit dat de nadelen van het samenwerken met sommige of alle andere ontwikkelaars de voordelen beginnen te overtreffen.

Wanneer er eenmaal een fork ontstaan is, bestaat er geen eenduidig antwoord meer op de vraag welke fork het 'echte', 'oorspronkelijke' project is. Mensen zullen onderling praten over fork F die voortgekomen is uit project P, alsof P onveranderd doorgaat op de gebaande weg, terwijl F uitwijkt naar nieuwe gebieden. Dit zegt in feite meer over hoe de persoon die dit zegt erover denkt. In feite is het gewoon een kwestie van perceptie. Als maar genoeg mensen het hiermee eens zijn, dan wordt de bewering vanzelf waar. Het is dus niet zo dat er van meet af aan sprake is van één objectieve waarheid, die we gewoon eerst niet goed hebben kunnen inzien. In plaats daarvan *zijn* de percepties de objectieve waarheid, omdat een project (of een fork) uiteindelijk ook alleen in de hoofden van de mensen bestaat.

Als de mensen die de fork maken het gevoel hebben dat ze een nieuwe branch maken die ontspruit uit het hoofdproject, dan is de perceptievraag direct en eenvoudig opgelost. Iedereen, zowel ontwikkelaars en gebruikers, zullen de fork als een nieuw project zien, met een nieuwe naam (misschien voortbouwend op de oude naam, maar makkelijk daarvan te onderscheiden), een afzonderlijke website en een eigen filosofie of doelstelling. De zaken liggen echter gecompliceerder als beide kanten het gevoel hebben dat zij de legitieme hoeders van het oorspronkelijke project zijn en daarom het recht hebben de oorspronkelijke naam te blijven gebruiken. Als een organisatie handelsmerkrechten voor de naam of juridische zeggenschap over het domein of de webpagina's heeft, wordt de kwestie meestal van hogerhand opgelost. De organisatie bepaalt wie het project is en wie de fork, omdat zij alle kaarten in handen heeft in de strijd om de public relations. Het komt echter zelden zover. Omdat iedereen al weet hoe de machtsverhoudingen liggen, gaat niemand een gevecht aan waarvan de uitkomst van tevoren al bekend is en legt men zich bij die uitkomst neer.

Gelukkig is er in de meeste gevallen weinig twijfel mogelijk over welke branch het project is en welke de fork, omdat de fork in feite een motie van wantrouwen is. Als meer dan de helft van de ontwikkelaars voorstander is van de richting die de fork voorstelt, is er meestal geen reden tot afsplitsing. Het project zelf gaat dan gewoon die kant op, behalve als het project wordt geleid door een bijzonder koppige 'dictator'. Aan de andere kant is het natuurlijk zo dat als minder dan de helft voorstander is, de fork duidelijk een opstand van een minderheid is en dus zowel de beleefdheid als het gezond verstand gebiedt dat men zichzelf als een afwijkende branch moet zien en niet als de hoofdlijn.

Omgaan met een fork

Als iemand dreigt een fork van uw project te maken, blijf dan kalm en denk aan uw langetermijndoelstellingen. Het *bestaan* van een fork is niet direct schadelijk voor een project, het verlies van ontwikkelaars en gebruikers wel. Uw werkelijke doel is dan ook niet de fork in de kiem te smoren, maar de schadelijke effecten ervan te minimaliseren. U kunt boos zijn, u kunt het gevoel hebben dat de fork onterecht en onnodig was, maar als u dat hardop zegt, loopt u alleen maar de kans om weifelende ontwikkelaars weg te jagen. Probeer in plaats daarvan mensen niet te dwingen een keuze te maken en wees zo coöperatief met de fork als mogelijk is. Om te beginnen moet u iemands commit access bij uw project niet afnemen alleen omdat hij heeft besloten aan de fork te werken. Werken aan de fork betekent niet dat de persoon plotseling zijn competentie kwijt is om aan het oorspronkelijke project te werken. Committers moeten dus committers kunnen blijven. Bovendien zou u moeten aangeven dat u zo compatibel mogelijk zou willen blijven met de fork en dat u hoopt dat ontwikkelaars veranderingen tussen de twee zullen uitwisselen als dat van toepassing is. Als u administratieve toegang hebt tot de servers van het project, biedt de forkers dan publiekelijk hulp aan met de infrastructuur tijdens de opstartperiode. Bied hun bijvoorbeeld een complete en helemaal tot aan het begin teruggaande kopie aan van de versiebeheerdatabase als zij er niet op een andere manier aan kunnen komen, zodat ze niet hoeven beginnen zonder historische gegevens (dit is, afhankelijk van het versiebeheersysteem, misschien niet nodig). Vraag ze of ze nog iets anders nodig hebben en geef ze dat ook als u dat kunt. Wring u in alle bochten om ze te laten zien dat u ze niets in de weg wilt leggen en dat u wilt dat de fork slaagt of mislukt op basis van zijn eigen kenmerken, en niets anders.

De reden om dit te doen (en dit vooral publiekelijk te doen) is in feite niet om de fork te helpen, maar om ontwikkelaars te overtuigen dat uw groep de veiligste keuze is omdat u niet rancuneus overkomt. In oorlogstijd is het soms zinvol (zo niet vanuit menselijk dan wel uit strategisch oogpunt) om mensen te dwingen een kant te kiezen, maar bij open source-software is dat bijna nooit het geval. In feite komt het zelfs vaak voor dat sommige ontwikkelaars openlijk aan beide projecten werken en hun best doen de twee compatibel te houden. Deze ontwikkelaars houden de communicatielijnen open na de afsplitsing van de fork. Ze geven uw project gelegenheid gebruik te maken van interessante nieuwe functies in de fork (inderdaad, de fork kan dingen hebben die u ook graag wilt hebben) en vergroten ook de kans dat het project en de fork op een later tijdstip weer bij elkaar komen.

Soms heeft een fork zoveel succes dat hij, hoewel hij zelfs door de initiatiefnemers ervan aanvankelijk als een fork werd beschouwd, op den duur de door iedereen geprefereerde versie wordt en uiteindelijk het origineel van de eerste plaats verdringt. Een beroemd voorbeeld hiervan is de GCC/EGCS-fork. De *GNU Compiler Collection* (*GCC*, voorheen de *GNU C Compiler*) is de populairste open source native-code-compiler en tevens een van de best porteerbare compilers ter wereld. Als gevolg van onenigheid tussen de officiële mensen van GCC en Cygnus Software[29] maakte een van de meeste actieve ontwikkelaarsgroepen van GCC, Cygnus, een fork van GCC onder de naam *EGCS*. De fork was met opzet niet vijandig opgezet. De EGCS-ontwikkelaars hebben op geen enkel moment hun versie van GCC afgeschilderd als de nieuwe officiële versie. In plaats daarvan concentreerden ze zich erop om EGCS zo goed mogelijk te maken en patches sneller op te nemen dan bij de officiële GCC-versie. EGCS werd steeds populairder en uiteindelijk besloten enkele grote distributeurs van besturingssystemen EGCS op te nemen als standaardcompiler in plaats van GCC. Op dat moment realiseerden de mensen van GCC zich dat vasthouden aan de naam 'GCC', terwijl iedereen overschakelde naar de EGCS-fork, door de onnodige naamsverandering voor iedereen extra moeite met zich mee zou brengen en dat ze niets zouden kunnen doen om deze overschakeling te voorkomen. Daarom nam GCC de gehele code van EGCS over en was (en is) er opnieuw sprake van één enkele GCC, maar dan aanzienlijk verbeterd dankzij de fork.

Dit voorbeeld laat zien waarom u een fork niet altijd hoeft te zien als iets puur slechts. Een fork kan pijnlijk en ongewenst zijn op het moment zelf, maar u kunt nooit zeker weten of hij wel of niet succesvol zal zijn. Daarom moeten u en de rest van het project de fork goed blijven volgen en bereid zijn niet alleen functies waar mogelijk over te nemen, maar in het uiterste geval zelfs aan te sluiten bij de fork wanneer deze populairder wordt dan uw project. Natuurlijk kunt u op basis van de mensen die zich bij de fork aansluiten van tevoren enigszins voorspellen hoe groot de kans op succes is. Als de fork is gestart door de grootste zeur van het project en wordt gevolgd door een handjevol ontevreden ontwikkelaars die sowieso al niet zo'n constructieve bijdrage aan het project leverden, hebben ze in feite een probleem voor u opgelost door zich af te splitsen en hoeft u zich waarschijnlijk geen zorgen te maken dat de fork veel vaart uit het oorspronkelijke project zal halen. Maar als u ziet dat invloedrijke en gerespecteerde ontwikkelaars de fork ondersteunen, moet u zichzelf natuurlijk wel afvragen waarom dat is. Misschien legde uw project wel overdreven veel beperkingen op. De beste oplossing is dan om enkele acties van de fork over te nemen voor de hoofdlijn van het project. Dat komt er op neer dat u de fork voorkomt door uzelf aan de fork aan te passen.

Een fork initiëren

Bij al het advies dat ik hier geef, ga ik ervan uit dat u geen andere mogelijkheid ziet dan afsplitsing. Probeer eerst alle andere mogelijkheden uit voordat u een fork start. Een fork maken betekent bijna altijd het verlies van ontwikkelaars, met alleen onzekerheid over de vraag of u daar later nieuwe voor terug krijgt. Het betekent ook dat u begint te concurreren om de aandacht van gebruikers. Iedereen die van plan is de software te downloaden, moet zichzelf afvragen: "Hmm, wil ik deze versie of de andere?" Welke van de twee u ook bent, de situatie is altijd verwarrend omdat er een vraag is ontstaan die er voorheen niet was. Sommige mensen beweren dat forks een gezond fenomeen zijn voor het ecosysteem van de software als geheel, op basis van het argument van natuurlijk selectie. De sterkste software zal overleven, wat uiteindelijk betekent dat iedereen betere software krijgt. Dit is misschien wel waar vanuit het oogpunt van het ecosysteem, maar het klopt niet voor de afzonderlijke projecten. De meeste forks worden geen succes en de meeste projecten zijn er niet blij mee wanneer een fork zich afsplitst.

Een voortvloeisel hiervan is dat u de dreiging van een fork niet zou moeten gebruiken als extreme discussietechniek ("Je moet het doen zoals ik het wil, anders maak ik een fork van het project!") omdat iedereen zich bewust is van het feit dat een fork die geen ontwikkelaars kan weghalen bij het oorspronkelijke project weinig overlevingskansen heeft. Iedereen die de discussie volgt, dus niet alleen ontwikkelaars maar ook gebruikers en distributeurs van besturingssystemen, beslissen zelf over welke partij ze kiezen. U moet het daarom laten lijken alsof u zeer terughoudend bent om een fork af te splitsen, zodat als u dat uiteindelijk wel doet u geloofwaardig kunt zeggen dat het uw enige overgebleven optie was.

Vergeet niet *alle* factoren in overweging te nemen wanneer u bekijkt of een fork kans van slagen heeft. Als bijvoorbeeld veel van de ontwikkelaars van een project voor dezelfde werkgever werken, dan zullen ze het niet snel zeggen wanneer ze ontevreden en persoonlijk voorstander zijn van een fork als ze weten dat hun werkgever er tegen is. Veel programmeurs van open source-software denken maar al te graag dat het hebben van een vrije licentie op de code betekent dat geen enkel bedrijf de ontwikkeling kan manipuleren. Het klopt dat de licentie een fundamentele garantie is voor vrijheid. Als anderen er voldoende van overtuigd zijn dat zij een fork willen maken van het project en daar ook de hulpmiddelen voor hebben, dan kunnen ze dat ook doen. In de praktijk echter worden ontwikkelingsteams van sommige projecten voornamelijk gefinancierd door één entiteit en het heeft geen zin net te doen alsof de ondersteuning van die entiteit niet belangrijk is. Als die tegen een fork is, is de kans klein dat de ontwikkelaars mee zullen werken, zelfs als ze dat stiekem wel graag zouden willen.

Als u nog steeds vindt dat u een fork moet maken, zoek dan eerst privé steun en kondig de fork daarna aan op een niet-vijandige manier. Zelfs als u boos bent op, of teleurgesteld in de huidige leiding, zeg dat dan niet in het bericht. Geef alleen kalm aan welke argumenten tot uw beslissing hebben geleid om een fork te maken en

dat u geen verkeerde bedoelingen hebt voor het project waarvan u zich afsplitst. Ervan uitgaande dat u uw actie als een fork ziet (en niet als een noodgreep om het oorspronkelijke project te behouden), benadruk dan dat u een fork maakt van de code en niet van de naam en kies een naam die niet conflicteert met de naam van het project. U kunt een naam kiezen die verwijst naar de oorspronkelijke naam, zolang het niet voor verwarring over de identiteit kan zorgen. Natuurlijk mag u op de website van de fork uitgebreid uitleggen dat het een afsplitsing is van het oorspronkelijke programma en zelfs dat u hoopt dat het dat programma uiteindelijk zal vervangen. Maar maak het gebruikers niet moeilijker door hen te dwingen eerst een identiteitspuzzel te moeten ontwarren.

Als laatste kunt u een goed begin maken door alle committers van het oorspronkelijke project automatisch commit access te geven voor de fork, ook de committers die het openlijk met u oneens waren dat een fork nodig was. Zelfs als ze deze toegang nooit gebruiken, is uw boodschap duidelijk: er is hier sprake van meningsverschillen, maar u bent geen vijanden van elkaar en u bent blij met codebijdragen van iedere competente bron.

24| Deze kwestie is diepgaand onderzocht, met interessante resultaten, in een artikel van Karim Lakhani en Robert G. Wolf, met de titel Why Hackers Do What They Do: Understanding Motivation and Effort in Free/Open Source Software Projects. Zie http://freesoftware.mit.edu/papers/lakhaniwolf.pdf.

25| Voor een goed tegenargument kunt u een kijkje nemen op de mailinglijst-thread met de titel 'having authors names in .py files' op http://groups.google.com/group/sage-devel/browse_thread/thread/e207ce2206f0beee, met name in de post van William Stein. Het belangrijkste in dat geval is, denk ik, dat veel van de auteurs uit een cultuur komen (de academische wiskundegemeenschap) waar naamsvermelding direct bij de bron de norm is en hoog wordt aangeslagen. In dergelijk omstandigheden kan het de voorkeur hebben de namen van de auteurs in de bronbestanden te vermelden, samen met exacte beschrijvingen van wat iedere auteur heeft gedaan, omdat de meerderheid van de mogelijke contribuanten een dergelijke vorm van erkenning zal verwachten.

26| Merk op dat het niet nodig is om alle bestaande tests te converteren naar het nieuwe systeem. De twee kunnen prima naast elkaar leven, waarbij oude tests alleen worden geconverteerd als ze moeten worden veranderd.

27| IssueZilla is de issue tracker die wij gebruiken. Het is een afstammeling van BugZilla.

28| Merk op dat commit access iets anders kan betekenen in gedecentraliseerde versiebeheersystemen, waar iedereen een database kan opzetten die naar het project is gelinkt, en zichzelf commit access kan geven voor die database. Het concept van commit access blijft echter van toepassing: 'commit access' is kort gezegd 'het recht om veranderingen aan te brengen in de code die bij de volgende release van de software onderdeel van deze software uit zullen maken.' Bij gecentraliseerde versiebeheersystemen betekent dit het hebben van directe commit access. In gedecentraliseerde systemen betekent dit dat iemands veranderingen standaard direct in de hoofdsoftware worden overgenomen. Het idee erachter is dus hetzelfde; de techniek waarmee het wordt gedaan is niet vreselijk belangrijk.

29| Nu onderdeel van RedHat (http://www.redhat.com/).

