Creando software libre

Como levar a cabo con éxito un proxecto de software libre

Dedicatoria

Este libro está dedicado a dous queridos amigos sen os que este libro non sería posible: Karen Underhill e Jim Blandy.

Índice de contidos

Capítulo 0. Preámbulo

Por que escribín este libro?

Nas festas, a xente xa non fica con expresión de dúbida cando lles digo que traballo con software libre. «Ah, si, código aberto, como Linux?» din, tras o que aceno con emoción: "si, iso mesmo!". É agradable deixar de ser un becho raro. Hai tempo, o que preguntarían a seguir era bastante previsible:

“e como gañas a vida con iso?”. Para responder a esta pregunta, facíalles un resumo dos aspectos económicos do código aberto; nomeadamente, que hai organizacións ás que lles interesa que exista un determinado software, mais que non precisan vender copias do mesmo, o que lles interesa é asegurarse de que ese software se mantén e de que está á súa disposición; ven nel unha ferramenta antes que un produto.

Ultimamente, porén, as preguntas que fan a seguir non son sempre de carácter pecuniario. O modelo económico do software libre perdeu boa parte do seu misterio, e moitas persoas alleas á programación xa comprenden (ou ao menos non se sorprenden ao escoitaren) que hai quen traballa con software libre a tempo enteiro. Antes ben, a pregunta que máis teño escoitado é a seguinte: “e como é que funciona iso?”.

Aínda non dei cunha resposta satisfactoria, e canto máis trato de atopar unha, máis me decato de que se trata dun tema moi complexo. Xerir un proxecto de software libre non é exactamente como xerir un negocio (imaxinade como sería ter que negociar constantemente a natureza do voso produto cunha turma de voluntarios, moitos dos cales non coñeceriades persoalmente), mais tampouco é exactamente como xerir unha organización non gobernamental clásica, nin tampouco un goberno. Ten analoxías con todo iso, mais, co tempo, cheguei á conclusión de que o software libre constitúe un xénero *per se*. Pode ser comparado con maior ou menor éxito con multitude de cousas, mais nunca en relación de igualdade. Mesmo a presunción de que os proxectos de software libre poden ser “xeridos” é unha esaxeración: un proxecto de software libre pode ser iniciado, e pode ser influenciado polas partes interesadas, frecuentemente de maneira considerable, mais o material non pode pasar a mans dun único propietario e, sempre que houber alguén (onde for) interesado en continuar co proxecto, este non pode ser cancelado de maneira unilateral. Todos os membros teñen poder ilimitado e todos son impotentes, o que crea unha dinámica interesante.

Eses son os motivos que me impulsaron a redactar este libro. Os proxectos de software libre tornáronse nunha cultura característica, un etos que ten como dogma central a liberdade de lograr que o software faga o que un quixer, sen que isto supoña a disgregación dos programadores; antes ben, que colaboren con máis forza. A capacidade de cooperación é, de feito, unha das habilidades máis cotizadas no ámbito do software libre. Xerir un proxecto desta índole implica mergullarse nunha especie de cooperación hipertrofiada, e a capacidade de descubrir novas vías de cooperación, para alén da capacidade de traballo en equipo, pode repercutir de maneira moi positiva sobre o produto. A finalidade deste libro é describir as técnicas que poden axudar a conseguilo. Non é unha obra exhaustiva, mais serve polo menos como punto de partida.

O software libre de calidade constitúe un obxectivo en si mesmo, e espero que esta obra satisfaga os lectores que se sirvan dela para o atinxiren. Para alén diso, espero tamén que participen do pracer que se obtén ao traballar nun equipo de desenvolvedores de código aberto motivados, e da relación directa co usuario que alenta o código aberto. Participar nun proxecto de software libre de éxito é divertido, e é, en definitiva, a causa de que o proceso non se deteña.

A quen está orientado este libro?

A presente obra está orientada a desenvolvedores de software e xerentes que teñen en mente iniciar un proxecto de software libre, ou que xa o iniciaran e non saben como continuar; tamén pode ser de utilidade a quen quixer participar nun proxecto de software libre pola primeira vez.

O lector non ten por que saber programar, porén, debería posuír nocións básicas de conceptos de

enxeñaría de software tales como código fonte, compiladores e parches (*patches*).

Non é necesario posuír experiencia previa como usuario ou desenvolvedor de software libre. É posible que quen tiver traballado con anterioridade en proxectos de software libre pense que certas partes da obra son obvias de mais, e talvez desexe evitar esas seccións; mais o grao de experiencia dos lectores é tan variable que fixen un esforzo para etiquetar as seccións con claridade, e indiquei cales poden ser ignoradas se xa se dominan os conceptos que nelas se tratan.

Fontes

Gran parte do material que forma a obra é debido aos cinco anos que traballei co proxecto Subversion (http://subversion.tigris.org/) . Subversion é un sistema de control de versións de código aberto escrito desde cero, e preténdese que substitúa CVS para tornarse no sistema de control de versións de preferencia na comunidade de código aberto. O proxecto foi iniciado pola empresa para a que traballaba, CollabNet (http://www.collab.net/), a comezos de 2000 e, afortunadamente, souberon administralo desde o inicio como un proxecto de colaboración de múltiples partes. Un elevado número de desenvolvedores voluntarios adheríronse ao proxecto desde o inicio, que conta na actualidade con máis de 50 desenvolvedores, dos que unicamente unha pequena parte traballan para CollabNet.

Subversion é un exemplo case perfecto do que debe ser un proxecto de software libre, e finalmente baseeime nel máis do que pensaba facer inicialmente; en parte polas vantaxes que ofrecía, xa que me fornecía de inesgotables exemplos para ilustrar determinados fenómenos, mais tamén por cuestións de fidelidade. Embora colabore con outros proxectos de software libre en maior ou menor grao, e teño amigos e coñecidos que traballan en moitos outros, un decátase logo de que, ao poñer as cousas sobre o papel, é necesario procurar referentes reais de todo o que se afirma. Non quería facer referencias a outros proxectos baseándome exclusivamente no que lía nas listas públicas de correo, xa que se alguén actuase deste xeito con Subversion acertaría só a metade das veces. Así, sempre que me mergullei en proxectos cos que non tiña experiencia de primeira man á procura de inspiración ou exemplos, tentei falar con algún membro dos mesmos para que me explicase a situación real dos mesmos.

Traballei en Subversion os últimos 5 anos, mais xa levo 12 anos no mundo do software libre. Outros proxectos que deron forma a esta obra son:

- O proxecto de editor de texto GNU Emacs da Fundación para o Software Libre (Free Software Foundation), no que manteño uns poucos pacotes.
- *Concurrent Version System* (CVS, Sistema de Versións Concorrentes), no que traballei a fondo en 1994-95 con Jim Bandy, mais co que colaboro só esporadicamente desde entón.
- A colección de proxectos de código aberto coñecida como Apache Software Foundation, en particular o *Apache Portable Runtime* (APR) e o Servidor HTTP Apache (Apache HTTP Server).
- OpenOffice.org, a base de datos de Berkeley de Sleepycat, e a base de datos MySQL; non formei parte deses proxectos persoalmente, mais os seguín de preto e en ocasións falei con algúns dos seus membros.
- GNU Debugger (GDB) (idem)
- O proxecto Debian (idem).

A lista apenas está completa, dado que, como a maior parte dos programadores de código aberto, non manteño unha memoria exhaustiva dos proxectos nos que colaboro, o xusto para ter unha idea global de como están as cousas. Non os vou enumerar todos nesta sección, mais si están citados en outras

partes da obra.

Recoñecementos

Tardei catro veces máis do que tiña previsto en redactar a presente obra, e durante a maior parte dese tempo o traballo pendente producíame un gran desacougo. Se non tivese recibido a axuda de numerosos colaboradores, non daría rematado o libro sen perder o xuízo.

Andy Oram, de O'Reilly, é o editor que todo autor desexaría ter. Para alén dun extraordinario coñecemento do ámbito (suxeriume moitos dos temas), posúe o escaso don de comprender o que un quere dicir e de axudarlle a atopar a mellor maneira de dicilo. Tamén quero agradecerlle a Chuck Toporek o terlle trasladado esta proxecto a Andy de maneira inmediata.

Brian Fitzpatrick revisou a maior parte do material a medida que o redactaba, algo que non só mellorou a obra, senón que tamén me obrigou a escribir mesmo cando o último que quería facer era sentarme diante do computador. Ben Collins-Sussman e Mike Pilato tamén supervisaban o avance da obra, e sempre estaban prontos a debater (ás veces durante horas) calquera tema no que estivese a traballar na altura. Tamén vixiaban o meu ritmo de traballo, e animábanme a seguir cando diminuía. Desde aquí agradézolles a súa axuda.

Biella Coleman estaba a redactar a súa tese na altura na que eu estaba a redactar este libro, e sabe o que é sentarse a escribir un día si e outro tamén, tamén foi unha fonte de inspiración e un gran apoio. A súa formación como antropóloga fai que posúa unha fascinante visión do movemento do software libre, e achegou ideas e referencias que empreguei nesta obra. Alex Golub (outro antropólogo que ten un pé no mundo do software libre, e quen tamén estaba a rematar a súa tese na altura) apoioume moito nas fases iniciais, o que foi unha gran axuda.

Agradecementos tamén a Micah Anderson, que nunca deixou que o atormentase a presión de redactar as súas obras, o que me serviu de inspiración (e me deu algo de envexa tamén). Porén, sempre estaba pronto a axudar, conversar e, polo menos nunha ocasión, prestar apoio técnico.

Jon Trowbridge e Sander Striker animáronme na miña empresa e brindáronse a colaborar nela, sen a súa experiencia en produtos de software libre nunca tería rematado a obra.

Quero agradecerlle tamén a Greg Stein non só a súa amizade e os seus ánimos no momento adecuado, senón tamén terlle mostrado ao proxecto Subversive a importancia de revisar o código con frecuencia cando se quere crear unha comunidade de programadores. Estendo o meu agradecemento tamén a Brian Behlendorf, quen con gran tacto nos gravou na mente a importancia de debater as cousas en público, espero que o libro recolla ese principio.

Agradézolle a Benjamin «Mako» Hill e a Seth Schoen as conversacións sobre software libre e a súa política, a Zack Urlocker e a Louis Suarez-Potts térense deixado entrevistar embora estiveran moi atarefados, a Shane da lista de Slashcode permitirme citar os seus posts, e a Haggen So as comparacións que fixo de aloxamentos web estándar.

Agradézolle a Alla Dekthyar, a Polina e a Sonya térenme animado con cariño e paciencia. Alégrame que a partir de agora non teñamos que rematar (ou debería dicir tratar de rematar) prematuramente as nosas veladas para voltar a casa a traballar no «Libro».

Agradézolle a Jack Repenning a súa amizade, as conversacións que mantivemos e a súa teimuda negativa a aceptar unha análise fácil, mais errada, se for posible facer unha análise correcta, mais difícil. Espero que esta obra se impregnase de parte da súa larga experiencia no desenvolvemento de software libre e a industria do software.

CollabNet fixo gala dunha gran xenerosidade ao facilitarme un horario flexible para poder escribir, e non protestaron cando a redacción do libro demorou máis do previsto. Ignoro os complicados métodos de xestión que existen tras ese tipo de decisións, mais sospeito que Sandra Klute, e máis tarde Mahesh Murthy, tiveron algo que ver. Agradézolle a ambos a súa axuda.

O equipo de desenvolvemento de Subversion serviume de inspiración durante os últimos cinco anos, e unha gran parte do que escribín neste libro aprendino deles. Non vou redactar os meus agradecementos un por un, xa que son demasiados, mais prégolle ao lector que, se coincidir cun membro de Subversion, o invite a unha copa (eu penso facelo).

Queixeime a Rachel Scollon de como levaba a redacción do libro en numerosas ocasións; nunca se negou a escoitarme e de algunha maneira conseguía que os problemas parecesen menos problemas. Agradézolle desde aquí a súa inestimable axuda.

Transmítolle (mais unha vez) o meu agradecemento a Noel Taylor, quen con certeza se ten preguntado por que quixen escribir outro libro despois de queixarme tanto a primeira vez, mais cuxa amizade e xestión de Golosá contribuíron a que non me faltase nin música nin boa compañía mesmo cando estaba moi atarefado. Agradézolle tamén a Matthew Dean e a Dorothea Samtleben, amigos e compañeiros de desventuras musicais, a súa comprensión ante as miñas cada vez máis frecuentes escusas para non ensaiar. Megan Jennings apoioume en todo momento, e sempre mostrou un grande interese pola obra embora non coñecese o tema de preto, o que é unha gran axuda para un autor inseguro. Agradézolle a súa axuda a ela tamén.

Esta obra foi revisada por catro persoas eruditas e dilixentes: Yoav Shapira, Andrew Stellman, Davanum Srinivas, and Ben Hyde. Este libro gañaría moito se puidese inserir todas as súas brillantes suxestións, mais a escaseza de tempo obrigoume a facer unha escolma; porén, a mellora foi significativa. Calquera erro que houber é responsabilidade exclusiva miña.

Os meus pais, Frances e Henry, ofrecéronme o seu apoio, como sempre, e dado que este libro é menos técnico que o anterior, espero que atopen a súa lectura máis agradable.

Para rematar, gustaríame estender o meu agradecemento ás persoas a quen está dedicada esta obra, Karen Underhill e Jim Blandy. A amizade e o apoio de Karen son todo para min, non só durante a redacción deste libro, senón tamén durante os últimos sete anos. Non o tería rematado se non for por ela. Idem a respecto de Jim, un grande amigo e hacker entre hackers, e a persoa que me introduciu no software libre pola primeira vez, algo en certa maneira semellante a un paxaro que lle aprendese a voar a un avión.

Renuncia de responsabilidade

As reflexións e opinións reflectidas nesta obra son exclusivamente do autor. Non representan necesariamente a postura de CollabNet ou do proxecto Subversion.

Capítulo 1. Introdución

Case todos os proxectos de software libre fracasan.

Poucas veces nos chegan noticias dos fracasos. Os únicos proxectos que chaman a atención son os ben sucedidos, e a cantidade total de proxectos é tan elevada[1] que, embora só unha pequena porcentaxe sobrevivan o fracaso, o número de éxitos é alto. E non nos chegan noticias dos fracasos porque un fracaso non é un acontecemento importante. Un proxecto non deixa de ser viable nunha altura concreta, máis ben pódese considerar que os membros vanse afastando pouco a pouco do mesmo e deixan de traballar nel. É posible que, nunha altura concreta, sexa feita unha mudanza definitiva no proxecto, mais é frecuente que os responsables non se decaten de que se trata da última modificación ata tempo despois. Nin sequera existe unha definición clara de cando remata un proxecto. Cando pasan seis meses sen que ninguén traballe nel? Cando a base de usuarios deixa de medrar, sen superar a base de desenvolvedores? Que acontece se os desenvolvedores dun proxecto o abandonan porque se decatan de que están a duplicar o traballo de outros, e que acontece se se adscriben a outro proxecto, e o amplían ata incluíren nel a maior parte do seu traballo anterior? Considérase que o proxecto inicial rematou, ou que simplemente mudou de nome?

Estas complicacións fan que sexa difícil especificar o número de proxectos que fracasan, mais a experiencia de máis de unha década no mundo do código aberto, unha pequena procura en SourceForge.net e un toque de Google lévanos a concluír que a porcentaxe de fracasos é moi elevada, posiblemente de 90-95%. Esta porcentaxe aumenta se incluírmos proxectos activos mais disfuncionais (aqueles que están a producir código operativo, mais que non son atractivos, ou cuxos avances non son tan rápidos ou fiables como deberían).

O obxectivo desta obra é aprender como evitar o fracaso. Non se limita a examinar como facer as cousas ben, senón tamén como facelas mal, para que poidas decatarte dos problemas a tempo. Espero que este libro che forneza un repertorio de técnicas non só para evitares os atrancos máis frecuentes no desenvolvemento do código aberto, senón tamén para administrares o desenvolvemento e mantemento dun proxecto ben sucedido. O éxito non é un xogo de soma cero, e esta obra non explica como gañar ou superar a concorrencia. Unha parte importante da xestión dun proxecto de código aberto é, de feito, traballar en harmonía con outros proxectos relacionados. A longo prazo, todo proxecto ben sucedido contribúe á saúde do software libre a nivel mundial.

Podemos sentir a tentación de dicir que os os proxectos de software libre fracasan polos mesmos motivos que os proxectos de software propietario. Os requisitos irreais, as especificacións vagas, a xestión inadecuada de recursos, as fases de deseño insuficientes, ou calquera outro atranco non son patrimonio exclusivo do software libre, tamén son ben coñecidos por calquera que traballe na industria do software. Existen numerosas publicacións que tratan eses temas, así que non os vou comentar nesta obra; vou tentar describir os problemas específicos do software libre. Con frecuencia, o fracaso dun proxecto de software libre é debido a que os desenvolvedores (ou os administradores) non se decatan dos problemas que presenta o desenvolvemento de código aberto, embora sexan que de identificar as dificultades, máis coñecidas, que presenta o desenvolvemento de código pechado.

Un dos erros máis frecuentes é crearse expectativas irreais sobre as vantaxes do código aberto en si mesmo. Unha licenza de código aberto non é garantía de que milleiros de desenvolvedores van participar no teu proxecto; nin liberar o código dun proxecto vai solucionar os problemas que este tiver. Antes ben, soe acontecer o contrario: liberar un proxecto pode acarretar novas dificultades, e ser máis custoso a curto prazo que se for mantido en privado. Liberar un proxecto implica ordenar o código para que persoas alleas ao mesmo o poidan comprender, crear un sito web de desenvolvemento e listas de correo, e en moitos casos redactar documentación pola primeira vez, o que supón unha carga considerable de traballo. E, por suposto, se algúns desenvolvedores si se prestan voluntarios, hai que resolver as súas dúbidas antes de comezar a ver os resultados do seu traballo. O desenvolvedor Jamie Zawinski dicía o seguinte sobre os duros comezos do proxecto Mozilla:

> *O código aberto funciona, mais non é unha panacea. Podemos tirar a seguinte conclusión: non se pode coller un proxecto moribundo, botarlle os pos máxicos do "código aberto" e esperar que todo saia ben. O*

1 SourceForge.net, un aloxamento web moi coñecido, tiña rexistrados 79.225 proxectos a mediados de abril de 2004. Este número é moi inferior ao total de proxectos en activo, trátase unicamente dos que escolleron Sourceforge.

software é un terreo difícil, os problemas non son sinxelos.

(de **http://www.jwz.org/gruntle/nomo.html**)

Un problema que ten a ver con isto é a escasa importancia que se lle dá á presentación e á configuración dos pacotes, partindo da premisa de que sempre se pode facer máis adiante. A presentación a configuración dos pacotes envolven múltiples tarefas, todas elas encamiñadas a mellorar a accesibilidade. Para tornar o proxecto atractivo para os non iniciados, é necesario redactar documentación para os usuarios e os desenvolvedores, crear un Web do proxecto con información para os que acaban de chegar, automatizar a compilación e instalación o máximo posible, etc. Infelizmente, moitos programadores contemplan estas tarefas como algo secundario en comparación co código en si mesmo. Isto é debido a dous motivos principalmente. En primeiro lugar, pode parecer traballo inútil, xa que os máis beneficiados son aqueles que menos familiarizados están co proxecto, e viceversa; despois de todo, os desenvolvedores do código non necesitan os pacotes, porque xa saben como administrar, instalar e utilizar o software, non hai que esquecer que o código foi escrito por eles. E en segundo lugar, as habilidades necesarias para configurar a presentación e os pacotes son frecuentemente moi distintas das habilidades necesarias para crear o código. Os individuos tenden a concentrarse no que destacan, mesmo se facendo outra cousa na que teñen menos destreza lle darían un maior pulo ao proxecto. O capítulo 2, *Primeiros pasos*, fala de maneira detallada sobre a presentación e a configuración de pacotes, e explica por que é extremadamente importante darlles prioridade desde o inicio do proxecto.

Outro erro común é crer que non se necesita xerir os proxectos de código aberto, ou pola contra, crer que as estratexias de xestión empregadas nos proxectos pechados terán o mesmo éxito se foren aplicadas a proxectos de código aberto. As tarefas de xestión dun proxecto de código aberto non sempre son aparentes, mais case sempre están, dunha maneira ou de outra, detrás dos proxectos ben sucedidos, como mostra un sinxelo experimento mental. Un proxecto de código aberto consta dun conxunto variable de programadores (ben coñecidos por seren unha turma anárquica de seu) que, na maioría dos casos, non se coñecen persoalmente, e que posiblemente participarán no proxecto por motivos distintos. O experimento mental consiste en imaxinar como acabaría un grupo destas características sen administradores. Se non intervir un milagre, desaparecería ou disolveríase logo. Os proxectos non avanzan de motu propio, por máis que quixeramos que así fose. Mais as tarefas de xestión, embora activas, acostuman a ser informais, sutís e elementais. O único que mantén unido o grupo de desenvolvedores é a crenza de que xuntos poden chegar máis lonxe que por separado. A finalidade das tarefas de xestión é, principalmente, garantir que manteñan esa crenza, establecendo regras básicas de comunicación, asegurando que bos desenvolvedores non sexan marxinados debido a idiosincrasias persoais e, en xeral, tornando o proxecto atractivo para os desenvolvedores. A presente obra explica con detalle as técnicas específicas para atinxir estes obxectivos.

En último lugar, existe unha categoría de problemas que se pode denominar “fracasos da navegación cultural”. Hai dez anos, mesmo hai cinco, sería prematuro falarmos dunha cultura global do software libre, mais isto xa non é así. unha cultura identificable apareceu de vagar e, embora non sexa monolítica (é tan propensa a sufrir discordancias internas e sectarismos como calquera outra cultura asociada a un lugar xeográfico), posúe un núcleo razoablemente sólido. A maior parte dos proxectos de software libre comparten unha ou varias das características deste núcleo: recompensan certo tipo de comportamentos e condenan outros, crean unha atmosfera que convida á participación espontánea (ás veces a expensas dunha coordinación central), a súa visión do que é correcto ou incorrecto pode diferir notablemente da visión maioritaria noutros ámbitos. E, sobre todo, moitos dos colaboradores a longo prazo xa interiorizaron estas características, polo que comparten un certo consenso sobre cal debe ser a conduta a seguir. Os proxectos que fracasan soen afastarse dese núcleo, embora o fagan inconscientemente, e xeralmente carecen de ese consenso a respecto do comportamento que cabe esperar. Cando aparecen os problemas, a situación pode empeorar rapidamente, xa que os participantes carecen dos reflexos culturais nos que apoiarse para resolveren a súas diferenzas.

Este libro é unha guía práctica, non unha historia nin un estudo antropolóxico. Porén, uns certos coñecementos útiles sobre as orixes da cultura do software libre actual constitúen o alicerce fundamental de calquera consello práctico. Un individuo con coñecementos sobre esta cultura pode chegar lonxe no mundo do software libre; atopará multitude de peculiaridades e “dialectos”, mais poderá participar cómoda e eficazmente en calquera lugar. Un individuo que non comprenda esta cultura, atopará o proceso de organización ou participación nun proxecto difícil e cheo de sorpresas. Dado que o número de desenvolvedores de software continúa a aumentar, hai moita xente nesta última categoría. É esta unha cultura de inmigrantes recentes, e continuará a selo durante un tempo. Se pensas que es un destes últimos, a seguinte sección proporcionarache conceptos básicos útiles para posteriores discusións, tanto na presente obra como en Internet. (Por outra banda, se levas tempo

traballando con código aberto, é posible que saibas moito sobre a súa historia, e talvez desexes saltar a seguinte sección).

Historia

O feito de compartir software é tan vello como o propio software. Nos inicios da era dos computadores, os fabricantes crían que as vantaxes competitivas dependían das innovacións no terreo do hardware, e en consecuencia non contemplaban o software como un factor de negocio. Moitos dos compradores deses computadores eran cientistas e técnicos, capaces de modificar e ampliar o software que acompañaba as máquinas. Con frecuencia, os clientes enviaban as súas versións non só aos fabricantes, senón tamén a outros propietarios de computadores semellantes. Os fabricantes con frecuencia toleraban e mesmo alentaban este comportamento: para eles, as melloras feitas no software, sen importar por quen, tornaban o computador máis atractivo para outros clientes potenciais.

Embora este período inicial se asemelle á cultura do software libre actual en moitos aspectos, diferénciase desta en dous puntos fundamentais. En primeiro lugar, a compatibilidade entre equipos era mínima (esta etapa caracterizouse por unha innovación constante no deseño dos computadores, mais debido á diversidade de arquitecturas case ningún equipo era compatible co resto), polo que o software escrito para un equipo poucas veces servía para outros. Os programadores habitualmente adquirían experiencia nunha arquitectura ou familia de arquitecturas determinada (na actualidade é frecuente que se especialicen nunha linguaxe ou nunha familia de linguaxes de programación en particular, coa convicción de que as destrezas adquiridas poderán ser transferidas a calquera tipo de hardware co que tiveren que traballar). Dado que as destrezas dun individuo estaban frecuentemente confinadas a un tipo concreto de computador, a acumulación de destrezas tornaba ese equipo máis atractivo para o resto dos seus colegas. A expansión deses coñecementos e dos códigos específicos para cada equipo beneficiaban por tanto os intereses dos fabricantes.

En segundo lugar, non había Internet. Embora as restricións legais sobre o acto de compartir fosen menores que hoxe, as restricións de carácter técnico eran maiores: trasladar datos de un sitio a outro era, en termos relativos, incómodo e custoso. Existían pequenas redes locais, idóneas para compartir información entre os empregados dunha mesma empresa ou laboratorio de investigación, mais era necesario superar barreiras para compartir material co resto da xente, sen importar onde vivisen. Estas barreiras eran superadas en moitos casos: ás veces algúns grupos contactaban con outros de maneira independente, e trocaban discos ou fitas por correo, ás veces os propios fabricantes obraban como intermediarios na elaboración de remendos (*patches*). Tamén axudou o feito de que moitos dos primeiros desenvolvedores informáticos pertencesen ao ámbito universitario, onde é frecuente publicar os novos avances. Mais a realidade física da transmisión de datos dificultaba o acto de compartir, tanto máis canto maior fose a distancia (real ou en termos organizativos) que tivese que cruzar o software. O acto de compartir a gran escala e con facilidade, como o coñecemos hoxe, era imposible.

A aparición do software privativo e do software libre

Ao ir madurando a industria, tiveron lugar de maneira simultánea mudanzas que estaban interrelacionadas. A enorme diversidade de deseños de hardware foi desaparecendo en favor duns poucos vencedores, que se impuxeron mercé de unha tecnoloxía superior, mellor publicidade ou unha combinación de ambos. Nesa altura, e non de maneira casual, o desenvolvemento das chamadas linguaxes de programación de “alto nivel” permitiron que programas escritos nunha linguaxe puidesen ser traducidos automaticamente (compilados) e utilizados en distintos computadores. Os fabricantes decatáronse logo das posibilidades deste método: un cliente podía levar a cabo un gran esforzo de creación de software sen restrinxirse a unha arquitectura determinada. Se combinarmos o anterior coa cada vez menor diferenza de rendemento entre os distintos computadores, consecuencia da desaparición dos deseños menos eficientes, os fabricantes que contemplaban o seu hardware como o seu único activo enfrontábanse a unha futura redución da súa marxe de beneficios. O rendemento bruto das máquinas estaba a se tornar nun ben funxible, en canto o software estaba a se tornar no elemento diferenciador. A venda de software, ou polo menos a súa integración na venda de hardware, tornouse nunha boa estratexia.

Como consecuencia do anterior, os fabricantes comezaron a impoñer dereitos de autor cada vez máis estritos sobre o seu código, dado que os usuarios, se continuaren a compartir e modificar o código entre eles, poderían chegar a reimplementar algúns dos avances comercializados como “valor engadido” polo distribuidor. Ironicamente, todo isto aconteceu na altura en que Internet comezaba a coller folgos. Apenas o acto de

compartir software comezaba a ser posible desde o punto de vista técnico, cando o as mudanzas no negocio dos computadores o tornaron en algo pouco desexable desde o punto de vista económico, polo menos desde o punto de vista de calquera empresa a nivel individual. Os fornecedores impediron o acceso, ben negándolle o acceso ao código aos usuarios dos seus equipos, ben impoñendo acordos de non divulgación que imposibilitaban o acto de compartir.

A resistencia consciente

En canto desaparecía o universo do código sen restricións, un programador reaccionou ante este feito. Richard Stallman traballaba no Laboratorio de Intelixencia Artificial do Instituto Tecnolóxico de Massachussets (MIT) na década dos 70 e inicios dos 80, durante o que viu a verificarse a mellor época e o mellor lugar para quen quixese compartir código. No laboratorio de IA reinaba unha forte "ética hacker", e animábase os traballadores a compartiren calquera mellora que fixesen no sistema. Tempo despois, o propio Stallman escribiu:

> *Non chamabamos o noso software "software libre" porque o termo aínda non existía, mais era xustamente software libre. Cando calquera persoa de outra universidade ou empresa quería levar e utilizar un programa, dabámoslle permiso encantados. Se vías alguén traballar cun programa novidoso e interesante, podías pedirlle que che deixase ver o código fonte, para poder mudalo ou copiar partes e facer un programa novo.*

(de http://www.gnu.org/gnu/thegnuproject.html)

Esta comunidade idílica desapareceu pouco despois de 1980, cando as mudanzas que tiveran lugar no resto do mundo chegaron ao Laboratorio de IA. Unha empresa de recente creación contratou a maior parte dos programadores do Laboratorio para traballaren nun sistema operativo moi semellante a aquel co que traballaran na etapa anterior, mais esta vez baixo licenza exclusiva. Na mesma altura, o laboratorio de IA adquiriu unha serie de equipos que xa traían un sistema operativo propietario.

Stallman recoñeceu un padrón na cadea de acontecementos:

> *Os computadores modernos naquela altura, como o VAX ou o 68020, posuían os seus propios sistemas operativos, mais non se trataba en ningún caso de software libre, xa que estabas obrigado a asinar un acordo de non divulgación mesmo para obter unha copia executable.*
>
> *Así, o primeiro paso para utilizar un computador pasaba por negarlle axuda ao veciño, a existencia dunha comunidade de cooperación era prohibida . A norma imposta polos donos do software propietario era: "se queres compartir co teu veciño, es un pirata; se quixeres modificar algo, préganos que o fagamos nós".*

Algo dentro del opúxose a esta tendencia. En lugar de continuar co seu traballo no (agora moi debilitado) Laboratorio de IA, ou de dixitar código nunha das novas compañías, onde o produto do seu traballo ficaría fechado nunha caixa, deixou o Laboratorio e iniciou o Proxecto GNU e a Fundación polo Software Libre (FSF). GNU1 procuraba desenvolver un sistema operativo e un conxunto de aplicacións completamente libres e gratuítas, que os usuarios puidesen hackear e compartir as modificacións que fixesen. Estaba a recrear, en esencia, o que se perdera no Laboratorio de IA, mais a escala mundial e sen as vulnerabilidades que tornaran a cultura do Laboratorio de IA en algo susceptible de desaparecer.

Para alén de traballar no novo sistema operativo, Stallman creou unha licenza de dereitos de autor cuxos termos garantían que o seu código ficaría libre para sempre. A Licenza Pública Xeral da GNU (GPL) constitúe unha obra maxistral de enxeñaría legal: establece que o código pode ser copiado e modificado sen restricións de ningún tipo, e que tanto as copias como os produtos derivados (por exemplo, as versións modificadas) deben ser distribuídas baixo a mesma licenza que o orixinal, sen ningunha clase de restricións adicionais. O que fai é aproveitarse das leis de dereitos de autor para atinxir o efecto contrario ao que perseguen as leis de dereitos de autor tradicionais: en lugar de limitar a distribución do software, evita que calquera, mesmo o autor, a limite. Stallman considerou que este sistema era mellor que limitarse a entregar o seu código ao dominio público. Se for de dominio público, calquera copia particular do mesmo podería ser incorporada a un programa propietario (como xa aconteceu con código protexido por licenzas de dereitos de autor permisivas). Embora tal incorporación non afectaría en absoluto á dispoñibilidade do código, si implicaría que os esforzos de Stallman poderían beneficiar o inimigo, o software propietario. A GPL pode ser contemplada como unha medida

protectora do software libre, en canto evita que o software non libre tire vantaxes do código protexido pola GPL. O capítulo 9, *Licenzas, dereitos de autor (copyright) e patentes*, analiza a GPL e a súa relación con outras licenzas de software libre.

Coa axuda de numerosos programadores, algúns dos que compartían a ideoloxía de Stallman e outros que simplemente querían que houbese unha gran cantidade de código libre en circulación, o proxecto GNU comezou a publicar substitutos libres da maior parte dos compoñentes fundamentais dun sistema operativo. A estandarización actual de hardware e software, amplamente estendida, posibilitou a utilización de substitutos GNU en sistemas non libres, algo do que se beneficiou moita xente. O editor de texto da GNU (Emacs) e o compilador C (GCC) tiveron un éxito especial, e gozaron de unha gran aceptación non por motivos ideolóxicos, senón pola súa perfección técnica. Por volta de 1990, a GNU producira a maior parte dun sistema operativo, excepto o núcleo (*kernel*), a parte da máquina responsable do arranque, de administrar a memoria, o disco e outros recursos do sistema.

Infelizmente, o proxecto GNU seleccionara un núcleo (*kernel*) cuxa implementación deu en ser máis complexa do que se esperara. A demora impediu que a Fundación polo Software Libre lanzase a primeira versión dun sistema operativo completamente libre. A peza final foi colocada por Linus Torvalds, estudante finés de ciencias da computación quen, axudado por voluntarios de todo o mundo, completara un núcleo libre de deseño máis tradicional. Púxolle o nome de *Linux*, e cando foi combinado cos programas GNU existentes, deu orixe a un sistema operativo totalmente libre. Pola primeira vez era posible ligar o computador e traballar sen utilizar software propietario.

Tecnicamente, Linux non foi o primeiro. Un sistema operativo libre para computadoras compatibles con IBM chamado 386BSD, tiña sido publicado antes que Linux. Porén, era máis complexo de configurar e executar o 386BSD. Linux tivo tanto éxito non so por ser libre, senón tamén porque as posibilidades de que o teu computador funcionase cando o instalabas eran moi elevadas.

Unha grande parte do software deste sistema operativo non foi producido polo proxecto GNU. De feito, a GNU non era o único grupo que estaba a traballar na produción dun sistema operativo libre (por exemplo, o código que co tempo desembocou en *NetBSD* e *FreeBSD* xa estaba en proceso de desenvolvemento nesta altura). A relevancia da Fundación polo Software Libre non se limita ao código que creou, senón que se estende tamén á súa retórica política. O feito de que contemplasen o software libre como unha causa e non como un produto despertaba unha certa conciencia política nos programadores. Mesmo aqueles que non concordaban cos postulados da FSL víanse na obriga de participaren no debate, embora só fose para manifestaren unha opinión diverxente. A efectividade da FSF como órgano propagandístico radicaba no feito de vincularen o código a unha mensaxe, coa axuda da GPL entre outros. Ao estenderse o código, estendeuse tamén a mensaxe.

