3

TECHNISCHE INFRASTRUCTUUR

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

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

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

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

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

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

3.1 PROJECTBENODIGDHEDEN

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

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

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

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

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

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

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

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

3.2 MAILINGLIJSTEN

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

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

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

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

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

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

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

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

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

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

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

Spampreventie

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

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

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

Filteren van posts

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

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

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

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

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

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

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

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

Adressen in archief verbergen

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

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

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

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

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

Management van identificatie en headers

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

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

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

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

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

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

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

De mail ziet echter in de lijst als volgt uit:

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

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

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

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

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

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

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

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

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

Het grote ‘reply-to’-debat

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

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

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

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


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

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

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

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

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

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

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

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

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

Twee fantasieën

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

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

Archivering

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

Onmiddellijke archivering

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

Referentiestabiliteit

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

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

Back-ups

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

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

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

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

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

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

Beheersoftware voor mailinglijsten:

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

Archiveringssoftware voor mailinglijsten:

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

3.3 VERSIEBEHEER

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

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

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

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

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

'Versie' versus 'revisie'

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

commit (committen)

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

logbericht (log message)

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

update

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

repository

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

checkout

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

working copy

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

revisie, *verandering*, *changeset*

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

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

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

diff

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

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

tag

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

branch (vertakking)

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

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

merge(n) (oftewel porten)

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

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

conflict

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

lock

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

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

Een versiebeheersysteem kiezen

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

CIA: Nog een manier om veranderingen te publiceren

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

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

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

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

Branches gebruiken om bottlenecks te voorkomen

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

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

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

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

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

Enkelvoudigheid van informatie

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

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

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

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

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

Autorisatie

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

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

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

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

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

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

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

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

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

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

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

3.4 BUG TRACKER

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

De noodzaak om tijdig te reageren heeft twee consequenties:

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

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

Interactie met mailinglijsten

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

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

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

De bug tracker voorfilteren

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

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

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

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

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

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

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

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

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

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

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

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

3.5 IRC- / REALTIME CHATSYSTEMEN

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

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

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

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

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

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

```
You are now talking on #svn

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

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

Paste-sites

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

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

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

Bots

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


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

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

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

Hetzelfde wordt bereikt met een handige verkorte versie:

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

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

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

IRC archiveren

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

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

3.6 RSS FEEDS

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

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

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

3.7 WIKI'S

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

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

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

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

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

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

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

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

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

3.8 WEBSITE

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

Canned Hosting

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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