9

LICENTIES, AUTEURSRECHTEN EN PATENTEN

De licentie die u kiest, heeft waarschijnlijk geen grote invloed op de aanvaarding van uw project, zolang het een open source-licentie betreft. Gebruikers kiezen software normaal gesproken om hun kwaliteit en gebruiksmogelijkheden, niet om de details van de licentie. Desalniettemin moet u wel het één en ander weten over opensoftwarelicenties, enerzijds om u ervan te verzekeren dat de licentie van het project past bij het doel van het project en anderzijds om over de keuze van de licentie te kunnen discussiëren. Houd wel in de gaten dat ik geen jurist ben en dat niets in dit hoofdstuk geïnterpreteerd mag worden als formeel juridisch advies. Daarvoor zult u een jurist moeten inschakelen, tenzij u er zelf één bent.

9.1 TERMINOLOGIE

In elke discussie over open source-licenties wordt direct duidelijk dat er veel verschillende benamingen lijken te zijn voor hetzelfde: *vrije software*, *open source*, *FOSS*, *F/OSS* en *FLOSS*. Laten we deze en wat andere termen eerst aan een nader onderzoek onderwerpen.

Free software

Software kan vrij gedeeld en aangepast worden, inclusief de broncode ervan. Deze term werd voor het eerst gebruikt door Richard Stallman. Hij is degene die het gecodificeerd heeft in de GNU General Public License (GPL) en is ook de oprichter van de Free Software Foundation (http://www.fsf.org/). Deze stichting is opgericht om het concept free software te promoten.

Hoewel ‘free software’ bijna de lading van ‘open source’-software dekt, geven de FSF en vele anderen de voorkeur aan de eerste term, omdat deze de nadruk legt op het idee van vrijheid en het concept van het vrij te verspreiden software, hoofdzakelijk als een sociale beweging in plaats van een technische. De FSF geeft toe dat de term dubbelzinnig is, het kan ‘vrij’ zijn in de zin van ‘kosteloos’, in plaats van ‘vrij’ als in ‘vrijheid’, maar ze vinden dat dit toch de beste term is. De alternatieven hebben ook hun dubbelzinnigheden. (In dit hele boek door is ‘vrij’ gebruikt in de zin van ‘vrijheid’ en niet van ‘kosteloos’.)

Open source-software

Open source-software is een andere naam voor vrije software. Deze andere naam staat echter voor een belangrijk filosofisch verschil. 'Open source' is geïntroduceerd door het Open Source Initiative (http://www.opensource.org/) als goed alternatief voor 'vrije software'. En om te laten zien dat dergelijke software een aantrekkelijke keus is voor bedrijven wordt het gepresenteerd als een ontwikkelingsmethodologie in plaats van een politieke beweging. Ze hebben wellicht ook een ander stigma willen overwinnen, namelijk dat alles wat 'vrij' is, haast wel van slechte kwaliteit moet zijn.

Omdat iedere licentie die gratis is ook open source is en vice versa (een enkele uitzondering daargelaten) hebben mensen de neiging één term te kiezen en daar ook aan vast te houden. Over het algemeen is het zo dat degenen die 'free software' prefereren waarschijnlijk een meer filosofische of morele kijk op de zaak hebben, terwijl degenen die 'open source' prefereren, het of niet zien als een kwestie van vrijheid, of niet de behoefte voelen om ermee te koop te lopen. Zie het gedeelte ''Vrij' versus 'open source'' in Hoofdstuk 1, *Introductie* voor een meer gedetailleerde uiteenzetting over deze tweedeling.

De Free Software Foundation geeft een goede (beslist niet objectieve, maar wel genuanceerde en behoorlijk eerlijke) uitleg over de twee termen op http://www.fsf.org/licensing/essays/free-software-for-freedom.html. De uitleg van het Open Source Initiative staat verspreid over twee sites: http://www.opensource.org/advocacy/case_for_hackers.php#marketing en http://www.opensource.org/advocacy/free-notfree.php.

FOSS, *F/OSS*, *FLOSS*

Als er twee zijn van iets, dan worden het er snel drie, en dat is precies wat er aan het gebeuren is met de termen voor vrije software. De academische wereld, in een poging precisie en alomvattendheid te verkrijgen in plaats van elegantie, lijkt gekozen te hebben voor FOSS, of soms F/OSS, wat staat voor 'Free / Open Source Software'. Een andere variant die aan populariteit wint, is FLOSS, wat staat voor 'Free / Libre Open Source Software' (*libre* is bekend in vele talen en niet onderhevig aan de dubbelzinnigheid van 'gratis/free'. Zie http://en.wikipedia.org/wiki/FLOSS voor meer hierover).

Al deze termen betekenen in essentie hetzelfde: software die door iedereen aangepast kan worden en doorgegeven, soms, maar niet altijd, met de vereiste dat afgeleide werken vrij verspreid kunnen worden maar onder de dezelfde voorwaarden.

DFSG-compliant

Compliant met the Debian Free Software-richtlijnen (http://www.debian.org/social_contract#guidelines). Dit is een veelgebruikte test om te kijken of een bepaalde licentie werkelijk open source (vrij, *libre* enz.) is. De missie van het Debian-project is om een geheel gratis besturingssysteem op te zetten, en wel zo dat iemand die het installeert zich nooit hoeft af te vragen of hij het mag aanpassen of het gedeeltelijk of helemaal mag doorgeven. De richtlijnen van Debian Free Software zijn de vereisten waaraan een licentie van een softwarepakket moet voldoen om in Debian opgenomen te worden. Omdat het Debian-project erg veel tijd besteed heeft aan de opzet van de test, zijn de richtlijnen waarmee ze kwamen zeer robuust gebleken (zie http://en.wikipedia.org/wiki/DFSG). Voor zover ik weet is er geen bezwaar tegen deze richtlijnen geuit, niet door de Free Software Foundation en ook niet door het Open Source Initiative. Als u weet dat een gegeven licentie conform DFSG is, dan weet u dat het alle belangrijke vrijheden garandeert (zoals forkability, zelfs tegen de wens van de oorspronkelijke auteur in), wat een vereiste is om de dynamiek van een open source-project in stand te houden. Alle besproken licenties in dit hoofdstuk zijn conform DFSG.

OSI-goedgekeurd

Goedgekeurd door het Open Source Initiative. Dit is een andere veelgebruikte manier om te zien of een licentie alle benodigde vrijheden biedt. De definitie van open source-software van OSI is gebaseerd op de Debian Free Software-richtlijnen, en licenties die voldoen aan de ene definitie doen dat bijna ook altijd aan de andere. Er zijn de laatste jaren enkele uitzonderingen geweest, maar deze betroffen alleen nichelicenties en zijn niet relevant hier. In tegenstelling tot het Debian-project houdt het OSI een lijst bij met alle licenties die ze ooit goedgekeurd hebben op http://www.opensource.org/licenses/, zodat 'OSI-approved' een ondubbelzinnige status is. Een licentie komt voor of komt niet voor op de lijst.

De Free Software Foundation houdt ook een lijst bij met licenties op http://www.fsf.org/licensing/licenses/license-list.html. De FSF categoriseert licenties niet alleen naar of ze vrij zijn maar ook of ze compatibel zijn met de GNU General Public-licentie. De compatibiliteit van een GPL is een belangrijk onderwerp. Dit wordt behandeld in het gedeelte 'De GPL en compatibiliteit van licenties' verderop in dit hoofdstuk.

Propriëtair, *closed-source*

Dit is het tegenovergestelde van 'vrije software' of 'open source'. Het betekent software verspreidt onder de traditionele, op royalty's gebaseerde licenties, waarbij gebruikers betalen per kopie, of onder andere licenties die voldoende beperkend zijn zodat de open source-dynamiek niet in werking kan treden. Zelfs software die gratis wordt verspreidt kan propriëtair zijn, als de licentie erop geen verdere verspreiding of aanpassing toestaat.

Over het algemeen worden 'propriëtair' en 'closed-source' gebruikt als synoniemen. Maar 'closed-source' impliceert ook nog dat zelfs de broncode niet gezien kan worden. En omdat de broncode bij de meeste propriëtaire software niet zichtbaar is, komt dit dus meestal op hetzelfde neer. In een enkel geval brengt iemand echter software uit onder een licentie die toestaat om de broncode te bekijken. Verwarrend is wel dat ze dit soms 'open source' of 'bijna open source' noemen, maar dit is misleidend. De *zichtbaarheid* van de broncode is niet van belang, de belangrijke vraag is wat iemand ermee mag *doen*. Daaruit volgt dat het verschil tussen propriëtaire software en closed-source bijna geheel irrelevant is en deze twee gezien kunnen worden als synoniemen.

Soms wordt het woord *commercieel* gebruikt als een synoniem voor 'propriëtair,' maar feitelijk zijn deze twee niet hetzelfde. Vrije software kan commerciële software zijn. Want uiteindelijk kan vrije software verkocht worden zolang de kopers er maar niet van weerhouden worden kopieën weg te geven. Het kan ook op andere manieren commercieel gebruikt worden, bijvoorbeeld door de verkoop van ondersteuning, service of certificering. Er bestaan bedrijven waarin vele miljoenen dollars omgaan en die gegrondvest zijn op vrije software. Vrije software is dus duidelijk intrinsiek anticommercieel noch antibedrijfsmatig. Aan de andere kant is de aard ervan wel antipropriëtair en dit is de essentie van het verschil met traditionele modellen waarbij *licenties per kopie* worden verkocht.

Publiek domein

Het feit dat software geen auteursrechten heeft, betekent dat er niemand is die het recht bezit om kopiëren van het werk te beperken. Bij het publieke domein horen is niet hetzelfde als geen auteur hebben. Alles heeft een auteur en al kiest de auteur of de auteurs ervoor het werk in het publieke domein te plaatsen, verandert dat niets aan het feit dat ze het geschreven hebben.

Wanneer een werk in het publieke domein is opgenomen, kan materiaal daarvan ingelijfd worden in werk waar wel auteursrechten op zit. Daarna valt *die kopie* van het materiaal onder dezelfde auteursrechten als het gehele werk. Maar dit heeft geen effect op de verkrijgbaarheid van het originele werk, wat beschikbaar blijft in het publieke domein. Dus iets vrijgeven in het publieke domein is technisch gezien een manier om het werk 'gratis of vrij' te maken, in overeenstemming met de richtlijnen van de meeste voor 'vrije software' certificerende organisaties. Hoe dan ook, er zijn meestal goede redenen om een licentie aan te vragen in plaats van het werk vrij te geven in het publieke domein. Zelfs bij vrije software kunnen bepaalde restricties zinvol zijn, niet alleen voor degene die het auteursrecht bezit maar ook voor de ontvanger van het werk. Dit wordt duidelijk in het volgende gedeelte.

Auteursrechtenvrij (copyleft)

Een licentie die auteursrechtelijke wetten toepast om het tegenovergestelde te bereiken van het traditionele auteursrecht. Al naar gelang aan wie je het vraagt, betekent het licenties die de vrijheden toestaan die hier ter discussie staan, of licenties die niet alleen die vrijheden toestaan maar ook afdwingen, door te bedingen dat deze vrijheden onlosmakelijk verbonden zijn met het werk. De Free Software Foundation gebruikt louter de tweede, meer specifieke, definitie. Bij de andere partijen is het een beetje een gok. Vele gebruiken de term op de manier waarop de FSF dat doet, maar andere (waaronder ook mensen die voor de reguliere media schrijven) neigen naar de eerste definitie. Het is niet voor iedereen die deze term bezigt duidelijk dat er een onderscheid gemaakt moet worden.

Het algemeen aanvaarde voorbeeld van de preciezere en striktere definitie is de GNU General Public-licentie, die bepaalt dat elk afgeleid werk ook onder de GPL-licenties moet vallen. Zie het gedeelte 'De GPL en licentiecompatibiliteit' verderop in dit hoofdstuk voor meer hierover.

9.2 ASPECTEN VAN LICENTIES

Hoewel er veel verschillende vrijesoftwarelicenties zijn, zeggen ze op de belangrijkste punten allemaal hetzelfde: dat iedereen de code aan mag passen, dat iedereen deze mag verspreiden, zowel in de originele als in de aangepaste vorm, en dat de houders van de auteursrechten geen garanties geven (wettelijke aansprakelijkheid vermijden is van uitzonderlijk belang gezien het feit dat mensen een aangepaste versie kunnen gebruiken zonder dit zelf te weten). De verschillen tussen licenties zijn in het kort samen te vatten tot enkele veel voorkomende kwesties:

Compatibiliteit met propriëtaire licenties

Sommige vrije licenties staan toe dat de betreffende code wordt gebruikt in propriëtaire programma's. Dit heeft geen gevolgen voor de licentievoorwaarden van het propriëtaire programma: het blijft net zo propriëtair, alleen bevat wat code uit een niet propriëtaire bron. De Apache-licentie, de X Consortium-licentie, de BSD-licenties en de MIT-licenties zijn allemaal voorbeelden van licenties die compatibel zijn met propriëtaire software.

Compatibiliteit met andere vrije licenties

De meeste vrije licenties zijn compatibel met elkaar. Dat wil zeggen dat code onder de ene licentie, gecombineerd kan worden met code onder een andere licentie, en dat het resultaat onder één van beide licenties verspreid kan worden zonder de voorwaarden van de ander te schenden. De grote uitzondering hierop is the GNU General Public-licentie, die vereist dat elk werk dat gebruik maakt van code met een GPL verder gedistribueerd moet worden onder de GPL, zonder toevoeging van extra restricties buiten die van de GPL.

De GPL is compatibel met enkele vrije licenties, maar niet met alle. Hierop wordt verder ingegaan in het gedeelte 'De GPL en compatibiliteit van licenties' verderop in dit hoofdstuk.

Handhaving van credits

Sommige vrije licenties vereisen dat elk gebruik van de bedoelde code vergezeld gaat van een mededeling, waarvan de plaatsing en presentatie meestal wordt aangegeven, om op deze manier credit te geven aan de auteurs van de code. Deze licenties zijn vaak nog steeds compatibel met propriëtaire software: ze vereisen niet per definitie dat het afgeleide werk vrij is, alleen maar dat de vrije code wordt voorzien van een credit.

Bescherming van handelsmerk

Dit is een variant op credithandhaving. Licenties die handelsmerken beschermen, geven aan dat de naam van de originele software (of van degenen die de auteursrechten hebben, of het instituut enz.) *niet* gebruikt mag worden bij afgeleide werken zonder voorafgaande schriftelijke toestemming. Terwijl crediting van een licentie vereist dat een bepaalde naam vermeld wordt, vereist een handelsmerkbescherming dat het niet gebruikt mag worden. Ze drukken echter beide dezelfde wens uit, namelijk dat de reputatie van de originele code bewaard en doorgegeven wordt, en niet bezoedeld wordt door de associatie met een ander product.

Bescherming van 'artistieke integriteit'

Sommige licenties (de Artistic-licentie, gebruikt voor de meest populaire implementatie van de programmeertaal Perl en Donald Knuths TeX-licentie bijvoorbeeld) vereisen dat de modificatie en verspreiding op een dusdanige manier gedaan worden dat er een duidelijk onderscheid gemaakt wordt tussen de onaangetaste originele versie van de code en de modificaties. In principe staan ze dezelfde vrijheden toe als andere licenties, maar leggen ze bepaalde eisen op die het makkelijk maken om de integriteit van het origineel te verifiëren. Deze licenties zijn buiten het specifieke programma waar ze voor gemaakt zijn nauwelijks gebruikt en zullen niet verder besproken worden in dit hoofdstuk. Ze worden hier alleen genoemd voor de volledigheid.

Bij de meeste van deze bepalingen sluit de ene de andere niet uit en sommige licenties bevatten verscheidene. De rode draad hier is dat ze eisen stellen aan de ontvanger in ruil voor het recht van de gebruiker op gebruik en/of verspreiding van de code. Bij sommige projecten willen ze bijvoorbeeld dat de naam en de reputatie doorgegeven wordt met de code en vinden ze dat de extra moeite van een credit- of handelsmerkclausule waard. De zwaarte van de verplichtingen echter, kan een reden zijn voor sommige gebruikers om voor een pakket met een minder dwingende licentie te kiezen.