Resistencia accidental

Porén, os inicios do software libre foron unha época de gran actividade, e poucas accións tiñan un compoñente ideolóxico tan explícito como o Proxecto GNU de Stallman. Un dos movementos máis destacados foi a *Berkeley Software Distribution* (BSD), unha reimplementación gradual do sistema operativo UNIX (que ata finais dos 70 foi un proxecto de investigación da AT&T de carácter lixeiramente propietario) xerido por programadores da Universidade de California en Berkeley. A turma da BSD non fixo declaracións de carácter político sobre a necesidade de os programadores se uniren e compartiren entre eles, mais puxeron a idea en práctica con talento e entusiasmo, coordinando un esforzo desenvolvedor distribuído no que os intérpretes de comandos e as librerías de códigos, e despois o propio núcleo do sistema operativo, foron reescritos desde cero, principalmente por voluntarios. O proxecto BSD tornouse no exemplo perfecto de desenvolvemento de software libre sen vinculacións ideolóxicas, e tamén lle serviu de adestramento a moitos desenvolvedores que pasaron a traballar no mundo do código aberto.

Outro crisol de desenvolvemento cooperativo foi o *X Window System*, un entorno gráfico de computación libre e transparente, desenvolvido polo MIT a mediados dos 80 en colaboración con distribuidores de hardware interesados en ofrecer un sistema de xanelas aos seus clientes. Lonxe de opoñerse ao sistema propietario, a licenza X permitía engadir extensións propietarias ao núcleo libre de maneira deliberada (cada un dos membros do consorcio quería ter a oportunidade de perfeccionar a distribución X, obtendo así unha vantaxe competitiva sobre o resto dos membros). *X Windows*[2] era software libre en si mesmo, mais o seu obxectivo era igualar as

2 Prefiren chamalo "X Window System", mais na práctica emprégase o nome "X Windows" por simple comodidade.

condicións de xogo entre intereses empresariais, non poñer fin ao dominio do software propietario. Outro exemplo que se anticipou ao proxecto GNU nuns poucos anos foi *TeX*, un sistema de tipografía de calidade profesional creado por Donald Knuth e publicado baixo unha licenza que permite modificar e distribuír o código, mais prohibe referirse a esas novas versións como TeX a menos que superen unha serie de probas de compatibilidade moi estritas (trátase dun exemplo de licenza libre con "protección da marca comercial", das que falarei en maior profundidade no capítulo 9, *Licenzas, dereitos de autor (copyright) e patentes*). Knuth non se posicionou no debate software libre fronte a software propietario, simplemente necesitaba un sistema tipográfico mellor para atinxir o seu auténtico obxectivo (un libro sobre programación de computadores) e non viu motivo ningún para non poñer o seus sistema ao alcance do resto do mundo unha vez rematase.

Sen enumerar todos os proxectos e licenzas, pódese dicir que, a finais dos 80, existía unha gran cantidade de software libre baixo unha ampla variedade de licenzas. Esta diversidade de licenzas correspondíase cunha gran diversidade de motivacións. Mesmo algúns dos programadores que escolleran a GNU GPL tiñan menos motivacións ideolóxicas ca o propio proxecto GNU. Embora traballasen a gusto con software libre, moitos desenvolvedores non percibían o software propietario como un mal social. Había quen sentía o compromiso moral de librar o mundo do "software acaparador" (termo coinado por Stallman para referirse ao software non libre), mais a outros animábaos o carácter técnico dos proxectos, ou traballar con colegas con esquemas mentais semellantes, outros simplemente ambicionaban a gloria. Con todo, esta gran variedade de motivacións non entraban en conflito entre elas. A causa radica en que o software, a diferenza de outras modalidades creativas como a prosa ou as belas artes, debe superar probas moderadamente obxectivas para ser considerado ben sucedido: debe funcionar, e non pode presentar demasiados erros (*bugs*). Isto proporciónalles aos participantes nun proxecto unha especie de base común, un motivo e un marco para cooperaren sen se preocuparen de demasiados detalles de outro carácter que non sexa o técnico.

Os desenvolvedores tiñan aínda outro motivo para traballaren xuntos: demostrouse que o universo do software libre estaba a crear código de moi boa calidade, que nalgúns casos mesmo chegaba a superar a outras alternativas non libres desde o punto de vista técnico; noutros casos era polo menos igual de bo, e por suposto sempre máis barato. Mentres que só unhas poucas persoas se sentían motivadas a empregar software libre por motivos puramente filosóficos, un gran número de individuos o empregaban porque funcionaba mellor, e unha porcentaxe dos que o utilizaban estaban dispostos a doar o seu tempo e habilidades para axudar a manter e a mellorar o software.

Esta tendencia cara a produción de código de calidade non era universal, mais estaba a gañar cada vez maior presenza nos proxectos de software libre a nivel mundial. Os negocios que dependían moito do software comezaron a decatarse diso, e moitos deles descubriron que xa estaban a utilizar software libre en tarefas do día a día, sen sabelo (os xestores de alto nivel moitas veces non saben o que fai o departamento de TIC). As corporacións comezaron a tomar parte de maneira máis activa e aberta en proxectos de software libre. Os beneficios derivados podían, nalgúns casos, multiplicar varias veces o investimento inicial. A empresa contrata unicamente un pequeno grupo de programadores expertos para traballaren a tempo enteiro no proxecto, mais benefíciase das contribucións de todo o mundo, voluntarios e programadores contratados por outras compañías incluídas.

"Libre" fronte a "código aberto (open source)"

A medida que aumentou o interese das corporacións polo software libre, os programadores enfrontáronse a novos retos de presentación, un dos cales era a propia palabra "libre". Ao escoitar o termo "libre" pola primeira vez, moita xente cre que se trata simplemente de "software gratuíto". É certo que todo o software libre é gratuíto, mais non todo o software gratuíto é libre. Por exemplo, na altura da guerra dos navegadores, nos anos 90, tanto Netscape como Microsoft regalaban copias dos seus respectivos navegadores co ánimo de ampliar a súa cota de mercado, mais en ningún caso se trataba de "software libre". Non se podía acceder ao código fonte, e se alguén o conseguía, non tiña dereito a modificalo nin a redistribuílo. O único que podía facer era descargalo como ficheiro executable e abrilo. Estes navegadores eran tan libres como calquera programa comprado na tenda, a única diferenza era o prezo.

A confusión causada pola palabra "*free*" é debida a unha ambigüidade do adxectivo en inglés. A maior parte das demais linguas diferencian entre a liberdade e o barato (a distinción entre gratis e libre é obvia para calquera falante dunha lingua romance, por exemplo). Mais o status que o inglés ten como lingua franca da Internet implica que un problema co inglés é, ata certo punto, un problema para todo o mundo. A confusión creada pola palabra "*free*" estaba tan estendida que o programadores crearon unha fórmula explicativa: *"It's free as in*

freedom- think free speech, not free beer" (libre de liberdade, pensa en "discurso libre", non en "cervexa gratis"). Porén, ter que explicalo constantemente cansa, e moitos programadores opinaban, con razón, que a palabra "*free*" impedía que a xente comprendese as características dese software.

Mais as raíces do problema eran máis profundas, xa que a palabra "libre" posuía connotacións morais evidentes: se a liberdade constituía un fin en si mesma, era indiferente que o software libre fose superior, ou máis beneficioso para determinados negocios en determinadas circunstancias. Estes eran simplemente efectos colaterais positivos dunha empresa que non era, en esencia, nin técnica nin mercantil, senón simplemente moral. Aínda máis, a filosofía *"free as in freedom"* (libre de liberdade) presentaba un problema de conciencia para as empresas que desexaban apoiar programas específicos de software libre, mais que ao mesmo tempo querían continuar a vender software propietario.

Estes dilemas aterraron nunha comunidade que xa era propensa de seu ás crises de identidade. Os programadores que producen software libre nunca foron unánimes a respecto de cal debería ser o obxectivo común do movemento polo software libre, se houber algún. Mesmo dicir que as opinións van de un estremo ao outro crearía confusión, xa que daría a entender a existencia de unha gradación linear en lugar da distribución multidimensional que existe na realidade. Porén, pódense distinguir dúas grandes categorías ideolóxicas, se ignorarmos as sutilezas polo momento. Uns comparten o punto de vista de Stallman de que que a liberdade de compartir e facer modificacións é o máis importante, e se perdérmonos de vista a liberdade, non atenderiamos ao elemento principal. Outros cren que o software en si mesmo constitúe a baza principal, e séntense incómodos ao identificaren o software propietario como algo malo de seu. Algúns programadores, mais non todos, cren que o autor (ou o contratista, se o traballo for remunerado), debería ter control sobre a distribución, e que a escolla dos termos non debería ficar supeditada a xuízos morais de ningún tipo.

Estas diferenzas non requiriron unha análise detallada nin unha articulación profunda durante moito tempo, mais o auxe do software libre no mundo dos negocios mudou esta situación. O termo código aberto (*open source*) foi creado en 1998 como alternativa ao termo "libre" (*free*) por un grupo de programadores que daría orixe á Iniciativa do Código Aberto (OSI1 polas siglas en inglés). Para a OSI, non só o termo "software libre" (*free software*) era potencialmente confuso, senón que a propia palabra "libre" (*free*) era un sintoma dun problema máis amplo, concretamente, de que o movemento necesitaba unha campaña de marketing para tornalo atractivo ante os ollos do mundo empresarial, e que falar de aspectos morais e dos beneficios sociais de compartir non espertarían interese nas reunións de executivos. Nas súas propias palabras:

> *A Iniciativa do Código Aberto é unha campaña de marketing de software libre. Dálle un pulo ao "software libre" desde unha visión pragmática, sen recorrer a discursos ideolóxicos inflamados. A esencia segue a ser a mesma, mais a actitude derrotista e o simbolismo mudaron. […]*
>
> *O que hai que lograr que comprendan a maior parte dos especialistas non é o concepto de código aberto, senón o nome. Por que non chamalo, como fixemos sempre, software libre?*
>
> *Un dos motivos principais é que o termo "software libre" (free software) crea tal confusión que pode propiciar conflitos...*
>
> *Mais a mudanza de nome é de carácter puramente mercantil, xa que queremos trasladar as nosas ideas ao mundo empresarial. Dispoñemos dun produto excelente, mais a nos posición no pasado era desastrosa. Os executivos non comprenden o significado real do termo "software libre", identifican o desexo de compartir con estratexias anticomerciais, ou aínda peor, con roubar.*
>
> *A maior parte dos grandes executivos nunca van comprar "software libre". Mais se collermos a mesma tendencia, a mesma xente e as mesmas licenzas, e mudarmos o nome para "código aberto", si van comprar.*
>
> *Para algúns hackers é difícil de crer, porque son especialistas que pensan en termos concretos e substanciais e que non comprenden a importancia da imaxe cando se quere vender algo.*
>
> *No marketing, as aparencias son a realidade. Aparentar que temos vontade de saír das barricadas e de cooperar co mundo empresarial é tan importante como os nosos principios, as nosas convicción e o noso software.*

(de http://www.opensource.org/. Parece que a OSI ten eliminado as páxinas desde entón, embora poidan ser lidas en http://web.archive.org/web/20021204155057/http://www.opensource.org/advocacy/faq.php e http://web.archive.org/web/20021204155022/http://www.opensource.org/advocacy/case_for_hackers.php#marketing.)

No texto pódense enxergar vestixios de controversia. Fai referencia a "as nosas convicción s", mais evita con habilidade concretar en que consisten esas convicción s. Algúns pensarán que as convicción s teñen a ver con que o código desenvolvido nun proceso aberto vai ser mellor; outros, que defenden que toda a información debe ser compartida. Fálase de "roubar" para referirse (presuntamente) ás copias ilegais, un concepto co que moita xente discrepa, xa que o elemento copiado segue en posesión do propietario orixinal despois de feita a copia. O texto deixa entrever que o movemento polo software libre pode ser acusado erroneamente de anticomercialismo, mais deixa no aire a cuestión de se tales acusacións son lexítimas.

Con isto non quero dicir que o web da OSI sexa incoherente ou conduza a equivocacións, non é así. Máis ben, trátase dun exemplo perfecto das carencias que ten o movemento polo software libre, e que a OSI leva tempo denunciando; en concreto, unha boa estratexia de marketing, onde por "boa" se entende "viable no mundo empresarial". A Iniciativa polo Código Aberto forneceulle a moita xente o que estaban a procurar, un vocabulario que permite falar sobre o software libre como unha metodoloxía de desenvolvemento e unha estratexia empresarial, en lugar de unha cruzada moral.

A creación da Iniciativa polo Código Aberto mudou a paisaxe do software libre. Formalizou unha dicotomía que ficara anónima durante moito tempo, o que forzou o movemento a recoñecer que posuía unha política tanto interna como externa. Como resultado, hoxe en día, ambas partes teñen que atopar un elemento común, xa que programadores de ambas faccións colaboran en moitos proxectos, e hai participantes que non poden ser adscritos a ningunha das dúas categorías. Isto non quere dicir que non se fale de motivacións de carácter moral (ás veces advírtese de problemas coa "ética hacker" tradicional, mais é infrecuente que un desenvolvedor de software libre/código aberto cuestione abertamente as motivacións do resto de persoas que participan no proxecto). A contribución é o que prevalece, se alguén escribe bo código non lle preguntas se o fai por convicción s morais, porque cobra por iso, porque quere ter un bo currículo, etc. A contribución é avaliada en termos técnicos, e respóndese en termos técnicos. Mesmo organizacións cunha política explícita como o proxecto Debian, cuxo obxectivo é ofrecer un entorno de computación 100% libre, non poñen impedimentos á hora de traballar con código non libre e de cooperar con programadores que non comparten os seus obxectivos.

A situación actual

Ao xestionares un proxecto de software libre non terás que tratar temas filosóficos tan profundos con frecuencia. Os programadores non van esixir que todos os participantes do proxecto compartan as súas ideas en todos os ámbitos (os que si o fan decátanse logo de que non poden colaborar en ningún proxecto). Mais tes que ser consciente da existencia das diferencias entre "software libre" e "código aberto", en parte para evitares ferir a sensibilidade de algúns dos colaboradores, e en parte tamén porque comprender as motivacións dos desenvolvedores é o mellor xeito (e ata certo punto o único) de xerir un proxecto.

O software libre é unha cultura de libre elección, e para desenvolvéreste nela con éxito tes que comprender os motivos que levan a determinadas persoas a desexar ser parte da mesma. A coerción non funciona; se os colaboradores dun proxecto non estiveren contentos co mesmo migrarán a outro. Un dos aspectos máis salientables do software libre, mesmo entre comunidades de voluntarios, é a pouca dedicación que esixe. A maioría dos colaboradores dun proxecto non se coñecen persoalmente, e limítanse a doar parte do seu tempo cando lles apetecer. Os condutos habituais de interacción entre os seres humanos vense reducidos a un único elemento: a palabra escrita, propagada a través de circuítos electrónicos. A consecuencia disto pode pasar moito tempo ata que se forma un grupo comprometido e, de igual maneira, é fácil perder voluntarios potenciais nos cinco primeiros minutos de "conversa". Se un proxecto non causar unha boa primeira impresión, os que acaban de chegar rara vez lle dan unha segunda oportunidade.

A volatilidade (ou mellor dito a volatilidade potencial) das relacións é quizais o maior desafío ao que se enfrontan os proxectos nas súas fases iniciais. Que vai manter a toda esa xente unida o bastante como para produciren algo de utilidade?. A resposta a esta pregunta é o bastante complexa como para ocupar o resto desta obra, mais se tivermos que resumila nunha oración, sería a seguinte:

Os colaboradores deberían percibir que a súa implicación nun proxecto e a súa influencia sobre o mesmo son directamente proporcionais ás súas colaboracións.

Ningún grupo de desenvolvedores, ou de desenvolvedores en potencia, debería sentirse discriminado por razóns de carácter non técnico. É obvio que os proxectos que contaren con patrocinio empresarial e/ou desenvolvedores asalariados deben prestar especial coidado neste sentido, como se explica no capítulo 5, *Diñeiro*. Isto non quere dicir que se non houber patrocinio empresarial non hai nada de que se preocupar; o diñeiro non é máis ca un dos múltiples factores que poden afectar o éxito dun proxecto. Tamén hai cuestións como que linguaxe empregar, que licenza, que proceso de desenvolvemento, que tipo de estrutura vai ser implementada, como publicitar a creación do proxecto de maneira eficaz, e moitas máis. O próximo capítulo explica como iniciar un proxecto con bo pé.

Capítulo 2. Primeiros pasos

O modelo clásico de como comezan os proxectos de software libre foi proposto por Eric Raymond, nun artigo xa famoso sobre procesos no software libre titulado *A catedral e o bazar*. Raymond escribiu:

> *Todos os bos proxectos de software inícianse para resolver un problema concreto dun desenvolvedor. (ver http://www.catb.org/~esr/writings/cathedral-bazaar/).*

Raymond non está dicindo que os proxectos de software libre unicamente xurdan cando un desenvolvedor ten un problema. Máis ben, está dicindo que os *bos* programas son resultado de que un programador teña un interese persoal en ver o problema resolto; porén, resolver un problema persoal é a motivación máis frecuente dos proxectos de software libre.

Aínda que esta segue sendo a razón máis habitual para a creación de moitos proxectos de software libre, é menos agora do que era en 1997, cando Raymond escribiu esas palabras. Hoxe en día, temos o fenómeno de organizacións -incluíndo as organizacións con ánimo de lucro- que inician grandes proxectos de software libre desde cero. O programador solitario, dixitando algún código para resolver un problema persoal para decatarse máis tarde de que o seu traballo ten unha aplicación máis ampla, é aínda a fonte da maior parte do software libre de nova factura, mais non a única.

Porén, o argumento de Raymond é perspicaz. A condición esencial é que os produtores do software teñan interese directo no seu éxito, xa que eles mesmos o utilizan. Se o programa non se comportar como debería, a persoa ou organización que o levan a cabo sentirá insatisfacción no seu día a día. Por exemplo, o proxecto OpenAdapter (http://www.openadapter.org/), que foi iniciado polo banco de investimentos Dresdner Kleinwort Wasserstein como unha ferramenta de traballo de software libre para integrar sistemas de información financeira dispares, é pouco probable que resolva a necesidade persoal dun desenvolvedor. Máis ben resolve a dunha institución. En particular, este problema xorde da experiencia da institución e dos seus socios, por tanto, se o programa non resolver o problema, eles o sufrirán. Esta situación provoca que se produza bo software, xa que a retroalimentación flúe na dirección correcta. O software non é escrito para venderllo a un terceiro, senón para resolver as *propias* necesidades, e posteriormente é compartido con todo o mundo, como se o problema fose unha doenza e o software o medicamento, que debe ser distribuído para erradicar a pandemia.

Este capítulo versa sobre como presentar ao mundo un novo proxecto de software libre, mais moitas das súas recomendacións seríanlle familiares a unha organización sanitaria que distribúe medicamentos. Os obxectivos son moi similares: desexas deixar claro o que fai o medicamento, facelo chegar ás mans da xente que o necesita e asegurarte de que eses que o reciben saben como usalo. Mais co software, desexas ademais atraer algúns dos receptores para que se sumen ao actual esforzo de investigación para mellorar o medicamento.

A distribución de software libre é unha tarefa a dúas bandas: o software precisa de usuarios e de desenvolvedores. Estas necesidades non están necesariamente en conflito, mais engaden algunha complexidade á presentación inicial do proxecto. Algunha información é útil para ambas audiencias, outra unicamente para unha delas. Toda esa información debería seguir o principio de presentación escalada: o grao de detalle presentado en cada etapa debe ser proporcional á cantidade de tempo e esforzo empregado polo lector. Máis esforzo debería acarretar unha recompensa maior. Cando non se equilibran ambos, as persoas poden perder rapidamente a fe e o impulso.

O corolario disto é que *as aparencias importan*. Habitualmente, os desenvolvedores non desexan crer nisto. O seu amor pola substancia sobre a forma é case un punto de orgullo profesional. Non é casualidade que tantos programadores amosen unha antipatía pola publicidade e as relacións públicas, nin que os deseñadores gráficos profesionais se horroricen co que os programadores propoñen.

Isto é unha mágoa, xa que existen situacións onde a forma *é* a substancia, e a presentación do proxecto é unha delas. Por exemplo, a primeira cousa que unha persoa descobre sobre un proxecto é a aparencia da súa páxina web. Esta información é absorbida antes que calquera outro contido - antes de ler un texto ou premer unha ligazón. Embora poida parecer inxusto, a xente non pode evitar facerse unha primeira impresión. A aparencia da web sinala se se dedicou tempo e esforzo á organización da presentación do proxecto. Os seres humanos temos unha antena extremadamente sensible para detectar o esforzo dedicado. Moitos de nós podemos dicir tras unha única ollada se un sitio web foi publicado rapidamente ou se foi pensado con mimo. Esta é a primeira peza de información que o proxecto mostra, e, por asociación, a impresión que crea será traspasada ao resto do proxecto.

Así, aínda que a maioría deste capítulo fala sobre o contido co que se debería iniciar o teu proxecto, recorda que a súa aparencia tamén importa. Debido a que a páxina web ten que abranguer dous tipos de visitantes distintos - usuarios e desenvolvedores - debe ser posta especial atención na claridade e concisión. Aínda que este non é lugar para un tratado xeral sobre deseño web, existe un principio o suficientemente importante como para prestarlle atención, particularmente cando a web informa a múltiples (é dicir, solapadas) audiencias: a xente debería ter unha vaga idea de a onde conduce unha ligazón antes de premela. Por exemplo, *con só ver a ligazón* á documentación de usuario ten que ficar claro que esta nos leva á documentación de usuario e non, por exemplo, á de desenvolvedor. Xerir un proxecto baséase parcialmente en subministrar información, mais tamén consiste en ofrecer comodidade. A mera presencia de estándares, nos lugares esperados, tranquiliza a usuarios e desenvolvedores que están decidindo se involucrarse ou non no proxecto. Comunica que o proxecto actúa como un todo, que anticipou as cuestións que a xente preguntará e respondeunas de maneira que o visitante teña que facer o mínimo esforzo. Dando esta sensación de preparación, o proxecto envía a seguinte mensaxe: "O teu tempo non será esbanxado se participas", que xustamente é a mensaxe que a xente precisa escoitar.

Antes de nada, bota unha ollada ao redor

Antes de comezar un proxecto de software libre, hai unha advertencia importante:

Olla sempre ao redor para ver se existe un proxecto que fai o que desexas. Son grandes as posibilidades de que calquera problema que desexas resolver, xa o tivese que resolver algún outro anteriormente. Se o fixeron e publicaron o código baixo unha licenza libre, non hai razón ningunha para reinventares a roda. Por suposto que hai excepcións: se desexas iniciar un proxecto como unha experiencia educativa, o código existente non é de axuda; ou se o proxecto que tes en mente é tan especializado que sabes que

non existe a posibilidade de que alguén o teña feito xa. Mais, en xeral, non hai escusa para non investigares e a recompensa pode ser grande. Se as ferramentas de procura habituais de internet non mostraren nada, tenta procurar en http://freshmeat.net/ (unha web sobre noticias de proxectos de software libre, sobre o cal se falará máis tarde), en http://www.sourceforge.net/ e no directorio de proxectos de software libre da Free Software Foundation http://directory.fsf.org/.

Mesmo se non atopares exactamente o que estás a procurar, podes atopar algo tan semellante que ten máis sentido unirse a ese proxecto e engadir a funcionalidade desexada que comezares un novo pola túa conta.

Comeza co que tiveres

Investigaches sen atopares nada que encaixe coas túas necesidades e decidiches comezar un novo proxecto.

E agora que?

O máis complexo sobre lanzar un proxecto de software libre é transformar unha visión privada nunha pública. Ti ou a túa organización podedes coñecer perfectamente o que desexades, mais expresar ese obxectivo dunha maneira comprensible ao mundo é complexo. Porén, é esencial que tomes tempo para facelo. Ti e os outros fundadores debedes decidir a temática real do proxecto - é dicir, decidir as súas limitacións, o que vai e o que *non* vai facer- así como redixirdes unha declaración de intencións. Esta parte non acostuma a ser moi complexa e algunhas veces pode revelar asuncións implícitas e mesmo desconformidades sobre a propia natureza do proxecto, o que é positivo: é mellor resolvelas agora que máis tarde. O seguinte paso é empaquetardes o proxecto para o público, o que é un traballo pesado.

O que o torna tan laborioso é que principalmente consiste en organizar e documentar o que xa todo o mundo sabe - "todo o mundo", é dicir, todos aqueles que teñen participado no proxecto ata agora. Entón, para a xente que fai o traballo non hai beneficio directo por realizar esta tarefa. Eles non precisan un ficheiro chamado *README* que resuma o proxecto, nin un documento de deseño ou un manual de usuario. Non precisan unha árbore de código meticulosamente ordeada conforme a estándares de distribución de código fonte informais mais amplamente difundidos. Sexa como for a maneira en que o código fonte estiver organizado, estará ben para eles, xa que están afeitos a el, e se o código funciona, saben como usalo. Nin sequera importa se as asuncións fundamentais da arquitectura do proxecto están sen documentar; tamén están familiarizados con iso.

Por outra banda, os neófitos precisan esas cousas. Afortunadamente, non as precisan todas á vez. Non é necesario proporcionar todos os recursos posibles antes de facer un proxecto público. Talvez nun mundo perfecto, cada novo proxecto de software libre comezaría a súa vida cun exhaustivo documento de deseño, un manual de usuario completo (con marcas especiais para funcionalidades ideadas mas aínda sen implementar), código bonito e portable, capaz de ser executado en calquera plataforma, etc. En verdade, preocuparse de todos eses flocos consumiría demasiado tempo e, de calquera forma, é traballo que un pode confiar en que fagan os voluntarios unha vez o proxecto estiver en marcha.

Porén, o que *si* é necesario, é investirmos suficiente esforzo na presentación do proxecto, de modo que os que acaban de chegar poidan superar a barreira inicial ao non estaren familiarizados co proxecto. Considérao como o primeiro paso dun proceso de arranque encamiñado a diminuír a enerxía de activación do proxecto. A este limiar soe chamárselle *enerxía de hacktivación*: a cantidade de enerxía coa que debe contribuír un que acaba de chegar antes de recibir algo en troca. Canto menor sexa o

punto de hacktivación, mellor. A primeira tarefa é baixar este limiar ata un nivel que anime a xente a participar.

Cada unha das seguintes subseccións describe un aspecto importante á hora de comezar un novo proxecto. Son presentadas aproximadamente na orde na que un novo visitante as atoparía, aínda que a orde en que sexan implementadas pode ser diferente. Poden ser entendidas como unha lista de tarefas. Cando se inicia un proxecto, é preciso revisar a lista e asegurarse de que cada unha das partes foron tidas en conta, ou que as potenciais consecuencias de deixar algunha sen realizar son asumibles.

Escolle un bo nome

Ponte no lugar de alguén que acaba de atopar o proxecto, talvez porque se atopou casualmente con el mentres procuraba software para resolver algún problema. A primeira cousa que vai ver é o nome.

Un bo nome vai tornar o proxecto nun suceso de maneira automática, e un mal nome non o vai condenar ao fracaso - ben, un nome *verdadeiramente* malo podería facelo, mais asumiremos que ninguén trata de sabotar o seu proxecto. Porén, un mal nome pode atrasar a adopción, ben porque a xente non o toma en serio, ben porque simplemente lle custe lembralo.

Un bo nome:

- Dá certa idea do que fai o proxecto, ou está polo menos relacionado dunha maneira obvia, tal que, se se ten oído o nome e se sabe o que fai o proxecto, o nome vai ser lembrado rapidamente.

- É sinxelo de lembrar. Non se pode obviar o feito de que o inglés se converteu no idioma franco da internet: "sinxelo de lembrar" quere dicir "sinxelo de lembrar para alguén que saiba inglés". Os nomes que son xogos de palabras dependentes da pronunciación do falante, por exemplo, serán opacos para moitos lectores non nativos. Se o xogo de palabras é particularmente rechamante e memorable, pode pagar a pena; unicamente é preciso ter en mente que moita xente que le o nome non o vai percibir da mesma maneira que un nativo.

- Non ten o mesmo nome que outro proxecto existente e non infrinxe ningunha marca comercial. É unha cuestión de educación e de cumprir a lei. Non se desexa crear confusión a respecto da marca. Xa é suficientemente complexo manterse ao día de todo o que está dispoñible na rede, sen necesidade de que proxectos distintos teñan o mesmo nome.

  Os recursos mencionados anteriormente (ver sección “Antes de nada, bota unha ollada ao redor”) son útiles para saber se outro proxecto xa posúe o nome no que se está pensando. É posible consultar ferramentas de procura gratuítas de marcas rexistradas en http://www.nameprotect.org/ e http://www.uspto.gov/.

- Se for posible,que estea dispoñible como un nome de dominio nos dominios de primeiro nivel *.com*, *.net* e *.org*. Probablemente se debería escoller o *.org* para promocionalo como o sitio principal do proxecto; os outros dous deberían encamiñar a este, son útiles unicamente para evitar que terceiras partes creen unha confusión da marca sobre o nome do proxecto. Mesmo se se tiver a intención de aloxar o proxecto nalgún outro lugar (ver “Aloxamento preconfigurado”), é posible rexistrar os dominios e encamiñalos ao lugar onde o proxecto está aloxado. É de utilidade para os usuarios ter que recordar unicamente unha URL sinxela.

Ten unha clara declaración de intencións

Unha vez que se atopa a web do proxecto, o seguinte que a xente fai é procurar unha descrición rápida, unha declaración de intencións para poderen decidir (en menos de 30 segundos) se están interesados ou non en saber máis. Esta debe ser colocada nun lugar prioritario da páxina principal, preferiblemente xusto debaixo do nome do proxecto.

A declaración de intencións debe ser concreta, restritiva e, sobre todo, curta. Un bo exemplo é a de http://www.openoffice.org:

> *Crear, en comunidade, a ferramenta ofimática líder a nivel internacional, que funcione nas principais plataformas proporcionando acceso a toda funcionalidade e datos a través de API's baseadas en compoñentes abertos e un formato de ficheiros baseado en XML.*

En poucas palabras, tocaron os puntos relevantes, explicando a misión tendo en conta os coñecementos previos do lector. Ao diciren "*en comunidade*", sinalan que ningunha corporación dominará o desenvolvemento; "*internacional*" significa que o programa estará dispoñible en diversas linguas e codificacións; "*nas principais plataformas*" quere dicir que funcionará en UNIX, Macintosh e Windows. O resto comunica que as interfaces abertas e os formatos de arquivo facilmente lexibles son un aspecto importante dos obxectivos. De boas a primeiras non din que están tratando de ser unha alternativa libre a Microsoft Office, mais a maioría da xente pode ler entre liñas. Aínda que esta declaración de intencións pode parecer ampla a primeira vista, a verdade é que está bastante circunscrita: as palabras "*ferramenta ofimática*" indican algo moi concreto a todos aqueles familiarizados con este tipo de programas. Asumir os coñecementos previos do lector (neste caso probablemente de MS Office) permite obter unha declaración de intencións concisa.

A natureza da declaración de intencións depende parcialmente de quen a escriba, non só do programa que intenta describir. Por exemplo, ten sentido que OpenOffice.org empregue as palabras "*en comunidade*" xa que o proxecto foi iniciado, e aínda é fortemente patrocinado, por Sun Microsystems. Incluíndo esas palabras, Sun amosa unha certa sensibilidade cara a aqueles que poden pensar que tentará controlar o proceso de desenvolvemento. Con esta clase de cousas, amosando preocupación por problemas *potenciais*, avánzase moito no camiño de evitar o problema por completo. Por outra banda, os proxectos que non están patrocinados por unha única corporación probablemente non necesiten tal linguaxe; despois de todo, o desenvolvemento feito pola comunidade é a norma, co cal non existiría razón algunha para listalo como parte dos obxectivos.

Declara que o proxecto é libre

Aqueles que seguiren interesados logo de ler a declaración de intencións, desexarán coñecer máis detalles, talvez a documentación de usuario ou a de desenvolvedor, mesmo poden desexar descarregalo. Mais antes de todo iso, necesitarán saber o que é software libre.

A páxina principal debe deixar claro, sen ambigüidades, que o proxecto é software libre. Aínda que poida parecer obvio, é sorprendente cantos proxectos esquecen este aspecto. Existen Webs de proxectos de software libre que non só non publican a licenza particular que usan, senón que nin sequera indican que o proxecto é libre. Algunhas veces, esta información crucial é relegada á páxina de descargas, á de desenvolvedores ou a algún outro lugar que require un click para chegar a el. En casos extremos, a licenza non se indica en ningún lugar do sitio web - a única maneira de atopala é descarregar o programa e ollar dentro.

Non cometas este erro. Tal omisión pode provocar a perda de moitos usuarios e desenvolvedores potenciais. Indícao desde o principio, xusto debaixo da declaración de intencións, que o proxecto é software libre, mostrando a licenza particular escollida. Pódese ver unha guía rápida para elixir licenza na sección "Elexir unha licenza e aplicala" máis adiante neste mesmo capítulo, e os aspectos concretos sobre o licenciamento son discutidos con detalle no capítulo 9, "Licenzas, dereitos de autor e patentes".

Chegados a este punto, un hipotético visitante da web, decidiu - probablemente nun minuto ou menos - que está interesado en investir outros cinco minutos informándose sobre o proxecto. As seguintes seccións describen o que debería atopar nos seguintes 5 minutos.

Lista de funcionalidades e requerementos

Debería existir unha breve lista das funcionalidades que o programa soporta (se algo aínda non estiver completo, podes listalo, mais poñendo "*planeado*" ou "*en progreso*" cerca da funcionalidade), así como o entorno necesario para que o programa funcione. Pensa na lista de funcionalidades/requirimentos como o que lle darías a alguén se pedir un resumo do proxecto. Habitualmente é unha expansión da declaración de intencións. Por exemplo, esta podería dicir:

> *Crear un indexador de texto completo e motor de procura cunha API usable por programadores, que permita fornecer servizos de procura para grandes coleccións de ficheiros.*

A lista de funcionalidades e requirimentos daría os detalles, restrinxindo o ámbito da declaración de obxectivos:

> *Funcionalidades*:
> - Procura en texto plano, HTML, e XML.
> - Procura de palabras ou frases.
> - (planeada) Coincidencias aproximadas (Fuzzy matching).
> - (planeada) Actualización incremental de índices.
> - (planeada) Indexación de sitios Web remotos.
>
> *Requerementos*:
> - Python 2.2 ou superior.
> - Espazo en disco suficiente para almacenar os índices (aproximadamente 2x o tamaño orixinal do conxunto de datos).

Con esta información, os lectores poden ter unha idea de se este programa é o que necesitan e valorar a súa implicación no proxecto como desenvolvedores.

Estado do desenvolvemento

Á xente gústalle saber como está evoluíndo un proxecto. En proxectos novos, desexan saber a distancia existente entre as palabras e a realidade. En proxectos maduros, desexan coñecer o grao de actividade do mantemento, con que frecuencia se publican novas versións, a receptividade do proxecto para recibir informes de erros, etc

Para responder a estas cuestións, deberíase crear unha sección na Web para desenvolvedores, listando os obxectivos a curto prazo do proxecto e as necesidades (por exemplo, se se están a procurar

desenvolvedores expertos nun tema en particular). A páxina pode tamén mostrar as anteriores publicacións do programa con listas de funcionalidades, de cara a que os visitantes poidan ter unha idea de como se define "progreso" neste proxecto e con que rapidez se avanza conforme a esa definición.

