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/).