9.3 DE GPL EN LICENTIECOMPATIBILITEIT

Verreweg de meest duidelijke scheidslijn tussen licenties is die tussen wel propriëtair-compatibele en niet-propriëtair-compatibele licenties. Dat is dus de scheidslijn tussen de GNU General Public-licentie en alle anderen. Omdat het primaire doel van de GPL-auteurs de promotie van vrije software is, hebben ze de licentie opzettelijk zo gemaakt dat het onmogelijk is om de code met GPL te gebruiken in propriëtaire programma's. Specifieke GPL-vereisten (zie http://www.fsf.org/licensing/licenses/gpl.html voor de volledige tekst) zijn deze twee:

1. Elk afgeleid werk, dat wil zeggen, elk werk dat een niet onbelangrijk deel code bevat met GPL, moet verspreid worden onder de GPL.

2. Er mogen geen restricties toegevoegd worden, niet aan het origineel en niet aan het afgeleide werk. (De exacte omschrijving is: "You may not impose any further restrictions on the recipients' exercise of the rights granted herein." NL: "U mag geen verdere beperkingen opleggen inzake de rechten van de ontvangers dan degenen die hierin zijn opgenomen.")

Met deze voorwaarden is de GPL erin geslaagd het open maken van software aanstekelijk te maken. Als van een programma de auteursrechten eenmaal vastgelegd zijn onder de GPL, worden de voorwaarden voor verspreiding als een *virus* verspreid. Dat wil zeggen dat ze overgaan op alles waarin de code opgenomen wordt, hetgeen het onmogelijk maakt om code met GPL te gebruiken in programma's met closed-source. Dezelfde clausules maken de GPL echter soms niet compatibel met sommige andere vrije licenties. Gewoonlijk gebeurt dit doordat andere licenties een eis opleggen (bijvoorbeeld een creditclausule die vereist dat de originele auteurs op een bepaalde wijze genoemd worden) die niet compatibel is met de vereiste van de GPL. "U mag geen verdere beperkingen opleggen ..." . Vanuit het oogpunt van de Free Software Foundation zijn de consequenties van deze bijkomende eisen wenselijk, of op zijn minst niet te negatief. De GPL houdt niet alleen uw software vrij, maar maakt uw software een doeltreffende vertegenwoordiger van vrije software en kan zo ook andere software pushen vrijheden toe te staan.

De vraag of dit een goede manier is om vrije software te promoten, blijft één van de meest hardnekkige heilige oorlogen op internet (zie het gedeelte 'Heilige oorlogen voorkomen' in Hoofdstuk 6, *Communicatie*)), hier zullen we dit niet verder bespreken. Wat belangrijk is voor ons doel is dat compatibiliteit met de GPL een belangrijke kwestie is wanneer u een licentie gaat kiezen. De GPL is verreweg de meest populaire open source-licentie. Op http://freshmeat.net/stats/#license scoort het 68% en de eerstvolgende in de rij van gekozen licenties staat op 6%. Als u wilt dat uw code vrij gebruikt kan worden met de code met GPL (en er is erg veel code met GPL in omloop) dan moet u een licentie kiezen die compatibel is met de GPL. De meeste open source-licenties die compatibel zijn met de GPL zijn ook compatibel met propriëtaire licenties, Oftewel, code onder een dergelijke licentie kan gebruikt worden in een programma met GPL en het kan gebruikt worden in een propriëtair programma. Het *resultaten* van combinaties van deze twee zijn uiteraard niet compatibel met elkaar, omdat de één onder de GPL zal vallen en de ander onder een closed-source-licentie. Maar die zorg treft alleen de afgeleide werken en niet de code die u oorspronkelijk verspreidde.

Gelukkig houdt de Free Software Foundation een lijst bij waarin alle licenties zijn opgenomen die compatibel zijn met de GPL en welke dat niet zijn, op http://www.gnu.org/licenses/license-list.html. Alle besproken licenties in dit hoofdstuk staan op die lijst, aan de ene kant of aan de andere.

9.4 EEN LICENTIE KIEZEN

Wanneer u een licentie voor uw project wilt kiezen, kies dan als het mogelijk is een bestaande licentie in plaats van een nieuwe te maken. Er zijn twee redenen waarom een bestaande licentie beter is:

- Bekendheid. Als u één van de drie of vier meest populaire licenties kiest dan hebben mensen niet het gevoel dat ze eerst al het juridische jargon moeten lezen om uw code te kunnen gebruiken, omdat ze dit lang geleden al gedaan hebben voor die licentie.
- Kwaliteit. Tenzij u een team juristen tot uw beschikking hebt, is het onwaarschijnlijk dat u met een legale en solide licentie op de proppen kunt komen. De hier genoemde licenties zijn het resultaat van veel overdenkingen en ervaring. Tenzij uw project uiterst ongewone behoeften heeft, is het onwaarschijnlijk dat u het er beter van afbrengt.

Raadpleeg het gedeelte 'Hoe u een licentie toepast op uw software' in Hoofdstuk 2,

Het begin om te weten te komen hoe u één van deze licenties kunt aanvragen voor uw project.

De MIT / X Window System-licentie
Als uw doel is om uw code toegankelijk te maken voor het grootst mogelijke aantal ontwikkelaars en afgeleide werken, en u hebt er geen bezwaar tegen dat uw code gebruikt wordt in propriëtaire programma's, kies dan de MIT / X Window System-licentie (deze is zo genoemd omdat het de licentie is waaronder the Massachusetts Institute of Technology de originele X Window System-code heeft uitgebracht). De primaire boodschap van deze licentie is "U bent vrij om deze code te gebruiken zoals u wilt." Hij is compatibel met de GNU GPL, hij is kort, simpel en makkelijk te begrijpen:

Copyright (c) <year> <copyright holders>

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the 'Software'), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

(Overgenomen van http://www.opensource.org/licenses/mit-license.php.)

De GNU General Public-licentie
Als u niet wilt dat de code van uw project gebruikt wordt in propriëtaire programma's of als het u niets kan schelen of het wel of niet gebruikt kan worden in propriëtaire programma's, kies dan de GNU General Public-licentie (http://www.fsf.org/licensing/licenses/gpl.html). De GPL is vandaag de dag waarschijnlijk de meest gebruikte licentie voor open source-software ter wereld. Directe herkenbaarheid is een van GPL's grootste voordelen.

Als u een codebibliotheek maakt die voornamelijk is bedoeld om gebruikt te worden als een gedeelte van andere programma's, overweeg dan zorgvuldig of de restricties die opgelegd zijn door de GPL stroken met het doel van uw project. In sommige gevallen (bijvoorbeeld als u probeert een concurrerende propriëtaire bibliotheek uit het zadel te wippen) kan het wat strategisch inzicht vereisen om voor uw code een licentie te vinden die gebruikt kan worden in combinatie met propriëtaire programma's, ook al zou u dit in eerste instantie niet gewild hebben. De Free Software Foundation heeft zelfs een alternatief gemaakt voor de GPL voor dergelijke situaties: de *GNU Library GPL*, die later de nieuwe naam *GNU Lesser GPL* kreeg (de meeste mensen gebruiken het acroniem *LGPL*). De LGPL heeft minder strenge restricties dan de GPL en kan makkelijker worden gecombineerd met niet-vrije code. Hoe dan ook, het blijft een beetje complex en het vraagt wat tijd om het te kunnen begrijpen. Dus als u de GPL niet gaat gebruiken, raad ik u aan de MIT/X-style-licentie te nemen.

Is de GPL gratis of niet gratis?
Een consequentie van de keuze voor de GPL is de kans (die kans is klein, maar niet oneindig klein) dat uw project verwikkeld raakt in een conflict over de vraag of de GPL daadwerkelijk 'vrij' is, gegeven het feit dat deze enkele restricties oplegt over wat u mag doen met de code, namelijk de restrictie dat de code niet meer verspreid mag worden onder een andere licentie. Voor sommige mensen betekent het bestaan van deze restrictie dat de GPL 'minder vrij' is dan de meer tolerante licenties als de MIT/X-licentie. Waar deze discussie meestal in uitmondt, is dat 'vrijer' beter is dan 'minder vrij' (wie is er per slot van rekening geen voorstander van vrijheid?) en dat deze licenties dus wel beter moeten zijn dan de GPL.

Deze discussie is een van de populaire heilige oorlogen (zie het gedeelte 'Heilige oorlogen voorkomen' in Hoofdstuk 6*, Communicatie*). Laat u zich niet tot dergelijke discussie verleiden, in ieder geval niet op de projectforums. Probeer niet te bewijzen dat de GPL minder vrij, even vrij of vrijer is dan andere licenties. Benadruk in plaats daarvan de specifieke redenen waarom er binnen uw project gekozen is voor de GPL. Als de herkenbaarheid van de licentie de reden was, zeg dat dan. Als de handhaving van een vrije licentie over de afgeleide werken ook een reden was, meld dat dan ook, maar weiger deel te nemen aan de discussie over of het deze code meer of minder 'vrij' maakt. Vrijheid is een ingewikkelde materie en er is weinig reden om over de terminologie te praten als dit gebruikt gaat worden als afleiding van het onderwerp waar het in wezen om draait.

Dit is een boek en geen thread op een mailinglijst, maar ik moet toegeven dat ik het argument 'GPL is niet vrij' nooit begrepen heb. De enige restrictie die de GPL oplegt, is degene die mensen er van weerhoudt *verdere* restricties op te leggen. Om te zeggen dat dit resulteert in minder vrijheid doet me voorgekomen als zeggen dat slavernij verbieden vrijheid reduceert, omdat het sommige mensen ervan zal weerhouden slaven te hebben.

(O ja, en als u zich dan toch bij deze discussie hebt laten betrekken, drijf de boel dan niet op de spits door opruiende analogieën aan te halen.)

En de BSD-licentie?
Een behoorlijk deel van alle open source-software is verspreid onder een *BSD-licentie* (of soms een *BSD-achtige licentie*). De originele BSD-licentie is gebruikt voor de Berkeley Software Distribution, waaronder de Universiteit van Californië belangrijke gedeeltes van de Unix-implementatie heeft uitgebracht. Deze licentie (de exacte

tekst kunt u vinden in sectie 2.2.2 op http://www.xfree86.org/3.3.6/COPYRIGHT2.html#6) is van gelijke strekking als de MIT/X-licentie, op één clausule na:

All advertising materials mentioning features or use of this software must display the following acknowledgement: This product includes software developed by the University of California, Lawrence Berkeley Laboratory.

De aanwezigheid van die clausule maakte niet alleen de oorspronkelijke BSD-licentie incompatibel met GPL, maar creëerde ook een gevaarlijk precedent. Toen andere organisaties gelijke advertentieclausules in *hun* vrije software gingen zetten (en hun eigen naam lieten toevoegen in plaats van 'the University of California, Lawrence Berkeley Laboratory') werd het voor softwareverspreiders steeds moeilijker te bepalen wat ze moesten vermelden. Gelukkig werden veel van de projecten die deze licentie gebruikten zich bewust van het probleem en hebben ze simpelweg de advertentieclausule geschrapt. In 1999 heeft zelfs de Universiteit van Californië dit gedaan.

Het resultaat is een herziene versie van de BSD-licentie, die gewoon de oorspronkelijke BSD-licentie is maar dan zonder de advertentieclausule. Deze achtergrond maakt de term 'BSD-licentie' echter een beetje dubbelzinnig. Verwijst hij naar de oorspronkelijke of de herziene versie? Dit is waarom de MIT/X-licentie is te prefereren. Deze is in essentie gelijkwaardig, maar heeft niet te lijden van enige dubbelzinnigheid. Er is misschien één reden om de voorkeur te geven aan de herziene versie van de BSD-licentie boven de MIT/X-licentie, en dat is deze clausule uit BSD-licentie:

Neither the name of the <ORGANIZATION> nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

Het is onduidelijk of zonder een dergelijke clausule een ontvanger van de software het recht zou hebben om de naam van de licentiehouder te gebruiken. Deze clausule echter, neemt iedere mogelijke twijfel weg. Voor organisaties die bezorgd zijn over de controle over hun handelsmerk, valt de herziene versie van de BSD-licentie wellicht net te prefereren boven die van de MIT/X. Algemeen impliceert een liberaal auteursrecht gewoonlijk niet dat de ontvangers uw handelsmerken mogen gebruiken of verzwakken. De wet op de auteursrechten en de wet op de handelsmerken zijn twee zeer verschillende dingen.

Als u de herziene versie van de BSD-licentie wilt gebruiken, is er een sjabloon beschikbaar op http://www.opensource.org/licenses/bsd-license.php.

9.5 TOEKENNEN VAN AUTEURSRECHT EN EIGENDOM

Er zijn drie manieren om om te gaan met het eigendom van auteursrechten voor vrije code en documentatie die ontwikkeld zijn door meerdere mensen. De eerste is om de kwestie van het auteursrecht geheel te negeren (ik raad dit niet aan).

De tweede is om een *contributor license agreement* (*CLA*) van iedere persoon die binnen het project werkzaam is te verkrijgen, waarin aan het project uitdrukkelijk het recht wordt toegekend om de bijdragen van die persoon te gebruiken. Dit is gewoonlijk voldoende voor de meeste projecten en het prettige is dat in sommige rechtsgebieden een CLA per e-mail verzonden mag worden. De derde manier is om daadwerkelijke eigendomsoverdracht te krijgen van auteursrechten van mensen die bijdragen, zodat het project (d.w.z. een legale eenheid, meestal een non-profit-organisatie) de eigenaar is van alle auteursrechten. Dit een legale en waterdichte manier, maar het geeft ook de meeste rompslomp voor degenen die bijdragen aan het project. Er zijn dan ook maar enkele projecten die er op staan dat het op deze manier gebeurt.

Er moet opgemerkt worden dat zelfs bij gecentraliseerd geregelde auteursrechten de code vrij blijft[31], omdat open source-licenties de auteursrechthebbende niet het recht geven om met terugwerkende kracht zich alle kopieën van de code toe te eigenen. Dus zelfs als het project, als legale eenheid, plotseling van mening zou veranderen en zou beginnen alle code onder een licentie met veel restricties uit te brengen dan zou dit geen problemen geven voor de publieke gemeenschap. De andere ontwikkelaars zouden simpelweg een fork starten gebaseerd op de laatste vrije kopie en verdergaan alsof er niets aan de hand is. Omdat men weet dat ze dit kunnen doen werken de meeste mensen die bijdragen mee als hun wordt gevraagd om een CLA of een toekenning van de auteurrechten te ondertekenen.

Niets doen

Bij de meeste projecten worden van de contribuanten geen CLA's of auteursrechtoverdracht gevraagd. In plaats daarvan accepteren ze code telkens wanneer het redelijkerwijze duidelijk is dat het de bedoeling was van de contribuant om de code bij het project in te lijven.

Onder normale omstandigheden is dit in orde. Zo nu en dan kan er echter iemand beslissen om een rechtzaak aan te spannen over schending van auteursrechten en beweren dat hij de ware eigenaar is van de code in kwestie en dat hij het project nooit toestemming gegeven heeft om de code te verspreiden onder een open source-licentie. De SCO Group deed bijvoorbeeld iets dergelijks bij het Linux project. Zie http://en.wikipedia.org/wiki/SCO-Linux_controversies voor details. Wanneer dit gebeurt, beschikt het project niet over documentatie om aan te kunnen tonen dat de contribuant het recht om de code te gebruiken heeft overgedragen, wat het verdedigen in een rechtzaak moeilijker kan maken.

Contributor-licentieovereenkomsten

CLA's bieden waarschijnlijk de beste balans tussen veiligheid en gemak. Een CLA is meestal een elektronisch formulier dat een ontwikkelaar invult en opstuurt naar het project. In vele rechtsgebieden is een verklaring per e-mail genoeg. Soms is een gewaarborgde digitale ondertekening vereist. Raadpleeg een jurist om na te gaan welke optie het beste is voor uw project.