Non te preocupes por se dás a sensación de non estares preparado, e non caias na tentación de acelerares o grao de desenvolvemento. Todo o mundo sabe que o software evolúe por etapas; non hai que se avergoñar ao dicir "Este é un programa en estado alpha con erros coñecidos. Corre, e funciona polo menos algunhas veces, así que úsao baixo a túa propia responsabilidade". Este tipo de linguaxe non asustará o tipo de desenvolvedor que precisas nese momento. Por outra banda, de cara ao usuario, unha das peores cousas que pode facer un proxecto é atraelo antes de o programa estar pronto para ser empregado. Unha mala reputación a respecto da estabilidade e erros é difícil de eliminar, unha vez adquirida. Ser conservador paga a pena a longo prazo; sempre é mellor que o software sexa *máis* estable do que o usuario espera, que o caso contrario. O mellor boca a boca vén de sorpresas agradables.

Alpha e Beta

O termo *alpha* refírese habitualmente a unha primeira publicación que se pode usar e ten toda a funcionalidade esperada, mais tamén ten bugs coñecidos. A principal razón de publicar programas en alpha é xerar feedback, de maneira que os desenvolvedores saiban onde concentrar o traballo. A seguinte etapa, *beta*, indica que o software ten todos os erros serios resoltos mais non está suficientemente probado para ser publicado como estable. O propósito dos programas beta é convertérense nunha publicación estable, asumindo que non se atoparon erros, ou achegaren feedback suficiente aos desenvolvedores para que se poida obter a estable rapidamente. A diferenza entre alpha e beta é unha cuestión de xuízo.

Descargas

O código fonte do programa debería poderse descarregar en formatos estándar. Cando se lanza un proxecto, os pacotes binarios (executables) non son precisos a menos que o proxecto teña requirimentos ou dependencias moi complexas de compilación, polo que simplemente facelo funcionar sería complexo para a maioría da xente. (Mais se este for o caso, o proxecto vaino ter complicado para atraer desenvolvedores!)

O mecanismo de distribución debería ser o máis estándar e sinxelo posible. Se estiveres tentando erradicar un virus, non distribuirías o medicamento de tal modo que requirise unha agulla especial para ser administrado. De igual modo, o software debería axeitarse a métodos de instalación e compilación estándares; canto máis se desviar do estándar, máis usuarios e desenvolvedores se van dar por vencidos e van abandonar o proxecto.

Aínda que pareza obvio, moitos proxectos non se molestan en estandarizar o proceso de instalación ata moi tarde, dicíndose a si mesmos que teñen tempo a facelo máis adiante: "*Enfrontarémonos a esas cuestións cando o código estea case pronto.*" Mais non se decatan de que deixando para o final os aburridos procesos de construción e instalación do código están a atrasar a finalización do proxecto- xa que desalentan a desenvolvedores que de outra maneira poderían ter contribuído no proxecto. O peor de todo é que *non saben* que están a perder desenvolvedores, xa que o proceso é unha acumulación de non-eventos: alguén visita a web, descarrega o proxecto, tenta construír o programa, non o consegue, abandona e vaise. Quen saberá algunha vez o que aconteceu, excepto a mesma persoa? Ninguén do proxecto vaise decatar de que o interese de alguén foi desaproveitado.

As tarefas aburridas cun alto beneficio deberían ser feitas nas etapas iniciais do proxecto. E baixar a barreira de entrada utilizando un bo sistema de empacotamento consegue un grande beneficio.

Cando se publica un pacote descarregable, é vital asignar un número único de versión á publicación, para que a xente poida comparar entre 2 publicacións diferentes e saber cal substitúe cal. Pódese atopar información detallada sobre a numeración de versións na sección "Numeración de versións", e os detalles de estandarización dos procesos de construción e instalación son tratados na sección "Empacotamento", ambos no capítulo 7 "Empacotamento, liberación e dia a dia no desenvolvemento".

Control de versións e acceso ao Sistema de notificación de erros (*bug tracking system*)

Descarregar pacotes co código fonte é suficiente para aqueles que unicamente desexan instalar e usar o programa, mais non o é para os que desexan depurar o código ou engadir novas funcionalidades. Empacotar o código diariamente pode axudar, mais non é suficiente para unha comunidade de desenvolvedores saudable. A xente precisa acceso en tempo real ao código fonte, e o modo de darllo é usar un sistema de control de versións. A presenza de código fonte baixo control de versión accesible anonimamente é un signo - tanto para usuarios como para desenvolvedores - de que o proxecto está facendo un esforzo por darlle á xente o que precisa para participar. Se non for posible ofrecer control de versións neste momento, indica que se está montando a infraestrutura. Todo o relativo ao control de versións estúdase en detalle no capítulo 3, "Infraestructura técnica".

O mesmo é aplicable para o sistema de notificación de erros do proxecto. A importancia deste sistema non só reside na utilidade para os desenvolvedores, senón o que isto significa para a xente que ve o proxecto desde fóra. Para moita xente, unha base de datos de erros accesible é un dos signos máis importantes de que o proxecto debe ser tomado en serio. Ademais, cantos máis erros rexistrados, mellor. Aínda que poida parecer pouco intuitivo, é preciso recordar que o número de erros almacenado depende de 3 claves: o número absoluto de erros presentes no programa, o número de usuarios que o usan e a facilidade coa que os usuarios poden rexistrar os erros. Deses 3 factores, os últimos 2 son máis significativos que o primeiro. Calquera aplicación de tamaño e complexidade suficientes ten unha cantidade arbitraria de erros esperando a seren descubertos. A verdadeira cuestión é o ben que o proxecto almacene e priorice eses erros. Un proxecto cunha base de datos de erros grande e ben mantida (é dicir, os erros aténdense sen demoras, os duplicados son unificados, etc) dá mellor impresión que un proxecto sen base de datos ou cunha case inexistente.

Obviamente, se o proxecto acaba de comezar, a base de datos de erros conterá poucos, non hai moito que facer ao respecto. Mais se a páxina do proxecto indica a súa xuventude e se a xente que usa a base de datos pode ver que a maioría dos erros foron notificados recentemente, é posible extrapolar que o proxecto ten un *ratio* de rexistros saudable e os visitantes non se alarmarán polo baixo número total de erros rexistrados.

É preciso sinalar que os sistemas de notificación de erros son habitualmente empregados non só para manter un rexistro de erros no software, senón tamén para indicar peticións de funcionalidades, mudanzas na documentación, tarefas pendentes, etc. Os detalles sobre a xestión dun sistema de notificación de erros son comentados na sección "Xestores de erros" dentro do capítulo 3 "Infraestructura técnica", polo que non se entrará en detalles aquí. Desde o punto de vista da presentación, a importancia radica en *ter* un sistema de notificación e asegurarse de que é visible desde a páxina principal do proxecto.

Canais de comunicación

Habitualmente, os visitantes da web desexan saber como pórse en contacto coa xente implicada no proxecto. Proporciona a dirección das listas de correo, salas de chat, canais de IRC e calquera outro foro onde a comunidade interaxa. Deixa claro que tanto ti como os outros participantes do proxecto estades subscritos a estas listas de correo, para que a xente vexa que existe un modo de enviar feedback aos desenvolvedores. A presenza nas listas de correo non implica a obrigación de responder todas as preguntas nin implementar todas as peticións de funcionalidades. A longo prazo, moitos usuarios non se van cadastrar nos foros, mais gustaralles saber que poderían facelo se algunha vez o precisaren.

En etapas iniciais do proxecto, non hai necesidade de ter foros separados para usuarios e desenvolvedores. É mellor ter a todo o mundo falando xuntos, nunha única "sala". Entre os primeiros participantes do proxecto, a distinción entre usuario e desenvolvedor é confusa; mais se a distinción tivese que ser feita, a relación entre desenvolvedores e usuarios é habitualmente máis alta no comezo. Aínda que non é posible asumir que cada participante nas etapas iniciais sexa un programador que desexa hackear o programa, é posible asumir que están polo menos interesados en seguiren os debates sobre o desenvolvemento e obter unha visión de cara a onde avanza o proxecto.

Xa que este capítulo unicamente versa sobre como iniciar un proxecto, chega con dicir que os foros de comunicacións teñen que existir. Máis adiante, dentro da sección “Xestionando o crecemento” do capítulo 6 “Comunicacións”, examinarase onde e como establecer as maneiras en que poden ser xestionados e moderados, e como separar foros de usuarios de foros de desenvolvedores, cando for preciso, sen crear un abismo infranqueable.

Pautas de desenvolvemento

Se alguén estiver a pensar en contribuír a un proxecto, buscará as pautas de desenvolvemento. Estas pautas non son tanto técnicas como sociais: explican como os desenvolvedores interaxen entre si e cos usuarios ou como se realiza o traballo.

Este tema trátase en detalle na sección “Poñer todo por escrito” no capítulo 4 “Infraestructura social e política”, mais os elementos básicos das pautas de desenvolvemento son:

- ligazóns aos foros onde interaxir con outros desenvolvedores
- instrucións sobre como notificar erros e enviar remendos
- indicacións sobre como o desenvolvemento se realiza habitualmente - é o proxecto unha ditadura benevolente? unha democracia? calquera outra cousa?

O temo "ditadura" non pretende ter sentido pexorativo. É perfectamente válido xestionar un proxecto como se de unha tiranía se tratase, onde un único desenvolvedor teña poder de veto sobre todos as mudanzas. Moitos proxectos con suceso funcionan deste xeito. O importante é que o proxecto sexa consciente diso e o comunique. Unha tiranía que pretenda ser unha democracia desalentará as persoas a participaren; unha tiranía que diga que é tal, funcionará ben sempre que sexa competente e de confianza.

Pódese observar aquí http://svn.collab.net/repos/svn/trunk/www/hacking.html un exemplo de pautas de desenvolvemento, ou http://www.openoffice.org/dev_docs/guidelines.html para pautas máis enfocadas á gobernanza e ao espírito de participación e menos en aspectos técnicos.

O tema de como fornecer unha introdución á aplicación para o programador discutirase máis adiante neste mesmo capítulo na sección “Documentación para desenvolvedores”.

Documentación

A documentación é esencial. É preciso que exista *algo* que a xente poida ler, embora sexa rudimentario e incompleto. Esta tarefa cae de cheo na categoría de "traballo pesado" antes citada e habitualmente é a primeira das áreas nas que un proxecto de software libre fracasa. Ter unha declaración de intencións e unha lista de funcionalidades, escoller unha licenza, resumir o estado do desenvolvemento - son pequenas tarefas que polo xeral se realizan e habitualmente non precisan ser revisadas unha vez feitas. Por outra banda, a documentación é algo que nunca remata, o que pode ser unha das razóns polas que sempre se atrasa ou non se realiza.

O máis insidioso da documentación é que a utilidade para quen o escribe é inversamente proporcional á utilidade para quen o vai ler. A documentación máis importante para usuarios iniciais é o básico: como configurar a aplicación, unha introdución ao seu funcionamento e talvez algunhas indicacións sobre como facer tarefas comúns. E xusto esas son as cousas que os *redactores* da documentación coñecen demasiado ben - polo que pode ser difícil para eles poñerse no punto de vista do lector e listar os pasos que (para o escritor) parecen tan obvios que non paga a pena mencionar.

Non hai unha solución máxica para este problema. Alguén ten que sentarse e escribila para logo presentarlla a un usuario novato e comprobar a calidade. Usa un formato sinxelo, fácil de editar como HTML, texto plano, Texinfo ou algunha variante do XML - algo que sexa facilmente editable. Isto vai facilitar as melloras incrementais tanto por parte dos escritores actuais como dos que se unan máis tarde ao proxecto e desexen traballar sobre a documentación.

Un modo de asegurarse de que se fai a documentación é limitar o seu alcance por adiantado. Desta forma, escribila non parecerá unha tarefa sen fin. Unha boa regra xeral é seguir uns criterios mínimos:

- Indicarlle ao lector claramente o nivel técnico que precisa ter.
- Describir clara e extensivamente como configurar a aplicación e, nalgunha parte ao principio da documentación, indicar como executar algún tipo de proba ou comando que confirme que todo funciona correctamente. A documentación sobre como comezar é ás veces mais importante que a documentación de uso. Canto máis esforzo tiver investido unha persoa en instalar e ter funcionando a aplicación, máis persistente será en descubrir funcionalidades que non estean ben documentadas. A xente que abandona soe facelo no comezo; por iso, son as etapas iniciais, como a instalación, as que precisan máis soporte.
- Dar un exemplo, tipo titorial, sobre como levar a cabo unha tarefa común. Obviamente, cantos máis exemplos mellor, mais se o tempo for limitado, escolle unha e descríbea a fondo. Unha vez unha persoa vexa que a aplicación pode ser utilizada para unha cousa, comezará a explorar que máis se pode facer - e, con sorte, comezará a xerar documentación por el mesmo. O que nos leva ao seguinte punto.
- Indica que áreas da documentación están incompletas. Mostrándolle aos lectores que te preocupas das deficiencias, estáste aliñando co seu punto de vista. A empatía dilles aos lectores que non terán que facer moito ruído para convenceren ao proxecto do que é importante. Esas indicacións non teñen por que ser entendidas como algo que teña que ser feito para unha data determinada, senón que poden verse como tarefas abertas a voluntarios.

O último punto é de maior importancia e pode ser aplicado ao conxunto do proxecto, non unicamente á

documentación. Unha detallada explicación das deficiencias do proxecto é a norma no mundo do software libre. Non esaxeres nos defectos do proxecto, limítate a identificalos escrupulosa e desapaixonadamente cando for necesario (na documentación, na base de datos de erros ou na lista de correo). Ninguén o vai ver como derrotismo nin como un compromiso para resolvelos no futuro, a menos que se faga esa promesa explícita. Xa que calquera que use a aplicación vai descubrir as deficiencias por si mesmos, é mellor que estean psicoloxicamente preparados - á vez que o proxecto mostrará ter un sólido coñecemento de como está progresando.

Manter unha FAQ (preguntas frecuentes)

Unha *FAQ* (un documento de preguntas frecuentes) pode ser un dos mellores investimentos que un proxecto pode facer en termos de documentación. As *FAQs* están altamente enfocadas ás preguntas que os usuarios e desenvolvedores preguntan - e que pode ser o contrario ás preguntas que ti *esperabas* que fixeran - unha *FAQ* detallada soe dar a quen procura a resposta exacta que estaba a esperar. A *FAQ* é habitualmente o primeiro lugar onde os usuarios consultan cando atopan un problema, mesmo antes do manual oficial, e é o documento do proxecto que probablemente máis se vai enlazar desde outros lugares.

Infelizmente, a *FAQ* non se pode realizar ao principio do proxecto. As boas *FAQ* non están escritas, medran. Son documentos reactivos, que evolúen a través do tempo en resposta ao uso diario da aplicación. Xa que é imposible anticipar correctamente as preguntas que van ser realizadas, é imposible sentarse e redixir unha *FAQ* útil desde cero.

Polo tanto, non esbanxes o tempo facéndoo. Porén, pode ser útil crear un molde inicial para as *FAQ*, de maneira que exista un lugar obvio onde as persoas contribúan con preguntas e respostas cando o proxecto comece a andar. Nesta etapa, o máis importante non é que estean completas, senón a conveniencia: se é sinxelo engadir contido á *FAQ*, a xente vaino facer. O mantemento correcto da *FAQ* non é un problema trivial e será comentado máis a fondo na sección “Xestor das *FAQ*” do capítulo 8 “Xestión de voluntarios”.

Dispoñibilidade da documentación

A documentación debe estar dispoñible desde 2 lugares: en liña (directamente desde a páxina web) *e* na distribución descarregable da aplicación (ver capítulo 7 “Empacotamento, publicación e desenvolvemento diario”). É preciso que estea en liña, de modo navegable, xa que a xente habitualmente le a documentación *antes* de descarregar o programa por primeira vez, para decidir se paga a pena facelo. Mais tamén debería ir acompañando o programa, baixo o principio de que a aplicación debería facer accesible todo aquilo que alguén necesitar para usar o programa.

Para a documentación en liña, é preciso asegurarase de que hai unha ligazón que mostre a documentación nunha única páxina HTML (pon unha nota como "monolítico ou "todo en un" ou "un único ficheiro" para que a xente saiba que pode levar un tempo que o documento carregue). Isto é útil porque a xente habitualmente desexa procurar unha palabra ou frase específica en todo o documento. Xeralmente, saben o que están a procurar, só que non recordan en que sección está. Para esta xente, nada será máis frustrante que atopar unha páxina HTML para a tabela de contidos, outra páxina para a introdución, outra para as instrucións de instalación, etc. Cando as páxinas se dividen dese modo, a función de procura do navegador é inútil. O estilo de páxinas separadas por sección é útil para aqueles que saben que sección precisan ou desexan ler a documentación enteira. Mais esta *non* é a maneira máis habitual en que se accede á documentación. É moito máis habitual que alguén que xa coñece o programa vaia procurar unha palabra ou frase específica. Non darlles un único documento sobre o que

facer as procuras tornaríalles a vida un pouco máis dura.

Documentación para desenvolvedores

A documentación para desenvolvedores está escrita para axudar os programadores a entenderen o código, para que poidan reparalo ou estendelo con máis funcionalidades. Isto é diferente das *pautas de desenvolvemento* comentadas anteriormente, as cales son máis sociais que técnicas. Estas dinlle aos programadores como interaxir uns cos outros; a documentación para programadores dilles como interaxir co código. Ambas son habitualmente empacotadas xuntas nun único documento por comodidade (como co http://svn.collab.net/repos/svn/trunk/www/hacking.html exemplo mostrado anteriormente), mais non ten por que ser así.

Aínda que a documentación para programadores pode ser moi útil, non hai razón para atrasar unha publicación por non posuíla. Para empezar, é suficiente que os autores orixinais estean dispoñibles (e dispostos) a contestaren cuestións sobre o código. De feito, ter que responder ás mesmas preguntas unha e outra vez é unha gran motivación para escribir documentación. Mais antes de que sexa escrita, os contribuidores dispostos atoparán o modo de colaborar. O que conduce á xente a investiren o seu tempo na aprendizaxe do código é que este faga algo útil para eles. Se a xente ten fe, tomaralles o seu tempo; se non teñen fe, ningunha documentación fará que fiquen.

Polo tanto, se unicamente tes tempo para escribires documentación para unha única audiencia, escríbea para os usuarios. Toda documentación de usuario é á súa vez documentación para programadores; calquera programador que desexe usar unha parte do software terá que familiarizarse co seu uso. Máis tarde, cando os programadores pregunten as mesmas cuestións unha e outra vez, tómate o teu tempo para escribires documentación específica para eles.

Algúns proxectos usan wikis para a súa documentación inicial e mesmo como primeira fonte de documentación. Pola miña experiencia, isto unicamente funciona se o wiki é editado activamente por unha pouca xente que decide como a documentación ten que ser organizada e que clase de "voz" debe ter. Ver sección “Wikis" no capítulo 3 “Infraestructura técnica” para máis información.

Exemplos de saídas e capturas de pantalla

Se o proxecto envolve o uso dunha interface de usuario gráfica ou se produce saídas gráficas ou distintivas dalgunha outra forma, publica algúns exemplos na páxina web do proxecto. No caso de interfaces, isto quere dicir capturas de pantalla; para resultados de saída, poden ser capturas de pantallas ou ficheiros. Ambas serán útiles para o usuario: unha simple captura de pantalla pode ser máis convincente que parágrafos de texto descritivo e barullo nas listas de correo, xa que a captura de pantalla é unha proba irrefutable de que *a aplicación funciona*. Pode ter erros, ser complicada de instalar ou ter documentación incompleta, mais esa captura de pantalla é a proba de que se un fai o suficiente esforzo, é posible facela funcionar.

Capturas de pantalla

Debido a que as capturas de pantalla poden ser desalentadoras ata que fas unhas poucas, aquí tes unhas instrucións básicas para facelas. En Gimp (http://www.gimp.org/), abre o menú *Arquivo-> Tomar > Captura de pantalla*, escoller *Unha única xanela* ou *Xanela completa*, preme en *Capturar*. A seguinte vez que premas unha xanela ou pantalla vai ser capturada como imaxe no Gimp. Recorta e redimensiona a imaxe tanto como for necesario, usando as instrucións que podes atopar en http://www.gimp.org/tutorials/Lite_Quickies/#crop.

Hai moitas máis cousas que se poden poñer na páxina web do proxecto, se tes tempo ou se por algunha razón son especialmente apropiadas: unha páxina de noticias, outra sobre a historia do proxecto, ligazóns relacionadas, unha ferramenta de procura do sitio web, ligazóns para facer doazóns, etc. Ningunha desas son necesarias ao comezo, mais tenas presentes para o futuro.

Aloxamento preconfigurado

Existen algúns lugares que facilitan aloxamento gratuíto e infraestrutura para proxectos de software libre: unha páxina web, sistema de control de versións, xestor de erros, área de descargas, salas de chat, backups regulares, etc. Os detalles varían entre sitio e sitio, mais os mesmos servizos básicos son ofertados en todos eles. Usando un destes servizos, obtense moito por nada; porén, cédese control sobre a experiencia do usuario. O servizo de aloxamento decide sobre que aplicacións funciona e pode controlar ou influír no "*look&feel*" das páxinas dos proxectos.

Ver sección “Aloxamento preconfigurado” no capítulo 3 “Infraestrcutura técnica” para obter máis detalles sobre as vantaxes e desvantaxes de usar aloxamento enlatado así como para obter unha lista de sitios que o ofrecen.

Escoller unha licenza e aplicala

Esta sección pretende ser unha rápida guía sobre como escoller licenzas. Le o capítulo 9 “Licenzas, dereitos de autor e patentes” para comprenderes as implicacións legais das diferentes licenzas, e como a licenza que escollas podo afectar a como a xente pode integrar o teu programa con outros.

Existen moitas e moi boas licenzas de software libre onde escoller. A maioría delas non precisan ser consideradas aquí, xa que foron escritas para satisfacer as necesidades legais dalgunha corporación ou persoa específica, e non serían apropiadas para o teu proxecto. Cinguirémonos ás máis comunmente usadas; na maioría dos casos, será suficiente con escoller unha delas.

As licenzas "Fai o que queiras"

Se che parece ben que o código do teu proxecto poida ser integrado en programas propietarios, usa una licenza estilo *MIT/X*. É a máis sinxela de moitas licenzas que unicamente declaran un *copyright* nominal (sen restrinxir a copia) e especifica que o código ven sen garantía. Ver a sección “A licenza MIT/X Window” no capítulo 9 para os detalles.

A licenza GPL

Se non desexas que o teu código poida ser usado en programas propietarios, usa a GNU General Public License (http://www.gnu.org/licenses/gpl.html). A GPL é probablemente a licenza máis coñecida e estendida no mundo a día de hoxe. Isto en si mesmo é unha vantaxe, xa que moitos usuarios potenciais e contribuidores estarán familiarizados con ela, polo que non terán que investir tempo extra en comprender que licenza usas. Ver a sección “Licenza GNU General Public License” no capítulo 9 para máis detalles.

Se os usuarios interaxiren co teu código a través da rede - isto é, se o programa é parte dun servizo aloxado - podes considerar o uso da *GNU Affero GPL*. Ver a sección “Licenza GNU Affero GPL” no capítulo 9 para máis información.

Como aplicar unha licenza ao teu código

Unha vez escolleres unha licenza, deberás indicalo na páxina principal do proxecto. Non é preciso incluíres o texto completo da licenza, unicamente dar o seu nome e pór unha ligazón ao texto completo.

Isto dille ao público baixo que licenza será publicado o software, máis non é suficiente para fins legais. Para iso, o software en si mesmo debe conter a licenza. A maneira habitual de facelo é poñendo o texto da licenza completa nun ficheiro chamado *COPYING* ou *LICENSE*, así como poñer un aviso no cabezallo de cada ficheiro de código fonte, indicando a data do "*copyright*", mantedor e licenza usada e indicando onde atopar o texto completo.

Existen moitas variacións deste padrón, así que unicamente imos ver un deles. A GNU GPL di que se debe poñer un aviso como o seguinte no cabezallo de cada ficheiro de código fonte:

```
Copyright (C) <ano> <nome do autor>

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program.  If not, see <http://www.gnu.org/licenses/>
```

O texto non indica especificamente que a copia da licenza se pode atopar no ficheiro *COPYING*, mais é alí onde habitualmente se pon. É posible mudar o aviso anterior para indicalo explicitamente. Este molde tamén dá un enderezo postal onde pedir unha copia da licenza. Outro método común é dar unha ligazón a unha páxina web que a conteña. Usa o teu xuízo para apuntares a onde considerares que se mantén a copia máis permanente da licenza, que pode ser algún lugar da páxina web do teu proxecto. En xeral, o aviso que poñas en cada ficheiro de código fonte non ter por que ser igual ao anterior, sempre e cando comece co mesmo aviso de data e mantedor do copyright, indique o nome da licenza e deixe claro onde obter a licenza completa.

Axustando o ton

Ata agora falouse de tarefas unitarias que se fan durante a etapa de inicio do proxecto: escoller unha licenza, organizar a páxina web, etc. Mais os aspectos máis importantes dun proxecto son dinámicos. Elixir un nome para unha lista de correo é sinxelo, asegurarse de que as conversas da lista se manteñen nas temáticas e son produtivas é outra cousa totalmente diferente. Se o proxecto se fai público logo de anos de desenvolvemento pechado interno, os procesos de desenvolvemento mudarán e será necesario preparar os desenvolvedores para esa mudanza.

Os primeiros pasos son os máis duros, xa que aínda non se sentaron precedentes e expectativas para condutas futuras. A estabilidade dun proxecto non vén de políticas formais, senón dunha intelixencia compartida e colectiva que se desenvolve co tempo. Habitualmente tamén existen regras escritas, mais en esencia, soen ser unha destilación do intanxible antes que acordos que realmente guían o proxecto. As políticas que se escriben non definen a cultura do proxecto, senón que a describen, e só aproximadamente.

Hai algunhas razóns polas que as cousas funcionan así. O crecemento e a alta taxa de rotación entre voluntarios non son tan nocivos como para precisar da acumulación de normas sociais, como un puidera pensar a primeira vista. Mentres a mudanza non teña lugar *demasiado* rápido, os novos teñen tempo de aprenderen como se fan as cousas; unha vez aprenderen, axudarán a reforzar os valores por si mesmos. Considera como as nanas sobreviven polos séculos dos séculos. Os nenos de hoxe cantan aproximadamente as mesmas rimas que as que cantaron outros nenos centos de anos atrás. Aínda é máis sorprendente se temos en conta que non hai nenos que estean agora vivos que tamén fosen nenos daquela. Os cativos máis pequenos oen as cancións que cantan os maiores, e cando sexan máis vellos cantaranas diante doutros rapaces máis novos ca eles. Os cativos non están comprometidos cun programa consciente de transmisión, desde logo. As razóns polas que as cancións sobreviven é, nada máis e nada menos, porque son transmitidas regular e repetidamente. A escala de tempo dos proxectos de software libre pode non ser medida en séculos (aínda non o sabemos) mais as dinámicas de transmisión de coñecemento son as mesmas. Porén, a taxa de rotación é máis rápida e debe ser compensada por un esforzo máis activo e deliberado de transmisión de prácticas e coñecemento compartido.

Este esforzo válese do feito de que a xente, en xeral, chega ao proxecto agardando e procurando normas sociais. Así é a raza humana. En calquera grupo unificado por un comportamento común, a nova xente que se une quere reafirmar comportamentos que os convertan en parte do grupo. O obxectivo de poñer precedentes ao principio é facer que eses comportamentos internos do grupo sirvan o proxecto; cando sexan establecidos, serán autoperpetuados por moito tempo.

A continuación hai algúns exemplos de cousas que se poden facer para sentar bos precedentes. Non pretenden ser unha lista exhaustiva, senón ser útiles como ilustracións de que investir en comportamentos e normas compartidas no inicio pode axudar moito no futuro do proxecto. Fisicamente, cada desenvolvedor pode estar traballando só nun cuarto, mais é posible facer moito para que se *sintan* como se estivesen todos no mesmo cuarto. Canto máis se sintan desta maneira, máis desexarán participar no proxecto. Escollín os seguintes exemplos particulares xa que foron os que se propuxeron no proxecto Subversion (http://subversion.tigris.irg), no cal participei desde os seus inicios. Mais non son únicos ao proxecto Subversion; situacións como as seguintes ocorren na maioría dos proxectos de software libre, e deberían ser vistas como oportunidades para comezar con bo pé.

Evita os debates privados

Mesmo despois de que o proxecto sexa público, ti e os demais fundadores habitualmente desexaredes xerir en privado temas difíciles. Isto é especialmente certo nos primeiros momentos do proxecto, cando hai moitas decisións importantes que tomar, e poucos voluntarios cualificados para tomalas. Todas as desvantaxes dos debates en listas de correo públicas vanse erguer ante ti: os atrasos inherentes a debates por correo electrónico, a necesidade de deixar tempo suficiente para que se forme o consenso, discutir con voluntarios inxenuos que pensan que comprenden todo mais en realidade non (todos os proxectos os teñen; algunhas veces se converten nos mellores contribuidores do proxecto, outras veces simplemente continúan na súa inxenuidade), a persoa que non comprende que só desexas resolver o problema X cando este é obviamente unha pequena parte do maior Y, etc. A tentación de tomar decisións a porta pechada e presentalas como *faits accomplis*, ou como as firmes recomendacións dun unido e influente bloco de votantes, será moi grande.

Non o fagas.

Embora o lentos e difíciles que os debates públicos poidan chegan ser, case sempre son preferibles a

longo prazo. Tomar as decisións en privado é como pulverizar repelente anti-contribuidores sobre o proxecto. Ningún voluntario serio permanecerá moito tempo nun entorno onde un consello secreto toma todas as grandes decisións. Ademais, os debates públicos teñen efectos positivos que permanecerán alén de calquera cuestión tecnolóxica efémera:

- O debate contribuirá ao adestramento e educación dos novos desenvolvedores. Non é posible saber cantos ollos están a seguir a conversa; e embora a maioría da xente non participe, moitos poden estar observando en silencio, obtendo información sobre o proxecto.

- O debate formarate na arte de explicar cuestións técnicas a xente que non está tan familiarizada co programa coma ti. Esta é unha capacidade que require práctica e non podes practicala con xente que xa sabe o que ti sabes.

- O debate e as súas conclusións estarán dispoñibles en arquivos públicos, evitando que futuros debates reincidan sobre os mesmos pasos. Ver sección "Uso intensido de arquivos" no capítulo 6 "Comunicacións".

Finalmente, existe a posibilidade de que alguén na lista faga unha contribución real á conversa, propoñendo unha idea que nunca se che ocorrería. É difícil calcular a probabilidade de que ocorra, dependerá da complexidade do código e do grao de especialización requirido. Mais se se me permite una pequena anécdota, apostaría a que isto é máis probable do que intuitivamente poida parecer. No proxecto Subversion, os fundadores críamos que nos enfrontabamos a un complexo conxunto de problemas, sobre os que tiñamos pensado moito durante algúns meses e -francamente- dubidabamos de que alguén na lista de correo creada recentemente contribuíse con algunha idea nova ao debate. Así, tomamos o camiño máis sinxelo e comezamos a compartir algunhas ideas en privado, ata que un observador do proxecto[3] decatouse do que ocorría e pediu que o debate se movese á lista pública. Así o fixemos, e ficamos abraiados polo número de comentarios e suxestións brillantes que súpeto xurdiron. En moitos casos, a xente deu ideas que nunca se nos terían ocorrido. Isto demostrou que había xente *moi* intelixente nesa lista de correo; unicamente estaban a agardar o momento adecuado para participar. Se ben é certo que os debates foron máis longos do que serían se fosen mantidos en privado, sen dúbida foron máis produtivos en aberto e o tempo extra pagou a pena.

Sen deixarnos levar por xeneralizacións como "o grupo sempre é máis intelixente que o individuo" (todos temos coñecido bastantes grupos como para saber que isto non é certo), hai que recoñecer que en certas actividades os grupos son excelentes. A revisión masiva entre pares é unha delas; xerar un número de ideas grande en pouco tempo é outra. A calidade das ideas dependerá da calidade dos pensadores que están detrás delas, desde logo, mais non saberás que clase de pensadores roldan o proxecto ata que os estimules cun problema desafiante.

Naturalmente, hai certos debates que deben ser mantidos en privado; máis adiante no libro se poderán ver exemplos. Mais a política debería ser sempre: *se non hai razón para que sexa privado, debe ser público*.

Isto non se fai só. Non é suficiente con que te asegures de que todas as túas mensaxes van á lista pública. Debes tamén facer que os demais eviten as conversas privadas. Se alguén tenta comezar un

3 Aínda non o puxemos na sección de créditos, mais por practicar o que máis tarde predicarei: o nome do observador é Brian Behlendorf, e foi el quen sinalou a necesidade de manter todos os debates en público a menos que existise algunha razón concreta para facelo en privado.

debate privado, e non hai razón para que así sexa, é a túa obriga abrir o meta-debate de inmediato. Nin sequera comentes no tema orixinal ata que a conversa se teña movido a un foro público ou descubras que a privacidade do caso é realmente precisa. Se fixeres isto de modo coherente, a xente rapidamente seguirá esta política e comezará a usar os foros públicos por defecto.

Tolerancia cero coa mala educación

Desde o primeiro momento en que o proxecto sexa público, débese manter tolerancia cero cos comportamentos maleducados ou insultantes nos foros de debate. Tolerancia cero non significa usar medidas técnicas per se: non é preciso eliminar a xente da lista de correo cando entraren en conflito con outro participante ou quitarlle o dereito de enviar commits debido a que fixo comentarios despectivos (en teoría, podes ter que levar a cabo esas accións, mais unicamente cando todos os outros intentos teñan errado - o que, por definición, non soe ser o caso ao principio do proxecto). Tolerancia cero simplemente significa que nunca deixes que os malos comportamentos pasen inadvertidos. Por exemplo, cando alguén publica un comentario técnico mesturado cun ataque *ad hominen* sobre outro desenvolvedor do proxecto, é necesario que primeiro se trate o ataque *ad hominen* como unha cuestión independente e unicamente despois diso debater o contido técnico.

