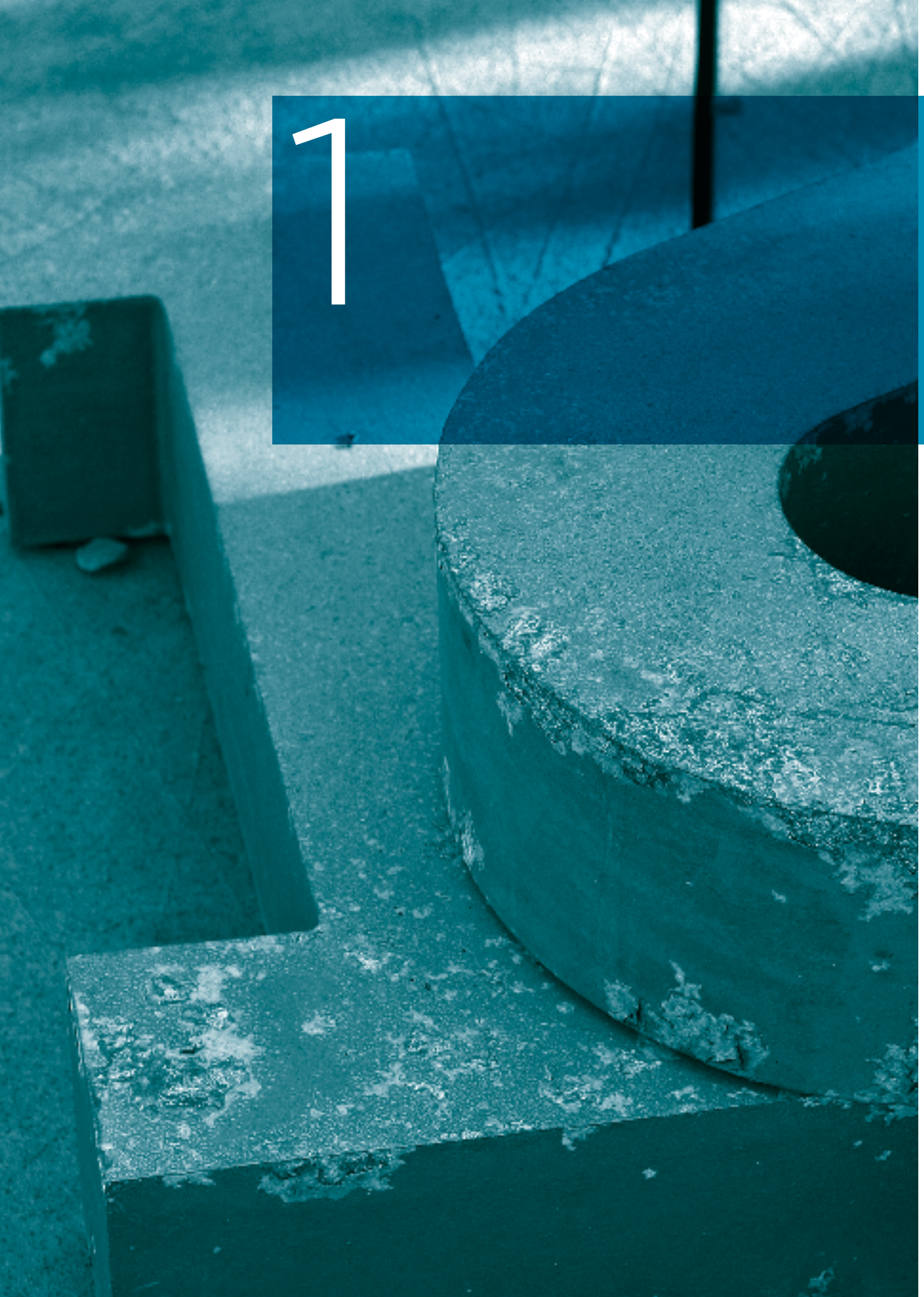

1

INLEIDING

De meeste open source-softwareprojecten mislukken.

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

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

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

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

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

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

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

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

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

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

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

1.1 GESCHIEDENIS

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

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

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

De opkomst van propriëtaire en vrije software

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

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

Bewuste weerstand

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

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

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

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

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

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

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

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

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

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

Onbewuste weerstand

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

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

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

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

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

'Vrij' versus 'Open source'

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

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

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

1.2 DE HUIDIGE STAND VAN ZAKEN

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

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

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

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

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

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

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

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

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

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

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

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