De meeste projecten gebruiken twee enigszins verschillende CLA's, één voor individuen en één voor bedrijven. Voor beide types geldt echter dat de kern hetzelfde is:

de contribuant kent het project *"...eeuwigdurende, wereldwijde, niets uitsluitende, kosteloze, royaltyvrije en onherroepelijke auteursrechtlicentie om te reproduceren, afgeleide werken te maken, publiekelijk te presenteren, publiekelijk uit te voeren, sublicenties op aan te vragen en bijdragen en daarvan afgeleide werken te distribueren"* toe. Ook hier zou u een jurist in de arm moeten nemen om de CLA goed te keuren, maar als u al deze toevoegingen erin zet, zit u waarschijnlijk goed.

Wanneer u uw contribuanten vraagt een CLA te ondertekenen, wees dan zeker dat u duidelijk maakt dat u *niet* om het feitelijke auteursrecht vraagt. Veel CLA's beginnen met een opmerking voor de lezer:

Dit is alleen een licentieovereenkomst; deze draagt geen auteursrechten over en verandert niets aan uw rechten om uw bijdragen te gebruiken voor welk ander doel dan ook.

Hier zijn enkele voorbeelden:

- CLA's voor individuele contribuanten:
  - http://apache.org/licenses/icla.txt
  - http://code.google.com/legal/individual-cla-v1.0.html
- CLA's voor bedrijven:
  - http://apache.org/licenses/cla-corporate.txt
  - http://code.google.com/legal/corporate-cla-v1.0.html

Overdracht van auteursrecht

Overdracht van auteursrecht betekent dat de contribuant het auteursrecht van zijn bijdragen toekent aan het project. Dit moet schriftelijk worden gedaan en per fax of per post naar het project worden gezonden.

Binnen sommige projecten staat men op volledige overdracht omdat het nuttig kan zijn om het auteursrecht van de gehele code onder te brengen bij één rechtspersoon voor het geval de voorwaarden van de open source-licentie afgedwongen moeten worden in een rechtzaak. Als er niet één rechtspersoon is die het recht heeft om dit te doen dan moeten alle ontwikkelaars meewerken. Sommigen hebben wellicht geen tijd echter, of zijn zelfs niet bereikbaar op het moment dat de zaak voorkomt.

Niet alle organisaties zijn even streng op het gebied van het inzamelen van auteursrechtoverdracht. Sommige vragen alleen om een informele verklaring van een codecontribuant op een publieke mailinglijst, vaak in de trant van "Ik verklaar hierbij het auteursrecht van deze code toe te kennen aan het project waarbij de code onder dezelfde licentie valt als al de andere code." Minstens één advocaat die ik hierover gesproken heb heeft gezegd dat dit voldoende moet zijn, vermoedelijk omdat het in een context staat waar het overdragen van auteursrecht normaal en verwacht is, en omdat het op een betrouwbare manier laat zien dat het project moeite doet om de werkelijke bedoelingen van de ontwikkelaar te achterhalen. De Free Software Foundation houdt echter vast aan het andere uiterste. Ze eisen van de contribuanten dat ze een formele schriftelijke verklaring ondertekenen waarop staat dat ze het auteursrecht overdragen, soms voor een enkele bijdrage en soms voor lopende of toekomstige bijdragen. Als de ontwikkelaar een werkbetrekking heeft vraagt de FSF de werkgever ook te tekenen.

Deze paranoia van de FSF is te begrijpen. Als iemand de rechten van de GPL schendt door iets van hun software in te lijven in een propriëtair programma, dan zal de FSF dat aanvechten in een rechtzaak, en ze willen een zo waterdicht mogelijk auteursrecht wanneer dit gebeurt. Sinds de FSF auteursrechthouder is van veel populaire software denken ze dat er een reële kans bestaat dat dit gebeurt. Of uw organisatie even nauwgezet te werk moet gaan, is iets dat alleen u kunt beslissen in overleg met uw advocaten. Tenzij er een specifieke reden is waarom uw project volledig auteursrecht nodig heeft, kunt u normaal gesproken gewoon de CLA's gebruiken; dat is het makkelijkst voor iedereen.

9.6 TWEEVOUDIGE LICENTIEPROGRAMMA'S

Sommige projecten proberen aan inkomsten te komen door een tweevoudig licentieprogramma te gebruiken, waarin afgeleide werken met auteursrecht de auteursrechthouder voor het gebruik van de code kunnen betalen, maar waarbij de code nog wel vrij te gebruiken blijft voor open source-projecten. Dit werkt doorgaans natuurlijk beter bij codebibliotheken dan losstaande toepassingen. De exacte voorwaarden verschillen van geval tot geval. Vaak is de licentie voor het vrije gedeelte de GNU GPL, omdat deze anderen toch al verbiedt de beschermde code in hun Auteursrechtelijk beschermde product te gebruiken zonder toestemming van de auteursrechthouder, maar het kan ook een op maat gemaakte licentie zijn met hetzelfde effect. Een voorbeeld van de eerstgenoemde is de MySQL-licentie, zoals te zien op http://www.mysql.com/company/legal/licensing/, een voorbeeld van de tweede is Sleepycat Softwares licentiestrategie, te zien op http://www.oracle.com/technology/software/products/berkeley-db/htdocs/licensing.html.

U zou zich af kunnen vragen hoe een houder van auteursrechten een licentie kan aanbieden voor een verplicht bedrag als de GNU GPL-licentie vereist dat de code verkrijgbaar moet zijn onder veel minder beperkende voorwaarden? Het antwoord hierop is dat de auteursrechthouder de GPL voorwaarden oplegt aan alle anderen; de eigenaar is daarom vrij om te besluiten dat deze voorwaarden *niet* voor hemzelf gelden. U kunt zich voorstellen dat de auteursrechthouder een oneindig aantal kopieën van de software opgeslagen heeft. Iedere keer dat hij er eentje pakt om de wereld in te sturen kan hij beslissen wat voor licentie hij daarop toepast: GPL, auteursrechten of iets anders. Zijn recht om dit te doen is niet gebonden aan de GPL of aan welke open source-licentie dan ook. Het is simpelweg een recht dat hij heeft op basis van de wet op auteursrechten.

De aantrekkelijkheid van tweevoudige licenties is dat het een manier verschaft aan open source-softwareprojecten zichzelf van een betrouwbare inkomstenstroom te voorzien. Helaas botst dit soms met de normale dynamiek van open source-projecten. Het probleem is dat iedere vrijwilliger die een code bijdraagt dit nu doet aan twee verschillende entiteiten: de vrije versie van de code en de versie met auteursrechten. Terwijl de contribuant zich prettig kan voelen met zijn bijdrage aan de vrije

versie, omdat dat nu eenmaal de norm is binnen open source-projecten, kan hij een onprettig gevoel overhouden aan het feit dat hij ook bijdraagt aan iemands semi-propriëtaire inkomstenbron. Dit is niet erg elegant, wat wordt verergerd door het feit dat bij tweevoudige licenties de eigenaar daadwerkelijk formele en getekende auteursrechttoekenning moet gaan verzamelen van alle contribuanten, om op deze manier te voorkomen dat een ontevreden contribuant van code later een percentage gaat claimen van de royalty's. Dit proces van het verzamelen van de toekenningspapieren confronteert de contribuanten met het feit dat ze bezig zijn geld te verdienen voor iemand anders.

Niet alle vrijwilligers zullen zich hieraan storen. Uiteindelijk komen hun bijdragen ook terecht in de open source-versie, wat precies in hun belang is. Desalniettemin, tweevoudige licenties zijn een voorbeeld van een speciaal recht dat de auteursrechthouder zich toekent, dat de anderen binnen het project niet hebben en daarom zeker tot spanning kan leiden, in ieder geval bij enkele vrijwilligers.

Wat er gebeurt in de praktijk is dat bedrijven die gebaseerd zijn op software met tweevoudige licenties niet daadwerkelijk een egalitaire ontwikkelaarsgemeenschap hebben. Ze krijgen kleinschalige bugfixes en patches van externe bronnen, maar uiteindelijk moeten ze het echte werk zien te klaren met interne middelen. Zack Urlocker, Vice President marketing bij MySQL, vertelde bijvoorbeeld dat de meeste bedrijven uiteindelijk toch de meest actieve vrijwilligers moesten inhuren. Dus hoewel het project zelf open source is met een licentie onder de GPL, wordt de ontwikkeling ervan in meer of mindere mate gecontroleerd door het bedrijf, ondanks het (zeer onwaarschijnlijke) feit dat iemand die ontevreden is met de manier waarop het bedrijf met de software omgaat een fork maakt van het project. Tot op welke hoogte dit invloed heeft op het beleid van een bedrijf weet ik niet, maar in ieder geval lijkt MySQL geen acceptatieproblemen te hebben, niet in de open source-wereld en ook niet daarbuiten.

9.7 PATENTEN

Softwarepatenten zijn momenteel een heet hangijzer binnen de vrije software, omdat ze de enige bedreiging zijn waartegen de vrije softwaregemeenschap zich niet kan verdedigen. Problemen met auteursrechten en handelsmerken zijn wel te omzeilen. Als het erop lijkt dat een gedeelte van uw code inbreuk maakt op het auteursrecht van een ander, dan kunt u dat gedeelte gewoon herschrijven. En als het blijkt dat iemand een handelsmerkrecht heeft op de naam van uw project dan moet u in het ergste geval de naam van het project veranderen. Alhoewel een naamsverandering tijdelijk ongemakken geeft, maakt het op de lange duur niet uit, omdat de code op zich nog steeds doet wat hij altijd deed.

Een patent is echter een allesomvattend verbod op het implementeren van een bepaald idee. Het maakt niet uit wie de code schrijft of welke programmeertaal gebruikt wordt. Op het moment dat iemand een open source-softwareproject ervan beschuldigt inbreuk te maken op een patent, dan moet het project of stoppen met het implementeren van die bepaalde functie of een dure en tijdrovende rechtszaak voeren. Omdat de aanstichters van dergelijke rechtszaken meestal bedrijven zijn met veel geld (dat zijn sowieso degenen die de middelen en de neiging hebben om patenten aan te vragen) kunnen de meeste open source-softwareprojecten zich de laatste mogelijkheid niet permitteren en moeten zich onmiddellijk gewonnen geven, zelfs als ze denken dat ze een goede kans maken dat het patent voor de rechter niet standhoudt. Om te voorkomen dat ze in een dergelijke situatie verzeild raken, zetten open source-softwareprojecten hun code steeds vaker defensief op, om op voorhand gepatenteerde algoritmes te vermijden ook al zij die de best bruikbare of de enige beschikbare oplossing voor een probleem in een programma.[32]

Samenvattingen en anekdotisch bewijs laten zien dat niet alleen de overgrote meerderheid van open source-programmeurs, maar een meerderheid van *alle* programmeurs, vindt dat patenten op software afgeschaft moeten worden.[33] Open source-programmeurs vinden dit vaak uitgesproken belangrijk en kunnen weigeren aan projecten mee te werken die te sterk verband houden met het verkrijgen of handhaven van softwarepatenten. Als uw organisatie softwarepatenten verzamelt, maak dan publiekelijk en onherroepelijk duidelijk dat deze patenten nooit zullen worden ingezet tegen open source-projecten en dat ze alleen gebruikt mogen worden als verdediging voor het geval een andere partij een rechtszaak wegens inbreuk begint tegen uw organisatie. Dit is niet alleen correct, het is ook goede reclame voor de open source-wereld.[34]

Jammer genoeg is het defensief verzamelen van patenten een rationele actie. Het huidige patenteersysteem, in ieder geval in de Verenigde Staten, lijkt wel op een wapenwedloop. Als uw mededingers veel patenten hebben verworven, dan is de beste verdediging om zelf een hoop patenten te verzamelen, zodat wanneer u een rechtszaak aangespannen krijgt wegens inbreuk op een patent u dit kunt beantwoorden met een vergelijkbare dreiging. Op dat moment gaan de beide partijen meestal om de tafel zitten om een afspraak over de licentie te maken zodat beide partijen niets hoeven te betalen, behalve hun advocaten natuurlijk.

De schade die open source-software ondervindt door softwarepatenten is hoe dan ook verraderlijker dan een direct gevaar voor de ontwikkeling van code. Softwarepatenten moedigen een sfeer aan van geheimzinnigheid rondom de ontwikkelaars van firmware, die terecht bezorgd zijn dat door het publiceren van details van hun interfaces ze technische hulp geven aan concurrenten die een manier zoeken om een rechtszaak aan te spannen wegens inbreuk op een patent. Dit is niet slechts een theoretisch gevaar. Het gebeurt klaarblijkelijk al heel lang in de videokaartensector bijvoorbeeld. Veel producenten van videokaarten zijn terughoudend met het vrijgeven van de gedetailleerde programmaspecificaties die nodig zijn om krachtige open source-drivers te produceren voor hun kaarten. Zo maken ze het onmogelijk voor vrij opererende systemen om deze kaarten voor hun volle capaciteit te ondersteunen. Waarom doen de producenten dit? Het lijkt onlogisch als ze softwareondersteuning *tegenwerken.* Uiteindelijk betekent compatibiliteit met meerdere besturingssystemen alleen maar meer verkoop van kaarten. Het blijkt echter dat, achter de deur van de ontwerpkamer, deze bedrijven allemaal elkaars patenten schenden, soms per ongeluk en soms willens en wetens. De patenten zijn zo onvoorspelbaar en potentieel zo breed dat geen enkele kaartproducent ooit zeker weet of hij goed

zit, zelfs niet na een patentenonderzoek. Daarom durven producenten hun volledige interfacespecificaties niet te publiceren omdat dat het voor de concurrenten veel gemakkelijker zou maken om uit te zoeken of er inbreuk op hun patenten gedaan gepleegd. (Uiteraard is de aard van deze situatie zo, dat u dit nergens zwart op wit en uit de eerste hand zult terugvinden; ik ben het te weten gekomen in een persoonlijk gesprek.)

Sommige vrijesoftwarelicenties hebben speciale clausules om softwarepatenten te bevechten of in ieder geval te ontmoedigen. De GNU GPL bijvoorbeeld bevat deze tekst:

7. If, as a consequence of a court judgment or allegation of patentinfringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence youmay not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

[...]

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

De Apachelicentie, versie 2.0 (http://www.apache.org/licenses/LICENSE-2.0) bevat ook antipatentvoorwaarden. Allereerst bepaalt de licentie dat iedereen die code verspreidt onder deze licentie impliciet een royaltyvrije licentie moet hebben voor eventuele patenten die op de code van toepassing zouden kunnen zijn. Ten tweede straft het iedereen op ingenieuze wijze af die een inbreuk claimt op het beschermde werk, door automatisch hun impliciete patentlicentie uit te schakelen op het moment dat ze een claim indienen:

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

Alhoewel het nuttig is, zowel juridisch als politiek gezien, om op deze manier een verdediging tegen patenten in te bouwen in open source-softwarelicenties, zullen deze stappen uiteindelijk niet genoeg zijn om het verkillende effect dat de dreiging van rechtszaken om patenten heeft op vrije software weg te nemen.

Alleen veranderingen aan de inhoud of de interpretatie van internationaal patentrecht kunnen dat doen.

Om meer te weten te komen over dit probleem en hoe het aangepakt kan worden, ga naar http://www.nosoftwarepatents.com/. Het volgende artikel op Wikipedia http://en.wikipedia.org/wiki/Software_patent biedt ook veel bruikbare informatie over softwarepatenten. Ik heb ook een blogpost geschreven met daarin de samenvatting van de argumenten tegen software patenten, op http://www.rants.org/2007/05/01/how-to-tell-that-software-patents-are-a-bad-idea/.