Infelizmente, é moi sinxelo e demasiado típico, que debates construtivos se convertan en longos debates destrutivos "*flames*"). A xente di cousas por email que nunca dirían cara a cara. Os temas de debate amplificarán este efecto: en cuestións técnicas, a xente habitualmente pensa que hai unha resposta única para a maioría das preguntas e que estar en desacordo só pode ser signo de ignorancia ou estupidez. Hai unha distancia moi pequena entre chamar estúpida unha proposta técnica a chamar estúpida a persoa que a propón. De feito, habitualmente é complexo dicir onde remata o debate técnico e comeza o ataque persoal, o que é unha das razóns polas que as respostas drásticas ou os castigos non son unha boa idea. En lugar diso, cando algo así acontecer, publica unha mensaxe que saliente a importancia de manter o debate amigable, sen acusar ninguén en particular de ser problemático. Desafortunadamente, esas mensaxes soen ser vistas como as de unha profesora de xardín de infancia dando aulas de bo comportamento:

> *Primeiramente, reduce os comentarios que sexan persoais; como por exemplo, dicir que o deseño de J para a camada de seguridade é "inxenuo e ignora os principios básicos da seguridade dos computadores". Pode ser certo ou non, mais en calquera caso non é xeito de manter o debate. J fixo a súa proposta de boa fe. Se ten deficiencias, que sexan sinaladas e resolvámolas ou obteñamos un novo deseño. Seguramente M non quixo insultar J persoalmente, mais a frase foi desafortunada e aquí trátase de manter os debates construtivos.*
>
> *Agora, sobre a proposta. Creo que M estaba no certo cando dixo que ...*

Embora esas respostas poidan parecer torpes, teñen un notable efecto. Se chamas a atención sobre os malos comportamentos mais non esixes unha desculpa pola parte que ofende, facilitas que a xente se calme e a seguinte vez amose un mellor comportamento - e farano. Un dos segredos para facer isto con éxito consiste en nunca deixar que o meta-debate se converta no tema principal. A resposta debería ser colateral, un breve prefacio á mensaxe completa. Sinalando que "as cousas non se fan así nesta comunidade" e continuando o debate sobre contido real, facilitas un tema sobre o que a xente pode seguir falando deixando detrás a mala contestación. Se alguén che recrimina que non merecían tal reprimenda, non entres nese debate. Ou non respondas (se considerares que simplemente precisan desestresarse) ou responde pedindo desculpas se a túa reacción foi excesiva e que é difícil detectar certos matices nun correo electrónico. E volta ao contido principal da conversación. Nunca xamais

esixas unha desculpa, sexa pública ou privada, de alguén que se comportou de modo inapropiado. Se por si mesmos deciden escribila, estupendo, mais esixila só causará resentimento.

O obxectivo xeral é facer ver que a boa educación é unha das políticas de comportamento do grupo. Isto axudará o proxecto xa que os desenvolvedores poden decidir non contribuír (incluso en proxectos que lles gustan e nos que desexan participar) debido ás continuas guerras de flames. É probable que nin te decates; alguén está unicamente lendo a lista de correo, ve que hai que ter coraxe para participar no proxecto e decide non facelo. Manter a netiqueta nos foros é unha estratexia de longo recorrido e é máis sinxela de implementar cando o proxecto é pequeno. Unha vez que é parte da cultura, non serás a única persoa en promocionala. Será mantida por todo o mundo.

Fomenta a revisión continua e pública do código

Unha das mellores maneiras de favorecer unha comunidade de desenvolvemento produtiva é facer que a xente revise o código de outros. Para atinxilo de modo efectivo é precisa certa infraestrutura técnica - en particular, o aviso de novos commits por email debe ser activado; ver a sección "Mensaxes de *commit*" no capítulo 3 para máis detalles. O efecto de activar o aviso por email dos commits é que cada vez que alguén faga un commit sobre o repositorio de código do proxecto, un email será enviado mostrando o log do mensaxe e as mudanzas introducidas (ver "*diff*" na sección "Vocabulario dos Sistemas de Control de Versións" no capítulo 3). A *revisión de código* é a práctica de revisar os emails de commit que chegan, buscando erros e posibles melloras.

Así é como habitualmente se fai a revisión do código en proxectos de software libre. En proxectos máis centralizados, a "revisión do código" pode mesmo envolver grupos de persoas que se reúnen para procuraren problemas e patróns específicos sobre o código impreso.

A revisión do código ten varias vantaxes. A máis obvia é que é un exemplo de revisión entre pares no mundo do software libre e axuda de maneira directa a manter a calidade do software. Cada erro que se empacota nunha aplicación chegou alí porque non foi detectado ao ser subido ao repositorio do código e, por iso, cantos máis ollos miren, máis erros van ser corrixidos. Mais a revisión do código tamén ten unha vantaxe indirecta: fai que a xente saiba que o que fan serve para algo, xa que un desenvolvedor non revisaría un commit se non estivese interesado polos seus efectos. A xente dá o mellor de si cando saben que os demais van avaliar o seu traballo.

As revisións deberían ser públicas. Mesmo cando estaba no mesmo cuarto que outro desenvolvedor e un de nós facía un commit, procurabamos non facer a revisión verbalmente no cuarto, senón enviala á lista de correo. Todo o mundo se beneficia de ver a revisión. A xente verá os comentarios e algunhas veces atopará erros neles; mais mesmo se non o fan, é algo que recorda que a revisión é unha actividade que se espera do grupo, como lavar os pratos ou cortar o céspede.

No proxecto Subversion non fixemos a revisión do código desde o inicio. Non había garantía ningunha de que un commit fose revisado, aínda que ás veces alguén podía revisar unha mudanza se estaba especialmente interesado nesa parte do código. Os erros incluídos puideron e deberon ter sido detectados. Un desenvolvedor chamado Greg Stein, quen coñecía a importancia da revisión do código por traballos anteriores, decidiu que ía sentar precedente mediante a revisión de cada liña de *cada commit* realizado sobre o repositorio do código do proxecto. Así, cada commit realizado viña seguido por un email de Greg á lista de correo dos desenvolvedores, diseccionándoo, analizando posibles problemas e ocasionalmente louvando un pedazo de código brillante. Desde o primeiro momento, comezou a a detectar erros e prácticas de código non óptimas que de outro modo se terían filtrado sen

que ninguén se decatase. Hai que salientar que nunca se queixou de ser a única persoa que revisaba cada commit, aínda tendo en conta que lle levaba unha gran cantidade de tempo, e louvaba as vantaxes da revisión do código cada vez que tiña ocasión. Máis xente, incluído eu mesmo, comezamos logo a revisar os commits regularmente. Cal foi a nosa motivación? Non foi que Greg se queixara porque nós non o fixeramos. A razón foi que demostrara que a revisión de código era unha actividade coa que se podía contribuír tanto ao proxecto como escribindo código novo. Unha vez o demostrou, converteuse nun hábito do grupo, ata tal punto de que cada commit que non tiña ningún comentario provocaba que o desenvolvedor se preocupase e mesmo preguntase na lista de correo si alguén tivera a oportunidade de revisalo. Máis tarde, Greg comezou un traballo que non lle deixou moito tempo para colaborar co proxecto Subversion. Para entón, o hábito estaba tan interiorizado en cada un de nós como se tivese sido sempre así.

Comeza a facer revisións desde o primeiro commit. A clase de problemas que son máis sinxelos de detectar mediante a revisión de diffs son as vulnerabilidades de seguridade, fugas de memoria, comentarios insuficientes ou documentación da API, erros "*off-by-one*", incumprimento das regras de estilo e outros problemas que requiren un mínimo de contexto para seren vistos. Porén, mesmo problemas maiores tales como non abstraer patróns que se repiten poden ser vistos despois de que un teña feito revisións regularmente, xa que a memoria das mudanzas anteriores axuda na revisión dos presentes.

Non te preocupes se non puideres atopar nada sobre o que comentares, ou se non souberes o suficiente sobre cada área do código. Habitualmente hai cousas que dicir sobre cada commit; mesmo onde non atopes nada que cuestionar, podes atopar algo que gabar. O importante é deixares claro a cada commiter que o que fan é visto e comprendido. Desde logo, a revisión de código non absolve os programadores da responsabilidade de revisaren e probaren as súas mudanzas antes de subírenas ao repositorio; ninguén debería deixar que a revisión do código faga o traballo que un debería ter feito por si mesmo.

Cando publicares un proxecto que estiver pechado ata agora, ten presente a magnitude da mudanza

Se vas facer público un proxecto existente, un que xa ten desenvolvedores activos afeitos a traballaren nun ambiente pechado, asegúrate de que todo o mundo comprende que a apertura supón unha gran mudanza - e de que comprendes como se van sentir con ela.

Trata de imaxinar como é a situación para eles: ata esa altura, todas as decisións de deseño e código foron feitas por un grupo de programadores que coñecían o software máis ou menos igual, que recibiron as mesmas presións desde o mesmo equipo de xestión e que todos coñecen os puntos fracos e fortes dos seus compañeiros. A apertura do proxecto supón expor o código ao escrutinio de descoñecidos, que se formarán xuízos de valor baseados unicamente no código, sen se preocuparen de que presións de negocio poden ter forzado a tomar certas decisións. Eses descoñecidos van facer moitas preguntas, preguntas que van facer que os desenvolvedores se decaten de que a documentación na que tan duramente traballaran é *aínda* inadecuada (isto é inevitable). E máis aínda, os que acaban de chegar son descoñecidos, entidades sen face. Se un dos teus desenvolvedores xa se sente inseguro sobre as súas capacidades, imaxina como ese sentimento será exacerbado cando os neófitos sinalen erros no código que escribiu, e aínda peor, diante dos seus colegas. A menos que teñas un equipo de perfectos programadores, isto é inevitable - de feito, probablemente lle ocorrerá a todos eles ao principio. Isto non se debe a que sexan malos programadores; é só que todo programa cunhas certas dimensións ten erros e a revisión entre pares deitará luz sobre algúns deles (ver sección “Revisión de código intensiva

e continua" anteriormente neste capítulo). Ao mesmo tempo, os que acaban de chegar non están baixo o mesmo nivel de revisión, xa que non poden enviar código ata que se familiarizaren co proxecto. Os teus desenvolvedores poden sentir como que todas as críticas van dirixidas a eles, mais ningunha para os demais. Así, existe o perigo de que se estenda un sentimento negativo entre o equipo.

O mellor modo de evitar esta sensación é tentar que todo o mundo saiba o que implica a mudanza, explicárllelo, dicirlles que o incomodo inicial é perfectamente normal e asegurarlles que co tempo a situación vaise normalizar. Algunhas desas recomendacións deben ser feitas en privado antes da apertura do proxecto. Mais pode ser mesmo útil recordarlle á xente a través das listas públicas que hai un novo modo de desenvolver o proxecto e que os axustes van levar tempo. O mellor que podes facer é liderar mediante o exemplo. Se non ves os teus desenvolvedores responderen a cuestións dos neófitos, dicirlles que o fagan non será suficiente. Poden aínda non ter un bo sentido do que se debe responder e como, ou poden non ter claro como priorizar o traballo de codificación fronte ás novas tarefas de relacións externas. A maneira de que participen é participares ti mesmo. Estares presente nas listas de correo públicas e responderes algunhas preguntas axudará. Se non tiveres a experiencia para responderes algunha das preguntas, pásalla a un desenvolvedor que si a teña - e asegúrate de que forneza unha solución ou polo menos responda. De modo natural, os teus desenvolvedores realizarán moitos dos seus comentarios en privado, entre eles, xa que é como o facían habitualmente. Subscríbete ás listas de correo internas nas que isto poida ocorrer e suxire que tales debates sexan movidos á lista pública.

Hai outras cuestións a longo prazo que un debe ter en conta cando decide abrir un proxecto. O capítulo 5 "Diñeiro" explora as técnicas para envolver con suceso desenvolvedores pagados e voluntarios, e o 9 "Licenzas, dereitos de autor e patentes" debate as dilixencias legais necesarias cando abrires un código que pode ter partes "propiedade" de terceiros.

Anunciar o proxecto

Unha vez o proxecto é presentable - non perfecto, só presentable - estás pronto para anuncialo ao mundo. Isto é un proceso sinxelo: vai a http://freshmeat.net/, pincha sobre *Enviar* na barra de navegación superior e preenche un formulario coas características do teu proxecto. Freshmeat é o lugar onde todo o mundo procura novos proxectos. Unicamente tes que facer que uns poucos ollos se fixen para que a noticia sobre o teu proxecto se expanda.

Se coñeces listas de correo ou foros onde a publicación do proxecto poida ser de interese, faino tamén aí, mais tenta escribir exactamente *unha* mensaxe por foro e dirixires a xente aos teus propios canais de comunicación para continuar a conversa (configurando o cabezallo *Reply-to*). As mensaxes debería ser curtas e directas:

```
To: discuss@lists.example.org
Asunto: [ANN] Proxecto Scanley: indexador de texto completo
Reply-to: dev@scanley.org

Esta mensaxe é para anunciar a creación do proxecto Scanley,
 un motor de indexado e procura de texto completo de código aberto cunha
API ampla, para ser usado por programadores que desexaren fornecer servizos de procura
de amplas coleccións de arquivos de texto. Scanley é funcional, está
baixo desenvolvemento activo e precísanse desenvolvedores e testadores.

Páxina web: http://www.scanley.org/

Funcionalidades:
   - Procuras de texto plano, HTML e XML
   - Procura de palabras e/ou frases
```

```
   - (planeada) Fuzzy matching
   - (planeada) Actualización incremental dos índices
   - (planeada) Indexación de sitios web remotos

Requerimentos:
   - Python 2.2 ou superior
   - Espazo en disco suficiente para manter os índices (aproximadamente 2x
     o tamaño orixinal)

Para máis información, ver scanley.org.
 Obrigado,
-J. Random
```

(Ver a sección “Publicidade” no capítulo 6 “Comunicacións” para consellos sobre como anunciar futuras liberacións do proxecto e outros eventos.)

Hai un debate non pechado no mundo do software libre sobre se é preciso comezar con código que funcione ou se un proxecto se pode beneficiar das vantaxes da apertura nas etapas de deseño e debate. Eu pensaba con frecuencia que o factor máis importante era comezar con código que funcionase, que iso era o que separaba os proxectos con suceso dos xoguetes e que os desenvolvedores serios unicamente eran atraidos por software que facía algo concreto.

Isto demostrou ser erróneo. No proxecto Subversion, comezamos cun documento de deseño, un grupo de desenvolvedores interesados e ben conectados, moita fanfarra e *sen* código que funcionase. Para a miña sorpresa, o proxecto adquiriu participantes activos desde o primeiro momento e para cando tiñamos algo funcional, había uns cantos desenvolvedores voluntarios profundamente implicados no proxecto. Subversion non é o único exemplo; o proxecto Mozilla foi tamén lanzado sen código que funcionase e agora é un navegador web que ten éxito.

Ante tales evidencias, teño que retractarme das miñas palabras. Comezar con código que funcione é unha das mellores formas para atinxir o éxito e unha boa regra sería esperar ata que tiveres algo para anunciares o proxecto. Porén, poden existir ocasións onde anunciar o proxecto en fases máis previas teña sentido. Creo firmemente que é necesario polo menos un bo documento de deseño ou algunha clase de código marco - desde logo, pode ser revisado en público, mais ten que existir algo concreto, algo máis tanxible que só boas intencións para que á xente lle interese participar.

Cando fixeres o anuncio, non esperes que unha horda de voluntarios se unan ao proxecto de maneira inmediata. Habitualmente, o resultado da publicación é que tes unhas poucas preguntas, algunha xente se unirá ás listas de correo e todo continuará case coma sempre. Mais ao pasar o tempo, observarás incrementos graduais na participación tanto de usuarios como de desenvolvedores. O anuncio pode ser considerado simplemente unha semente. A expansión da noticia pode demorar. Se o proxecto premiar eses que participaren, as noticias vanse estender rapidamente xa que á xente lle gusta compartir que atoparon algo que vale a pena. Se todo vai ben, as dinámicas de redes de comunicacións exponenciais transformaranse lentamente nunha comunidade complexa, onde non necesariamente vas coñecer todo o mundo nin vas poder seguir cada conversa. Os seguintes capítulos versan sobre como traballar nese entorno.

Capítulo 3. Infraestrutura Técnica

Os proxectos de software libre apóianse en tecnoloxías que facilitan a captura selectiva e a integración da información. Canto máis habilidoso sexas usando estas tecnoloxías, e persuadindo a outros a empregalas, maior éxito terá o teu proxecto. Esta afirmación só vai comezar a ser certa segundo vaia medrando o proxecto. Unha boa xestión da información é o que evita que os proxectos de software libre sexan esmagados polo peso

da Lei de Brooks[4], que establece que engadir man de obra a un proxecto que vai con atraso fará que se atrase aínda máis. Fred Brooks observou que a complexidade dun proxecto aumenta a razón do *cadrado* do número de participantes. Mentres só estiver implicado un número reducido de persoas, cada un pode facilmente comunicarse cos demais; mais cando están implicadas centos de persoas, non é xa posible que cada persoa permaneza constantemente informada do que están a facer todos os demais. Se o segredo dunha boa xestión dun proxecto de software libre reside en conseguir que todo o mundo sinta que está a traballar cos demais na mesma habitación, a pregunta obvia é: que pasa cando todos os que están nunha habitación abarrotada tratan de falar ao mesmo tempo?

Este problema non é novo. En habitacións abarrotadas, reais, a solución consiste en adoptar o denominado*procedemento parlamentario*: un conxunto de normas formais sobre como manter discusións en tempo real no seo de grandes grupos, como estar seguros de que ningunha manifestación relevante se perde entre moreas de comentarios tipo "eu tamén", como formar subcomités, como estar seguros sobre cando foi tomada unha decisión, etc. Unha parte importante do procedemento parlamentario consiste en determinar como interaxe o grupo co seu sistema de xestión da información. Algúns comentarios son feitos coa vontade de que fiquen rexistrados nos arquivos, outros non. O propio arquivo está suxeito a manipulación directa, e é entendido non coma unha transcrición literal do que ocorreu, se non como a transcrición do que o grupo pretende *decidir* que ocorreu. Os arquivos non son monolíticos, adoptan diferentes formas para diferentes usos. Comprenden tanto as actas de cada reunión como as coleccións completas de todas as actas de todas as reunións, resumos, axendas e as súas anotacións, informes de comités, informes de colaboradores non presentes nas reunións, listas de temas a tratar, etc.

Como Internet non é realmente unha habitación, non temos que preocuparnos de replicar as partes do procedemento parlamentario que teñen a ver con manter unhas persoas caladas mentres outros falan. Mais cando se trata de técnicas de xestión da información, os proxectos de software libre ben xestionados poden compararse a procedementos parlamentarios dopados con esteroides. Dado que nos proxectos de software libre practicamente toda a comunicación se fai por escrito, os sistemas elaborados evoluíron para faceren axeitadamente o seguimento e a etiquetaxe dos datos; cara á minimización das repeticións para evitaren diverxencias espúreas; para o correcto almacenamento e recuperación dos datos; cara a corrixir a información errónea ou obsoleta; e cara a asociación entre si de bits de datos desaparellados en canto se observan novas conexións. Quen participa activamente en proxectos de software libre adopta moitas destas técnicas, e frecuentemente é quen de executar complexas operacións manuais para garantir que a información está a ser xestionada axeitadamente. Mais o traballo completo depende en grande medida da axuda de software sofisticado. Na medida do posible son precisamente as ferramentas de comunicación as que deberían asumir as máis das tarefas de direccionamiento, etiquetaxe, e gravación, e deberían achegarlle a información aos seres humanos da maneira máis axeitada. É obvio que na vida real os seres humanos van ter que intervir en moitas etapas do proceso, e é importante que o software facilite esas intervencións. Mais en xeral, se os seres humanos toman a precaución de etiquetar e encamiñar a información correctamente ao inserila pola primeira vez no sistema, o software debería estar configurado para tirar o máximo partido dos metadatos.

As recomendacións recollidas neste capítulo son principalmente prácticas, baseadas en experiencias con padróns de software específico e de uso. Mais a cuestión non é aprenderche unha colección particular de técnicas. É tamén a de demostrar, a través do uso de moitos pequenos exemplos, a actitude xeral que mellor se vai adecuar á correcta xestión da información nos teus proxectos. Esta actitude envolve un conxunto de habilidades técnicas e habilidades persoais. As habilidades técnicas son esenciais dado que o software de xestión da información sempre ten que ser correctamente configurado, ademais de precisar de unha certa cantidade de mantemento permanente e de modificación en canto xurdir unha nova necesidade (como exemplo, bótalle unha ollada á discusión sobre como manexar o crecemento do proxecto n a sección “Prefiltrando o xestor de erros” máis adiante neste mesmo capítulo). As habilidades persoais son necesarias xa que a comunidade humana tamén require de mantemento: non sempre é inmediatamente obvio o modo de empregar estas ferramentas para obter o mellor rendemento, e nalgúns casos os proxectos adoptan convencións conflitivas (como exemplo, podes ver a discusión sobre a configuración dos cabezallos *Reply-to* das mensaxes saíntes nas listas de correo). Todos os que estiveren implicados no proxecto van precisar que, nalgún momento ou noutro e da maneira correcta, se lles force a facer a súa parte do traballo para manter ben organizada a información relativa ao mesmo. Canto máis implicado estiver o colaborador, máis complexas e especializadas serán as técnicas que se espera que aprenda.

---

4 Do seu libro *The Mythical Man Month*, 1975. Ver http://en.wikipedia.org/wiki/The_Mythical_Man-Month e http://en.wikipedia.org/wiki/Brooks_Law.

Non existen solucións á medida para a xestión da información. Hai demasiadas variables. Podes ter absolutamente todo configurado da maneira que queres telo, e coa maioría da comunidade participando xa, mais entón o crecemento do proxecto fará que non sexa posible incrementar una desas prácticas. Ou o crecemento do proxecto pode terse estabilizado, e as comunidades de desenvolvedores e de usuarios manter unha confortable relación coa infraestrutura técnica, mais entón alguén se presentará cunha solución de xestión da información completamente nova, e os recentemente incorporados ao proxecto logo van comezar a preguntar por que non está a ser empregada; por exemplo, isto está a ocorrer actualmente cunha chea de proxectos de software libre anteriores á invención das wikis (véxase http://en.wikipedia.org/wiki/Wiki). Moitas decisións dependerán de matices, xa que van ser resoltas mediante compromisos entre os intereses de quen produce a información e os de quen a consume, ou do tempo requirido para configurar o software de xestión da información fronte aos beneficios que este poida achegar ao proxecto.

Loita contra a tentación de hiper-automatizar, e dicir, de automatizar procesos que en realidade requiren da atención humana. A infraestrutura técnica é importante, mais o que fai que un proxecto de software libre funcione é o coidado—e a manifestación intelixente dese coidado— por parte dos individuos involucrados. A infraestrutura técnica ten a ver principalmente con fornecerlles aos humanos formas convenientes de facelo.

Que é o que un proxecto precisa

Os máis dos proxectos de software de código aberto ofrecen cando menos un conxunto mínimo e estándar de ferramentas para a xestión da información:

Sitio Web

> É ante todo un conduto centralizado e unidireccional de transmisión de información desde o proxecto cara ao público. O sitio web pode tamén funcionar como interface administrativa para outras ferramentas do proxecto.

Listas de correo

> Xeralmente constitúen os foros de comunicación máis activos do proxecto, ademais do "medio de arquivamento"

Control de versións

> Permite aos desenvolvedores xestionaren convenientemente as mudanzas no código, incluíndo a reversión e o traslado de mudanzas ("change porting"). Permite tamén que todo o mundo saiba o que se está a facer co código.

Seguimento de erros ("Bug Tracking")

> Permite aos desenvolvedores manteren o control sobre aquilo no que están a traballar, coordinárense cos demais e planificaren as publicacións de novas versións. Permite que todos os colaboradores poidan facer consultas sobre o estado en que se atopa a resolución de erros e gravar información (p. ex.: procedementos de reprodución) sobre erros concretos. Pode ser usado non só para facer seguimento dos erros, senón tamén de tarefas, versións, novas funcionalidades, etc

Chat en tempo real (IRC)

> É o lugar onde se celebran discusións rápidas e pouco transcendentes e se fan intercambios de preguntas/respostas. Non sempre se arquivan completamente.

Cada ferramenta das descritas aborda unha necesidade concreta, mais as súas funcións están interrelacionadas, e deben estar configuradas de xeito que poidan traballar conxuntamente. Máis adiante examinaremos como facer iso e, máis importante, como conseguir que a xente as empregue. O sitio web non se trata ata o final, dado que actúa máis como aglomerante dos demais compoñentes que como unha ferramenta en si mesma.

Podes evitar moitas das dores de cabeza derivadas da selección e configuración destas ferramentas empregando un sitio web do tipo *canned hosting* (aloxamento enlatado): un servidor que ofrece áreas web preempacotadas e regularizadas que conteñen todas as ferramentas necesarias para manexar un proxecto de software libre. Podedes consultar o apartado “Aloxamento preconfigurado” máis adiante neste mesmo capítulo, no que se discute sobre as vantaxes e desvantaxes deste tipo de sitios web.

Listas de Correo

As listas de correo son o pan e o sal das comunicacións nun proxecto. Se un usuario está exposto a un foro para alén das páxinas web, o máis probable é que se trate dunha das listas de correo do proxecto. Mais antes de chegar a usar as listas en si mesmas, primeiro terán que experimentar o uso das súas interfaces, é dicir, o mecanismo polo cal se subscriben (*"subscribe to"*) á lista en cuestión. Isto nos leva á Regra nº1 das listas de correo:

> *Non tratedes de xerir as listas de correo manualmente; empregade software de xestión de listas*

Deixar esta regra de lado pode resultar tentador. Configurar software para a xestión das listas pode parecer excesivo. Xestionar listas pequenas, con pouco tráfego de mensaxes a man pode resultar sedutoramente sinxelo: simplemente creas un enderezo de subscrición encamiñado a ti, e cando alguén manda unha mensaxe a ela, ti engades (ou eliminas) o seu enderezo de email dun arquivo de texto que contén todos os enderezos da lista. Podería ser máis sinxelo?

O truco está en que unha boa xestión da lista de correo, que é o que a xente espera, non é sinxela en absoluto. No se trata tan só de subscribir ou dar de baixa os usuarios cando o pediren. Ten tamén a ver con moderala, evitar o *spam*, ofertar a lista en formato de resumo de mensaxes ou nun formato que mostre todas as mensaxes, fornecer información estándar sobre a lista e o proxecto mediante mensaxes de resposta automática, e moitas outras cousas. Un ser humano que faga seguimento da lista de subscricións e baixas estará a fornecer tan só unha mínima parte das funcionalidades requiridas e, mesmo entón, non en forma tan segura e áxil como o faría o software.

O software de xestión de listas actual ofrece cando menos as seguintes utilidades:

Subscrición tanto por email como a través dunha páxina web

> Cando un usuario se subscribe a unha lista debería recibir *inmediatamente* unha mensaxe automática de benvida, dicíndolle a que se subscribiu, como interaxir en adiante co software de xestión da lista e (moi importante) como darse de baixa. Esta resposta automática pode ser personalizada para que conteña información especifica do proxecto, desde logo, tal como o enderezo da súa páxina web,a localización das FAQ (*Frequently Asked Questions,* Preguntas Respondidas Frecuentemente), etc.

Subscrición en modo "digest" (só resumos das mensaxes) ou en modo de mensaxes completas

> En modo digest, o subscritor recibe só unha mensaxe por día coa totalidade da actividade da lista nese día. Para xente que está a seguir unha lista de xeito desligado, sen participar realmente na súa actividade, o modo digest é con frecuencia o máis axeitado, xa que lle permite explorar a totalidade das materias tratadas dunha vez e evitar a distracción que supón que as mensaxes estean entrando de maneira aleatoria.

Utilidades de moderación

> "Moderar" consiste en controlar que as mensaxes entrantes: a)non son tópico da *spam*, e b) correspóndense co tópico da lista, antes da súa distribución aos restantes usuarios da mesma. O feito de moderar as listas é unha tarefa tipicamente humana, mais o software pode contribuír enormemente a facela máis sinxela. Falaremos máis sobre a moderación máis adiante.

Interface administrativa

> Entre outras cousas, permite ao administrador dar de baixa enderezos obsoletos de maneira sinxela. Isto pode ser urxente se o enderezo dun usuario comeza a emitir mensaxes automáticas do tipo "Non estou xa neste enderezo" en resposta a cada mensaxe enviada á lista. Algúns softwares de listas de

correos poden mesmo detectar estas situacións por si mesmos e dar de baixa automaticamente á persoa en cuestión.

Manipulación de cabezallos

Moita xente ten activadas no seu software de correo electrónico regras de filtraxe e de resposta realmente sofisticadas. O software de xestión de listas de correo pode engadir e manipular os cabezallos estándar para que esta xente se beneficie delas (algúns detalles máis adiante).

Arquivamento

Todas as mensaxes enviadas ás listas xestionadas son almacenadas e postas á disposición dos interesados na web; alternativamente, algúns softwares de listas de correo ofrecen interfaces especiais para enlazar ferramentas externas de arquivamento tales como *MHonArc* (http://www.mhonarc.org/). Como se comenta no capítulo "Uso intensivo dos arquivos" do capítulo 6 "Comunicacións", o arquivamento é unha tarefa crucial.

A intención de todos estes comentarios é salientar que a xestión das listas de correo é un problema complexo sobre o que se ten pensado moito, e que foi en grande medida solucionado. Certamente, ti non precisas tornarte en experto na materia. Mais deberías saber que sempre hai máis sobre o que aprender, e que a xestión das listas vai ocupar a túa atención de cando en vez ao longo da vida dun proxecto de software libre. Máis adiante examinaremos unhas poucas das cuestións máis frecuentes relativas á configuración das listas de correo.

Prevención do *spam*

Entre o momento en que esta frase foi escrita e aquel en que foi publicada, seguramente o problema do *spam* ten duplicado a súa severidade, ou cando menos se percibirá dese xeito. Houbo un tempo, non hai tanto, cando un podía pór en marcha unha lista de correo sen ter que tomar ningunha medida *anti-spam*. A mensaxe perniciosa ocasional podía certamente aparecer, mais de xeito tan pouco frecuente que se consideraba un inconveniente de baixo nivel. Eses tempos desapareceron para sempre. Hoxe en día, unha lista de correo que non adopte ningunha precaución *anti-spam* verase rapidamente asolagada por correos lixo, ata o punto de se tornar inutilizable. A prevención *anti-spam* é polo tanto obrigatoria.

A prevención *anti-spam* pode ser dividida en dúas categorías: a que prevén que entren mensaxes *spam* na túa lista, e aquela que trata de evitar que a lista se converta nunha fonte de enderezos de correo electrónico para os *spammers* que as colleitan. A primeira é a máis importante, así que a analizaremos en primeiro lugar.

Filtraxe das mensaxes

Hai tres técnicas básicas para evitar a entrada de mensaxes lixo, e os máis dos software de listas fornecen as tres. Traballan mellor cando se usan en tándem:

1. **Soamente permitir automaticamente aquelas mensaxes enviadas á lista por subscritores.** Esta é unha medida efectiva cando está activa, e implica moi pouco traballo administrativo, xa que normalmente obriga a mudar tan só unha opción na configuración do software de xestión da lista. Mais ten en conta que as mensaxes que non son automaticamente aprobadas non deben ser simplemente descartadas. Pola contra, deben pasar a ser moderadas por dúas razóns. A primeira é que vas querer que os non subscritores poidan contribuír. Unha persoa que quere facer un comentario ou unha suxestión non debería ter que subscribirse tan só para enviar unha única mensaxe. En segundo lugar, mesmo os subscritores poden, en ocasións, enviar mensaxes desde enderezos distintos a aqueles cos que se cadastraron na lista. Os enderezos de correo electrónico no son un método fiable de identificación das persoas, e non deberían ser tratados como tales.
2. **Filtraxe de mensaxes a través de software de control de spam**. Se o software da lista de correo o permitir (os máis deles o fan), podes facer que as mensaxes sexan controladas por unha aplicación de filtraxe de *spam*. A filtraxe automática de *spam* non é perfecta, e nunca o vai ser, xa que hai unha carreira inacabable entre os *spammers* e os escritores de filtros. Porén, pode reducir sensiblemente a cantidade de *spam* que atravesa a ringleira de moderación, e dado que canto máis longa é a ringleira máis tempo de dedicación require dos humanos, calquera cantidade de filtraxe automática resulta beneficiosa. Non hai espazo dabondo neste libro para transmitirche instrucións detalladas sobre como

configurares os filtros *anti-spam*. Terás que consultar as instrucións do software da túa lista (ver máis na sección “Aplicacións de roldas de correo”). O software de listas de correo inclúe frecuentemente algunha utilidade preconfigurada de prevención *anti-spam*, mais pode que queiras engadir algúns filtros de terceiros. Persoalmente teño boas experiencias con estes dous: *SpamAssassin* (http://spamassassin.apache.org/) e *SpamProbe* (http://spamprobe.sourceforge.net/). Isto non supón ningunha valoración de outros filtros *anti-spam* libres, algúns dos cales son aparentemente bastante bos. É só que eu usei os dous mencionados e quedei moi satisfeito con eles.

3. **Moderación**. Para aquelas mensaxes que non foren automaticamente autorizadas por procederen dun subscritor da lista, e que pasaren a través do software de filtraxe de *spam*, se for o caso, o último paso é o da *moderación*: a mensaxe é enviada a un enderezo especial, no que un individuo a examina e decide se é aprobada ou rexeitada. A aprobación dunha mensaxe pode facerse de dúas maneiras: pódese aceptar a mensaxe só por esta vez, ou podes dicirlle á lista que acepte esta e calquera outra mensaxe que no futuro proceda do mesmo remitente. As máis das veces é isto último o que se fai, xa que así se reduce a carga de traballo da moderación. Os detalles sobre como se fai varían de sistema a sistema, mais xeralmente consiste en responder a un determinado enderezo de correo coa mensaxe "aceptar" (o que quere dicir que se acepta tan só esta mensaxe) ou "permitir" (aprobando esta e outras mensaxes futuras). O rexeitamento faise xeralmente ignorando a mensaxe de moderación. Se o software da lista non dá recibido a confirmación de que algo é unha mensaxe válida, non a pasará á lista, así que o efecto desexado se produce se simplemente non se dá curso á mensaxe de moderación. Ás veces tamén tes a opción de responderes cos comandos "rexeitar" ou "denegar", para que ningunha mensaxe do mesmo remitente sexa aceptada no futuro, sen necesidade de sometelas a moderación. Xeralmente non ten moito sentido facer isto, xa que a moderación tenta controlar o *spam*, e os *spammers* moi raramente empregan dúas veces o mesmo enderezo.

Ten coidado de empregares a moderación *tan só* para filtrares o *spam* e as mensaxes claramente fóra de tema, como cando alguén manda accidentalmente un correo á lista equivocada. O sistema de moderación permite habitualmente a resposta directa ao remitente, mais non se debe usar este método para dar resposta a preguntas que en realidade entran no ámbito de actividade da propia lista, aínda que teñas absolutamente clara a resposta. Facer iso privaría á comunidade do coñecemento exacto acerca de que clase de preguntas fai a xente, e impediríalles responder por eles mesmos ou ver as respostas que outros poidan enviar. A moderación das listas de correo ten a ver exclusivamente con mantelas libres de correo lixo e mensaxes fóra de tema, nada máis.