9.8 VERDERE BRONNEN

Dit hoofdstuk is alleen een introductie over open source-softwarelicenties. Ik hoop hoe dan ook dat het genoeg informatie bevat om u op gang te helpen met uw eigen open source-project. Ieder serieus onderzoek naar licentiekwesties zal meer opleveren dan dit boek aan informatie kan bieden. Hier volgt een lijst met meer bronnen over open source-licenties:

- *Understanding Open Source and Free Software Licensing* door Andrew M. St. Laurent. Gepubliceerd door O'Reilly Media, eerste editie August 2004, ISBN: 0-596-00581-4.

  Dit is een boek van gemiddelde lengte over open source-licenties in al hun complexiteit, inclusief veel onderwerpen die zijn weggelaten in dit hoofdstuk. Kijk op http://www.oreilly.com/catalog/osfreesoft/ voor details.

- *Make Your Open Source Software GPL-Compatible. Or Else.* Door David A. Wheeler, op http://www.dwheeler.com/essays/gpl-compatible.html.

  Dit is een goed geschreven en gedetailleerd artikel over waarom het belangrijk is om een GPL-compatibele licentie te nemen, ook al gebruikt u de GPL-licentie zelf niet. Het artikel behandelt ook veel andere licentiekwesties en bevat een groot aantal zeer goede links.

- *http://creativecommons.org/*
  Creative Commons is een organisatie die een stelsel van meer flexibele en

liberale auteursrechten promoot dan de huidige toepassingen van het auteursrecht. Ze bieden niet alleen licenties voor software, maar ook voor tekst, kunst en muziek, allemaal toegankelijk via een gebruiksvriendelijke licentiekiezer. Sommige licenties zijn copylefts, sommigen zijn niet-copyleft maar nog steeds gratis, andere zijn gewoonweg traditionele auteursrechten met wat minder strikte beperkingen. De website van Creative Commons geeft zeer duidelijke uitleg over waar het allemaal om gaat. Al ik een site zou moeten kiezen om de bredere filosofische implicaties van de vrijesoftwarebeweging te demonstreren, dan zou ik deze kiezen.

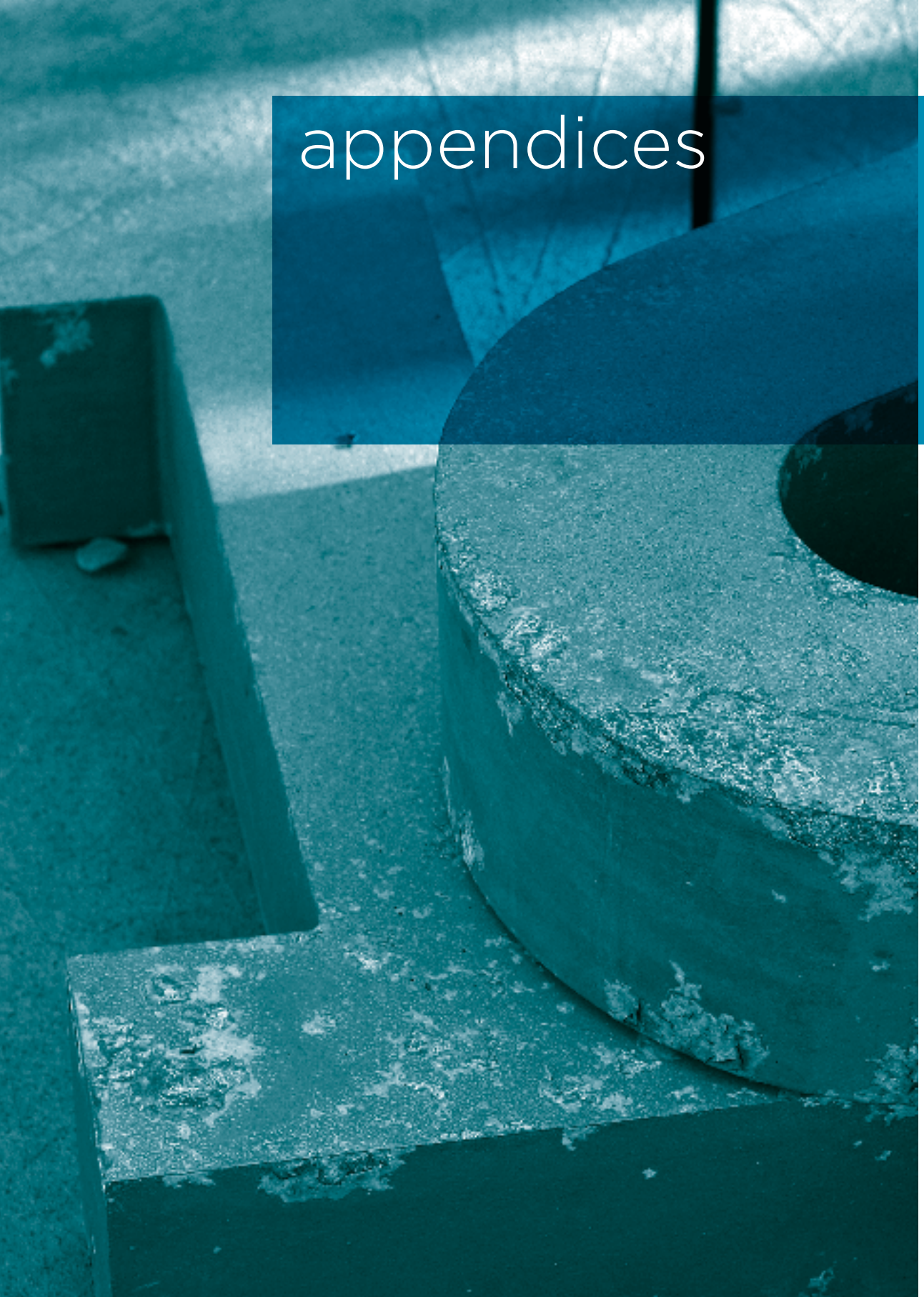
30| De overdracht van auteursrechten is op dit moment onderhevig aan nationale wetgeving en licenties ontworpen voor de Verenigde Staten kunnen ergens anders problemen geven (bijv. in Duitsland, waar het klaarblijkelijk niet mogelijk is auteursrechten over te dragen).

31| Ik zal vanaf nu het woord 'code' gebruiken om te verwijzen naar zowel code als documentatie.

32| Sun Microsystems en IBM hebben in ieder geval een gebaar gemaakt naar aanleiding van dit probleem door een groot aantal softwarepatenten vrij te geven (respectievelijk 1600 en 500) voor gebruik door de open source-gemeenschap. Ik ben geen jurist en kan daarom de werkelijke bruikbaarheid niet beoordelen van deze concessies, maar zelfs als het allemaal belangrijke patenten zijn, en de voorwaarden maken ze daadwerkelijk vrij voor gebruik door ieder open source-project, dan nog is het een druppel op een gloeiende plaat.

33| Zie http://lpf.ai.mit.edu/Whatsnew/survey.html voor een samenvatting.

34| RedHat heeft bijvoorbeeld toegezegd dat open source-projecten veilig zijn voor hun patenten, zie http://www.redhat.com/legal/patent_policy.html.



appendices

A OPEN SOURCE-VERSIEBEHEER-SYSTEMEN

Dit zijn alle open source-versiebeheersystemen die ik in 2007 kon vinden. Het enige systeem dat ik zelf regelmatig gebruik is Subversion. Ik heb zelf weinig of geen ervaring met de meeste van deze systemen, behalve dan met Subversion en CVS; de onderstaande informatie is afkomstig van de betreffende websites. Zie ook http://en.wikipedia.org/wiki/List_of_revision_control_software.

CVS — http://www.nongnu.org/cvs/
CVS bestaat al lang en veel ontwikkelaars kennen het. Bij zijn introductie was het een revolutionair systeem. Het was het eerste open source-versiebeheersysteem met wide area-netwerktoegang voor ontwikkelaars (voor zover ik weet) en het eerste dat anonieme, read-only proefversies kon leveren, zodat nieuwe ontwikkelaars zich makkelijk konden inwerken in een project. CVS beheert alleen bestandsversies en geen mappen. Het biedt branching en tagging en werkt snel op client-computers, maar het heeft moeite om grote en binaire bestanden te verwerken. Het ondersteunt ook geen individuele commits (atomic commits). *[NB: Ik heb zo'n vijf jaar lang CVS helpen ontwikkelen en heb daarna geholpen om het vervangende Subversion-project te starten.]*

Subversion — http://subversion.tigris.org/
Subversion is in de eerste plaats geschreven om CVS te vervangen. Dat wil zeggen, om versiebeheer mogelijk te maken op ruwweg dezelfde manier als CVS, maar dan zonder de tekortkomingen daarvan en mét de functies die de gebruikers van CVS moeten missen. Een van de doelstellingen van Subversion is om bestaande gebruikers van CVS heel gemakkelijk van CVS naar Subversion te kunnen laten overstappen. Er is hier niet genoeg plaats om op alle functies van Subversion in te gaan. Zie de website voor meer informatie. *[NB: Ik ben betrokken bij de ontwikkeling van Subversion en het is het enige systeem dat ik zelf ook regelmatig gebruik.]*

SVK — http://svk.elixus.org/
Hoewel dit een schil is om Subversion, lijkt SVK meer op de hieronder genoemde decentrale systemen dan op Subversion zelf. SVK ondersteunt gedistribueerde ontwikkeling, lokale commits, het geavanceerd invoegen van wijzigingen en de mogelijkheid om trees te spiegelen van niet-SVK-versiebeheersystemen. Zie de website voor meer details.

Mercurial — http://www.selenic.com/mercurial/
Mercurial is een gedistribueerd versiebeheersysteem met, onder andere: 'volledige indexering van bestanden en wijzigingen; efficiënte HTTP- en SSH-synchronisatieprotocollen die weinig bandbreedte en processortijd vragen; arbitrair samenvoegen van ontwikkelings-branches; geïntegreerde standalonebrowser; [te porteren naar] UNIX, MacOS X en Windows' en nog veel meer (deze opsomming is overgenomen van de Mercurial-website).

GIT — http://git.or.cz/
GIT is een project van Linus Torvalds om de broncodeontwikkeling van Linux te beheren. Eerst richtte GIT zich uitsluitend op het ontwikkelen van de kernel, maar het is inmiddels uitgebreid naar andere projecten dan alleen de Linux-kernel. De website zegt dat GIT "... is bedoeld om zeer grote projecten efficiënt en snel te beheren. Dit zijn doorgaans allerlei open source-projecten, maar met name ook de Linux-kernel. GIT is een beheertool voor gedistribueerde broncode, vergelijkbaar met GNU Arch en Monotone (of BitKeeper in commerciële omgevingen). Elke Git-werkmap bevat een complete werkversie met wijzigingenbeheer en is niet afhankelijk van netwerktoegang of van een centrale server."

Bazaar — http://bazaar.canonical.com/
Bazaar is nog steeds in ontwikkeling. Het is een implementatie van het GNU Archprotocol dat ook tijdens zijn verdere ontwikkeling compatibel zal blijven met GNU Arch, waarbij er met de GNU Arch-gemeenschap wordt samengewerkt als er protocolwijzigingen nodig zijn in verband met de gebruiksvriendelijkheid.

Bazaar-NG — http://bazaar-ng.org/
Bazaar-NG (of bzr) wordt op dit moment door Canonical (http://canonical.com/) ontwikkeld. U kunt binnen één en hetzelfde project zowel centraal als decentraal werken. Zo kunt u op kantoor centraal aan een gemeenschappelijke ontwikkeling werken. Maar om experimentele veranderingen aan te brengen of om offline te werken, kunt u ook een versie op uw laptop aanmaken en die later weer samenvoegen.

Darcs — http://abridgegame.org/darcs/
"Ook dit 'David's Advanced Revision Control System' is een vervanger van CVS. Het is geschreven in Haskell en kan worden gebruikt onder Linux, MacOS X, FreeBSD, OpenBSD en Microsoft Windows. Darcs omvat een cgi-script, waarmee u de inhoud van uw repository kunt bekijken."

Arch — http://www.gnu.org/software/gnu-arch/
GNU Arch ondersteunt zowel gedistribueerde als gecentraliseerde ontwikkelingen. Ontwikkelaars schrijven hun wijzigingen weg naar 'een archief', al dan niet lokaal, waarna deze wijzigingen actief naar andere archieven worden verstuurd of door beheerders van andere archieven zelf worden opgehaald. Deze werkwijze verraadt al dat Arch meer merge-functies ondersteunt dan CVS. Met Arch kunt u heel eenvoudig versies creëren op basis van archieven waar u zelf geen commit access toe hebt. Dit is slechts een korte omschrijving; zie de Arch-website voor meer details.

monotone — http://www.venge.net/monotone/
"Monotone is een vrij en gedistribueerd versiebeheersysteem. Het biedt eenvoudige opslagruimte voor enkelvoudige bestandstransacties met volledige offlinefunctionaliteit, in combinatie met een efficiënt peer-to-peer synchronisatieprotocol. Het voorziet in merge-functies op basis van de historie, light-versies, geïntegreerde code review en externe testfuncties. Ook zijn cryptografische versienamen en client-side RSA-certificaten mogelijk. Het beschikt over goede lokalisatiefuncties, kent geen externe afhankelijkheden, draait onder Linux, Solaris, OSX en Windows en valt onder de GNU GPL-licentie."

Codeville — http://codeville.org/
"Waarom nóg een versiebeheersysteem? Bij alle andere versiebeheersystemen moet u de afhankelijkheden tussen de verschillende branches nauwkeurig in de gaten houden zodat u niet steeds weer dezelfde conflicten oproept. Codeville werkt veel anarchistischer. U kunt te allen tijde iets uit een archief ophalen of naar een archief committen zonder overbodige merges uit te voeren."

"Codeville werkt met een identificatiecode voor elke verandering en houdt een lijst bij met alle veranderingen aan alle bestanden en de laatste regelwijzigingen van alle bestanden. Bij conflicten wordt er gecontroleerd of één van de twee al eerder op de ander is uitgevoerd, en laat de ander automatisch voorgaan als dat het geval is. En als er bij het samenvoegen ooit versieconflicten optreden die niet automatisch kunnen worden opgelost, gaat Codeville daar op vrijwel dezelfde manier mee om als CVS."

Vesta — http://www.vestasys.org/
"Vesta is een porteerbaar SCM-systeem [Software Configuration Management] bedoeld voor het ondersteunen van softwareontwikkelingen in diverse ordegroottes, van vrij klein (minder dan 10.000 broncoderegels) tot zeer groot (10.000.000 broncoderegels)."

"Vesta is een volwassen systeem. Dit systeem is het resultaat van meer dan tien jaar onderzoek en ontwikkeling in het Compaq/Digital Systems Research Center, waarbij het door de Alpha-microprocessorgroep van Compaq meer dan tweeëneenhalf jaar lang is gebruikt voor productiedoeleinden. De Alpha-groep bestond uit meer dan 150 actieve ontwikkelaars op twee locaties aan respectievelijk de oost- en westkust van de Verenigde Staten die duizenden kilometers uit elkaar liggen. Deze groep gebruikte Vesta om builds te beheren met meer dan 130 MB aan broncode en met elk 1,5 GB aan afgeleide gegevens. De versies die in de locatie aan de oostkust werden gemaakt, produceerden elke dag gemiddeld zo'n 10-15 GB aan afgeleide gegevens. Dit alles werd met Vesta beheerd. Hoewel Vesta speciaal was bedoeld voor softwareontwikkeling, toonde de Alpha-groep aan dat dit systeem ook flexibel genoeg was voor hardwareontwikkeling. De broncodecontrolefuncties van Vesta konden ook hardwarebeschrijvingen controleren, en de compilerfuncties van Vesta konden ook simulaties en andere afgeleide objecten creëren. De leden van de oude Alpha-groep, nu een onderdeel van Intel, gebruiken Vesta nog steeds voor nieuwe microprocessorprojecten."