Ocultación de enderezos nos arquivos

Para evitares que as túas listas de correo se convertan nunha fonte de subministro de enderezos para os *spammers*, unha técnica habitual é a de que os arquivos oculten os enderezos da xente, por exemplo substituíndo

```
jrandom@somedomain.com
por jrandom_AT_somedomain.com
ou jrandomNOSPAM@somedomain.com
```

ou algunha maneira igualmente obvia (para os humanos) de codificalas. Dado que os colleitadores de enderezos para *spam* funcionan frecuentemente uliscando nas páxinas web; incluíndo os arquivos web da túa lista de correo; e buscando secuencias que conteñan "@", codificar os enderezos é a maneira de tornalos inútiles ou invisibles para os *spammers*. Por suposto que isto non evita que o *spam* sexa enviado directamente á propia lista, mais si que minimiza a cantidade de lixo que chega aos enderezos persoais dos membros da lista.

A ocultación de enderezos pode ser discutida. Gústalle moito a algunha xente , que se sorprenderá se os teus arquivos non o fan de maneira automática. Outra xente, pola contra, pensará que é unha incomodidade (xa ques as persoas terán tamén que traducir os enderezos antes de poder usalos). E hai xente que afirma que é unha práctica inútil, xa que un colleitador podería, en teoría, compensar calquera padrón de codificación. En calquera caso, hai evidencia empírica dabondo de que a ocultación de enderezos *é* efectiva, consulta http://www.cdt.org/speech/spam/030319spamreport.shtml.

Idealmente, o software de xestión da lista debería deixar a escolla en mans de cada subscritor, ben mediante un cabezallo especial tipo "Si/Non" ou cunha opción de configuración nas preferencias da conta da lista de ese

subscritor en particular. Porén, non coñezo ningún software que ofreza esa opción para cada subscritor ou para cada mensaxe, así que polo de agora será o xestor da lista quen teña que tomar a decisión por todos (dando por sentado que o arquivo ofrece esa utilidade, o que non sempre é así). Eu inclínome lixeiramente cara a que a ocultación de enderezos estea activada. Algunhas persoas poñen moito coidado en non publicar os seus enderezos en páxinas web ou en calquera outro sitio onde un colleitador de *spam* poida localizalos, e sentiríanse moi incómodos se esa precaución fose tirada ao lixo por un arquivo de listas de correo; mentres tanto, o inconveniente que supón a ocultación de enderezos para as persoas é moi pouco importante, xa que a transformación de un enderezo oculto nun utilizable é trivial no caso de que precisares acceder á persoa en cuestión. Mais non esquezas que, ao final, non deixa de ser unha carreira armamentística: para cando leas isto, os colleitadores poden perfectamente ter evolucionado ata o punto de seren quen de recoñecer as formas máis habituais de ocultación, e teremos que empezar a pensar noutra solución.

Identificación e xestión de cabezallos

Frecuentemente, os subscritores das listas quererán arquivar as mensaxes nunha pasta específica para o proxecto, distinta das de outras mensaxes de correo. O seu software de lectura de correo pode facer isto automaticamente sen máis que examinar os *cabezallos* das mensaxes. Os cabezallos son os campos situados no inicio das mensaxes que indican quen é o remitente, o destinatario, cal é o asunto, a data e outra información variada acerca da mensaxe. Algúns cabezallos son ben coñecidos e certamente imprescindibles:

```
De: ...
Para: ...
Asunto: ...
Data: ...
```

Outras son opcionais, aínda que bastante habituais. Por exemplo, os emails non teñen por que levar o cabezallo

```
Responder a: remitente@enderezo de correo,
```

mais as máis delas o levan, xa que lle ofrece ao destinatario un camiño seguro para acceder ao autor da mensaxe (o que é especialmente útil cando o autor tivo que enviar a mensaxe desde un enderezo distinto a aquel ao que desexa que se lle remitan as respostas).

Algún software de lectura de correo ofrece unha interface de uso sinxelo para filtrar mensaxes baseándose en padróns do cabezallo "asunto". Isto fai que a xente pida que a lista engada un prefixo automático a todos os asuntos, de xeito que se poida configurar o programa para que recoñeza ese prefixo e arquive automaticamente na pasta correcta as mensaxes que o levan. A idea é que o autor da mensaxe escribiría:

```
Asunto: preparando a liberación 2.5.
```

mais a mensaxe se vería na lista deste xeito:

```
Asunto: [discusions@listas.exemplo.org] preparando a liberación 2.5.
```

Aínda que a maioría dos programas de xestión de listas ofrecen esta opción, persoalmente recomendo non activala. O problema que resolve pode ser solucionado de maneira máis sinxela por métodos menos intrusivos, e o custo de reducir o espazo do cabezallo "asunto" é considerablemente superior. Os usuarios experimentados de listas de correo acostuman a ver os asuntos da lista de mensaxes entrantes do día para decidiren cal ler ou a cal responder. Se o nome da lista antecede o asunto, o contido deste pode ser desprazado cara ao lado dereito da pantalla, fóra do campo de visión. Isto oculta información que a xente necesita para decidiren que mensaxes abrir, reducindo así a utilidade xeral da lista para todo o mundo.

En lugar de mudar excesivamente o cabezallo "asunto", aprende aos teus usuarios a tiraren partido de outros cabezallos estándar, comezando polo cabezallo "para:", que debería conter o nome da lista:

```
Para: <discusion@listas.exemplo.org>
```

Calquera lector de correo que poida filtrar o "asunto" debería tamén poder filtrar o "para:" coa mesma facilidade.

Hai uns poucos cabezallos máis de listas de correo que son opcionais mais estándar. Filtralos é mesmo máis preciso que facelo sobre os "para:" ou "CC"; dado que estes cabezallos son engadidos ás mensaxes polo propio software de xestión de listas, algúns usuarios poden contar coa súa presenza:

```
axuda-lista: <mailto:discuss-help@lists.example.org>
baixa-lista: <mailto:discuss-unsubscribe@lists.example.org>
envio-a-lista: <mailto:discuss@lists.example.org>
Distribuído-a:: mailing list discuss@lists.example.org
Lista: contactar discuss-help@lists.example.org; run by ezmlm
```

Os máis deles son obvios. Consulta http://www.nisto.com/listspec/list-manager-intro.html para máis explicacións, ou se precisares especificacións realmente detalladas e formais, podes ver http://www.faqs.org/rfcs/rfc2369.html.

Ten en conta que estes cabezallos implican que, se tiveres unha lista denominada "lista", tamén vas dispor de enderezos administrativos "axuda-lista" e "baixa-lista". Ademais, é habitual ter "subscrición-lista" para cadastrarse, e "propietario-lista" para contactar cos seus administradores. Xeralmente se envía automaticamente a explicación sobre a utilidade e todos estes enderezos aos novos subscritores como parte dunha mensaxe de benvida. Probablemente ti mesmo teñas copia desta mensaxes de benvida. Se non for así, pídelle a alguén que cha pase, para que poidas saber o que é que os teus usuarios reciben cando se dan de alta na lista. Ten a copia a man de xeito que poidas responder preguntas sobre o funcionamento da lista ou, mesmo mellor, publícaa nalgunha páxina web. Deste xeito, cando alguén perda a súa copia e che mande unha mensaxe preguntando "Como podo darme de baixa nesta lista?", ti só terás que enviarlle a correspondente URL.

Algúns programas de listas de correo ofrecen unha opción para engadir ao pé das mensaxes as instrucións para darse de baixa. Se for así, actívaa. Só produce un par de liñas de máis por mensaxe, nunha localización onde non estorban, e pode aforrarche unha boa cantidade de tempo diminuíndo a cantidade de xente que che envía consultas, ou peor, que manda unha mensaxe á lista! preguntando como darse de baixa.

O gran debate sobre "responder-a" (*Reply-To*)

Antes, na sección "Evita as discusión privadas" no capítulo 2 "Primeiros pasos", salientei a importancia de manter discusións seguras nos foros públicos, e falei sobre como é as veces necesario adoptar medidas activas para evitares a conversión de discusións en fíos privados; máis adiante, este capítulo trata completamente sobre a configuración do software de comunicacións para que faga por ti tanto traballo como for posible. Polo tanto, se o software de administración ofrece unha maneira automatizada para evitar que se establezan discusións na lista, deberías pensar que activar esa opción é a decisión obvia.

Ou talvez non tanto. Existe esa utilidade, mais presenta algunhas desvantaxes realmente importantes. A cuestión de se usala ou non constitúe un dos debates máis quentes no mundo da xestión de listas, admitámolo, non unha controversia como para saír nas noticias da noite na túa cidade, mais si que pode xurdir de tempo en tempo en todo proxecto de software libre. Máis adiante describirei esta utilidade, darei os principais argumentos a favor e en contra, e darei a mellor recomendación que poida.

A utilidade en si mesma é moi simple: o software de listas de correo pode, se quixeres, configurar automaticamente o cabezallo "responder-a" (*Reply-to*) en todas as mensaxes, de xeito que as respostas se encamiñen á propia lista. Isto é, independentemente do que o remitente poña no cabezallo "contestar a "(e mesmo se non poñen nada en absoluto), cando os subscritores vexan a mensaxe, o cabezallo conterá o enderezo da lista:

```
Responder a: discusións@listas.exemplo.org
```

De primeiras, parece unha boa idea. Xa que practicamente todo o software de lectura de correo ten en conta o cabezallo "responder a", cando alguén responder a unha mensaxe, a súa resposta vai ser automaticamente dirixida a toda a lista, non só ao remitente da mensaxe orixinal. Por suposto, quen contesta pode mudar

manualmente o destinatario da súa mensaxe, mais o importante é que *por defecto* as respostas son dirixidas á lista. É un exemplo perfecto do uso da tecnoloxía para fomentar a colaboración.

Infelizmente, hai algunhas desvantaxes. A primeira é coñecida como o problema de *Non podo dar co camiño de volta*: ás veces o remitente pon o seu enderezo "real" de correo electrónico no cabezallo "responder a", porque por unha razón ou outra envía o seu correo desde un enderezo distinto de onde o recibe. A xente que sempre envía a recibe o seu correo desde o mesmo enderezo non ten este problema, e pode que se sorprenda mesmo de que exista. Mais para aqueles que teñen configuracións de coreo inusitadas, ou que non poden controlar que enderezo ocupa o campo "de" das súas mensaxes (talvez porque envían os correos desde o seu traballo e non teñen influencia ningunha sobre o seu departamento de TI), usar o "responder a" pode ser a única maneira de asegurar que lles chegan as respostas. Cando alguén así publica algunha mensaxe nunha lista á que non está subscrito, a súa configuración do "responder a" tórnase unha información esencial. Se o software da lista sobrescribe esa información, é posible que nunca chegue a ver as respostas á súa mensaxe.

A segunda desvantaxe ten a ver coas expectativas, e na miña opinión é o argumento máis poderoso contra o uso de forzar o "responder a". Os usuarios de correo electrónico máis experimentados están afeitos a dous métodos básicos de resposta: *"responder a todos"* e *"responder ao remitente"*. Todos os programas de lectura de correo electrónico modernos teñen botóns diferentes para estas dúas accións. Os usuarios saben que para responder a todo o mundo (incluíndo a lista) deben escoller "responder a todos", e para responder en privado ao autor orixinal, deben escoller "responder ao remitente". Aínda que desexes fomentar que a xente responda á lista sempre que for posible, hai certas circunstancias nas que a resposta privada é unha decisión de quen responde; por exemplo, poden querer dicirlle algo confidencial ao autor da mensaxe inicial, algo que resultaría inapropiado publicar na lista.

E agora considera o que pasa cando a lista sobrescribe o "responder a" do remitente orixinal. Quen responde preme a tecla de "responder ao remitente", esperando que a mensaxe lle sexa enviada directamente ao autor orixinal. Xa que ese é o comportamento esperado, pode que non se preocupe de comprobar o enderezo do destinatario na nova mensaxe. Compón a súa mensaxe privada e confidencial, que quizais conteña cousas embarazosas sobre alguén da lista, e preme a tecla de enviar. De súpeto, poucos minutos máis tarde, a mensaxe aparece *na lista!.* Certo, en teoría el tiña que ter comprobado convenientemente o que puña no campo "destinatario", e moitos usuarios expertos esperan que se faga así. De feito, cando alguén configura o "responder a" con calquera outro enderezo, como a da lista, xeralmente ten o coidado de mencionar este feito no corpo da mensaxe, de xeito que a xente non se sorprenda do que pasa cando responden.

Como resultado das consecuencias deste comportamento inesperado, a miña preferencia pasa por configurar o software de xestión da lista de xeito que nunca toque o cabezallo "responder a". Esta é unha das ocasións nas que usar a tecnoloxía para fomentar a colaboración ten, paréceme, efectos colaterais potencialmente perniciosos. En calquera caso, hai tamén algúns argumentos poderosos a favor do outro bando deste debate. Independentemente do que decidas, recibirás de cando en vez mensaxes de xente preguntando por que non o escolles a outra maneira. Como seguramente non queres que esta cuestión se converta no principal fío da lista, sería bo que tiveses preparada unha resposta por anticipado, das do tipo que conducen ao abandono da discusión, en lugar de fomentala. Debes estar seguro de que *non* insistes en que a túa decisión, sexa esta a que for, é a única correcta (aínda que o penses). Pola contra, explica que este é un debate moi vello, que hai boas razóns nos dous bandos, que ningunha escolla vai satisfacer a todos os usuarios, e que por tanto tomaches a mellor decisión posible. Moi amablemente indica que non se debe reiniciar a discusión sobre a materia salvo que alguén tiver algo realmente novo que dicir, despois mantente á marxe do fío e espera a que morra de morte natural.

Alguén pode suxerir que se faga unha votación para decidir unha solución ou a outra. Podes facelo se quixeres, mais persoalmente non me parece que contar cabezas sexa unha boa maneira de resolver a cuestión. O castigo para quen é sorprendido polo comportamento do software da lista (enviando por accidente unha mensaxe privada á lista) é tan grande, e os inconvenientes para os demais son tan leves (tendo que recordarlle de tanto en tanto a alguén que responda á lista en lugar de ao teu enderezo privado), que non está claro que a maioría, aínda que sexa precisamente maioría, poida poñer a minoría en tal risco.

Non estou a tratar aquí todos os aspectos do asunto, tan só aqueles que semellan ser da maior importancia. Para un tratamento máis completo, le estes dous documentos canónicos, que son os que sempre se citan cando xorde este debate:

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

A pesar da miña lixeira preferencia indicada antes, non creo que haxa unha única resposta "correcta" para esta cuestión, e de feito participo en moitas listas que escolleron *empregar* o "responder a". O mellor que podes facer é decidirte por unha ou outra solución o antes posible, e tratar de non enredarte en debates sobre o tema máis adiante.

Dúas fantasías

Algún día, alguén vai ter a feliz idea de inserir unha tecla *"responder á lista"* no seu software de correo electrónico. Empregará algún dos cabezallos estándar mencionados anteriormente para deducir o enderezo da lista, e entón dirixir as respostas tan só á lista, deixando de lado todos os demais enderezos dos destinatarios, xa que os máis deles estarán, de todos modos, subscritos. É posible que outros programas de correo copien a idea, e todo este debate desapareza finalmente. (De feito, o programa http://www.mutt.org/ ofrece xa esta característica[5].

Unha solución aínda mellor sería que a opción de "responder a" estivese da parte de quen resposta. Aqueles que desexan que a lista teña o "responder a" preconfigurado (ben nas mensaxes dos outros ou nas propias) poderían escoller esta opción, e os demais escollerían que o "responder a" fose escollido por casa usuario. Porén, non coñezo ningún programa que o faga. De momento, parece que estamos estancados na opción da solución global.

Desde que escribín o anterior, descubrín que hai polo menos un sistema de xestión de listas que ofrece esta posibilidade: Siesta (http://siesta.unixbeard.net/). Tamén podes ler este artigo: http://www.perl.com/pub/a/2004/02/05/siesta.html.

Arquivamento

Os detalles técnicos acerca da configuración do arquivamento de listas de correo son específicos do software que xestiona a lista, e fican alén o ámbito deste libro. Cando escollas ou configures un arquivador, ten en conta as seguintes características:

Actualización áxil

> Frecuentemente a xente vai querer referirse a unha mensaxe enviada durante as últimas dúas horas. Se for posible, o arquivador debería almacenar cada mensaxe instantaneamente, de xeito que en canto aparece na lista de correo estea xa dispoñible nos arquivos da mesma. Se esa opción non estiver dispoñible, entón tenta cando menos configurar o arquivador para que se actualice cada hora. Por defecto, algúns arquivadores lanzan os seus procesos de actualización unha vez cada noite, mais iso supón unha demora claramente inasumible para unha lista de correo medianamente activa.

Estabilidade referencial

> Cando unha mensaxe é arquivada nunha URL dada, debería permanecer accesible para sempre nese mesmo enderezo, ou durante tanto tempo como for posible. Mesmo se os arquivos son reconstruídos, restaurados desde un ficheiro de respaldo (*backup*), ou reparados de calquera outra maneira, as URL que foran publicamente accesibles nun momento dado deben permanecer accesibles. As referencias estables fan posible que os motores de procura en Internet indexen correctamente os arquivos, o que supón unha vantaxe considerable para usuarios que buscan respostas a preguntas concretas. As estabilidade das referencias é tamén importante porque as mensaxes e fíos das listas de correo acostuman estar enlazadas ao xestor de erros (*bug tracker*) (véxase “Xestor de erros” máis adiante neste mesmo capítulo) ou a outros documentos do proxecto.
>
> Idealmente, o software de listas de correo debería incluír a URL do arquivo de mensaxes, ou polo menos a parte específica da URL que se refire á mensaxe, nun cabezallo, cando distribúa as mensaxes

5 Pouco despois de que este libro fora publicado, Michael Bernstein (http://www.michaelbernstein.com/) escribiume para dicirme: "Hai outros programas de correo que incorporan una función de "responder á lista" ademais de Mutt. Por exemplo, Evolution ten esta función en forma de atallo de teclas, aínda que non cun botón (Ctrl+L).

aos seus destinatarios. Deste xeito, a xente que ten unha copia da mensaxe coñecerá a localización do seu arquivo sen ter que ver realmente os arquivos, o que é moi útil porque calquera operación que implica o uso do navegador de Internet supón necesariamente un certo consumo de tempo. Non sei se todo o software de listas de correo ofrece esta característica; infelizmente, os que eu teño usado non o fan. Porén, é algo sobre o que pesquisar (ou, se escribires software para listas de correo, é unha característica que deberías considerar implementar, por favor)

Copias de seguridade (Backups)

Como facer copias de seguridade dos arquivos debería ser razoablemente obvio, e a receita de restauración non debería ser moi difícil. Noutras palabras, non trates o teu arquivador como unha caixa negra. Ti (ou alguén do teu proxecto) deberías saber onde se almacenan as mensaxes, e como rexenerar as páxinas de arquivamento desde o almacén de mensaxes no caso de que nalgún momento fose necesario. Eses arquivos conteñen datos preciosos—un proxecto que os perde, perde tamén unha parte considerable da súa memoria colectiva.

Soporte aos fíos (Threads)

Debería ser posible ir dunha mensaxe dada ao correspondente *fío* (grupo de mensaxes relacionadas entre si) ao que pertence esa mensaxe en particular. Cada fío debería tamén ter a súa propia URL, distinta das correspondentes ás mensaxes que compoñen o fío.

Pesquisas

Un arquivador que non soporte pesquisas (sobre os corpos das mensaxes, e sobre os seus autores ou asuntos) é practicamente inútil. Ten en conta que algúns arquivadores soportan pesquisas simplemente derivándoas a un motor externo tal como Google. Isto é aceptable, mais o soporte a pesquisas directas normalmente permite un mellor afinado, xa que deixa que a ferramenta de procura especifique que o resultado debe procurarse na liña de asunto e non no corpo, por exemplo.

O que antecede é simplemente unha lista de control para axudarte a avaliar e configurar un arquivador. Conseguir que a xente faga un *uso* real do arquivador en beneficio do proxecto se discute en capítulos posteriores, nomeadamente na sección “Uso intensivo de arquivos” no capítulo 6 “Comunicacións”.

Software

De seguido se comentan algunhas ferramentas de software libre para a xestión e o arquivamento de listas de correo. Se o sitio no que tes o teu proxecto aloxado ten xa unha configuración por defecto, non vas ter que escoller ningunha ferramenta en particular. Mais si vas ter que instalar unha por ti mesmo, estas son algunhas das posibles. As que eu teño usado son *Mailman*, *Ezmlm*, *MHonArc*, e *Hypermail*, mais iso non quere dicir que non haxa outras igual de boas (e seguro que existen por aí adiante outras ferramentas que eu nin sequera dei atopado, así que non penses que esta é unha lista completa).

Software de xestión de listas de correo:

- Mailman - http://www.list.org/

  (Ten un arquivador incorporado, así como conexións para arquivadores externos)

- SmartList - http://www.procmail.org/

  (Deseñado para ser usado co sistema de procesamento de correo electrónico *Procmail*)

- Ecartis - http://www.ecartis.org/

- ListProc - http://listproc.sourceforge.net/

- **Ezmlm** - http://cr.yp.to/ezmlm.html

  (Deseñado para traballar co sistema de distribución de correo Qmail - http://cr.yp.to/qmail.html)

- **Dada** - http://mojo.skazat.com/

  (Embora os insistentes esforzos da súa web para ocultalo, é software libre, liberado baixo a licenza GNU. Tamén inclúe un arquivador)

Software de arquivamento de listas de correo:

- **MhonArc** - http://www.mhonarc.org/
- **Hypermail** - http://www.hypermail.org/
- **Lurker** - http://sourceforge.net/projects/lurker/
- **Procmail** - http://www.procmail.org/

  (Software de acompañamento de *Smartlistk*, é un sistema procesador de correo que pode, aparentemente, ser configurado como arquivador)

Control de versións

Un *sistema de control de versións* (ou *sistema de control de revisións*) é unha combinación de tecnoloxías e prácticas para facer seguimento e control das mudanzas que se van producindo nos ficheiros dun proxecto, nomeadamente nos de código fonte, documentación e páxinas web. Se nunca antes usaches control de versións, o primeiro que deberías facer é buscar alguén que si o teña usado e fichalo para o teu proxecto. Hoxe en día, todo o mundo espera que cando menos o código fonte do teu proxecto estea subido a un sistema de control de versións, e seguramente non se tomará o proxecto en serio se non se utiliza este sistema cun mínimo de solvencia.

A razón pola que o control de versións é tan universal é que axuda a practicamente todos e cada un dos aspectos relacionados coa xestión dun proxecto: comunicacións entre desenvolvedores, xestión de liberacións, seguimento de erros (*bug tracking*), estabilidade do código e esforzos de desenvolvemento experimental, e atribución e autorización de mudanzas por certos desenvolvedores. O sistema de control de versións ofrece un centro de mando para todas estas tarefas. O núcleo do control de versións reside na *xestión de mudanzas*: identificando cada mudanza concreta realizada aos ficheiros do proxecto, anotando cada mudanza con metadatos tales como a súa data e o seu autor, e repetindo estes procesos para quen o solicitar, sen importar como. É un mecanismo de comunicación no que a mudanza é a unidade básica de información.

Esta sección non vai revisar a totalidade dos aspectos derivados do uso dun sistema de control de versións. É tan transversal que vai ser comentado ao longo do libro. Aquí concentrarémonos na escolla e a configuración dun sistema de control de versións de xeito que facilite o desenvolvemento cooperativo durante a existencia do proxecto.

Vocabulario do control de versións

Este libro non pode aprenderte a usares o control de versións se nunca antes o usaches, mais sería imposible falar sobre a materia sen definirmos primeiro algúns conceptos chave. Estes termos son útiles independentemente do sistema de control de versións seleccionado: son os verbos e substantivos básicos da colaboración en rede, e serán usados de xeito xenérico ao longo da obra. Aínda que non existisen sistemas de control de versións, o problema da xestión de mudanzas persistiría, e estes termos vannos fornecer unha linguaxe para falar concisamente deste problema.

"Versión" fronte a "Revisión"

A palabra *versión* ás veces é usada como sinónimo de "revisión", mais eu non o vou usar nese sentido neste libro, porque é demasiado fácil confundilo co termo "versión" tal como cando se fala de versións dun software,

isto é, o número de liberación ou distribución, como en "Versión 1.0". De todos os xeitos, como a expresión "control de versións" é de feito estándar, eu vou continuar a empregala como sinónimo de "control de revisión" e "control de mudanzas".

Commit

Facer unha mudanza no proxecto; máis formalmente, almacenar unha mudanza na base de datos de control de versións de tal xeito que poida ser incorporada en futuras versións do proxecto. En inglés, "*commit*" pode ser usado como substantivo ou como verbo. Como substantivo, pode ser considerado como sinónimo de "mudanza". En galego a forma verbal pode ser substituída por "facer un *commit*". Por exemplo: "fixen o *commit* dun remendo para o fallo de servidores en Mac OS X que os de xestión de erros estaban a comentar. Jay, poderías revisar a mudanza e comprobar que non estou a facer mal uso do localizador (*allocator*)?"

Mensaxe de log

Un pequeno comentario engadido a cada *commit*, describindo a natureza e o propósito da mudanza. As mensaxes de *log* cóntanse entre os documentos máis importantes dun proxecto: constitúen a ponte entre a linguaxe altamente técnica das mudanzas no código e a máis orientada aos usuarios de características, remendo de erros e progreso do proxecto. Máis adiante nesta sección, falaremos de como distribuír as mensaxes de *log* ás audiencias apropiadas; tamén, na sección “Tradición na documentación” do capítulo 6 “Comunicacións” se discute acerca dos modos en que os contribuíntes deben escribir mensaxes de *log* sinxelas e útiles.

Actualizar (update)

Solicitar que as mudanzas (*commits*) feitas por outros se incorporen á túa copia local do proxecto; isto é, actualizar a túa copia. É unha operación moi frecuente; os máis dos desenvolvedores actualizan o seu código varias veces ao longo do día, de xeito que están seguros de estaren traballando sobre practicamente o mesmo código sobre o que están a traballar os outros colaboradores, e se eles atopan un erro, poden estar bastante seguros de que aínda non foi reparado. Por exemplo: "Ei!, decateime de que o código de indexación perde o último byte constantemente. Trátase dun novo erro?" "Si, mais foi corrixido hai unha semana, tenta facer unha actualización, debería solucionarse."

Repositorio

Unha base de datos na que se almacenan as mudanzas. Algúns sistemas de control de versións están centralizados: hai un único repositorio no que se almacenan todos as mudanzas do proxecto. Outros son descentralizados: cada desenvolvedor garda o seu propio repositorio, e as mudanzas poden ser trocadas entre repositorios arbitrariamente. O sistema de control de versións mantén o seguimento das dependencias entre mudanzas, e cando chega a hora de liberar unha nova versión, un conxunto concreto de mudanzas é aprobado para esa versión. A cuestión sobre se é mellor a centralización ou a descentralización é unha das guerras santas máis longas do desenvolvemento de software; tenta non caer na trampa de discutires sobre este asunto nas listas de correo do teu proxecto.

Checkout

O proceso de descarregar unha copia do proxecto desde un repositorio. Un *checkout* xera normalmente unha árbore de directorios chamada "copia de traballo" (ver debaixo), desde a que as mudanzas poden ser enviadas novamente ao repositorio orixinal. Nalgúns sistemas de control de versións descentralizados, cada copia de traballo é ao tempo un repositorio, e as mudanzas poden ser enviadas (ou traídas de) calquera outro repositorio que as acepte.

Copia de traballo

A árbore de directorios dun desenvolvedor que contén os ficheiros de código fonte e, posiblemente, as súas páxinas web ou outros documentos. Unha copia de traballo contén tamén un certo número de metadatos xeridos polo sistema de control de versións, que lle din de que repositorio vén, que "revisións" (ver máis abaixo) contén, etc. Xeralmente, cada desenvolvedor mantén a súa propia copia de traballo, na que fai e comproba as mudanzas, e desde a que fai os *commits*.

Conxunto de mudanzas (changeset), mudanza, revisión

Unha "revisión" é xeralmente a encarnación dun ficheiro ou directorio en particular. Por exemplo, se o proxecto comeza coa revisión 6 do ficheiro F, e despois alguén fai un *commit* cunha mudanza a F, isto produce a revisión 7 de F. Algúns sistemas usan tamén "revisión", "mudanza" ou "*changeset*" para se referiren a un conxunto de mudanzas subidas todas xuntas nun único *commit* como unha unidade conceptual

Estes termos teñen ocasionalmente diferentes significados técnicos en diferentes sistemas de control de versións, mais a idea máis estendida é sempre a mesma: son a maneira de falar con precisión acerca de puntos temporais concretos na historia dun ficheiro ou un conxunto de ficheiros (digamos, inmediatamente antes e despois de que un erro é reparado). Por exemplo: "ó, si, reparou ese problema na revisión 10" ou "reparou iso na revisión 10 de foo.c"

Cando un fala sobre un ficheiro ou colección de ficheiros sen especificar unha revisión en particular, xeralmente se asume que se está a referir a(s) revisión(s) máis recentes.

Diff

Representación textual dunha mudanza. Un "*diff*" mostra que liñas foron mudadas e como, ademais dunhas poucas liñas de contextualización desa información. Un desenvolvedor que estea familiarizado cun código dado, poderá ler o *diff* comparándoo co código e entender que foi o que se mudou, e mesmo localizar erros.

Etiqueta (tag)

Designan coleccións particulares de ficheiros pertencentes a unha revisión específica. As etiquetas xeralmente son usadas para preservar fitos interesantes dun proxecto. Por exemplo, frecuentemente se lle adxudica unha etiqueta a cada versión liberada, de tal xeito que se poida obter, directamente desde o sistema de control de versións, o conxunto exacto de ficheiros que compoñen esa versión. Os nomes que se empregan con frecuencia para as etiquetas poden ser do estilo de: *Versión_1_0*, *Distribución_00456*, etc.

Rama (branch)

Unha copia do proxecto, baixo o control de versións mais individualizada, tal que as mudanzas feitas á rama non afectan ao resto do proxecto, e viceversa, excepto cando as mudanzas son fusionadas (*merged*) desde unha rama a outra (ver debaixo). As ramas son tamén coñecidas como "liñas de desenvolvemento". Mesmo cando un proxecto non ten ramas, considérase que o desenvolvemento se está a facer na "rama principal" (*main branch*), tamén coñecida como "liña principal" (*main line*) ou" "tronco" (trunk).

As ramas ofrecen unha maneira de isolar entre si diferentes liñas de desenvolvemento que poderían desestabilizar o tronco principal. Ou alternativamente, unha rama pode ser empregada para estabilizar unha nova versión. Durante o proceso de liberación, o desenvolvemento normal do proxecto pode continuar na rama principal do repositorio; mentres tanto, na rama da liberación, non se permite introducir novas mudanzas distintas das aprobadas polos xestores da liberación. Así, facer unha liberación non ten por que interferir cos traballos de desenvolvemento. Podes consultar a definición de ramas máis adiante neste capítulo, se precisares máis detalles sobre a creación de ramas.

Fusionar ("merge", ou "port")

Trasladar unha mudanza entre ramas. Inclúe fusionar desde o tronco principal a unha rama ou viceversa. De feito, estas son precisamente as fusións máis frecuentes; é raro facer fusións entre ramas secundarias do proxecto. Podes ver máis información sobre este tema na sección "Singularidade da información" neste mesmo capítulo.

"Fusionar" (*merge*) ten un segundo significado, relacionado co anterior: é o que fai o sistema de control de versións cando detecta que dúas persoas fixeron mudanzas a un mesmo ficheiro, mais en aspectos que non se solapan. Dado que as dúas mudanzas non interfiren entre si, cando un dos desenvolvedores actualiza a súa copia do ficheiro (contendo aínda as súas propias mudanzas), as

mudanzas feitas polo segundo desenvolvedor fusiónanse automaticamente cos anteriores. Isto é moi frecuente, especialmente en proxectos nos que moitas persoas están traballando sobre o mesmo código. Cando dúas mudanzas diferentes *interfiren*, o resultado é un "conflito".

Conflito

É o que sucede cando dúas persoas tratan de facer mudanzas diferentes sobre a mesma porción de código. Todos os sistemas de control de versións detectan automaticamente os conflitos, e lle notifican a cando menos un dos desenvolvedores implicados que as súas mudanzas entran en conflito coas de alguén máis. *Resolver* o conflito e comunicarllo ao sistema fica entón na man desa persoa.

Bloquear (lock)

É unha maneira de declarar un intento de introducir mudanzas en exclusiva nun ficheiro ou directorio dado. Por exemplo: "Non podo subir ningunha mudanza ás páxinas web neste momento. Alfred as ten bloqueadas mentres corrixe as imaxes do fondo". Non todos os sistemas de control de versións ofrecen a posibilidade de facer bloqueos, e entre os que si o fan, non todos esixen que se empregue o bloqueo. Isto é así porque o desenvolvemento simultáneo e en paralelo é a norma, e bloquear a xente que pretende traballar sobre os ficheiros é (xeralmente) contrario a este ideal.

Os sistemas de control de versións que requiren do uso do bloqueo para poder subir mudanzas son coñecidos como modelos do tipo *bloqueo-modificación-desbloqueo*. Os que non, son denominados modelos *copia-modificación-refundido*. Unha comparación excelente entre os dous modelos pode ser consultada en http://svnbook.red-bean.com/svnbook-1.0/ch02s02.html. En xeral, o modelo copia-modificación-refundido considérase mellor para o desenvolvemento de software libre, e todos os sistemas de control de versións que se comentan neste libro soportan o mencionado modelo.

Seleccionando un sistema de control de versións

Cando este libro foi escrito, os dous sistemas de control de versións máis populares no mundo do software libre eran *Concurrent Versions System* (*CVS*, http://www.cvshome.org/) e *Subversion* (*SVN*, http://subversion.tigris.org/).

CVS está dispoñible desde hai moito tempo. Os desenvolvedores máis experimentados están familiarizados dabondo con el, fai máis ou menos o que precisas, e como é popular desde hai tempo, probablemente non teñas que enredarte en longas discusións sobre se é ou non a opción máis axeitada. Porén, CVS tamén ten algunhas desvantaxes. Non subministra unha maneira doada de tratar cen mudanzas multi-ficheiro; non te deixa renomear ou copiar os ficheiros dentro do sistema de control (así que tentar reorganizar a árbore do código do teu proxecto, unha vez xa iniciado, pode tornarse nun auténtico pesadelo); o seu soporte para as fusións é moi pobre; non manexa nada ben os ficheiros moi grandes ou os binarios; e algunhas operacións son moi lentas cando se fan sobre un número grande de ficheiros.

Ningún dos fallos de CVS é fatal, e aínda é moi popular. Porén, nos últimos anos o máis recente Subversion está a gañar terreo, especialmente nos proxectos máis novos[6]. Se estás iniciando un proxecto novo, eu recoméndoche Subversion

Pola contra, dado que eu estou participando no proxecto Subversion, a miña obxectividade pode ser razoablemente discutida. E nos últimos anos teñen xurdido un certo número de novos sistemas de control de versións en software libre. Na sección “Sistemas de control de versións”, presento unha listaxe , ordenadas máis ou menos por orde de popularidade. Como a listaxe deixa claro, decidirse por un sistema de control de versións podería tornarse nun proxecto vitalicio en si mesmo. Posiblemente vaste ver liberado desta decisión porque o fará por ti o sitio no que teñas hospedado o teu proxecto. Mais se es ti quen ten que tomar a decisión, consúltao co resto dos desenvolvedores, investiga que experiencia teñen ao respecto, e despois escolle un sistema e traballa con el. Calquera sistema de control de versións estable e listo para a produción vai satisfacer as túas necesidades; non deberías preocuparte moito sobre se tomaches unha decisión radicalmente errónea. Se simplemente non dás tomado unha decisión, entón elixe Subversion. É moi doado de aprender, e seguramente vai seguir sendo un estándar por moitos anos.