Aegis — http://aegis.sourceforge.net/
"Aegis is softwareconfiguratiebeheersysteem op basis van transacties. Het levert een kader waarin een team van ontwikkelaars tegelijk aan allerlei programmawijzigingen kan werken, waarbij Aegis zorgt voor het integreren van deze wijzigingen in de master-broncode van het programma, met zo min mogelijk storingen."

CVSNT — http://cvsnt.org/
"CVSNT is een geavanceerd multiplatformversiebeheersysteem. Het is compatibel met het populaire CVS-protocol maar beschikt over nog meer functies. [...] CVSNT is open source-software onder de GNU General Public License-licentie." Tot de functies behoren, onder andere, authenticatie via alle standaard-CVS-protocollen, én SSPI en Active Directory van Windows; 'secure transport support' via sserver of encrypted SSPI; platformonafhankelijkheid (werkt zowel in Windows- als Unix-omgevingen); de NT-versie is volledig Win32-geïntegreerd; met MergePoint-processing voor samenvoegen zonder tagging; en hij wordt actief verder ontwikkeld.
META-CVS — http://users.footprints.net/~kaz/mcvs.html
"Meta-CVS is een versiebeheersysteem op basis van CVS. Hoewel de meeste functies met die van CVS overeenkomen, inclusief de complete netwerkondersteuning, is het krachtiger en gebruikersvriendelijker dan CVS." De productkenmerken die op de website van META-CVS worden genoemd, zijn: mapstructuurversies, verbeterde bestandstypeafhandeling, eenvoudige en gebruikersvriendelijke branching en merging, ondersteuning van symboolkoppelingen, eigenschappenlijsten van gegevensversies, betere importfunctie voor gegevens van derden en gemakkelijk upgraden vanaf standaard-CVS.

OpenCM — http://www.opencm.org/
"OpenCM is ontworpen als veilige vervanger voor CVS met nadruk op de integriteit. Op de 'features'-pagina staat een overzicht van de belangrijkste productkenmerken. Hoewel het minder functies heeft dan CVS zelf, bevat het wel enkele functies die in CVS ontbreken. Kort samengevat: OpenCM levert hoogwaardige ondersteuning voor hernoemen en configureren, cryptografische authenticatie en toegangscontrole, en eersteklas-branches."

Stellation — http://www.eclipse.org/stellation/
"Stellation is een geavanceerd, schaalbaar softwareconfiguratiebeheersysteem dat is ontwikkeld door IBM Research. Stellation levert niet alleen alle standaardfuncties die u van een SCM-systeem mag verwachten, het beschikt ook over een aantal opvallende functies zoals taakgericht wijzigingenbeheer, consistente projectversies en light-versies, die het ontwikkelen van softwaresystemen vergemakkelijken met grote groepen ontwikkelaars en een minimale aansturing."

PRCS — http://prcs.sourceforge.net/
"PRCS is het 'Project Revision Control System' en vormt de interface voor een aantal tools die (net zoals CVS) de omgang regelt met groepen losse bestanden en mappen en het geheel tot coherente versies verwerkt. ... De doelstelling is dezelfde als die van SCCS, RCS en CVS, maar het werkt (volgens de makers) veel eenvoudiger dan die systemen."

ArX — http://www.nongnu.org/arx/
ArX is een gedistribueerd versiebeheersysteem met functies als branching en merging, cryptografische verificatie van de data-integriteit en de mogelijkheid om archieven gemakkelijk op elke gewenste http-server te publiceren.

SourceJammer — http://sourcejammer.org/
"SourceJammer is een broncode- en versiebeheersysteem dat is geschreven in Java. Het omvat een serverdeel dat bestanden en de versiehistorie bijhoudt en dat de check-in, check-out en andere commando's afhandelt; en een client-deel dat requests indient bij de server en de client-bestanden beheert."

FastCST — http://www.zedshaw.com/projects/fastcst/index.html
"Dit is een 'modern' systeem dat wijzigingen hanteert in plaats van bestandsrevisies en dat gedistribueerd werkt in plaats van centraal. Om FastCST te gebruiken is een e-mailaccount voldoende. Voor grotere distributies is slechts een FTP- en/of een HTTP-server vereist. Of gebruik het interne 'serve'-commando om uw resultaten te uploaden. Alle changesets zijn uniek en omvatten uitgebreide metadata zodat u zelf precies kunt bepalen wat u wel [en niet] wilt proberen. Bij het samenvoegen wordt de changeset eerst vergeleken met de actuele mapinhoud en niet direct op een andere changeset toegepast."

Superversion — http://www.superversion.org/
"Superversion is een gedistribueerd multi-user versiebeheersysteem dat werkt met changesets. Het streven is om hiermee een professioneel open source-alternatief te leveren voor de commerciële pakketten dat net zo gemakkelijk is in het gebruik en dat dezelfde krachtige functies levert. In feite staan het intuïtief en efficiënt gebruik al vanaf het begin op de prioriteitenlijst van Superversion."

B GRATIS BUG TRACKERS

Welke bug tracker een project ook gebruikt, er zijn altijd ontwikkelaars die erover klagen. Dit gaat meer op voor bug trackers dan voor iedere andere standaardontwikkelingstool. Ik denk dat dit komt omdat bug trackers zo zichtbaar en interactief zijn, dat je het je makkelijk voor kunt stellen welke verbeteringen je aan zou kunnen brengen (als je de tijd maar had) en daarom die verbeteringen dan maar luid en duidelijk beschrijft. Neem die onvermijdelijke klachten met een korreltje zout. Veel van de hieronder genoemde trackers zijn behoorlijk goed.

Bij de opsomming wordt het woord 'issue' gebruikt om te verwijzen naar de items die de trackers opsporen. Bedenk echter dat elk systeem zijn eigen terminologie kan hanteren en dit begrip wellicht aanduidt met 'artefact' of 'bug' of iets anders.

Bugzilla — http://www.bugzilla.org/
Bugzilla is erg populair, wordt actief bijgehouden en lijkt behoorlijk tevreden gebruikers te hebben. Ik gebruik voor mijn werk nu al vier jaar een gemodificeerde variant ervan en ik ben er tevreden over. Het valt niet uitgebreid op maat aan te passen, maar op een bepaalde manier is dat wellicht een van de goede kenmerken: Bugzilla-installaties zien er overal waar je ze tegenkomt vrijwel hetzelfde uit. Dat betekent dat veel ontwikkelaars gewend zijn aan de interface en zich op bekend terrein weten.

GNATS — http://www.gnu.org/software/gnats/
GNU GNATS is een van de oudste open source-bug trackers en wordt alom gebruikt. De sterke punten zijn interfacediversiteit – het programma kan niet alleen via een webbrowser worden gebruikt, maar ook via e-mail en command line-tools - en issueopslag in plaintext (ongeformatteerde tekst). Het feit dat alle issuegegevens worden opgeslagen in tekstbestanden op schijf maakt het makkelijker om speciale tools te schrijven om de gegevens te trawlen en te parsen, bijvoorbeeld om statistische rapporten te genereren. GNATS is ook in staat via diverse middelen e-mails te absorberen en deze op basis van patronen in de e-mailheaders toe te voegen aan de toepasselijke issues. Dit maakt het loggen van gebruikers-/ontwikkelaarsdiscussies erg eenvoudig

RequestTracker (RT) — http://www.bestpractical.com/rt/
Op de website van RT staat het volgende: "RT is an enterprise-grade ticketing system which enables a group of people to intelligently and efficiently manage tasks, issues, and requests submitted by a community of users." Dat zegt het wel zo'n beetje. RT heeft een redelijk gepolijste webinterface en lijkt een behoorlijk breed geïnstalleerde basis te hebben. Visueel is de interface wat ingewikkeld, maar dat wordt steeds minder verwarrend naarmate je eraan went. RT wordt gelicentieerd onder de GNU GPL (om de een of andere reden wordt dit niet duidelijk op de website).

Trac — http://trac.edgewall.com/
Trac is wat meer dan alleen een bug tracker. Het is eigenlijk een wiki en een bug tracker in één. Het gebruikt *wiki linking* om issues, bestanden, versiebeheer-changesets en gewone wikipagina's te koppelen. Het is tamelijk makkelijk in te stellen en integreert met Subversion (zie Appendix A, *Gratis versiebeheersystemen*).

Roundup — http://roundup.sourceforge.net/
Roundup is redelijk makkelijk te installeren (alleen Python 2.1 of hoger is vereist) en eenvoudig te gebruiken. het beschikt over web-, e-mail- en command line-interfaces. De issuegegevenssjablonen en webinterfaces kunnen gecustomiseerd worden, net als een gedeelte van de *state-transition logic*.

Mantis — http://www.mantisbt.org/
Mantis is een webgebaseerd bug tracking-systeem, geschreven in PHP. Het gebruikt MySQL-database voor opslag. Het heeft de functies die je zou verwachten. Ikzelf vind de webinterface duidelijk, intuïtief en prettig aan de ogen.

Flyspray — http://www.flyspray.org/
Flyspray is een webgebaseerd bug tracking-systeem, geschreven in PHP. De makers beschrijven het programma op de website als ongecompliceerd. De functies zijn: meervoudige databaseondersteuning (op dit moment MySQL en PGSQL); meervoudige projecten; 'in de gaten houden' van taken, met melding van veranderingen (via e-mail of Jabber); uitgebreide takengeschiedenis; CSS theming; bestandsattachments; geavanceerde zoekfuncties (niettemin eenvoudig te gebruiken); RSS/Atom feeds; wiki- en plaintext-invoer; stemmen; afhankelijkheidsgrafieken.

Scarab — http://scarab.tigris.org/
Scarab is ontworpen als een zeer customiseerbare, uitgebreide bug tracker, die min of meer alle functies biedt die de andere bug trackers te bieden hebben: gegevensinvoer, query's, rapporten, meldingen naar geïnteresseerde partijen, gezamenlijke cumulatie van opmerkingen en afhankelijkheids-tracking.

Het programma is customiseerbaar via administratieve webpagina's. In een enkele Scarab-installatie kun je diverse 'modules' (projecten) actief hebben. Binnen een gegeven module kun je nieuwe issuetypes maken (defecten, uitbreidingen, taken, verzoeken om support enz.) en willekeurige attributen toevoegen om de tracker af te stemmen op de specifieke behoeften van uw project.
Eind 2004 was Scarab dichtbij zijn release 1.0.

Debian Bug Tracking System (DBTS) — http://www.chiark.greenend.org.uk/~ian/debbugs/
Het Debian Bug Tracking System is in die zin ongewoon dat alle invoer en manipulatie van issues gedaan wordt via e-mail. Elke issue krijgt zijn eigen e-mailadres. De DBTS is behoorlijk goed schaalbaar: http://bugs.debian.org/ heeft bijvoorbeeld maar liefst 277.741 issues.

Omdat de interactie plaatsvindt via gewone mail-clients – een omgeving die voor de meeste mensen bekend en makkelijk toegankelijk is – is de DBTS goed voor het verwerken van grote hoeveelheden binnenkomende rapporten die snel geclassificeerd en beantwoord moeten worden. Natuurlijk zijn er ook nadelen. Ontwikkelaars moeten tijd investeren om het e-mailopdrachtsysteem te leren kennen en gebruikers moeten hun bugrapporten schrijven zonder een webformulier dat ze helpt bij het selecteren van de gegevens. Er zijn tools beschikbaar die gebruikers helpen om betere bugrapporten te verzenden, zoals het op command-line gebaseerde programma **reportbug** of het pakket `debbugs-el` voor Emacs. Maar de meeste mensen zullen deze tools niet gebruiken. Ze schrijven hun e-mails gewoon met de hand en volgen daarbij al dan niet de richtlijnen die uw project hanteert ten aanzien van het melden van bugs.

De DBTS heeft een read-only webinterface, voor het bekijken van en zoeken naar issues.

Trouble-Ticket Trackers
Deze zijn meer gericht op helpdesk ticket tracking dan software bug tracking. U hebt waarschijnlijk meer aan een gewone bug tracker. Ik vermeld ze voor de volledigheid en omdat er mogelijk een ongewoon project zou kunnen zijn waarvoor een trouble-ticketsysteem geschikter is dan een gewone bug tracker.

WebCall — http://myrapid.com/webcall/
Teacup — http://www.altara.org/teacup.html
(Ik geloof niet dat Teacup nog actief ontwikkeld wordt, maar de downloads zijn nog steeds beschikbaar. Het heeft zowel een web- als een e-mailinterface.)

Bluetail Ticket Tracker (BTT) — http://btt.sourceforge.net/
BTT houdt een beetje het midden tussen een standaard-trouble-ticket tracker en een bug tracker. Het beschikt over privacyfuncties die voor een bug trackeer nogal ongewoon zijn. Gebruikers van het systeem worden namelijk gecategoriseerd als Medewerker, Vriend, Klant of Anoniem en al naar gelang de categorie waartoe je behoort, heb je toegang tot meer of minder gegevens. Het biedt enige e-mailintegratie, een command line-interface en mechanismes om e-mails in tickets om te zetten. Het heeft ook functies voor het bijhouden van gegevens niet met een bepaalde ticket verband houden, zoals interne documentatie en FAQ's.

C WAAROM ZOU IK ME DRUK MAKEN OVER DE KLEUR VAN HET FIETSENHOK?

Dat hoeft u ook niet. Het doet er ook niet toe; bovendien hebt u wel wat beters te doen.

De beroemde 'fietsenhok'-post van Poul-Henning Kamp, waarvan een uittreksel in Hoofdstuk 6, *Communicatie* staat, is een welbespraakte uiteenzetting over wat er verkeerd kan gaan in groepsdiscussies. Ik druk het stuk hier met zijn toestemming nog een keer af, vertaald en wel. De oorspronkelijk URL is http://www.freebsd.org/cgi/getmsg.cgi?fetch=506636+517178+/usr/local/www/db/text/1999/freebsd-hackers/19991003.freebsd-hackers.

```
Onderwerp: Een fietsenhok (de kleur maakt niet uit) op groener
           gras …
Van: Poul-Henning Kamp <phk@freebsd.org>
Datum: Zat 2 okt 1999 16:14:10 +0200
Message-ID: <18238.938873650@critter.freebsd.dk>
Afzender: phk@critter.freebsd.dk
Bcc: Blind Distribution List: ;
MIME-Version: 1.0

[bcc's naar committers, hackers]

Mijn laatste pamflet is goed genoeg ontvangen om mij er niet van
te laten weerhouden er nog een in te sturen. Vandaag heb ik de
tijd en de zin om dat te doen.

Ik had wat problemen om te beslissen over de juiste ontvangers
voor dit soort tekst; deze keer bcc ik 'm naar committers en
hackers. Iets beters kan ik geloof ik niet verzinnen.  Ik sub-
scribe zelf niet op hackers, maar daarover later meer.

Wat me ertoe heeft gezet deze keer in de pen te klimmen, is de
thread "sleep(1) should do fractional seconds", die onze levens
nu al dagenlang verziekt. Het kan ook wekenlang zijn, maar ik
```

heb gewoon geen zin om dat te checken.

Voor wie niet bekend is met deze thread: Gefeliciteerd.

Het was het voorstel om van sleep (1) DTRT, indien gegeven, een niet-integer argument te maken dat deze prairiebrand deed ontstaan. Meer zeg ik er niet over, omdat het een veel kleiner probleem is dan je zou denken op grond van de lengte van de thread. En het onderwerp heeft al veel meer aandacht gekregen dan enkele van de *problemen* die we hier hebben.

De sleep(1)-kroniek is het meest flagrante staaltje van een fietsenhokdiscussie dat we ooit hebben gehad binnen FreeBSD. Het voorstel was doordacht. We zouden compatibel worden met OpenBSD en NetBSD, en nog steeds compatibel blijven met alle code die iemand ooit had geschreven.

Toch kwamen er zoveel bezwaren, tegenvoorstellen en veranderingen, dat je zou denken dat de voorgestelde verandering net zo megabelangrijk was als het vullen van alle gaten in alle Zwitserse kazen of het veranderen van de smaak van Coca Cola.