6 Véxase <ulink url="http://cia.vc/stats/vcs"></ulink> e <ulink url="http://subversion.tigris.org/svn-dav-securityspace-survey.html"></ulink> onde se comentan probas deste crecemento.

Usando o sistema de control de versións

As recomendacións desta sección non se dirixen a un sistema de control de versións en particular, e deberían ser doadas de poñer en práctica en calquera deles. Consulta a documentación do teu sistema específico se precisares máis detalles.

Versiona todo.

Debes manter no teu sistema de control de versións non só o código fonte do teu proxecto, senón tamén as súas páxinas web, documentación, FAQ, notas de deseño, e calquera outra cousa que a xente poida querer editar. Mantenas xusto a carón do código, na mesma árbore do repositorio. Calquera peza de información que pague a pena escribir, paga igualmente a pena versionar, isto é, calquera peza de información que poida ser modificada. As cousas que non están sometidas a mudanzas deberían ser arquivadas, non versionadas. Por exemplo, un email, unha vez enviado, non vai mudar; polo tanto, non tería sentido pensar que pode ser versionado (a non ser que forme parte dun documento máis longo e que si evolúa no tempo).

A razón pola que é importante versionar todo nun único sitio é que deste xeito a xente soamente terá que aprender un mecanismo para incorporar mudanzas. Frecuentemente, un colaborador vai comezar por enviar mudanzas ás páxinas web ou á documentación, e comezará máis tarde a facer pequenas contribucións ao código, por exemplo. Cando o proxecto empregue o mesmo sistema para todos os tipos de contribucións, a xente só terá que afacerse a el unha vez. Manter todas as versións xuntas significa tamén que as novas características poden ser incorporadas xunto coa súa documentación, e que distribuír o código por varias ramas vai supor que a documentación tamén se distribúe, etc.

Non manteñas os *ficheiros xerados* no sistema de control de versións. Non son propiamente datos editables, xa que son xerados automaticamente a partir de outros ficheiros. Por exemplo, algúns sistemas de montaxe (*build*) crean o ficheiro *configure* a partir do molde *configure.in*. Para faceres unha mudanza a *configure*, terías que editar *configure.in* e posteriormente rexenerar o primeiro; así, tan só o molde *configure.in* é un ficheiro "editable". Polo tanto, mantén no control de versións tan só os moldes (se incluíres os ficheiros resultantes, a xente vai inevitablemente esquecer rexeneralos cando incorporaren unha mudanza a un molde, e a correspondente falta de cohesión non vai causar máis que confusión).

Mais se quixeres ler unha segunda opinión acerca da cuestión dos ficheiros *configure*, podes revisar o comentario de Alexey Makhotkin "*configure.in and version control*" en http://versioncontrolblog.com/2007/01/08/configurein-and-version-control/.

A regra de que todo dato editable debe ser mantido nun sistema de control de versións ten infelizmente unha excepción: o xestor de erros (*bug tracker*). As bases de datos de erros conteñen multitude de datos editables, mais por cuestións técnicas non se pode, xeralmente, ter eses datos almacenados nun sistema de control de versións. (Algúns xestores de erros incorporan de seu algunhas funcionalidades moi primitivas de control de versións, mais en calquera caso independentes do repositorio principal do proxecto).

Navegabilidade

O repositorio do proxecto debe ser navegable desde a web. Isto inclúe non só a habilidade para ver a última revisión dos ficheiros do proxecto, senón tamén poder ir atrás no tempo e visitar revisións anteriores, ver as diferencias entre elas, ler as mensaxes de seguimento (*log*) de mudanzas concretas, etc.

A navegabilidade é importante porque supón o uso dun portal lixeiro de acceso aos datos do proxecto. Se o repositorio non pode ser visitado desde un navegador web, entón alguén que pretenda ver un ficheiro en particular (nomeadamente: comprobar se unha corrección dun erro foi xa incorporada ao código), primeiro tería que instalar localmente un cliente do sistema de control de versións, o que podería tornar a súa tarefa inicial de un par de minutos nun traballo de polo menos media hora.

Implica tamén a existencia de URL normalizadas para acceder a revisións específicas dos ficheiros, e para visitar a última revisión nun momento dado. Isto pode ser moi útil cara a discusións técnicas ou cando se trata de dirixir a xente cara á documentación. Por exemplo, en lugar de dicir "Para informarse sobre instrucións para reparar o servidor, diríxase ao ficheiro www/hacking.html na súa copia local", podería dicirse"Para informarse

sobre instrucións para reparar o servidor, visite *http://svn.collab.net/repos/svn/trunk/www/hacking.html"* proporcionando unha URL que vai dirixir sempre á última revisión do ficheiro *hacking.html*. A URL é mellor xa que carece totalmente de ambigüidade, e evita o problema de se o destinatario ten unha copia de traballo completamente actualizada.

Algúns sistemas de control de versións incorporan mecanismos internos de navegación polo repositorio, mentres que outros empregan ferramentas de terceiros. Estas ferramentas son: *ViewVC* (http://viewvc.org/), *CVSWeb* (http://www.freebsd.org/projects/cvsweb.html), e *WebSVN* (http://websvn.tigris.org/). A primeira traballa sobre CVS e Subversion, a segunda só sobre CVS, e a terceira só sobre Subversion.

Mensaxes de incorporación de mudanzas (commit emails)

Toda incorporación de mudanzas ao repositorio (*commit*) debería xerar unha mensaxe na que conste quen fixo a mudanza, cando a fixo, que ficheiros e directorios mudou, e en que consistiron as mudanzas. A mensaxe debería ser enviada a unha lista de correo específica, distinta das empregadas para xestionar as mensaxes dos colaboradores. Deberíase recomendar aos desenvolvedores e outras persoas interesadas que se desen de alta na lista de mensaxes de mudanzas, xa que este é o modo máis efectivo de manter o coñecemento sobre que é o que se está a facer no proxecto a respecto do código. Para alén das evidentes vantaxes técnicas da revisión entre iguais (véxase a sección “Revisión continua e intensiva do código” no capítulo “Primeiros pasos”) as mensaxes de incorporación de mudanzas axudan a crear un espírito de comunidade, xa que dotan ao proxecto dun entorno común onde a xente pode reaccionar ante os eventos (incorporación de mudanzas) que saben que están visibles tamén para os demais.

As cuestións específicas sobre a configuración de mensaxes de incorporación de mudanzas varían dependendo de cal sexa o teu sistema de control de versións, mais xeralmente hai un *script* ou outra clase de utilidade para facelo. Se tiveres problemas para atopala, tenta buscar na documentación por *hooks*, e especialmente *post-commit hook*, tamén chamado *loginfo hook* en CVS. Os "*post-commit hooks*" constitúen un modo xeral de lanzar tarefas automatizadas en resposta á incorporación de mudanzas. O "*hook*" dispárase ante a incorporación dunha mudanza en concreto, aliméntase da información acerca desa mudanza e é logo libre de usar esa información para realizar calquera outra operación (por exemplo, enviar un correo electrónico).

No caso dos sistemas de envío de mensaxes de actualización pre-empacotados, talvez queiras modificar algúns dos comportamentos por defecto:

1. Algúns destes sistemas non inclúen os "*diff*" no email, mais inclúen unha URL na que consultar as mudanzas no repositorio a través do navegador web. Aínda que fornecer a URL está ben, de xeito que se poida facer referencia á mudanza máis tarde, é tamén *moi* importante que a mensaxe de envío da mudanza inclúa os propios *diff*. A lectura dos emails é parte da rutina da xente, así que se o contido das mudanzas é visible xusto no mesmo email de mudanzas, os desenvolvedores revisarán a mudanza sobre a marcha, sen abandonar o seu lector de emails. Se teñen que premer unha ligazón para poder revisalo, os máis non o farán, xa que isto require unha nova acción en lugar da continuación do que xa estaban a facer. Aínda máis, se o revisor quixer preguntar algo sobre a mudanza feita, é considerablemente máis sinxelo responder á mensaxe e incluír os comentarios sobre o rexistro de diferencias directamente sobre o propio texto do que é visitar a páxina web e cortar e pegar partes do rexistro no cliente de email.

   (Desde logo, se o reporte de diferencias é moi grande, como no caso de que se teña engadido ao repositorio un corpo de texto completamente novo, terá sentido omitir a información do "*diff*" e achegar tan só a URL. Os máis dos programas de envío de correos de actualización poden facer este tipo de discriminación automática. Se o teu non pode, será mellor manteres a inclusión dos "*diff*" e resignáreste a sufrir mensaxes ocasionalmente enormes, que eliminárelos por completo. A facilidade para facer revisións e comentarios é de vital importancia para o desenvolvemento colaborativo, demasiado para deixalos de lado)

2. Os correos de subida de mudanzas deberían ter o cabezallo "*Reply-to*" dirixido á lista normal de desenvolvemento, non á de mudanzas. É dicir, cando alguén revisa unha mudanza e remite un comentario á mesma, a súa resposta debería ser automaticamente encamiñada á lista dos desenvolvedores humanos, na que normalmente se discuten os asuntos técnicos. Hai unhas cantas

razóns para facelo así. Primeiro, todas as discusións técnicas deben facerse nunha única lista xa que iso é o que a xente espera que ocorra, e porque dese xeito hai un só arquivo no que facer pesquisas. En segundo lugar, talvez haxa colaboradores que non estean subscritos á lista de mudanzas. Terceiro, a lista de correo de mudanzas se autopublicita como un servizo para facer seguimento das mudanzas, non para facer seguimento das mudanzas *e* de discusións técnicas ocasionais. Cando un se subscribe á lista de correo de mudanzas, non se está subscribindo ademais a ningunha outra lista; enviarlles outros materiais a través da lista violaría un contrato implícito. En cuarto lugar, a xente escribe frecuentemente pequenos programas que len a lista de mudanzas e procesan os resultados (de cara á súa publicación nunha páxina web, por exemplo). Eses programas están preparados para manipular mensaxes de correo-e formatadas de forma consistente, mais non mensaxes incoherentes escritas por humanos.

Ten en conta que esta recomendación acerca da configuración do cabezallo "*Reply-to*" non refuta o comentado anteriormente neste capítulo. Normalmente o *remitente* dunha mensaxe vai considerar correcto establecer o "*Reply-to*". Neste caso, o remitente é o propio sistema de control de versións, e configura o "*Reply-to*" para indicar que o lugar apropiado para responder ás súas mensaxes é a lista de desenvolvemento, non a de mudanzas.

CIA: outro mecanismo de publicación de mudanzas (Another Change Publication Mechanism)

As mensaxes de mudanzas non son a única maneira de propagar as novas sobre as mudanzas feitas. Recentemente desenvolveuse un novo mecanismo denominado CIA (http://cia.navi.cx/). CIA é un agregador e distribuidor en tempo real de estatísticas sobre mudanzas. O uso máis popular de CIA é o de enviar notificacións de agregación de mudanzas a canais IRC, de xeito que a xente que está conectada a eses canais pode ver a incorporación das mudanzas en tempo real. Aínda que menos útil tecnicamente que as mensaxes de mudanzas, xa que os observadores poden estar conectados ao canal ou non cando aparece no IRC unha nova sobre unha mudanza, esta técnica ten unha inmensa utilidade *social*. A xente terá a sensación de formar parte de algo vivo e activo, e sentirá que se progresa xusto ante os seus ollos.

A maneira en que traballa consiste en que ti invocas o programa CIA de notificación desde o gancho (*hook*) do teu sistema de publicación de mudanzas. O notificador formata a XML a información da mudanza, e a envía como mensaxe a un servidor central (típicamente *cia.navi.cx*). A continuación, ese servidor distribúe a información sobre a mudanza a outros foros.

CIA pode ser configurado tamén para enviar comunicacións RSS (http://www.xml.com/pub/a/2002/12/18/dive-into-xml.html). Podes consultar os detalles na documentación do proxecto en http://cia.navi.cx/.

Para veres un exemplo de CIA en acción, accede desde o teu cliente IRC a *irc.freenode.net*, canal *#commits*.

Usa as ramas para evitares gargalos

Os usuarios non expertos dos sistemas de control de versións ás veces lle teñen un certo medo á creación de ramas (*branching*) e á posterior fusión (*merging*). Trátase probablemente dun efecto colateral da popularidade dos CVS: a súa interface de creación de ramas e fusión é pouco intuitiva, de xeito que moita xente aprendeu a evitar esas operacións por completo.

Se ti es unha destas persoas, toma a decisión xa mesmo de superares eses medos e toma o tempo preciso para aprenderes como facer a creación de ramas e a fusión. Non son operacións difíciles, unha vez que estás afeito a elas, e tórnanse máis e máis importantes segundo o proxecto vai incorporando máis desenvolvedores.

As ramas son valiosas xa que converten un recurso escaso (espazos de traballo no código do proxecto) nun considerablemente máis abundante. Normalmente todos os desenvolvedores traballan conxuntamente sobre unha mesma versión do software, construíndo un único castelo. Cando alguén quere engadir unha nova torreta, mais non é quen de convencer o resto de que se debería facer iso, a creación dunha rama vaille dar a oportunidade de facelo nunha esquiniña. Se o esforzo ten éxito, pode convidar os outros desenvolvedores a examinaren os resultados. Se todos estiveren de acordo en que o resultado paga a pena, entón poden dicirlle ao sistema de control de versións que mova (fusione -*merge*) a torreta desde a rama ao castelo principal.

É doado ver como esta habilidade axuda ao desenvolvemento colaborativo. A xente precisa da liberdade de

probar cousas novas sen sentir que está a interferir co traballo dos demais. Igualmente importante, hai momentos en que o código precisa ser isolado das tarefas habituais, para corrixir un erro ou estabilizar unha versión para ser liberada (véxase as seccións “Estabilizando unha publicación” no capítulo 7 “Empacotamento, publicación e desenvolvemento diario”) sen ter que se preocupar acerca de manter controlada unha diana móbil.

Usa as ramas con liberalidade, e aconsella os demais a que tamén o fagan. Mais tamén debes estar seguro de que cada rama vai permanecer activa exactamente o tempo estritamente necesario. Cada rama é unha pequena fuga na atención da comunidade. Mesmo os que non están a traballar na rama manteñen un interese periférico sobre o que está a acontecer nela. Ese interese é desexable, por suposto, e deberíanse enviar correos de incorporación de mudanzas exactamente igual que no caso dos correspondentes ao corpo principal do proxecto. Mais as ramas non deben tornar nun mecanismo de división da comunidade de desenvolvemento. Salvo raras excepcións, o obxectivo final das máis das ramas debería ser o de que as súas mudanzas sexan reintegradas á liña principal e que a rama desapareza.

Singularidade da información

A fusión (*merging*) ten un corolario importante: nunca envíes unha mesma mudanza dúas veces. Isto é, cada mudanza dada debería ser incorporada ao sistema de control de versións exactamente unha soa vez. A revisión (ou conxunto de revisións) na que a mudanza foi incorporada será o seu identificador único de aí en diante. Se for necesario incorporala a outras ramas distintas daquela na que tivo a súa orixe, entón debería ser fusionada (*merged*) desde o seu punto de entrada orixinal cara a eses outros destinos (en oposición á integración dunha mudanza textualmente idéntica, que tería o mesmo efecto sobre o código, mais que tornaría imposibles o arquivamento preciso e a xestión de liberacións).

Os efectos prácticos deste consello van diferir dun sistema de control de versións a outro. Nalgúns sistemas, as fusións son acontecementos especiais, fundamentalmente diferentes da incorporación de mudanzas, e presentan o seu propio conxunto de metadatos. Noutros, o resultado das fusións é incorporado exactamente igual que calquera outra mudanza de xeito que o medio fundamental para distinguir unha "incorporación dunha fusión" dunha "incorporación dunha mudanza" estará nos contidos da mensaxe de rexistro. Nunha mensaxe de rexistro dunha fusión nunca debes repetir a mensaxe de rexistro da mudanza orixinal. Pola contra, debes indicar tan só que se trata dunha fusión, e asígnalle o identificador de revisión da mudanza orixinal, engadindo como moito un breve resumo dos seus efectos. Se alguén quixer ver a mensaxe de rexistro completa, deberá consultar a revisión orixinal.

A razón pola que a repetición das mensaxes de rexistro é importante é que ás veces son editadas despois de enviada a mudanza. Se a mensaxe de rexistro de cada mudanza fose repetida en cada destino de fusión, ao editar alguén a mensaxe orixinal, quedarían sen corrixir as distintas repeticións, o que simplemente causaría confusión no futuro.

O mesmo principio pode ser aplicado cando se reverte unha mudanza. No caso de que unha mudanza sexa descartada do código, a única mensaxe de rexistro que debería engadirse debería tan só especificar que unha revisión concreta está a ser descartada, *nunca* se debería describir a mudanza resultante desa operación, xa que a semántica da mudanza pode derivarse mediante a simple lectura da mensaxe de rexistro da mudanza orixinal. Por suposto, a mensaxe de rexistro do descarte debería igualmente deixar claro por que se decide ese descarte, mais non debería duplicar ningunha información procedente da mensaxe orixinal de rexistro. Sempre que for posible, o que si se debe facer é editar esa mensaxe orixinal para explicar nela que a mudanza á que se refire foi finalmente descartada.

Todo o que antecede implica que se debería empregar unha sintaxe coherente para referirse ás revisións. Isto non é só importante para as mensaxes de rexistro, senón tamén nas de correo electrónico, o xestor de erros (*bug tracker*), e en toda a documentación do proxecto. Se estás a usar CVS, suxírote *path/to/file/in/project/tree:REV*, na que REV é un número de revisión concreto do CVS tal como "1.76". Se estás a empregar Subversion, a sintaxe estándar para as revisións 1729 é "r1729" (as rutas dos arquivos non son precisas xa que Subversion emprega números de revisión globais). Noutros sistemas, normalmente hai tamén unha sintaxe estándar para denominar o conxunto de mudanzas. Calquera que for a sintaxe axeitada para o teu sistema, anima a xente a usárena para referírense ás mudanzas. A expresión coherente dos nomes de mudanzas axuda a facer máis doado o arquivamento do proxecto (como veremos nos capítulos 6 “Comunicacións” e 7 “Empacotamento, publicación e desenvolvemento diario”) e como unha boa parte do

traballo de arquivamento vai ser feita por voluntarios, mellor mantelo tan simple como for posible.

Podes ver tamén a sección “Liberación e desenvolvemento diario” no capítulo 7 “Empacotamento, liberación e desenvolvemento diario”.

Autorización

Os máis dos sistemas de control de versións ofrecen unha característica pola cal determinadas persoas poden ser autorizadas ou non a introduciren mudanzas en subáreas concretas do repositorio do proxecto. Seguindo o principio de que cando agarras un martelo, a xente comeza a ollar ao redor en busca de cravos, moitos proxectos empregan esta característica con abandono, garantindo coidadosamente que a xente teña acceso a xusto aquelas áreas para as que están autorizados a subiren mudanzas, e que non poidan facelo en ningunha outra parte. (Podes consultar a sección “Commiters” no capítulo 8 “Xestión de voluntarios” para entenderes como se decide quen vai poder introducir mudanzas en que partes do proxecto)

Probablemente haxa pouco risco de danos por exercer un control tan estrito, mais unha política máis relaxada é tamén viable. Algúns proxectos simplemente usan un sistema honorífico: cando alguén recibe autorización para acceder ao sistema de control de mudanzas, mesmo para unha subárea do repositorio, o que en realidade están a recibir é un pasaporte para poderen introducir mudanzas en calquera parte do proxecto. Simplemente se lles suxire que manteñan as súas mudanzas na súa área. Ten en conta que non existe ningún risco real en facelo así: nun proxecto activo, todos as mudanzas integradas son revisadas por alguén. Se un colaborador introduce mudanzas nunha área na que non está autorizado a traballar, outros vanse decatar e o van comentar. Se for preciso desfacer unha mudanza, chega con iso (todo está sometido ao sistema de control de versións, así que simplemente terás que reverter a mudanza).

Hai moitas vantaxes nunha aproximación relaxada. Primeiro, segundo os desenvolvedores van expandíndose por outras áreas (o que con seguridade vai ocorrer, se é que se manteñen no proxecto), non hai ningunha dificultade administrativa en ampliarlles os privilexios de acceso. Unha vez que se toma a decisión, a persoa en cuestión pode comezar inmediatamente a traballar na nova área.

Segundo, a expansión pode ser feita dun xeito máis granular. Xeralmente, un desenvolvedor da área X que queira expandir o seu interese á área Y, comezará a publicar contribucións a Y e solicitar que sexan revisadas. Se alguén que xa tiña privilexios de acceso a Y ve unha das mencionadas contribucións e a aproba, simplemente poden indicarlle ao desenvolvedor que incorpore a mudanza directamente ao repositorio (mencionando o nome do revisor na mensaxe de rexistro, por suposto). Así, a incorporación da mudanza vai ser feita por quen escribiu a mudanza, o que é máis axeitado tanto desde o punto de vista da xestión da información como desde o da acreditación de autorías.

Por último, e pode que sexa o máis importante, empregar un sistema meritocrático crea unha atmosfera de confianza e respecto mutuos. Darlle a alguén acceso a un subdominio é un recoñecemento da súa preparación técnica (;é como dicir: "Ei!, xa vemos que tes experiencia como para incorporar mudanzas neste dominio, así que adiante"). Pola contra, establecer un sistema moi estrito de control de accesos é como dicir: "Non só estamos poñendo en dúbida a túa experiencia, tamén temos sospeitas sobre as túas *intencións*." E ese é o tipo de frases que non che gustaría empregar , se é que podes evitalo. Incorporar alguén ao proxecto como desenvolvedor é unha oportunidade para inicialo no círculo de confianza mutua. Unha boa maneira de facelo é concederlle máis poder do que se lle supón que vai usar, e despois informarlles que está nas súas mans manterse dentro dos límites acordados.

O proxecto Subversion vén traballando co sistema meirotcrático desde hai máis de catro anos, con 33 colaboradores sobre a totalidade do proxecto e 43 sobre partes do mesmo cando foi escrita a presente obra. A única distinción que o sistema mantén realmente por si mesmo é entre quen pode subir mudanzas e quen non; as restantes divisións son mantidas exclusivamente por humanos. A día de hoxe non tivemos ningún problema con alguén que pretendera incorporar mudanzas fóra do seu dominio deliberadamente. Unha vez ou dúas houbo algunha falta de entendemento acerca de ata onde chegan exactamente os privilexios de alguén, mais sempre foron resoltas rápida e amigablemente.

Obviamente, en situacións nas que o autocontrol non é práctico, deberás establecer controis de autenticación severos. Mais estas situacións son pouco frecuentes. Mesmo cando se está a falar de millóns de liñas de código e centos ou milleiros de desenvolvedores, a incorporación dunha mudanza a un módulo de código

concreto debería ser revisada por quen traballa nese módulo preciso, e son eles quen poden recoñecer se alguén que non debería está a enviar mudanzas. Se a revisión regular de mudanzas a incorporar *non* está a funcionar axeitadamente, entón é que o proxecto ten problemas máis importantes que resolver que os relativos ao sistema de autenticacións.

En resumo, non pases moito tempo chafullando no sistema de autenticación do control de versións, non sendo que teñas unha razón específica para facelo. Xeralmente non vai producir ningún beneficio tanxible e, porén, atoparás moitas vantaxes en deixar o control en mans dos humanos.

Nada do que antecede debe ser tomado no sentido de que as restricións carecen de importancia en si mesmas, por suposto. Pode ser desastroso para un proxecto deixar que a xente incorpore mudanzas en áreas para as que non están cualificados. Aínda máis, en moitos proxectos, o acceso completo (sen restricións) responde a un status especial: implica gozar do dereito a votar en cuestións que afecten a calquera das partes do proxecto. Este aspecto político dos accesos á incorporación de mudanzas é discutido máis a fondo na sección “Quen vota?” do capítulo 4 “Infraestructura social e política”.

Xestor de erros (Bug Tracker)

A xestión de erros é unha cuestión moi ampla; varios dos aspectos que lle afectan son discutidos ao longo deste libro. Aquí eu vou tratar de concentrarme principalmente na súa configuración e máis en consideracións técnicas, mais antes de chegar a ese punto, temos que comezar cunha cuestión política: exactamente que información debe ser mantida nun xestor de erros?

O termo *bug tracker* é confuso. Os sistemas de xestión de erros son usados frecuentemente para facer seguimento de requirimentos de novas características, tarefas unitarias, remendos non solicitados (en realidade calquera cousa que presenta distintos estados de inicio e de fin, con estados opcionais de transición no medio, o que fai incrementar a información asociada ao longo do tempo). Por esta razón os xestores de erros son tamén denominados *xestores de asuntos (issue trackers), xestores de defectos (defect trackers), xestores de artefactos (artifact trackers), xestores de requisitos (request trackers), sitemas de tickets de erros (trouble ticket systems)*, etc. Podes consultar unha lista de software na sección “Xestores de erros”.

Neste libro, eu vou continuar a falar de "xestor de erros” (*bug tracker*) cando me refira ao software que fai o seguimento, xa que é así como a maioría da xente lle chama, mais vou empregar asunto (*issue*) para referirme a unha instancia individual na base de datos do xestor de erros. Isto vainos permitir distinguir entre o comportamento ou fallo que o usuario ten atopado (isto é, o erro ou "*bug*" en sí mesmo), e os *rexistros* referidos ao descubrimento, diagnose e eventual resolución do erro. Ten en conta que aínda que moitos asuntos se refiren a erros reais, tamén poden referirse igualmente ao seguimento de outros tipos de tarefas.

O clásico ciclo de vida dun asunto é semellante a:

1. Alguén detecta o erro. Subministra un resumo, unha breve descrición do mesmo (incluíndo unha receita para reproducir o fallo, no seu caso; bótalle unha ollada á sección “Trata a cada usuario como un potencial voluntario” no capítulo 8 “Xestión de voluntarios” para entenderes como se debe pedir á xente que faga bos informes de erros), así como calquera outra información que o xestor de erros pida. A persoa que detecta o asunto pode ser totalmente descoñecida para o proxecto (os informes sobre erros e o requirimento de novas funcionalidades terán as mesmas probabilidades de seren solicitados tanto por usuarios como polos desenvolvedores).
   Unha vez reportado, o asunto fica rexistrado no estado coñecido como *aberto*. Como de momento non se realizou ningunha operación sobre el, algúns xestores de erros o denominan *non verificado* e/ou *sen iniciar.* Non está aínda asignado a ninguén; ou, nalgúns xestores, se lle asigna a un usuario ficticio, o que vén a representar a ausencia de asignación. Neste punto, está nun espazo de espera: o asunto foi arquivado, mais aínda non está integrado na conciencia activa do proxecto.
2. Outra xente vai ler o asunto, engadirlle comentarios, e talvez preguntarlle ao remitente orixinal sobre detalles que clarifiquen algún punto escuro.
3. O erro é entón *reproducido*. Este pode ser o momento máis importante no seu ciclo de vida. Aínda que o erro non está corrixido, o feito de que alguén ademais do remitente orixinal sexa quen de reproducilo vai demostrar que é auténtico, e , non menos importante, vai confirmar a quen o detectou por primeira vez que contribuíu ao proxecto cun erro real.

4. O erro é entón *diagnosticado*: identifícase a causa que o produce, e se for posible, estímase o esforzo requirido para corrixilo. Estes procesos deben ficar gravados no sistema; se a persoa que fai o diagnóstico desaparece repentinamente do proxecto por un tempo (o que vai suceder frecuentemente cos desenvolvedores voluntarios), calquera outro deberá ser quen de retomar a xestión do erro onde o outro o deixou.
   Neste paso, ou ás veces no anterior, un desenvolvedor vaise "apropiar" do asunto e *autoasignalo a el mesmo* (máis información na sección “Distingue entre petición e asignación” do capítulo 8 “Xestión de voluntarios”) examina o proceso con máis detalle). A *prioridade* pode tamén ser establecida neste paso. Por exemplo, se for tan severo que podería atrasar a seguinte liberación, ese asunto precisa ser identificado logo, e o xestor debería proporcionar algunha maneira de notificalo.
5. O asunto é entón programado para ser resolto. A programación non ten por que incluír a determinación dunha data límite para a súa corrección. Ás veces implica simplemente determinar en que liberación futura (non necesariamente a seguinte) o erro debería estar solucionado, ou decidir que non é preciso bloquear ningunha liberación en particular. A programación pode tamén ser ignorada, se o erro for doado de resolver.
6. O erro é resolto (ou a tarefa completada, ou o remendo aplicado, ou o que for). A mudanza ou conxunto de mudanzas resultantes da súa resolución deberán ser gravadas nun comentario, e posteriormente o asunto é *pechado* e/ou marcado como *resolto*.

Hai algunhas variacións frecuentes sobre este ciclo de vida. Ás veces un asunto é pechado moi cedo tras ser comunicado, porque resulta non ser realmente un erro, senón un fallo de utilización por parte do usuario. Segundo o proxecto vai tendo máis usuarios, máis destes falsos erros van ser comunicados, e os desenvolvedores van proceder a pechalos con comentarios cada vez máis intempestivos. Tenta evitar esta última tendencia. Non lle fai ningún ben a ninguén, xa que o usuario que comunica o erro en cada caso non é responsable do todas as comunicacións previas; a tendencia estatística é visible só desde o punto de vista dos desenvolvedores, non dos usuarios.(Máis adiante neste capítulo, na sección “Pre-filtrando erros”, veremos algunhas técnicas destinadas a reducir o número de falsos erros). Igualmente, se diferentes usuarios están a cometer o mesmo fallo de operación unha e outra vez, poderíase pensar que esa parte do software precisa ser redeseñada. Este tipo de padróns son máis doados de detectar cando hai un xestor de asuntos monitorizando a base de datos de erros; véxase “Xestor de incidencias” no capítulo 8 “Xestión de voluntarios”.

Outra variación frecuente do ciclo de vida consiste en pechar o asunto como *duplicado* xusto despois do Paso 1. Un duplicado é un asunto que xa é coñecido polos responsables do proxecto. Os duplicados non afectan exclusivamente a asuntos abertos: é posible que un erro volva manifestarse mesmo despois de ter sido resolto (isto é coñecido como *regresión*), en cuxo caso o procedemento preferido consiste en reabrir o asunto orixinal e pechar calquera nova notificación do mesmo como duplicados do anterior. O sistema de xestión de erros debería facer seguimento bidireccional desta relación, de xeito que a información sobre reprodución contida nos duplicados estea dispoñible para o orixinal, e viceversa.

Unha terceira variación consiste en que os desenvolvedores pechen o asunto, pensando que o resolveron, e ver que o informante orixinal rexeita a solución e o reabre. Isto xeralmente ocorre porque os desenvolvedores simplemente non teñen acceso ao entorno necesario para reproduciren o fallo, ou porque non o reproduciron seguindo exactamente a mesma receita achegada polo informante para a súa reprodución.

Para alén destas variacións, poden darse outros aspectos do xestor de erros que poden variar dependendo do software empregado. Mais o aspecto básico é o mesmo, e mentres o ciclo de vida en si mesmo non é especifico do software de código aberto, ten implicacións concretas en como os proxectos de software aberto usan os seus xestores de erros.

Tal como se deduce do paso 1, o xestor é tan "fachada pública" do proxecto como poidan selo as listas de correo ou as páxinas web. Calquera pode rexistrar un asunto, calquera pode facer seguimento dun asunto, e calquera pode explorar a lista de asuntos actualmente abertos. Ocorre que non podes coñecer canta xente está pendente de como se resolve un asunto. Mentres que o tamaño e a habilidade da comunidade van reducir o prazo en que os asuntos son resoltos, o proxecto debería en todo caso ser quen de tomar coñecemento de cada asunto xusto no momento en que é detectado. Mesmo se un asunto permanece activo durante un tempo, unha resposta vai favorecer que o informante permaneza interesado, xa que se vai decatar de que alguén tomou nota do que el reportou (lembra que preencher un formulario de notificación dun asunto leva máis tempo que, por exemplo, escribir un correo electrónico). Aínda máis, unha vez que un asunto é revisado por un

desenvolvedor, pasa a formar parta da "consciencia" do proxecto, no senso de que ese desenvolvedor pode estar xa analizando outras instancias do asunto, pode falar sobre o tema con outros colegas, etc.

A necesidade de reaccionar a tempo implica dúas cousas:

- O xestor debe estar conectado a unha lista de correo, de xeito que cada mudanza sobre un asunto, incluíndo a súa notificación inicial, debe producir unha mensaxe de correo que describa o que aconteceu. Esta lista de correo é xeralmente distinta da de desenvolvemento, xa que non todos os desenvolvedores poden querer recibir mensaxes automatizadas referidas a erros, mais (igual que nas mensaxes de inserción de mudanzas) o cabezallo "*Reply-to*" debería ser dirixido á lista xeral de desenvolvemento.
- O formulario de notificación de asuntos debería capturar o enderezo de correo do remitente, de xeito que poida ser contactado para pedirlle máis información. (Porén, non debería *requirir* ese enderezo, dado que hai xente que prefire notificar erros de xeito anónimo. Véxase “Confidencialidade” máis adiante neste capítulo para ler información adicional sobre a importancia de poder manter o anonimato.)

Interacción con listas de correo

Asegúrate de que o xestor de erros non se convirte nun foro de discusión. Aínda que é importante manter a presencia humana no sistema, non está preparado para soportar discusións en tempo real. Máis ben debes pensar nel como un arquivador, un xeito de organizar feitos e referencias cara a outras discusións, basicamente aquelas que teñen lugar nas listas de correo.

Hai dúas razóns para facer esta distinción. Primeiro, o xestor de erros e máis complicado de usar que as listas de correo (ou que os chats en tempo real). Isto non é así debido a que o xestor teña unha interface de usuario mal deseñada, senón a que as súas interfaces foron deseñadas para capturar e presentar estados discretos, non discusións libres. En segundo lugar, non todo o mundo que está a seguir unha determinada discusión sobre un asunto ten por que estar conectado ao xestor de erros. En parte a boa xestión de asuntos (véxase “Xestión compartida” no capítulo 8 “Xestión de voluntarios”) ten a ver con asegurar que a información correcta está sendo posta a disposición da xente axeitada, máis que con pretender que todos os desenvolvedores estean pendentes de todos os asuntos. En “Uso do xestor de erros” no capítulo 6 “Comunicacións” veremos algunhas maneiras de asegurar que a xente non lance discusións fora dos foros apropiados e no xestor de erros.

Algúns xestores de erros poden monitorizar as listas de correo e rexistrar automaticamente todas as mensaxes que teñen a ver cun asunto coñecido. Tipicamente fan isto mediante o recoñecemento do identificador do erro na liña de "asunto" das mensaxes para atraeren a atención do xestor. Este ou ben gardará toda a mensaxe, ou (aínda mellor) gravará unha ligazón á mensaxe gardada nos arquivos da lista de correo. En calquera caso, esta é unha característica moi útil; se o teu xestor dispón dela, asegúrate de activala e de recomendar á xente a súa utilización.