“Hoezo fietsenhok?” vroegen sommigen me.

Het is een lang verhaal, of eigenlijk een heel oud verhaal, en eigenlijk toch een kort verhaal. C. Northcote Parkinson schreef in het begin van de jaren 60 van de vorige eeuw een boek met de titel “De wet van Parkinson”, met veel inzichten in de dynamiek van management.

Je kunt het vinden op Amazon en misschien in je vaders boekenkast. Het boek is z’n geld en het lezen meer dan waard. Als je van Dilbert houdt, hou je van Parkinson.

Iemand vertelde me onlangs dat hij het had gelezen en vond dat maar zo’n 50% ervan vandaag de dag nog van toepassing was. Dat is bepaald niet slecht, zou ik zeggen. Veel van de hedendaagse managementboeken scoren veel lager dan dat en dit boek is al 35 jaar oud.

In het specifieke voorbeeld van het fietsenhok is de andere essentiële component een kerncentrale. Dat plaatst het boek, denk ik, mooi in zijn tijd.

Parkinson laat zien hoe u in de Raad van Bestuur goedkeuring kunt krijgen voor de bouw van een kerncentrale van miljoenen of zelfs miljarden dollars, maar dat u in eindeloze discussies verzeild raakt als u een fietsenhok wilt neerzetten.

Parkinson legt uit dat dit komt doordat een kerncentrale zo enorm groot, duur en gecompliceerd is dat mensen er met hun hoofd niet bij kunnen. In plaats van te proberen het toch te begrijpen, vertrouwen ze op de aanname dat iemand anders de informatie in een vorig stadium wel gecontroleerd zal hebben. Richard P. Feynmann geeft in zijn boeken een paar interessante en zeer treffende voorbeelden met betrekking tot Los Alamos.

Aan de andere kant hebben we het fietsenhok. Iedereen kan in een weekend een fietsenhok bouwen en dan nog genoeg tijd over houden om op zondag naar Studio Sport te kijken. Het maakt niet uit hoe goed voorbereid u bent en hoe redelijk uw voorstel is, iemand zal altijd de kans grijpen om te laten zien dat hij zijn werk goed doet, dat hij goed oplet, dat hij er is.

We noemen dit “je geur achterlaten”. Het heeft te maken met persoonlijke trots en status; je moet iets aan kunnen wijzen en zeggen “Kijk! Dat heb *IK* gedaan.” Politici bijvoorbeeld hebben sterk die neiging, maar iedereen loopt kans ermee besmet te raken. Denk maar aan het achterlaten van voetstappen in nat cement.

Ik neem mijn hoed af voor de indiener van het oorspronkelijke voorstel omdat hij voet bij stuk hield terwijl hij vanuit de nekloge met pinda’s bekogeld werd. Het voorstel maakt tegenwoordig deel uit van onze tree. Ik zou na een paar van die berichtjes in de thread de handdoek in de ring hebben gegooid.

En daarmee kom ik, zoals ik eerder beloofde, bij de reden waarom ik niet subscribe op -hackers. Ik heb geünsubscribed van -hackers omdat ik de hoeveelheid mailtjes namelijk niet kon bijbenen. Sindsdien heb ik om dezelfde reden verschillende andere lijsten ook vaarwel gezegd.

En nog steeds krijg ik veel e-mail. Veel daarvoor wordt door filters naar /dev/null gerouteerd: Mensen als [naam doorgekrast] bereiken mijn scherm nooit, commits naar documenten in talen die ik niet begrijp evenmin, net zo min als commits naar ports. Al deze zaken gaan de prullenmand in zonder dat ik er ooit weet van heb.

Maar ondanks die grommende tanden onder mijn postbus krijg ik nog steeds te veel mail.

Hier verschijnt het groenere gras in beeld. Ik wou dat we de hoeveelheid ruis in onze lijsten konden verminderen en ik wou dat we mensen regelmatig een fietsenhok konden laten bouwen en

het kan me eigenlijk niet schelen welke kleur ze dat schilderen.

De eerste wens gaat over beleefd, smaakvol en intelligent gebruik van onze mail.

Als ik bondige en nauwkeurige criteria zou kunnen formuleren voor wanneer wel en wanneer niet te reageren op een e-mail, waaraan iedereen zich zou willen houden, zou ik een gelukkig mens zijn. Maar ik ben verstandig genoeg om dat niet eens te proberen.

Maar laat ik niettemin een paar popup-vensters voorstellen die ik graag zou terugzien in e-mailprogramma's. Ze zouden telkens moeten verschijnen als je e-mail verstuurt naar of op e-mail antwoord van de lijsten waarop ze me graag willen laten subscriben:

```
+-----------------------------------------------------------+
| Uw e-mail staat op het punt naar honderdduizenden mensen te
worden verstuurd | allemaal mensen, die ten minste 10 seconden
kwijt zijn aan het lezen van die e-mail | voordat ze kunnen be-
slissen of die interessant is. Ten minste | twee mensweken gaan
besteed worden aan het lezen van uw e-mail. Veel van de geadres-
seerden moeten betalen om uw e-mail te kunnen downloaden.| Bent
u er absoluut zeker van dat uw e-mail belangrijk genoeg is om al
die mensen lastig te vallen? |

|                [JA]  [WIJZIGEN]  [ANNULEREN]                |
+-----------------------------------------------------------+
+-----------------------------------------------------------+
| Waarschuwing:  U hebt nog niet alle e-mails in deze thread
gelezen. | Iemand anders heeft wellicht al gezegd wat u nu wilt
gaan zeggen in uw antwoord. Gelieve de gehele thread te lezen
voordat u antwoordt op een e-mail in die thread. |

|                        [WIJZIGEN]                         |
+-----------------------------------------------------------+
+-----------------------------------------------------------+
| Waarschuwing:  Uw e-mailprogramma heeft nog niet eens het hele
bericht laten zien. Hieruit volgt logischerwijs dat u onmoge-
lijk alles gelezen en begrepen kunt hebben. Het is onbeleefd op
een e-mail te reageren voordat u deze helemaal gelezen en erover
nagedacht hebt.|
| De afkoeltimer voor deze thread verhindert dat u het komende
uur op een e-mail in deze thread kunt antwoorden.
|                        [Annuleren]                        |
+-----------------------------------------------------------+
+-----------------------------------------------------------+
| U hebt deze e-mail getypt met een snelheid van meer dan N.NN
tps | Over het algemeen is het niet mogelijk om te denken en te
typen met een snelheid van meer dan A.AA tps. Om die reden is
het waarschijnlijk dat uw antwoord onsamenhangend, slecht bear-
gumenteerd en/of emotioneel is.
| Een afkoeltimer voorkomt dat u het komende uur e-mails kunt
versturen.
|                        [Annuleren]                        |
+-----------------------------------------------------------+
+-----------------------------------------------------------+
```

Het tweede deel van mijn wens is meer emotioneel. Overduidelijk waren de deskundigen die het vuurtje in de sleep(1)-thread brandende hielden, ondanks hun jarenlange ervaring bij het project, niet bereid om deze kleine daad te verrichten. Waarom waren ze dan plotseling zo verontwaardigd dat iemand die zo veel hun junior was dat deed? Ik wou dat ik het wist.

Ik weet dat de reden onmachtig is om dergelijk "reactionair conservatisme" te stoppen. Het kan zijn dat deze mensen gefrustreerd zijn over hun eigen magere bijdrage recentelijk of misschien is het een kwestie van "Wij zijn oud en chagrijnig en WIJ weten hoe de jeugd zich te gedragen heeft."

Hoe dan ook, voor het project is het zeer contraproductief. Ik heb echter geen suggesties om het te laten ophouden. Het beste advies dat ik kan geven, is om de monsters die in de mailinglijsten op de loer liggen niet van brandstof te voorzien. Negeer ze, antwoord er niet op, vergeet dat ze bestaan.

Ik hoop dat we een sterkere en bredere basis van contribuanten aan FreeBSD krijgen en ik hoop dat we samen de brommerige oude mannen en [weggelaten]s van deze wereld ervan kunnen weerhouden om ze op te rakelen, door ze uit te spugen en ze weg te jagen voordat ze een voet aan de grond krijgen.

Tegen die mensen die zich daarbuiten ergens verborgen houden, door de draken afgeschrikt om deel te nemen: Ik kan alleen maar mijn excuses aanbieden en u aansporen om het toch te proberen. Dit is niet de omgeving die ik voor het project wil.

Poul-Henning

D VOORBEELDINSTRUCTIES VOOR HET RAPPORTEREN VAN BUGS

Dit is een enigszins bewerkte versie van de online-instructies van het Subversion-project voor gebruikers over hoe bugs te rapporteren. Raadpleeg om te zien waarom het belangrijk is dat een project dergelijke instructies heeft het gedeelte "Behandel iedere gebruiker als een mogelijke vrijwilliger" in Hoofdstuk 8, *Het managen van vrijwilligers*. Het oorspronkelijke document kunt u vinden op http://svn.collab.net/repos/svn/trunk/www/bugs.html.

```
Bugs rapporteren in Subversion. Dit document legt uit hoe en
waar u bugs moet rapporteren. (Het is geen lijst met alle uit-
staande bugs. Die kunt u hier in de plaats krijgen.)

WAAR EEN BUG TE RAPPORTEREN ----------------------------------
* Als de bug in Subversion zelf zit, stuur dan een mail naar
users@subversion.tigris.org. Als deze eenmaal is bevestigd als
een bug, kan iemand, mogelijk uzelf, hem in de issue tracker in-
voeren. (Of als u behoorlijk zeker van de bug bent, post dan di-
rect op onze ontwikkelingslijst, dev@subversion.tigris.org. Maar
als u er niet zeker van bent, is het beter om naar users@ first
te posten; iemand daar kan u vertellen of het gedrag dat u bent
tegengekomen normaal is of niet.)

* Als de bug al in de APR library zit, rapporteer 'm dan naar
deze beide mailinglijsten: dev@apr.apache.org, dev@subversion.
tigris.org.

* Als de bug al in de Neon HTTP library zit, rapporteer 'm dan
naar: neon@webdav.org, dev@subversion.tigris.org.

* Als de bug al in de Apache HTTPD 2.0 zit, rapporteer 'm dan
naar deze beide mailinglijsten: dev@httpd.apache.org, dev@sub-
version.tigris.org. De Apache httpd-ontwikkelaarsmailinglijst is
zeer druk, dus het is mogelijk dat uw bugrapport over het hoofd
wordt gezien. U kunt een bug ook rapporteren op http://httpd.
apache.org/bug_report.html.

* Als 't een bug is op uw rug, pak 'm dan vlug en behandel 'm
als een mug.

HOE EEN BUG TE RAPPORTEREN ----------------------------------
Ten eerste, zorg dat u zeker weet dat het een bug is. Als Sub-
version zich niet op de verwachte manier gedraagt, zoek dan in
de documentatie en mailinglijstarchieven naar bewijs dat het
programma zich inderdaad zo zou moeten gedragen als u had ver-
wacht. Het gaat natuurlijk om gezond verstand. Als Subversion
net al uw gegevens vernietigd heeft en er rook uit uw beeld-
scherm komt, dan kunt u gerust afgaan op uw oordeel. Maar als
u twijfelt, vraag dan eerst de gebruikers op de mailinglijst
users@subversion.tigris.org, of vraag in IRC, irc.freenode.net,
channel #svn.

Als u hebt vastgesteld hebt dat het inderdaad een bug is, is het
belangrijkste dat u kunt doen een simpele beschrijving en repro-
ductierecept opstellen.  Als de bug zoals u die in eerste in-
stantie hebt aangetroffen bijvoorbeeld vijf bestanden verspreid
over tien commits betreft, probeer 'm dan te reproduceren met
één bestand en één commit. Hoe eenvoudiger het reproductiere-
cept, des te waarschijnlijker het is dat een ontwikkelaar de bug
kan reproduceren en herstellen.

Bij het schrijven van het reproductierecept moet u geen prozaï-
sche beschrijving geven van wat u deed dat deze bug deed optre-
den. Geef in plaats daarvan een letterlijke transcriptie van de
exacte serie opdrachten die u gaf en hun resultaat. Doe dit met
knippen-en-plakken. Als er bestanden bij betrokken zijn, geef
dan ook de namen van die bestanden en ook de inhoud, als u denkt
dat dat belangrijk is. Het beste is om uw reproductierecept te
verpakken als een script; dat is enorm handig.

Even de zit-de-stekker-wel-in-het-stopcontact-vraag: u werkt
toch met de laatste versie van Subversion? :-) Misschien is de
bug al gefixt. Test uw reproductierecept uit op de meest recente
ontwikkelings-tree van Subversion.

Naast het reproductierecept hebben we ook een volledige be-
schrijving nodig van de omgeving waarin u de bug hebt gereprodu-
ceerd.  Dat betekent:
* Uw bedieningssysteem
* De release en/of revisie van Subversion
* De compiler- en configuratieopties waarmee u Subversion hebt
ge-build
```

```
* Alle privémodificaties die u aan uw Subversion hebt aangebracht
* De versie van Berkeley DB waarmee u Subversion runt, indien
van toepassing
* Al het andere dat relevant zou kunnen zijn. Neig naar te veel
informatie, in plaats van naar te weinig.

Als u dit allemaal hebt, bent u klaar om uw melding te schrij-
ven. Begin met een heldere beschrijving van de bug. Dat wil zeg-
gen: vertel hoe u verwachtte dat Subversion zich zou gedragen
en vervolgens hoe Subversion zich werkelijk gedroeg. De bug kan
voor u zeer helder zijn, voor een ander is dat wellicht niet het
geval; dus zorg dat het geen partijtje raden wordt. Vervolg met
de omgevingsbeschrijving en het reproductierecept. Als u ook een
speculatie wilt toevoegen over de oorzaak en zelfs een patch:
des te beter. Zie http://svn.collab.net/repos/svn/trunk/www/
hacking.html#patches voor aanwijzingen over het versturen van
patches.

Post al deze informatie naar dev@subversion.tigris.org, of, als
u daar al bent geweest en u gevraagd is om een issue te melden,
ga naar de Issue Tracker en volg de aanwijzingen aldaar op.

Bedankt. We weten dat het veel werk is om een effectief bugrap-
port in te dienen, maar een goed rapport kan uren kostbare tijd
van een ontwikkelaar schelen en het veel waarschijnlijk maken
dat de bug wordt gefixt.
```


E AUTEURSRECHT

LICENSE

THE WORK (AS DEFINED BELOW) IS PROVIDED UNDER THE TERMS OF THIS CREATIVE COMMONS PUBLIC LICENSE ("CCPL" OR "LICENSE"). THE WORK IS PROTECTED BY COPYRIGHT AND/OR OTHER APPLICABLE LAW. ANY USE OF THE WORK OTHER THAN AS AUTHORIZED UNDER THIS LICENSE OR COPYRIGHT LAW IS PROHIBITED.

BY EXERCISING ANY RIGHTS TO THE WORK PROVIDED HERE, YOU ACCEPT AND AGREE TO BE BOUND BY THE TERMS OF THIS LICENSE. TO THE EXTENT THIS LICENSE MAY BE CONSIDERED TO BE A CONTRACT, THE LICENSOR GRANTS YOU THE RIGHTS CONTAINED HERE IN CONSIDERATION OF YOUR ACCEPTANCE OF SUCH

TERMS AND CONDITIONS

1. Definitions

a. "Adaptation" means a work based upon the Work, or upon the Work and other pre-existing works, such as a translation, adaptation, derivative work, arrangement of music or other alterations of a literary or artistic work, or phonogram or performance and includes cinematographic adaptations or any other form in which the Work may be recast, transformed, or adapted including in any form recognizably derived from the original, except that a work that constitutes a Collection will not be considered an Adaptation for the purpose of this License. For the avoidance of doubt, where the Work is a musical work, performance or phonogram, the synchronization of the Work in timed-relation with a moving image ("synching") will be considered an Adaptation for the purpose of this License.

b. "Collection" means a collection of literary or artistic works, such as encyclopedias and anthologies, or performances, phonograms or broadcasts, or other works or subject matter other than works listed in Section 1(f) below, which, by reason of the selection and arrangement of their contents, constitute intellectual creations, in which the Work is included in its entirety in unmodified form along with one or more other contributions, each constituting separate and independent works in themselves, which together are assembled into a collective whole. A work that constitutes a Collection will not be considered an Adaptation (as defined below) for the purposes of this License.

c. "Creative Commons Compatible License" means a license that is listed at http://creativecommons.org/compatiblelicenses that has been approved by Creative Commons as being essentially equivalent to this License, including, at a minimum, because that license: (i) contains terms that have the same purpose, meaning and effect as the License Elements of this License; and, (ii) explicitly permits the relicensing of adaptations of works made available under that license under this License or a Creative Commons jurisdiction license with the same License Elements as this License.

d. "Distribute" means to make available to the public the original and copies of the Work or Adaptation, as appropriate, through sale or other transfer of ownership.

e. "License Elements" means the following high-level license attributes as selected by Licensor and indicated in the title of this License: Attribution, ShareAlike.

f. "Licensor" means the individual, individuals, entity or entities that offer(s) the Work under the terms of this License.

g. "Original Author" means, in the case of a literary or artistic work, the individual, individuals, entity or entities who created the Work or if no individual or entity can be identified, the publisher; and in addition (i) in the case of a performance the actors, singers, musicians, dancers, and other persons who act, sing, deliver, declaim, play in, interpret or otherwise perform literary or artistic works or expressions of folklore; (ii) in the case of a phonogram the producer being the person or legal entity who first fixes the sounds of a performance or other sounds; and, (iii) in the case of broadcasts, the organization that transmits the broadcast.

h. "Work" means the literary and/or artistic work offered under the terms of this License including without limitation any production in the literary, scientific and artistic domain, nwhatever may be the mode or form of its expression including digital form, such as a book, pamphlet and other writing; a lecture, address, sermon or other work of the same nature; a dramatic or dramatico-musical work; a choreographic work or entertainment in dumb show; a musical composition with or without words; a cinematographic work to which are assimilated works expressed by a process analogous to cinematography; a work of drawing, painting, architecture, sculpture, engraving or lithography; a photographic work to which are assimilated works expressed by a process analogous to photography; a work of applied art; an illustration, map, plan, sketch or three-dimensional work relative to geography, topography, architecture or science; a performance; a broadcast; a phonogram; a compilation of data to the extent it is protected as a copyrightable work; or a work performed by a variety or circus performer to the extent it is not otherwise considered a literary or artistic work.

i. "You" means an individual or entity exercising rights under this License who has not previously violated the terms of this License with respect to the Work, or who has received express permission from the Licensor to exercise rights under this License despite a previous violation.

j. "Publicly Perform" means to perform public recitations of the Work and to communicate to the public those public recitations, by any means or process, including by wire or wireless means or public digital performances; to make available to the public Works in such a way that members of the public may access these Works from a place and at a place individually chosen by them; to perform the Work to the public by any means or process and the communication to the public of the performances of the Work, including by public digital

performance; to broadcast and rebroadcast the Work by any means including signs, sounds or images.

k. "Reproduce" means to make copies of the Work by any means including without limitation by sound or visual recordings and the right of fixation and reproducing fixations of the Work, including storage of a protected performance or phonogram in digital form or other electronic medium.

2. Fair Dealing Rights.
Nothing in this License is intended to reduce, limit, or restrict any uses free from copyright or rights arising from limitations or exceptions that are provided for in connection with the copyright protection under copyright law or other applicable laws.

3. License Grant.
Subject to the terms and conditions of this License, Licensor hereby grants You a worldwide, royalty-free, non-exclusive, perpetual (for the duration of the applicable copyright) license to exercise the rights in the Work as stated below:

a. to Reproduce the Work, to incorporate the Work into one or more Collections, and to Reproduce the Work as incorporated in the Collections;

b. to create and Reproduce Adaptations provided that any such Adaptation, including any translation in any medium, takes reasonable steps to clearly label, demarcate or otherwise identify that changes were made to the original Work. For example, a translation could be marked "The original work was translated from English to Spanish," or a modification could indicate "The original work has been modified.";

c. to Distribute and Publicly Perform the Work including as incorporated in Collections; and,

d. to Distribute and Publicly Perform Adaptations.

e. For the avoidance of doubt:
   I. Non-waivable Compulsory License Schemes. In those jurisdictions in which the right to collect royalties through any statutory or compulsory licensing scheme cannot be waived, the Licensor reserves the exclusive right to collect such royalties for any exercise by You of the rights granted under this License;
   II. Waivable Compulsory License Schemes. In those jurisdictions in which the right to collect royalties through any statutory or compulsory licensing scheme can be waived, the Licensor waives the exclusive right to collect such royalties for any exercise by You of the rights granted under this License; and,
   III. Voluntary License Schemes. The Licensor waives the right to collect royalties, whether individually or, in the event that the Licensor is a member of a collecting society that administers voluntary licensing schemes, via that society, from any exercise by You of the rights granted under this License.

The above rights may be exercised in all media and formats whether now known or hereafter devised. The above rights include the right to make such modifications as are technically necessary to exercise the rights in other media and formats. Subject to Section 8(f), all rights not expressly granted by Licensor are hereby reserved.

4. Restrictions.
The license granted in Section 3 above is expressly made subject to and limited by the following restrictions:

a. You may Distribute or Publicly Perform the Work only under the terms of this License. You must include a copy of, or the Uniform Resource Identifier (URI) for, this License with every copy of the Work You Distribute or Publicly Perform. You may not offer or impose any terms on the Work that restrict the terms of this License or the ability of the recipient of the Work to exercise the rights granted to that recipient under the terms of the License. You may not sublicense the Work. You must keep intact all notices that refer to this License and to the disclaimer of warranties with every copy of the Work You Distribute or Publicly Perform. When You Distribute or Publicly Perform the Work, You may not impose any effective technological measures on the Work that restrict the ability of a recipient of the Work from You to exercise the rights granted to that recipient under the terms of the License. This Section 4(a) applies to the Work as incorporated in a Collection, but this does not require the Collection apart from the Work itself to be made subject to the terms of this License. If You create a Collection, upon notice from any Licensor You must, to the extent practicable, remove from the Collection any credit as required by Section 4(c), as requested. If You create an Adaptation, upon notice from any Licensor You must, to the extent practicable, remove from the Adaptation any credit as required by Section 4(c), as requested.

b. You may Distribute or Publicly Perform an Adaptation only under the terms of: (i) this License; (ii) a later version of this License with the same License Elements as this License; (iii) a Creative Commons jurisdiction license (either this or a later license version) that contains the same License Elements as this License (e.g., Attribution-ShareAlike 3.0 US)); (iv) a Creative Commons Compatible License. If you license the Adaptation under one of the licenses mentioned in (iv), you must comply with the terms of that license. If you license the Adaptation under the terms of any of the licenses mentioned in (i), (ii) or (iii) (the "Applicable License"), you must comply with the terms of the Applicable License generally and the following provisions:(I) You must include a copy of, or the URI for, the Applicable License with every copy of each Adaptation You Distribute or Publicly Perform; (II) You may not offer or impose any terms on the Adaptation that restrict the terms of the Applicable License or the ability of the recipient of the Adaptation to exercise the rights granted to that recipient under the terms of the Applicable License; (III) You must keep intact all notices that refer to the Applicable License and to the disclaimer of warranties with every copy of the Work as included in the Adaptation You Distribute or Publicly Perform; (IV) when You Distribute or Publicly Perform the Adaptation, You may not impose

any effective technological measures on the Adaptation that restrict the ability of a recipient of the Adaptation from You to exercise the rights granted to that recipient under the terms of the Applicable License. This Section 4(b) applies to the Adaptation as incorporated in a Collection, but this does not require the Collection apart from the Adaptation itself to be made subject to the terms of the Applicable License.

c. If You Distribute, or Publicly Perform the Work or any Adaptations or Collections, You must, unless a request has been made pursuant to Section 4(a), keep intact all copyright notices for the Work and provide, reasonable to the medium or means You are utilizing: (i) the name of the Original Author (or pseudonym, if applicable) if supplied, and/or if the Original Author and/or Licensor designate another party or parties (e.g., a sponsor institute, publishing entity, journal) for attribution ("Attribution Parties") in Licensor's copyright notice, terms of service or by other reasonable means, the name of such party or parties; (ii) the title of the Work if supplied; (iii) to the extent reasonably practicable, the URI, if any, that Licensor specifies to be associated with the Work, unless such URI does not refer to the copyright notice or licensing information for the Work; and (iv) , consistent with Ssection 3(b), in the case of an Adaptation, a credit identifying the use of the Work in the Adaptation (e.g., "French translation of the Work by Original Author," or "Screenplay based on original Work by Original Author"). The credit required by this Section 4(c) may be implemented in any reasonable manner; provided, however, that in the case of a Adaptation or Collection, at a minimum such credit will appear, if a credit for all contributing authors of the Adaptation or Collection appears, then as part of these credits and in a manner at least as prominent as the credits for the other contributing authors. For the avoidance of doubt, You may only use the credit required by this Section for the purpose of attribution in the manner set out above and, by exercising Your rights under this License, You may not implicitly or explicitly assert or imply any connection with, sponsorship or endorsement by the Original Author, Licensor and/or Attribution Parties, as appropriate, of You or Your use of the Work, without the separate, express prior written permission of the Original Author, Licensor and/or Attribution Parties.

d. Except as otherwise agreed in writing by the Licensor or as may be otherwise permitted by applicable law, if You Reproduce, Distribute or Publicly Perform the Work either by itself or as part of any Adaptations or Collections, You must not distort, mutilate, modify or take other derogatory action in relation to the Work which would be prejudicial to the Original Author's honor or reputation. Licensor agrees that in those jurisdictions (e.g. Japan), in which any exercise of the right granted in Section 3(b) of this License (the right to make Adaptations) would be deemed to be a distortion, mutilation, modification or other derogatory action prejudicial to the Original Author's honor and reputation, the Licensor will waive or not assert, as appropriate, this Section, to the fullest extent permitted by the applicable national law, to enable You to reasonably exercise Your right under Section 3(b) of this License (right to make Adaptations) but not otherwise.

5. Representations, Warranties and Disclaimer

UNLESS OTHERWISE MUTUALLY AGREED TO BY THE PARTIES IN WRITING, LICENSOR OFFERS THE WORK AS-IS AND MAKES NO REPRESENTATIONS ORWARRANTIES OF ANY KIND CONCERNING THE WORK, EXPRESS, IMPLIED, STATUTORY OR OTHERWISE, INCLUDING, WITHOUT LIMITATION, WARRANTIES OF TITLE, MERCHANTIBILITY, FITNESS FOR A PARTICULAR PURPOSE, NONINFRINGEMENT, OR THE ABSENCE OF LATENT OR OTHER DEFECTS, ACCURACY, OR THE PRESENCE OF ABSENCE OF ERRORS, WHETHER OR NOT DISCOVERABLE. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO SUCH EXCLUSION MAY NOT APPLY TO YOU.

6. Limitation on Liability.

EXCEPT TO THE EXTENT REQUIRED BY APPLICABLE LAW, IN NO EVENT WILL LICENSOR BE LIABLE TO YOU ON ANY LEGAL THEORY FOR ANY SPECIAL, INCIDENTAL, CONSEQUENTIAL, PUNITIVE OR EXEMPLARY DAMAGES ARISING OUT OF THIS LICENSE OR THE USE OF THE WORK, EVEN IF LICENSOR HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

7. Termination

a. This License and the rights granted hereunder will terminate automatically upon any breach by You of the terms of this License. Individuals or entities who have received Adaptations or Collections from You under this License, however, will not have their licenses terminated provided such individuals or entities remain in full compliance with those licenses. Sections 1, 2, 5, 6, 7, and 8 will survive any termination of this License.

b. Subject to the above terms and conditions, the license granted here is perpetual (for the duration of the applicable copyright in the Work). Notwithstanding the above, Licensor reserves the right to release the Work under different license terms or to stop distributing the Work at any time; provided, however that any such election will not serve to withdraw this License (or any other license that has been, or is required to be, granted under the terms of this License), and this License will continue in full force and effect unless terminated as stated above.

8. Miscellaneous

a. Each time You Distribute or Publicly Perform the Work or a Collection, the Licensor offers to the recipient a license to the Work on the same terms and conditions as the license granted to You under this License.

b. Each time You Distribute or Publicly Perform an Adaptation, Licensor offers to the recipient a license to the original Work on the same terms and conditions as the license granted to You under this License.

c. If any provision of this License is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this License, and without further action by the parties to this agreement, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable.

d. No term or provision of this License shall be deemed waived and no breach consented to unless such waiver or consent shall be in writing and signed by the party to be charged with such waiver or consent.

e. This License constitutes the entire agreement between the parties with respect to the Work licensed here. There are no understandings, agreements or representations with respect to the Work not specified here. Licensor shall not be bound by any additional provisions that may appear in any communication from You. This License may not be modified without the mutual written agreement of the Licensor and You.

f. The rights granted under, and the subject matter referenced, in this License were drafted utilizing the terminology of the Berne Convention for the Protection of Literary and Artistic Works (as amended on September 28, 1979), the Rome Convention of 1961, the WIPO Copyright Treaty of 1996, the WIPO Performances and Phonograms Treaty of 1996 and the Universal Copyright Convention (as revised on July 24, 1971). These rights and subject matter take effect in the relevant jurisdiction in which the License terms are sought to be enforced according to the corresponding provisions of the implementation of those treaty provisions in the applicable national law. If the standard suite of rights granted under applicable copyright law includes additional rights not granted under this License, such additional rights are deemed to be included in the License; this License is not intended to restrict the license of any rights under applicable law.

CREATIVE COMMONS NOTICE

Creative Commons is not a party to this License, and makes no warranty whatsoever in connection with the Work. Creative Commons will not be liable to You or any party on any legal theory for any damages whatsoever, including without limitation any general, special, incidental or consequential damages arising in connection to this license. Notwithstanding the foregoing two (2) sentences, if Creative Commons has expressly identified itself as the Licensor hereunder, it shall have all rights and obligations of Licensor.

Except for the limited purpose of indicating to the public that the Work is licensed under the CCPL, Creative Commons does not authorize the use by either party of the trademark "Creative Commons" or any related trademark or logo of Creative Commons without the prior written consent of Creative Commons. Any permitted use will be in compliance with Creative Commons' then-current trademark usage guidelines, as may be published on its website or otherwise made available upon request from time to time. For the avoidance of doubt, this trademark restriction does not form part of the License.

Creative Commons may be contacted at http://creativecommons.org/.

OPEN SOURCE-SOFTWARE PRODUCEREN

Met succes een open source-softwareproject runnen

Karl Fogel

Open source-software produceren is een boek waarin alle aspecten van het ontwikkelen van open source-software aan bod komen. Daarbij ligt de nadruk op de menselijke kant van ontwikkeltrajecten. In dit boek wordt beschreven hoe succesvolle projecten in elkaar steken, hoe de verwachtingen van gebruikers en ontwikkelaars gestuurd kunnen worden en met welke cultuursaspecten rekening moet worden gehouden.

Deze vertaling is mogelijk gemaakt door SURFfoundation, SURFdiensten, SURFnet en Stichting Kennisnet.

SURF

SURFnet bv
Postbus 19035
3501 DA Utrecht
Telefoon 030 2 305 305
Fax 030 2 305 329
admin@surfnet.nl
www.surfnet.nl

Kennisnet

Stichting Kennisnet
Postbus 778
2700 AT Zoetermeer
Telefoon 0800 KENNISNET
Fax 079 3 212 322
surfnetkennisnet@kennisnet.org
kennisnet.nl