Prefiltrando o xestor de erros

As máis das bases de datos de asuntos sofren de cando en vez o mesmo problema: unha cantidade crítica de asuntos duplicados ou inválidos procedentes de usuarios ben intencionados mais inexpertos ou mal informados. O primeiro paso para combater esta tendencia consiste en colgar un aviso ben visible na páxina de entrada ao xestor de erros no que se explique como saber se un erro é realmente un erro, como buscar se xa foi notificado e, finalmente, como facer correctamente a notificación se se segue a pensar que se trata realmente dun erro novo.

Ese sistema reduciría o nivel de ruído por un tempo, mais segundo aumenta o número de usuarios, o problema vai retornar. Ningún usuario individual pode ser culpado diso. Cada un deles está pretendendo contribuír ao benestar do proxecto, e mesmo se a primeira notificación do erro non é útil, deberás continuar animándoos a permaneceren involucrados e a que fagan mellores contribucións no futuro. Mentres tanto, porén, o proxecto vai precisar manter a base de datos de asuntos o máis libre de lixo posible.

As dúas cousas que mellor van contribuír a previr este problema son: estares seguro de que vai haber xente suficientemente capacitada controlando o que está a ocorrer no sistema de xestión de erros como para seres

quen de pechares notificacións inválidas ou duplicadas no momento xusto en que entran, e requirires (ou recomendares fortemente) que os usuarios confirmen con outros usuarios os erros detectados por eles antes de reportalos ao sistema.

Semella que a primeira das técnicas mencionadas está a ser usada universalmente. Mesmo proxectos con bases de datos inmensas (nomeadamente, o xestor de erros de Debian en http://bugs.debian.org/, que no momento de escribir este texto contiña 315.929 asuntos[7] aínda organizan as cousas de xeito que *alguén* ve os asuntos que van chegando ao sistema. Pode ser unha persoa diferente para cada categoría de asuntos. Por exemplo, o proxecto Debian é unha colección de pacotes de software, así que Debian encamiña automaticamente cada novo asunto aos mantedores do correspondente pacote. Por suposto, os usuarios poden ás veces trabucarse na identificación das categorías ás que corresponden os erros detectados, co resultado de que o erro é enviado inicialmente á persoa incorrecta, quen deberá encargarse de encamiñalo novamente. De calquera maneira, o importante é que a carga de traballo é compartida, de todos modos (se o usuario tiver dúbidas no momento de reportar un asunto, a vixilancia sobre a entrada de asuntos vai estar mellor ou peor distribuída entre os desenvolvedores, de xeito que cada asunto vai recibir sempre unha resposta a tempo).

A segunda técnica está menos estendida, probablemente porque é máis difícil de automatizar. A idea esencial é que cada asunto novo é "coleguizado" na base de datos. Cando un usuario pensa que localizou un problema, solicítaselle que o describa nunha lista de correo, ou nun canal IRC, e recibe a confirmación por parte de alguén que lle di que efectivamente é un erro. Dispoñer cedo deste segundo par de ollos pode previr unha chea de reportes falsos. Ás veces, o segundo implicado é quen de identificar que o comportamento reportado non é un erro, ou que xa está corrixido nalgunha versión recente. Ou pode estar familiarizado cos síntomas dun asunto anterior, e evitar unha segunda incorporación do erro dirixindo o usuario cara ao asunto orixinal. Frecuentemente abonda con preguntarlle ao usuario: "Pesquisaches no sistema de xestión de erros se xa foi comunicado?". Moita xente simplemente nin pensa nesa posibilidade, aínda que farán a pesquisa de bo grado se saben que alguén *espera* que a fagan.

O sistema de colegas pode manter a base de datos de asuntos realmente limpa, mais tamén ten algunhas desvantaxes. Moita xente vai reportar en solitario, de todas maneiras, ben por non teren atopado, ou por ignoraren deliberadamente, as instrucións sobre pesquisa dun colega para reportar asuntos novos. Así que aínda vai ser necesario que haxa voluntarios revisando a base de datos. Aínda máis, como os máis dos remitentes novos non van ser conscientes da dificultade de manter a base de datos de asuntos, non é axeitado reprendelos por teren ignorado as instrucións. Así que os voluntarios van ter que permanecer atentos, e controlar como lles van devolver os asuntos non "colegueados" aos seus remitentes. O obxectivo é adestrar cada remitente no uso do sistema de colegas, de tal xeito que haxa sempre unha reserva crecente de xente que entende o sistema de filtraxe. Cando se detecta un asunto non revisado por un colega, os pasos ideais son:

1. Responder inmediatamente ao asunto, agradecendo amablemente o usuario por telo remitido, mais dirixíndoos ás instrucións sobre revisión por terceiros (que, por suposto, deberán estar ben visibles na web do proxecto).
2. Se o asunto for claramente válido e non estiver duplicado, apróbao de todas maneiras, e inicia o seu ciclo de vida normal. Despois de todo, o remitente xa estará informado sobre o sistema de revisión, e non ten sentido perder o tempo e o traballo feito pola vía de pechar un asunto perfectamente válido.
3. Pola contra, se o asunto non for claramente válido, péchao, mais pídelle ao remitente que o reabra se recibir confirmación dun colega. Cando a recibir, deberá facer mención ao fío de confirmación (p. ex.: mediante unha URL dos arquivos da lista).

Lembra que aínda que este sistema vai mellorar co tempo a relación sinal/ruído na base de datos, non vai poder evitar completamente a introdución de asuntos incorrectos. A única maneira de previr totalmente este problema é pechar o xestor de erros para todo o mundo salvo para os desenvolvedores (un remedio que é case sempre peor que a enfermidade). É mellor aceptar que a limpeza de asuntos inválidos vai constituír sempre unha tarefa máis da rutina de mantemento do proxecto, e tratar de contar co máximo número de persoas botando unha man.

Podes ver máis sobre este tema en “Xestor de incidencias” no capítulo 8 “Xestión de voluntarios”.

7 N. do T.: máis de 500.000 no momento de facer esta tradución.

IRC / Sistema de comunicacións en tempo real

Moitos proxectos ofrecen salas de conversación en tempo real usando foros *Internet Relay Chat (IRC)* nos que usuario e desenvolvedores poden responder ás preguntas dos demais e obter respostas instantáneas. Embora poidas manter un servidor IRC na túa propia sé web, normalmente non paga a pena facelo. Pola contra, fai o que fai todo o mundo: establece os teus canais IRC en *Freenode* (http://freenode.net/). Freenode ofréceche o control que precisas para administrares os canais IRC do teu proxecto[8].

A primeira tarefa é a de buscar un nome para o canal. A elección máis obvia é a do nome do teu proxecto (se estiver dispoñible en Freenode, úsao, se non , tenta dar con algo que se lle pareza o máis posible, e que sexa o máis doado de lembrar). Anuncia a dispoñibilidade do canal na páxina do proxecto, de xeito que un visitante que teña necesidade de facer unha pregunta urxente o atope á primeira. Por exemplo, esta mensaxe aparece nunha caixa moi ben situada na páxina de Subversion:

> *Se estás a empregar Subversion, recomendámosche que te unas á lista users@subversion.tigris.org, e que leas o Subversion Book (http://svnbook.red-bean.com/) e as FAQ (http://subversion.tigris.org/faq.html). Podes tamén facer preguntas no IRC en irc.freenode.net, canal #svn.*

Algúns proxectos ofrecen varios canais, un por subtema. Por exemplo, un canal para problemas de instalación, outro para cuestións sobre o uso, outro para conversas sobre desenvolvemento, etc. ("xestionando o crecemento" no capítulo 6 "Comunicacións") trata sobre como subdividir en múltiples canais). Mentres o teu proxecto sexa novo, debería haber un só canal, con todo o mundo charlando nel. Máis tarde, cando a relación usuario-desenvolvedor aumente, a separación de canais pode volverse necesaria.

Como vai saber a xente cales son os canais dispoñibles, e en que canal charlar?. E cando falen nun canal, como van saber cales son as regras locais?. A resposta está en dicirllo pola vía de establecer o *tema do canal*[9]. O tema do canal é unha mensaxe breve que cada usuario ve no momento de entrar no canal. Inclúe instrucións para os novatos e ligazóns a máis información. Por exemplo:

> *Estás charlando en #svn*
>
> *O tema para #svn é o Foro para as cuestións dos usuarios de Subversion, podes visitar tamén http://subversion.tigris.org/. || As discusións sobre desenvolvemento teñen lugar en #svn-dev. || Por favor non coles transcricións longas aquí, pola contra usa un sitio como http://pastebin.ca/. || NOVAS: Subversion 1.1.0 foi liberado, máis detalles en http://svn110.notlong.com/.*

Xa sei que é pesado, mais infórmalles aos novatos sobre todo o que precisaren saber. Di exactamente para que é o canal, facilita o enderezo da páxina do proxecto (para o caso de que alguén se pasee polo canal sen ter visitado antes o sitio web do proxecto), menciona un canal relacionado e dá algúns consellos sobre pegar textos.

Sitios de pegado

Un canal IRC é un espazo compartido: todo o mundo pode ver o que calquera outro está a escribir. Normalmente isto é bo, xa que permite que a xente se meta nunha conversa cando cre que ten algo co que contribuír, e permite que os observadores aprendan mirando. Mais empeza a ser un problema cando alguén ten unha chea de información que compartir dunha vez, tal como a transcrición dunha sesión de corrección de erros, xa que pegar demasiadas liñas no canal vai entorpecer outras conversas.

A solución está en usar un dos sitios *pastebin* ou *pastebot*. Cando se precise que alguén achegue unha cantidade grande de datos, non lle pidas que os pegue no chat, senón que vaia (por exemplo) a http://pastebin.ca/, pegue os seus datos no formulario da páxina, e comunique a URL resultante no canal.

---

8 Non hai obriga nin se espera que fagas unha doazón a Freenode, mais se ti ou o teu proxecto podedes afrontalo, por favor, pensa en contribuíres economicamente. Están exentos de impostos nos EEUU, e están a fornecer un servizo realmente valioso.

9 Para facelo, usa o comando */topic*. Todos os comandos no IRC comezan polo carácter "/". Bótalle unha ollada a http://www.irchelp.org/ se non estás familiarizado co uso e administración do IRC; en particular, http://www.irchelp.org/irchelp/irctutorial.html é un excelente titorial.

Calquera poderá visitar a URL e ver os datos.

Hai uns cantos sitios gratuítos para pegado de datos na web, demasiados para unha lista entendible, mais estes son algúns dos que eu teño usado: 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, e http://www.pastebin.com/.

Bots

Moitos canais IRC orientados a asuntos técnicos inclúen un membro non humano, chamado *bot*, que é quen de almacenar e devolver información en resposta a certos comandos. Tipicamente, a comunicación co *bot* faise de xeito exactamente igual que con calquera outro membro do canal, é dicir, os comandos se executan pola vía de "dicirllo" ao *bot*. Por exemplo:

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

Así lle dixo ao *bot* (que está rexistrado no canal como ayita) que lembre unha certa URL como resposta á consulta "*diff.cmd*". Agora poderemos dirixirnos a ayita para que informe outro usuario acerca de *diff-cmd*:

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

O mesmo pode facerse mediante un atallo axeitado:

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

A configuración exacta do comando e os comportamentos varían dun *bot* a outro. O exemplo anterior é para *ayita* (http://hix.nu/svn-public/alexis/trunk/) do que normalmente hai unha instancia funcionando no canal *#svn* en Freenode. Outros *bots* son *Dancer* (http://dancer.sourceforge.net/) e *Supybot* (http://supybot.com/). Non son precisos privilexios especiais para executar un *bot*. Un *bot* é un programa cliente; calquera pode pór un en marcha e dirixilo para que escoite un determinado servidor/canal.

Se no teu canal se tende a repetir as mesmas preguntas unha e outra vez, eu recoméndoche implementares un *bot*. Tan só un número pequeno de usuarios do canal vai adquirir a experiencia necesaria para manipular o *bot*, mais son os mesmos usuarios que van responder a un número desproporcionadamente grande de cuestións, xa que o *bot* vailles permitir facelo de xeito moito máis eficiente.

Arquivando o IRC

Aínda que é posible arquivar todo o que ocorre nun canal IRC, non necesariamente se espera que o fagas. As conversacións IRC poden ser nominalmente públicas, mais moita xente pensa que son semiprivadas e informais. Talvez os usuarios non teñan coidado coa gramática, e a miúdo expresen opinións (por exemplo, acerca de outro software ou outros programadores) que non queren que sexan gardadas para sempre nun arquivo en liña.

Por suposto, sempre haberá *citas* que deberían ser conservadas, e iso está ben. Os máis dos clientes IRC poden rexistrar unha conversación nun ficheiro, ou se iso non for posible, un sempre pode "copiar e pegar" a conversación do IRC a un foro máis permanente (frecuentemente o sistema de xestión de incidencias). Mais tamén o rexistro indiscriminado pode facer que algúns usuarios se sintan incómodos. Se arquivares absolutamente todo, deberías ter a seguridade de que iso está claramente establecido no tema do canal, e que hai unha URL coñecida do arquivo.

RSS Feeds

RSS (Really Simple Syndication) é un mecanismo para distribuír resumos de noticias, moi ricos en metadatos,

aos usuarios subscritos, é dicir: á xente que manifestou o seu interese en recibir eses resumos. Unha fonte RSS determinada é normalmente denominada un *feed*, e a interface de subscrición é chamada *lector de feed* ou *agregador de feed*. RSS Bandit (http://www.rssbandit.org) e o epónimo Feedreader (http://www.feedreader.com/) son dous lectores de RSS de código aberto, por exemplo.

Non hai espazo aquí para facer unha explicación técnica detallada de RSS[10], mais deberías ter en conta dúas cousas. A primeira é que o software de lectura de *feeds* é seleccionado polo usuario e vai ser *o mesmo* para todos os *feeds* que o subscritor estiver a monitoriza (de feito, este é o maior atractivo de RSS: que o subscritor escolle usar unha interface para todos os *feeds*, así que cada *feed* ten que concentrarse exclusivamente en servir os contidos). A segunda é que o RSS é xa ubicuo, tanto que as máis das persoas que están a empregalo nin saben que o usan. Cara á xente en xeral, RSS ten o aspecto dun pequeno botón nunha páxina web, cunha etiqueta que di "Subscríbete a este sitio" ou "*Feed* de noticias". Ti premes o botón, e de aí en diante o teu lector de *feed* (que pode perfectamente ser unha *applet* embebida na túa páxina web) actualízase automaticamente cada vez que for introducida unha novidade na web de orixe.

Isto quere dicir que o teu proxecto de software de código aberto debería probablemente ofrecer un *feed* RSS (os máis dos sitios de aloxamento preconfigurado xa o ofrecen como opción por defecto, véxase “Aloxamento preconfigurado”). Mais ten coidado de non publicares tantas novas ao día que os subscritores non sexan quen de separar o gran da palla. Se houber demasiados eventos de novas, a xente simplemente vai ignorar o *feed* ou, aínda peor, vaise dar de baixa por desesperación. Idealmente, un proxecto debería ofertar *feeds* separados: un para os grandes anuncios, outro seguindo (digamos) os eventos do xestor de incidencias, outro para cada lista de correo, etc. Na práctica, isto é difícil de facer en condicións: pode producir confusión de interfaces tanto para os visitantes da web do proxecto como para os administradores. Mais como mínimo, o proxecto debería ofrecer un *feed* RSS na portada da web para enviar novas sobre os eventos máis relevantes, tales como novas versións ou avisos de seguridade.

Ao César o que é do César[11]: esta sección non estaba na primeira edición do libro, mais a entrada "*Release Criteria, Open Source, Thoughts On...*" (http://krow.livejournal.com/564980.html) no blog de Brian Aker fíxome decatarme da utilidade dos *feed* RSS para os proxectos de software de código aberto.

Wikis

Unha *wiki* é un sitio web que permite que cada visitante poida editar e estender os seus contidos; o termo "*wiki*" (que significa "áxil" ou "super-rápido" en hawaiano) úsase tamén para denominar o software que permite facer esa edición. As *wikis* foron inventadas en 1995, mais a súa popularidade comezou a aumentar a partir de 2000 ou 2001, acelerada en parte polo éxito da Wikipedia (http://www.wikipedia.org/ ou a súa versión en galego: http://gl.wikipedia.org/), unha enciclopedia con contidos libres baseada en *wiki*. Podes pensar nunha *wiki* como algo intermedio entre o IRC e as páxinas web: as *wikis* non se editan en tempo real, de feito que a xente ten tempo de pulir e ponderar as súas contribucións, mais engadirlle contidos é tamén moi doado, implicando menor complexidade que a edición dunha páxina web.

As *wikis* non son aínda equipamentos estándar para os proxectos de software de código aberto, mais probablemente o sexan logo[12]. Como son unha tecnoloxía relativamente nova, e a xente está a experimentar con formas diversas de usalas, vou simplemente aconsellarvos unhas poucas precaucións (neste punto, é máis doado analizar os usos incorrectos das *wikis* que os seus éxitos).

Se decidires montar unha *wiki*, dedica bastante esforzo a conseguires unha organización clara da páxina e unha distribución visual agradable, para que os visitantes (p. ex.: editores potenciais) saiban de xeito instintivo como encaixar as súas contribucións. Igualmente importante, publica as instrucións na propia *wiki*, así a xente terá un sitio ao que acudiren en busca de axuda. Con demasiada frecuencia, os administradores de *wikis* caen vítimas da fantasía de que, xa que hordas de visitantes van engadir á *wiki* contidos de alta calidade, a suma de todas as contribucións individuais vai ser en consecuencia de alta calidade. Non é así como traballan os sitios web. Cada páxina individual ou parágrafo pode ser bo se se considera isoladamente, mais pode non selo se está integrado nun conxunto confuso ou desorganizado. Moi frecuentemente, as *wikis* sofren de:

10 Podes ver http://www.xml.com/pub/a/2002/12/18/dive-into-xml.html para maior información.

11 Nota do tradutor: tradución libre do orixinal inglés: "Credit where credit is due"

12 Nota do tradutor: de feito, xa no momento de facerse esta tradución (decembro de 2009), practicamente todos os proxectos que se desenvolven de xeito colaborativo contan con cadansúa *wiki,* ben para publicar a documentación e a información de soporte ao usuario, ben como "centro de mando" da totalidade do proxecto

- **Ausencia de principios de navegación.** Un sitio web ben organizado fai que os visitantes sintan que saben onde están en cada momento. Por exemplo, se as páxinas están ben deseñadas, a xente vai saber intuitivamente a diferenza entre unha rexión de "táboa de contidos" e unha rexión de "contidos". Os contribuidores dunha *wiki* van respectar esa diferenza, mais só se a diferenza estiver claramente establecida desde o principio.
- **Duplicación da información.** As *wikis* acaban frecuentemente con varias páxinas dicindo cousas similares, dado que os contribuidores individuais non van ser conscientes da duplicación. Isto pode ser consecuencia en parte da ausencia de principios de navegación comentados antes, debido a que a xente pode non atopar os contidos duplicados se non están onde esperan que estean.
- **Audiencia obxectivo mal definida.** Ata un certo punto, este problema é inevitable cando hai tantos autores diferentes, mais pode ser minimizado se previamente se escriben unha serie de consellos e guías acerca de como crear novos contidos. Editar unha chea de novos contidos a modo de exemplo ao principio tamén axuda, de xeito que os estándares fiquen xa afirmados.

A solución máis común a todos estes problemas é a mesma: fixar estándares editoriais, e demostralos non só publicándoos, senón publicando páxinas que os poñan en práctica. En xeral, as *wikis* van amplificar calquera erro nos seus materiais orixinais, xa que os colaboradores tenden a imitar os padróns que teñen á vista. Non poñas en marcha a *wiki* e fiques esperando a que todo axuste axeitadamente. Deberás tamén alimentala con materiais escritos, para que a xente teña un modelo que seguir.

O exemplo máis brillante dunha *wiki* ben administrada é a Wikipedia, aínda que isto pode ser así en parte porque os seus contidos (entradas dunha enciclopedia) encaixan de forma natural co formato *wiki*. Mais se examinares atentamente a Wikipedia, verás que os seus administradores deseñaron unha *moi* ben pensada infraestrutura para a cooperación. Hai unha boa cantidade de documentación sobre como escribir novas entradas, como manter un axeitado punto de vista, que clase de edicións facer, que edicións evitar, un procedemento para a resolución de disputas a respecto de edicións controvertidas (a base de varios pasos, incluíndo eventualmente a arbitraxe), etc. Tamén teñen controis de autoría, de xeito que se unha páxina for obxecto de repetidas edicións inapropiadas, poden bloqueala ata que o problema for resolto. Noutras palabras, os seus autores non se limitaron a pór unha serie de moldes nunha web e esperar o mellor. Wikipedia funciona porque os seus fundadores pensaron coidadosamente acerca de como conseguir que milleiros de descoñecidos axustasen os seus escritos a unha visión común. Aínda que ti seguramente non vas precisar un nivel preciso de preparación para montares a *wiki* dun proxecto de software de código aberto, paga a pena emular o seu espírito.

Para máis información acerca das *wikis*, podes consultar http://en.wikipedia.org/wiki/Wiki. Tamén a primeira *wiki* está viva e goza de boa saúde, e contén unha chea de discusións acerca de como administrar *wikis*: podes ver http://www.c2.com/cgi/wiki?WelcomeVisitors, http://www.c2.com/cgi/wiki?WhyWikiWorks, e http://www.c2.com/cgi/wiki?WhyWikiWorksNot para analizares varios puntos de vista sobre o tema.

Páxina web

Non hai moito que dicir acerca da posta en marcha da web dun proxecto desde un punto de vista técnico: montar un sitio web e escribir páxinas son tarefas realmente simples, e as máis das cousas importantes que se poden dicir acerca do seu deseño e distribución de contidos foron xa comentadas no capítulo anterior. A principal función do sitio web é a de presentar unha clara e amena visión xeral do proxecto, e enmarcar as restantes ferramentas (o sistema de control de versións, o xestor de incidencias, etc.) Se non es experto na construción de webs, xeralmente non che vai ser difícil atopares alguén que si o é e que estea desexando axudar na materia. Porén, para aforrar tempo e esforzos, a xente xeralmente prefire facer uso de webs pre-enlatadas.

Aloxamento preconfigurado

Usar un sitio preconfigurado ten moitas vantaxes. A primeira é a capacidade do servidor e o largo de banda: os seus servidores son caixas enormes conectadas a cabos moi grosos. Por moito éxito que tiver o teu proxecto, nunca vas esgotar o espazo en disco ou colapsar a conexión de rede. A segunda vantaxe é a simplicidade. Xa che van dar escollidos un xestor de incidencias, un sistema de control de versións, un xestor de listas de correo, un arquivo, e calquera outra utilidade que poidas precisar para controlar o teu proxecto. As ferramentas estarán xa configuradas, e serán eles quen se ocupen de facer as copias de seguridade de todos os datos almacenados

nas ferramentas. Terás poucas decisións que tomar. Todo o que terás que facer e preencher un formulario, premer un botón, e terás de súpeto un sitio web para o teu proxecto.

Estes son beneficios significativos dabondo. A desvantaxe, claro, é que ti vas ter que aceptar *as súas* eleccións e configuracións, aínda que algo diferente puidese ser mellor para o teu proxecto. Xeralmente os sitios preenlatados poden ser axustados dentro dun certo número de parámetros, mais nunca chegarás a ter o control refinado que atinxirías se instalases e configurases por ti mesmo o sitio e tiveses acceso administrativo completo ao servidor.

Un exemplo perfecto do que digo é o manexo de ficheiros xerados. Algunhas páxinas web do proxecto poden ser ficheiros xerados (por exemplo, hai sistemas para manter os datos das FAQ nun formato mestre doado de editar, desde o que se poden xerar HTML, PDF, e outros formatos de presentación). Como se explica en "Versiona todo" mais adiante neste mesmo capítulo, non deberías pretender versionar os formatos xerados, só o ficheiro mestre. Mais cando o teu sitio web está aloxado no servidor dun terceiro, pode que sexa imposible dispor dun gancho (*hook*) personalizado para xerar a versión HTML da FAQ cada vez que se modifique o ficheiro mestre. A única solución será versionar tamén os formatos xerados, para que se poidan publicar no sitio web.

Pode tamén haber consecuencias máis importantes. Talvez non teñas tanto control sobre a presentación como che gustaría. Algúns dos sitios preenlatados permitiranche personalizar as páxinas web, mais o deseño por defecto da web vaise facer presente en variadas e molestas formas. Por exemplo, algúns dos proxectos hospedados en SourceForge teñen as súas páxinas completamente personalizadas, mais aínda dirixen os desenvolvedores á súa "páxina en SourceForge" para información adicional. A páxina en SourceForge é o que sería a páxina do proxecto se non tivese un sitio web propio. Ten ligazóns ao xestor de incidencias, ao repositorio do CVS, descargas, etc. Por desgraza, unha páxina de SourceForge contén tamén unha certa cantidade de ruído alleo. A parte superior é unha banda publicitaria, e frecuentemente unha imaxe animada. No lado esquerdo contén unha banda de ligazóns de escasa relevancia para quen está interesado no proxecto. O lado dereito contén frecuentemente máis publicidade. Tan só o centro de páxina está dedicado a materiais realmente específicos do proxecto, e mesmo neste caso están dispostos dun xeito confuso que frecuentemente fai que o visitante non saiba onde premer a continuación.

Detrás de cada aspecto individual do deseño de SourceForge, hai sen dúbida unha boa razón (boa desde o punto de vista de SourceForge, tal como a publicidade). Mais desde o punto de vista dun proxecto individual, o resultado pode ser unha web máis que afastada do ideal. Non estou a botar pedras contra SourceForge; problemas similares ocorren cos máis dos servizos de hospedaxe enlatada. A cousa é que hai un compromiso. Ficas liberado das cargas técnicas derivadas da xestión do sitio do proxecto, mais polo prezo de teres que aceptar a maneira en que outros decidan como xestionalo.

Soamente ti podes decidir se os sitios enlatados son a mellor solución para o teu proxecto. Se elixires un sitio enlatado, deixa aberta a posibilidade de migrar aos teus propios servidores máis tarde, empregando un nome de dominio personalizado para a URL do proxecto. Podes reenviar a URL ao sitio enlatado, ou crear unha páxina de inicio completamente persoal na URL pública e mandar os usuarios ao sitio enlatado para faceren uso das funcionalidades sofisticadas. Tan só asegúrate de arranxares as cousas de xeito que se posteriormente decides empregar unha solución de aloxamento distinta, o enderezo do proxecto non teña que ser modificado.

Escollendo un sitio preconfigurado

O sitio enlatado máis grande e mellor coñecido é SourceForge (http://www.sourceforge.net/). Outros dous sitios que fornecen servizos iguais ou similares son savannah.gnu.org (http://savannah.gnu.org/) e BerliOS.de (http://www.berlios.de/)[13]. Algunhas organizacións, tales como a Apache Software Foundation (http://www.apache.org/) e Tigris.org (http://www.tigris.org/)[14], fornecen aloxamento gratis a proxectos de código aberto que encaixan coas súas misións e coa súa comunidade de proxectos existentes.

Haggen So fixo unha avaliación de varios sitios de aloxamento de proxectos como parte do seu traballo de investigación para a súa tese de doutoramento, *Construction of an Evaluation Model for Free/Open Source Project Hosting (FOSPHost) sites*. Os resultados están publicados en http://www.ibiblio.org/fosphost/, onde

13 Nota do tradutor: en Galiza, o sitio máis coñecido é a Forxa de Mancomún (https://forxa.mancomun.org/)

14 Aviso: eu son empregado de CollabNet (http://www.collab.net/), que patrocina Tigris.org, e emprego Tigris regularmente.

podes ver especialmente a tabela excelente comparativa en http://www.ibiblio.org/fosphost/exhost.htm.

Anonimato e participación

Un problema que non se dá só nos sitios enlatados, mais que se ten atopado especialmente neles, é o abuso da funcionalidade de *login* (rexistro de entrada). En si mesma é considerablemente simple: o sitio permite que cada visitante se cadastre cun nome de usuario e unha seña. Desde ese momento, mantén un perfil do usuario, e os administradores do proxecto poden asignarlle certos permisos, por exemplo o dereito a fornecer mudanzas ao repositorio.

Isto pode ser extremadamente útil e, de feito, é unha das vantaxes principais do aloxamento enlatado. O problema é que ás veces o rexistro de usuario acaba sendo necesario para tarefas que deberían ser permitidas a visitantes non rexistrados, especialmente a capacidade de rexistrar asuntos no xestor de incidencias, e de poder facer comentarios sobre asuntos abertos. Requirindo un nome de usuario rexistrado para tales accións, o proxecto eleva o limiar para o que debería poder facerse de xeito áxil e doado. Por suposto, é interesante poder contactar con quen ten introducido asuntos no xestor de incidencias, mais tendo un campo onde introducir o enderezo de correo-e (se o usuario o desexa) chega dabondo. Se un novo usuario localiza unha incidencia e quere reportala, sentirase molestado por ter que preencher un formulario de creación dunha conta antes de poder rexistrar o asunto no xestor de incidencias. Talvez simplemente renuncie a rexistrar a incidencia.

As vantaxes da xestión de usuarios xeralmente superan as desvantaxes. Mais se podes escoller que accións poden levarse a cabo de xeito anónimo, asegúrate de que non só están permitidas para visitantes non rexistrados *todas* as accións de só lectura, senón que tamén o están algunhas de entrada de datos, especialmente no xestor de incidencias e, se a tiveres, nas páxinas *wiki*.

Capítulo 4. Infraestrutura social e política

Xeralmente, as primeiras cuestións que a xente acostuma facer sobre o software libre son: Como traballa? Que mantén un proxecto activo? Quen toma as decisións? Eu sempre me dou por satisfeito con respostas conciliadoras sobre meritocracia, o espírito de cooperación, o código fala por si mesmo, etc. O feito é que a cuestión non é fácil de responder. Meritocracia, cooperación e executar código son parte deste, mais non chegan a explicar como os proxectos funcionan no día a día, e non din nada sobre como son resoltos os conflitos.

Este capítulo trata de mostrar a estrutura subxacente que os proxectos de éxito teñen en común. Refírome a "éxito" non só en termos de calidade técnica, senón tamén á saúde operacional e de supervivencia. Saúde operacional é a capacidade efectiva do proxecto de incorporar as novas contribucións de código e novos desenvolvedores, e para dar resposta aos informes de erros. Supervivencia é a capacidade do proxecto de existir independentemente dalgún participante ou patrocinador, entendendo isto como a probabilidade de que o proxecto continúe, ata se todos os seus membros fundadores tivesen que pasar a outras cousas. O éxito técnico non é difícil de atinxir, mais sen unha base robusta de desenvolvemento e base social, un proxecto pode ser incapaz de manexar o crecemento que trae o éxito inicial ou a saída de persoas carismáticas.

Hai varios xeitos de lograr este tipo de éxito. Nalgúns casos supón dispoñer dunha estrutura formal de goberno, na que se resolven os debates, convídanse (ou expúlsanse) novos desenvolvedores, planifícanse novas características, etc. Outros requiren dunha estrutura menos formal, mais tense que facer un esforzo maior de autocontrol para producir unha atmosfera de equidade na que a xente pode confiar coma unha forma de goberno *de facto*. Ambas formas conducen ao mesmo resultado: un sentimento de permanencia institucional, apoiado por hábitos e procedementos que son entendidos por todas as persoas que participan. Estas características son aínda máis importantes nos sistemas auto-organizados que nos centralizados, xa que nos sistemas auto-organizados, todos son conscientes de que unhas poucas mazás apodrecidas poden apodrecer todo o cesto, polo menos durante un tempo.

Posibilidade de crear ramas (forkability)

O ingrediente indispensable que une os desenvolvedores nun proxecto de software libre, e fai que estean dispostos a comprometérense cando for necesario, é a *capacidade de crearen ramas (forkability)* do código: a capacidade de calquera de coller unha copia do código fonte e utilizala para iniciar un proxecto en concorrencia, coñecido como "fork". O paradoxo é que a *posibilidade* de crear ramas é unha forza moito maior nos proxectos de software libre que as ramificacións reais, as cales son moi raras. Xa que a creación dunha rama é mala para todos (por motivos examinados en detalle na sección *Ramas* do capítulo 8, *Xestionando Voluntarios*), canto mais alto for o risco de ramificación, máis dispostos a comprometerse para evitalo estarán os membros.

As ramificacións, ou máis ben as ramificacións potenciais, son a razón de que non existan verdadeiros ditadores nos proxectos de software libre. Isto pode parecer unha afirmación sorprendente, considerando que é común oír como alguén é cualificado de "ditador" ou "tirano" nalgún proxecto de código aberto. Mais este tipo de tiranía é especial, moi diferente da interpretación convencional da palabra. Imaxinádevos un rei cuxos súbditos puidesen copiar todo o seu reino en calquera momento e pasar á copia para gobernala como mellor lles parecese. Non sería o goberno dese rei moi diferente doutro cuxos súbditos estiveren obrigados a permanecer baixo o seu mandato, sen importar o que fixer?

Por esta razón mesmo os proxectos que non están organizados formalmente como democracias, na práctica, compórtanse como tales cando se trata de afrontar decisións importantes. A capacidade de replicar implica a posibilidade de escindirse; a posibilidade dunha ramificación implica consenso. É moi posible que todo o mundo estea disposto a someterse a un líder (o exemplo máis famoso é Linus Torvalds no desenvolvemento do *kernel* de Linux), mais isto é así porque eles *escollen* facelo, dunha forma totalmente libre de cinismo ou dobreces. O ditador non ten un dominio máxico sobre o proxecto. Unha propiedade clave de todas as licenzas de código aberto é que non conceden máis poder a unha parte sobre as decisións de como mudar o código ou usalo. Se o ditador, de repente, comezase a tomar malas decisións, produciríase unha axitación, seguida finalmente dun levantamento e unha ramificación.

Mais porque exista a posibilidade dunha ramificación, o que impón un gran límite en canto a poder que alguén exerce nun proxecto, non quere dicir que non existan importantes diferenzas en como son dirixidos os proxectos. Non é desexable que cada decisión sexa condicionada pola posibilidade de que alguén estea considerando unha ramificación. Isto pode converterse en algo pesado e esbanxar enerxías destinadas ao auténtico traballo. As dúas seccións seguintes examinan os modos de organizar os proxectos para que a maioría das decisións sexan tomadas sen problemas. Estes dous exemplos son os casos estremos idealizados; moitos proxectos estarán situados entre estes estremos.

Ditadores benevolentes

O modelo de *ditador benevolente* é exactamente o que parece: a autoridade de tomar a decisión final radica nunha soa persoa, da que, en virtude da súa personalidade e experiencia, espérase que a use sabiamente.

Aínda que "ditador benevolente" (ou *DB*) é o termo estándar para este rol, sería mellor pensar nel coma

un "árbitro autorizado da comunidade" ou "xuíz". En xeral, os ditadores benevolentes realmente non toman todas as decisións, nin mesmo a maioría das decisións. É pouco probable que unha persoa poida ter os suficientes coñecementos para a toma de decisións apropiadas en todas as áreas dun proxecto, e ademais, os desenvolvedores máis cualificados non permanecerán no proxecto a non ser que teñan algunha influencia sobre a dirección do proxecto. Polo tanto, os ditadores benevolentes non acostuman a ditar moito. Pola contra, deixan que as cousas funcionen por si soas a través do debate e a experimentación sempre que for posible. Participan nesas discusións, mais como un desenvolvedor máis , con frecuencia delegando nun mantedor desa área que teña máis pericia. Só cando é evidente que non se logra o consenso, e que a maioría do grupo *quere* que alguén guíe a decisión para que o desenvolvemento poida seguir adiante, pisan firme e din: "Esta é a forma en que vai ser". As reticencias a tomar decisións por decreto é unha característica compartida por case todos os ditadores benevolentes con éxito, é unha das razóns polas que logran manter este rol.

Quen pode ser un bo ditador benevolente?

Ser un DB (ditador benevolente) require unha combinación de características. Necesítase, en primeiro lugar, unha certa delicadeza para xulgar a súa propia influencia no proxecto, o que á súa vez leva á moderación. Nas primeiras etapas dunha discusión, un non debe expresar opinións e conclusións con tanta seguridade que os outros sintan que é inútil opinar en contra. A xente debe ser libre de lanzar ideas, mesmo ideas estúpidas. É inevitable que o DB expoña unha idea estúpida de cando en vez tamén, por suposto, e polo tanto este rol require a capacidade de recoñecer cando un toma unha mala decisión; aínda que isto é unha característica básica que calquera bo desenvolvedor debe ter, especialmente se se mantén no proxecto durante tempo. Mais a diferenza é que o DB pode darse o luxo de equivocarse de cando en vez sen ter que lamentar danos permanentes na súa credibilidade. Os desenvolvedores que levan menos tempo poden sentirse máis inseguros, polo que o DB debería expoñer as críticas ou decisións contrarias con a suficiente sensibilidade tendo en conta o peso das súas palabras, tanto tecnicamente como psicoloxicamente.

O DB *non* necesita ter a maior capacidade técnica do proxecto. Debe ser suficiente como para traballar sobre o código, e entender e poder opinar sobre calquera mudanza en cuestión, mais iso é todo. A posición do DB non se adquire nin mantén en virtude dunha habilidade de xerar código que intimide. O que *si* é realmente importante é a experiencia e un bo sentido de deseño global do proxecto; non necesariamente a habilidade de xerar un bo deseño baixo demanda, senón a capacidade de recoñecelo, sexa cal for a súa orixe.

É común que un ditador benevolente sexa o fundador do proxecto, mais isto é máis unha correlación que unha causa. O tipo de calidades que permite pór en marcha con éxito un proxecto, competencia técnica, capacidade de persuadir a outros a unírense, etc, son exactamente as calidades que calquera DB necesita. E, por suposto, os fundadores iníciansе cunha especie de antigüidade automática, que a miúdo pode ser suficiente para que o ditador benevolente xurda cunha menor resistencia por parte de todos os involucrados.

Debe recordarse que a posibilidade de ramificación vai en ambos os sentidos. Un DB pode facer unha ramificación dun proxecto tan facilmente como calquera outro, e ocasionalmente fixérono así, cando se considerou que a dirección que está tomando o proxecto é diferente ao que a maioría dos desenvolvedores desexan. Debido á posibilidade dunha ramificación, non importa se o ditador benevolente ten privilexios de *root* (administrador do sistema) nos servidores principais do proxecto ou

non. Ás veces a xente fala de control do servidor coma se fose a última fonte de poder nun proxecto, mais en realidade é irrelevante. A posibilidade de agregar ou quitar as señas de *commit* dos membros nun servidor só afecta á copia do proxecto que reside no servidor. Un abuso prolongado dese poder, xa sexa polo DB ou doutra persoa, simplemente levaría ao traslado a un servidor diferente.

Se o teu proxecto debería ter un ditador benevolente ou se vai funcionar mellor cun sistema menos centralizado, depende en gran medida de que exista alguén adecuado para exercer este papel. Como regra xeral, se for moi obvio para todos quen debería ser o DB, entón ese é o camiño a seguir. Porén, se ningún candidato para BD é inmediatamente obvio, entón o proxecto probablemente deba utilizar unha toma de decisións descentralizada, como se describe na seguinte sección.

Democracia baseada no consenso

A medida que os proxectos van madurando, tenden a afastarse do modelo do ditador benevolente cara a sistemas democráticos máis abertos. Isto non se debe necesariamente á insatisfacción causada por un DB en concreto. Trátase simplemente de que unha forma de goberno baseada no grupo é máis "evolutivamente estable", usando unha metáfora biolóxica. Sempre que un ditador benevolente renuncia, ou intenta repartir a responsabilidade na toma de decisións máis equitativamente, é unha oportunidade para o grupo de asentarse nun sistema non ditatorial -establecer unha constitución, por así dicilo. O grupo pode non aproveitar esta oportunidade a primeira vez, ou na segunda, mais finalmente vaino facer; unha vez que o fan, a decisión é pouco probable que se inverta. O sentido común explica o porqué: se un grupo de persoas N tivese que investir unha persoa con poder especial, iso significaría que N - 1 persoas tiveron que aceptar diminuíren a súa influencia individual. Normalmente a xente non quere facer iso. Mesmo se o fixesen, a ditadura resultante aínda sería condicional: o grupo que designa un DB, é claramente o grupo que pode depor ao DB. Polo tanto, unha vez que o proxecto pasou dun liderado carismático individual a un sistema máis formal baseado no grupo, é raro que volte ao estado anterior.

Os detalles de como funcionan eses sistemas varían amplamente, mais hai dous elementos comúns: un, o grupo funciona por consenso na maioría dos casos, dous, conta cun mecanismo formal de votacións alternativo para as situacións nas que o consenso non poida ser alcanzado.

Consenso simplemente quere dicir un acordo que todo o mundo está disposto a aceptar. Non é un estado ambiguo: un grupo chegou a un consenso sobre un asunto particular cando alguén propón que o consenso foi alcanzado, e ninguén contradí esa afirmación. A persoa que propón o consenso debe, por suposto, establecer especificamente o que supón este consenso, e que accións deben tomarse en consecuencia, se non resultaren obvias.

A maioría das conversacións nun proxecto tratan sobre temas técnicos, como a forma correcta de corrixir algún erro, engadir ou non unha característica, como de estrita ten que ser a documentación das interfaces, etc. O goberno baseado no consenso funciona ben porque mestura á perfección a transparencia e a discusión técnica en si mesma. Ao final dun debate, a miúdo hai un acordo xeral sobre que camiño tomar. Alguén fai unha intervención conclusiva, que é á vez un resumo do que se decidiu e unha proposta implícita de consenso. Isto ofrece unha última oportunidade para que alguén diga "Espera, non estou de acordo con iso. Temos que reconsiderar isto un pouco máis".

Para as pequenas decisións, non controvertidas, a proposta de consenso é implícita. Por exemplo, cando

un desenvolvedor realiza un *commit* non solicitado para a corrección dun erro, o mesmo *commit* é a proposta de consenso: "Supoño que todos estamos de acordo en que este erro debe ser solucionado, e que este é o xeito de arranxalo". Por suposto, o desenvolvedor non o di explicitamente; só leva a cabo a reparación, e os outros membros do proxecto non se molestan en manifestar o seu acordo, porque o silencio é consentimento. Se alguén realiza un commit que *non* alcanza o consenso, para o proxecto é simple analizar a mudanza como se aínda non se realizase. A razón de que isto funcione é o tema da próxima sección.

Control de versións quere dicir que te podes relaxar

O feito de que o código fonte do proxecto fique baixo un sistema de control de versións quere dicir que a maioría das decisións poden facilmente desfacerse. O caso máis común no que isto ocorre é que alguén realiza un *commit* pensando, erroneamente, que todo o mundo estará de acordo con esa mudanza, soamente atopándose coas obxeccións despois de que se realizou. É típico desas obxeccións comezar obrigatoriamente desculpándose por non intervir no debate previo, aínda que isto pode omitirse se o obxector non atopa ningún rexistro dese debate nos arquivos da lista de correo. De todos modos, non hai ningunha razón para que o ton da discusión sexa diferente despois do *commit* que antes. Calquera mudanza pode ser revertida, polo menos ata que se introduzan mudanzas dependentes (é dicir, novo código que fallaría se as mudanzas realizadas foren eliminadas subitamente). O sistema de control de versións ofrece ao proxecto un xeito de desfacer os efectos de ideas malas ou apresuradas. Isto, á súa vez, outorga liberdade á xente para confiar nos seus instintos sobre canto "*feedback*" é necesario antes de facer algo.

Isto tamén quere dicir que o proceso de establecer un consenso non necesita ser moi formal. Moitos proxectos manexan isto por instinto. As mudanzas menores pódense levar a cabo sen discusión, ou cunha discusión mínima seguida por algúns xestos de asentimento. Para as mudanzas máis significativas, especialmente aquelas co suficiente potencial para desestabilizar unha cantidade considerable de código, as persoas deben esperar un día ou dous antes de asumiren que existe consenso. A razón de ser disto é que ninguén debe ser marxinado nunha negociación importante simplemente porque non comproba o seu correo electrónico coa frecuencia suficiente.

Así, cando alguén estiver seguro de que sabe o que hai que facer, debería seguir adiante e facelo. Isto aplícase non só ao *software*, senón tamén ás actualizacións do sitio web, mudanzas na documentación, e calquera outra cousa que non suscite controversia. Polo xeral só haberá uns poucos casos en que unha acción deba ser rectificada, e estas poden ser tratadas caso por caso. Por suposto, non se debe animar a xente a que sexa demasiado obstinada. Hai aínda unha diferenza psicolóxica entre unha decisión baixo discusión e unha que xa foi efectuada, mesmo se for tecnicamente reversible.
A xente sempre sente que o impulso está unido á acción, e serán un pouco máis relutantes a reverteren unha mudanza que a previla nun primeiro momento. Se un desenvolvedor abusa deste feito, efectuando *commits* potencialmente controvertidos demasiado rápido, a xente pode e debe queixarse, aplicándolle a este desenvolvedor unhas normas máis estritas ata que isto mellore.

Cando o consenso non pode ser alcanzado, votación

Inevitablemente, algúns debates non chegarán ao consenso. Cando fallan todos os outros medios de saír do estancamento, a solución é votar. Mais antes de levar a cabo a votación, debe existir un claro

conxunto de opcións nas papeletas. Aquí, de novo, o proceso normal de discusión técnica intégrase, case casualmente, cos procedementos de toma de decisións do proxecto. Os tipos de preguntas que teñen lugar na votación entrañan a miúdo asuntos complexos e polifacéticos. En calquera debate tan complexo, hai xeralmente unha ou dúas persoas exercendo o papel de *intermediario honesto*: expondo periodicamente resumos dos diversos argumentos e facendo un seguimento de onde residen os puntos centrais de desacordo (e acordo). Estes resumos axudan a medir o grao de progreso alcanzado, e recordar que cuestións quedan por resolver. Estas sínteses poden servir como modelos para unha papeleta de votación, en caso de ser necesaria esta. Se os intermediarios honestos fixeron ben o seu traballo, estarán en condicións de chamar á votación cando chegar o momento, e o grupo estará disposto a usar a papeleta de votación baseada no seu resumo das cuestións a resolver. Os intermediarios tamén participarán no debate; non é necesario que fiquen fóra da contenda, a condición de que poidan entender e representar as opinións dos demais, e non deixen que os seus sentimentos partidistas lles impidan sintetizar o estado do debate dunha forma neutral.

O contido real da votación non acostuma ser polémico. Por cuestións de tempo, ao chegar a unha votación, o desacordo foi reducido a unhas poucas cuestións, ben etiquetadas e con descricións breves. Ocasionalmente, un desenvolvedor oporase á forma da votación mesma. Ás veces esta preocupación é lexítima, por exemplo, que unha opción importante sexa deixada de lado ou que non se describa con precisión. Mais outras veces un desenvolvedor pode tratar de impedir o inevitable, tal vez por saber que o voto non vai ir no seu sentido. Ver sección *Xente Problemática* do capítulo 6, *Comunicacións*, para saber como facer fronte a este tipo de obstrucionismo.

Recorda que se debe especificar o sistema de votación, xa que hai moitos tipos diferentes, e a xente pode ter falsas expectativas sobre o procedemento que se utiliza. Unha boa elección na maioría dos casos é o *voto de aprobación*, a través do cal cada elector pode votar por tantas opcións como desexe. O voto de aprobación é fácil de explicar e recontar, e a diferenza doutros métodos, só require unha rolda de votación. Consulta *http://en.wikipedia.org/wiki/Voting_system#List_of_systems* para obteres máis detalles acerca da votación de aprobación e outros sistemas de votación, mais debes tentar evitar entrar nun longo debate sobre que sistema de votación utilizar (porque, por suposto, pódeste atopar nun debate sobre que sistema de votación utilizar para decidir o sistema de votación!). Unha razón para escolleres o voto de aprobación como unha boa opción é que é moi difícil para alguén oporse a este -é tan xusto como un sistema de votación pode chegar a ser.

Para rematar, débense administrar as votacións publicamente. Non hai necesidade de segredos ou anónimos nunha votación sobre os asuntos que se debateron publicamente. Cada participante envía o seu voto á lista de correo do proxecto, de modo que calquera observador poida facer o reconto e verificar os resultados por si mesmo, e así fique todo rexistrado nos arquivos.

Cando votar

O máis difícil da votación é determinar cando é necesario realizala. Levar a cabo unha votación debería ser algo pouco frecuente -o último recurso cando todas as opcións fracasaron. Non se debe pensar na votación como unha gran solución para resolver discusións. Non o é. Finaliza a discusión, e polo tanto, tamén acaba coa creatividade sobre como resolver o problema. Mentres a discusión continúa, existe a posibilidade de que alguén apareza cunha nova solución que lle guste a todos. Sorprendentemente, isto ocorre a miúdo: un debate aberto pode producir unha nova forma de pensar sobre o problema, e dar lugar a unha proposta que finalmente satisfaga a todos. Embora non xurda unha proposta nova, é mellor

negociar unha solución intermedia a que teña lugar unha votación. Logo de alcanzar un compromiso, todos están un pouco tristes, mentres que logo dunha votación, algunhas persoas non están contentas e outros son felices. Desde un punto de vista político, a primeira situación é preferible: polo menos cada persoa pode sentir que obtén algo en troca da súa infelicidade. Pode estar insatisfeito, mais todos o están.

A principal vantaxe da votación é que se pecha a cuestión e pódese seguir adiante. Mais o acordo alcánzase por un reconto, en lugar dun diálogo racional que conduza a todos á mesma conclusión. As persoas con máis experiencia en proxectos de software libre, son as que menos ansiosas atopei por alcanzar solucións a través da votación. No seu lugar, tratarán de buscar solucións previamente non consideradas ou transixir máis do que planearan. Existen varias técnicas para previr unha votación prematura. A máis obvia é simplemente dicir "eu non creo que esteamos listos para unha votación", e explicar por que non. Outra é a de solicitar un método informal (non vinculante) a man alzada. Se a resposta tende claramente cara a un lado ou outro, isto fará que, de súpeto, algunhas persoas estean máis dispostas a chegar a un acordo, evitando a necesidade dunha votación formal. Mais a forma máis efectiva é simplemente ofrecer unha solución nova, ou un novo punto de vista sobre unha proposta antiga, para que a xente volva ocuparse dos problemas a tratar en lugar de limitárense a repetir os mesmos argumentos.

Nalgúns casos isolados, todos poden estar de acordo en que as solucións intermedias son peores que calquera das que non o son. Cando iso ocorre, a votación non é tan cuestionable, tanto porque é máis probable que conduza a unha solución superior como porque a xente non vai estar demasiado descontenta, non importando como resulte. Ata entón, a votación non debe precipitarse. A discusión que desemboca nunha votación é o que educa o electorado, e deter pronto unha discusión pode diminuír a calidade do resultado.

(Ten en conta que este consello de ser relutante ás votacións non se aplica á votación sobre mudanza-inclusión que se describe na sección *Estabilizando unha versión* do capítulo 7, *Empacotamento, liberación e dia a dia no desenvolvemento*. Alí, a votación é máis un mecanismo de comunicación, un medio de rexistrar o propio compromiso no proceso de revisión de mudanza para que todos poidan dicir canta atención recibiu unha mudanza determinada).

Quen vota?

Ter un sistema de votación pon de manifesto a cuestión do electorado: quen vai votar? Isto pode ser un tema delicado, porque obriga a que o proxecto recoñeza oficialmente que hai xente máis involucrada, ou máis valorada que outros.

A mellor solución é simplemente tomar unha distinción existente, o acceso a *commit*, e asociar a isto o privilexio de voto. Nos proxectos que ofrecen permisos tanto parciais como totais, a cuestión de se os *committers* con acceso parcial poden votar, depende en gran medida do proceso polo cal se concede o acceso parcial a *commit*. Se o proxecto é administrado dunha forma liberal, por exemplo, como un medio de manter moitas ferramentas de colaboración para terceiras partes no repositorio, debe ficar claro que o acceso a *commit* parcial é só o dereito de realizar *commits*, non de voto. Naturalmente a implicación inversa mantense: como os *committers* con acceso total *terán* privilexios de voto, estes deben ser elixidos non só como programadores, senón como membros do electorado. Se alguén mostra tendencias que poden producir rupturas ou obstrucións na lista de correo, o grupo debe ser moi

cauteloso á hora de facelo *committer*, mesmo se a persoa estiver capacitada tecnicamente.

O sistema de votación debería ser utilizado para elixir os novos committers, tanto con privilexios completos como parciais. Mais aquí é un dos raros casos en que o segredo é apropiado. Non se pode ter os votos sobre os potenciais *committers* nunha lista de correo pública, porque os sentimentos dos candidatos (e a reputación) poderían resultar feridos. No canto disto, a forma habitual é que un *committer* actual envíe a unha lista de correo privada, formada só por *committers*, a proposta de que lle sexan concedidos permisos de *commit* a alguén. Deste xeito todos poden expresarse libremente, sabendo que a discusión é privada. A miúdo, non haberá desacordo, e polo tanto non se necesitará votar. Logo de esperar uns días para asegurarse de que todos tiveron a oportunidade de responder, o propoñente envía un *mail* ao candidato e ofrécelle permisos de *commit*. Se houber desacordo, iníciase unha discusión como para calquera outra cuestión, resultando posiblemente nunha votación. Para que este proceso sexa aberto e franco, o mero feito de que se está a levar a cabo debería ser totalmente secreto. Se a persoa en cuestión sabía o que estaba pasando, e logo non se ofrece o acceso a *commit*, pode concluír que perdeu a votación, e probablemente sentirse ferido. Por suposto, se alguén explicitamente pide o acceso ao *commit*, entón non hai máis remedio que considerar a proposta e explicitamente aceptalo ou rexeitalo. Neste último caso, entón debe facerse con sumo tacto, cunha explicación clara: "Gústannos os teus remendos, mais aínda non demostran o suficiente" ou "apreciamos todos os teus remendos, mais necesitaron considerables axustes antes de poder ser aplicados, polo que aínda non nos sentimos cómodos concedéndoche permisos de *commit*. Aínda que esperamos que isto mude co tempo". Recorda que o que se di pode caer como un golpe, dependendo do nivel de confianza coa persoa. Trata de velo desde o seu punto de vista cando escribas o email.

Posto que adicionar un novo *committer* é máis importante que a maioría de decisións puntuais, algúns proxectos teñen requirimentos especiais para o voto. Por exemplo, poderase esixir que a proposta reciba polo menos n votos positivos e ningún voto negativo, ou que certa supermaioría vote a favor. Os parámetros exactos non son importantes, a idea principal é conseguir que o grupo debe ser coidadoso ao engadir novos *committers*. Similares requirimentos especiais, ou ata máis estritos, débense aplicar nas votacións para *eliminar committers*, aínda que oxalá que iso nunca sexa necesario. Ver sección *Commiters* do capítulo 8, *Xestionando voluntarios*, para máis información sobre aspectos ademais dos relacionados coa votación engadindo e eliminando committers.

Inquéritos fronte a votacións

Para certas clases de votacións, pode ser útil ampliar o electorado. Por exemplo, se os desenvolvedores non poden determinar se unha interface dada coincide co xeito coa xente realmente usa o software, unha solución é pedir que voten todos os subscritores das listas de correo do proxecto. Estas votacións son realmente *inquéritos* en lugar de votacións, mais os desenvolvedores poden optar por tratar os resultados como vinculantes. Como en calquera votación, asegúrate de deixares claro aos participantes que existe a opción de incluír novas opcións: se alguén pensa nunha opción mellor que non está na lista, esta resposta pode chegar a ser o resultado máis importante do inquérito.

Vetos

Algúns proxectos permiten un tipo especial de dereito que se coñece como *veto*. O veto é unha forma para que un desenvolvedor poida deter unha mudanza apresurada ou mal considerada, polo menos o

tempo suficiente para que todos poidan discutila máis a fondo. Pensa no veto como algo que se sitúa entre unha forte obxección e unha manobra obstrucionista. O seu significado exacto varía dun proxecto a outro. Algúns proxectos fan que sexa moi difícil invalidar un veto; outros permiten que sexa superado polo voto da maioría, quizais logo dunha demora forzada para poder examinar máis o tema. Un veto debe ser acompañado por unha explicación exhaustiva; o veto presentado sen esa explicación debe ser considerado inválido.

Cos vetos introdúcese o problema do abuso do veto. Ás veces os desenvolvedores están demasiado ansiosos por aumentar a presión cun veto, cando o que realmente se require é máis discusión. Podes previr o abuso do veto sendo moi relutante ao uso deste, e facendo notar con tacto cando alguén usa o veto demasiado a miúdo. Se for necesario, tamén podes lembrarlle ao grupo que os vetos son de obrigado cumprimento a condición de que o grupo estea de acordo -despois de todo, se a maioría de desenvolvedores queren X, X vai ocorrer dun modo ou outro. Ou ben o desenvolvedor do veto retírao, ou o grupo vai quitarlle importancia a este dereito de veto.

Pode verse que a xente escribe "-1" para expresar un dereito de veto. Este costume vén da Apache Software Foundation, que ten moi estruturados o sistema de voto e o proceso de veto, que se describe en http://www.apache.org/foundation/voting.html. As normas de Apache estendéronse a outros proxectos, e pódese ver como as súas convencións son utilizadas en diversos graos en moitos lugares no mundo do código aberto. Tecnicamente "-1" non sempre indica un veto formal de acordo coas normas de Apache, mais informalmente considérase que representa un veto, ou polo menos unha obxección moi forte.

Do mesmo xeito que as votacións, os vetos pódense aplicar con carácter retroactivo. Non é correcto rexeitar un veto sobre a base de que a mudanza en cuestión xa se cometeu, ou das medidas adoptadas (a menos que sexa algo irrevogable, por exemplo unha nota de prensa). Doutra banda, un veto que chega con semanas ou meses de atraso, non é probable que se tome moi en serio, nin debería selo.

Poñer todo por escrito

Nalgún momento, o número de convenios e acordos que circulan no seu proxecto poden chegar a ser tan grandes que se necesiten rexistrar nalgún lugar. A fin de dar lexitimidade a eses documentos, debe ficar claro que se basean nas discusións da lista de correo e nos acordos xa en vigor. A medida que o redactes, remítete aos fíos relevantes dos arquivos das listas de correo, e cada vez que haxa un punto no que non esteas seguro, pregunta de novo. O documento non debería conter ningunha sorpresa: non é a fonte dos acordos, é simplemente unha descrición destes. Por suposto, se ten éxito, a xente comezará a citalo como fonte de autoridade en si mesmo, mais iso só significa que reflicte a vontade xeral do grupo con precisión.

Este é o documento aludido na sección *Pautas de desenvolvemento* do capítulo 2, *Primeiros pasos*. Naturalmente, cando o proxecto é moi recente, terá que establecer os criterios a seguir sen os beneficios cos que contan os proxectos con máis historia. Mais a medida que a comunidade de desenvolvemento madura, pode axustar a linguaxe para reflectir a forma en que actualmente son levadas a cabo as cousas.

Non trates de abranguelo todo. Ningún documento pode capturar todo o que a xente necesita saber acerca da participación nun proxecto. Moitas das convencións que un proxecto contrae son tácitas,

nunca mencionadas explicitamente, mais aceptadas por todos. Outras cousas son simplemente demasiado evidentes para seren mencionadas, e só iría en detrimento de material importante mais non evidente. Por exemplo, non ten sentido redactar directrices como "Sexa cortés e respectuoso cos demais nas listas de correo, e non inicie apaixonadas guerras", ou "Escriba código limpo, fácil de ler e libre de erros." Por suposto, estas cousas son desexables, mais xa que non existe un universo concibible no que puidesen *non* ser desexables, non son dignas de mención. Se a xente for groseira na lista de correo, ou escribir código con erros, non van deixar de facelo só porque a guía do proxecto o di. Estas situacións requiren atención no momento en que aparecen, e non con admonicións xerais para ser bos. Por outra parte, se o proxecto tiver liñas específicas sobre *como* escribir bo código, entón esas directrices da guía deberían escribirse co máximo detalle posible.

Un bo xeito de determinar o que debe incluírse é basear o documento nas preguntas que fan os chegados recentemente, e nas queixas que expoñen os desenvolvedores con experiencia máis a miúdo. Isto non quere dicir que necesariamente teñan que converterse nun *FAQ* -posiblemente o documento necesite unha estrutura narrativa máis coherente que a que pode ofrecer o *FAQ*. Mais debería seguir o mesmo principio baseado en abordar as cuestións que realmente se producen, en lugar de anticiparse ás que poidan xurdir.

Se o proxecto é unha ditadura benévola, ou ten axentes dotados de poderes especiais (presidente, secretario, ou calquera outro), entón o documento é tamén unha boa oportunidade para codificar os procedementos de sucesión. Ás veces isto pode ser tan simple como nomear a persoas específicas como sucesores no caso de que o DB abandone repentinamente o proxecto por calquera razón. Xeralmente, se hai un DB, é só el quen pode designar un sucesor. Se se elixe unha comisión, entón o procedemento da elección e o nomeamento dos integrantes da comisión ten que estar descrito no documento. Se orixinariamente non existe un procedemento, entón hai que conseguir un consenso nas listas de correo *antes* de escribir sobre isto. A xente, algunhas veces, pode ser susceptible coas estruturas xerárquicas, de modo que o tema debe ser tratado con delicadeza.

Quizais o máis importante é deixar claro que as normas poden ser reconsideradas. Se os acordos descritos no documento comezan a frear o proxecto, recordar a todos que se supón que é un vivo reflexo das intencións do grupo, non unha fonte de frustración e estagnación. Se alguén toma, de forma inapropiada, por hábito pedir que as regras sexan reconsideradas cada vez que unha regra lle afecta, non sempre convén debater o tema con el -ás veces o silencio é a mellor táctica. Se outras persoas están de acordo coas queixas, a campá soou, e é obvio que será necesario mudar algo. Se ninguén está de acordo, entón a persoa non obterá demasiados froitos, e as regras manteranse como están.

Dous bos exemplos de directrices dun proxecto son o arquivo de *Subversion HACKING* (http://svn.collab.net/repos/svn/trunk/www/hacking.html), e os documentos de xestión da *Apache Software Foundation* (http://www.apache.org/foundation/how-it-works.html e http://www.apache.org/foundation/voting.html). A *ASF* é, realmente, unha colección de proxectos de software, organizada legalmente como unha corporación sen fins de lucro, de modo que os seus documentos tenden a describir os procedementos de goberno máis que as convencións de desenvolvemento. De todos os xeitos, paga a pena lelos, porque representan a experiencia acumulada de moitos proxectos de código aberto.

Capítulo 5. Diñeiro

Este capítulo examina como conseguir fondos nun entorno de software libre. Está dirixido non só a desenvolvedores aos que se lles paga por traballaren en proxectos de software libre, senón tamén aos

seus xestores, que precisan entender as dinámicas sociais do entorno de desenvolvemento. Nas seguintes seccións, o destinatario ("ti") enténdese tanto como un desenvolvedor que cobra como aquel que coordina a tales desenvolvedores. O consello será frecuentemente o mesmo para ambos; cando non o for, a audiencia específica ficará clara a partir do contexto.

Os fondos corporativos dun desenvolvemento de software libre non son un fenómeno novo. Moitos desenvolvementos estiveron sempre informalmente subvencionados. Cando un administrador de sistemas escribe unha ferramenta de análise de rede para axudarlle no seu traballo e a colga na rede, e consegue correccións de erros e contribucións con novas características froito da colaboración de outros administradores de sistemas, o que está a ocorrer é que se crea un consorcio non oficial. Os fondos do consorcio veñen dos salarios dos administradores de sistemas; e o seu espazo de traballo e largo de banda de rede son cedidos, sen coñecemento por parte da organización na que traballan. Estas organizacións beneficianse do investimento, por suposto, aínda que institucionalmente ao comezo non son conscientes diso.

A diferenza é que hoxe moitos deses esforzos están sendo formalizados. As corporacións tomaron consciencia dos beneficios do software de código aberto, e comezaron a participar máis directamente no seu desenvolvemento. Os desenvolvedores hoxe en día tamén esperan que os proxectos realmente importantes atraian cando menos doazóns, e posiblemente mesmo patrocinadores de longa duración. Embora a presenza de diñeiro non mudase as dinámicas básicas do software libre, si que mudou dun xeito enorme a escala na que ocorren as cousas, tanto en termos de número de desenvolvedores como en tempo por desenvolvedor. Tamén tivo efecto en como se organizan os proxectos, e en como interaxen as partes involucradas neles. A cuestión non se reduce unicamente a como se gasta o diñeiro, ou como medir o retorno do investimento, senón que tamén envolve a xestión e os procesos: como poden traballar de maneira produtiva en colaboración as estruturas de mando xerárquico das corporacións e as comunidades semi-descentralizadas de voluntarios? Coincidirán no significado de "produtividade"?

A cobertura financeira é, en xeral, benvida polas comunidades de desenvolvemento de software de código aberto. Este patrocinio pode reducir a vulnerabilidade dun proxecto perante as Forzas do Caos, as cales barren moitos proxectos antes de que realmente despeguen, e ademais pode facer que a xente estea máis desexosa de darlle unha oportunidade ao software en cuestión; senten que están a investir o seu tempo en algo que vai durar polo menos seis meses. Despois de todo, a credibilidade é contaxiosa. Cando, se di, IBM apoia un proxecto de código aberto, a xente máis ou menos asume que non van deixar que o proxecto fracase, e a vontade resultante de dedicarlle esforzos pode convertela nunha profecía autocumprida.

Porén, os fondos tamén acarretan unha percepción de control. Se for manexado con coidado, o diñeiro pode dividir o proxecto en grupos. Se os voluntarios non pagados teñen a sensación de que as decisións de deseño ou de adición de novas funcionalidades están só dispoñibles para quen máis paga, marcharanse a outro proxecto que se asemelle máis a unha meritocracia e menos a un traballo non pagado para o beneficio dun terceiro. Pode que nunca se queixen abertamente nas listas de correo; en lugar diso, simplemente haberá menos e menos ruído desde fontes externas a medida que os voluntarios gradualmente deixen de tentar ser tomados en serio. O rumor da actividade a pequena escala continuará na forma de informes de erros e ocasionalmente corrección de pequenos erros, mais non haberá máis contribucións con moito código ou participación externa en discusións sobre o deseño. A xente sente que é o que se espera dela, e vive (ou desfalece) da man desas expectativas.

O diñeiro debe ser usado con coidado, mais iso non quere dicir que non poida comprar influencia

Desde logo que pode. O truco está en que non pode comprar influencia directamente. Nunha sinxela transacción comercial, trocas diñeiro polo que quixeres; se precisares engadir unha funcionalidade, asinas un contrato, pagas por el e farase. Nun proxecto de código aberto isto non é tan sinxelo. Claro que podes asinar un contrato con algúns desenvolvedores, mais eles ían pensar que es parvo, tanto ti como eles mesmos, se che garantisen que o traballo que pagaches vai ser aceptado pola comunidade de desenvolvemento simplemente porque pagaches por el. O traballo só pode ser aceptado polos seus propios méritos e por como encaixa na visión do software da comunidade. Pode que teñas algo que dicir nesta visión, mais non serás a única voz.

Polo tanto, o diñeiro non pode comprar influencia, mais pode comprar cousas que **conducen á** influencia. O exemplo máis obvio son os programadores. Se son contratados bos programadores, e aguantan abondo para colleren experiencia co software e credibilidade na comunidade, entón van poder influenciar no proxecto polos mesmos medios que calquera outro membro. Terán voto, ou se hai varios, terán un bloque de voto. Se son respectados no proxecto, terán influencia para alén dos seus votos. Tampouco hai necesidade de que os desenvolvedores con salario disimulen os seus motivos. Despois de todo, calquera que quixer facer unha mudanza no software quéreo por algunha razón. As razóns da túa empresa non son menos lexítimas que as de calquera outro. Simplemente o peso dado aos obxectivos da túa empresa vai ser determinado polo status dos teus representantes no proxecto, e non polo teu tamaño, orzamento ou plan de negocio.

Tipos de participación

Existen múltiples razóns distintas polas que os proxectos de código aberto conseguen fondos. Os elementos desta lista non son mutuamente excluentes; a miúdo o financiamento dun proxecto será o resultado de varias, ou mesmo todas estas motivacións:

Compartindo a carga

Distintas organizacións con necesidades de software similares, a miúdo atópanse a si mesmas duplicando esforzos, tanto escribindo internamente código similar, como comprando produtos similares de vendedores propietarios. Cando se decatan do que ocorre, as organizacións poden reunir os seus recursos e crear (ou unirse a) un proxecto de Código Aberto adaptado ás súas necesidades. As vantaxes son obvias: o custo de desenvolvemento divídese, mais os beneficios acumúlanse para todos. Aínda que este escenario pareza máis intuitivo para organizacións sen ánimo de lucro, pode ter sentido estratéxico mesmo para competidores con ánimo de lucro.

Exemplos: http://www.openadapter.org http://www.koha.org

Aumentando servizos

Cando unha empresa vende servizos que dependen de, ou se tornan máis atractivos con, programas de código aberto, naturalmente un dos intereses desta compañía é asegurar que estes programas sexan mantidos activamente.

Exemplo: o apoio de Collab Net (http://www.collab.net) ao soporte de http://subversion.tigris.org (Nota: este é o meu traballo diario, mais tamén é un exemplo perfecto deste modelo).

Apoiando as ventas de hardware

O valor dos ordenadores e dos seus compoñentes está directamente relacionado coa cantidade de software dispoñible para eles. Os fabricantes de hardware (non só de máquinas completas, senón tamén os fabricantes de periféricos e microprocesadores) descubriron que ter software libre de alta calidade dispoñible para ser executado no seu hardware é importante para os clientes.

Minando a competencia

Algunhas veces as empresas patrocinan certos proxectos de código aberto como un medio de minar o produto dun concorrente, o cal á súa vez pode ser ou non de código aberto. Comerlle cota de mercado aos competidores soe non ser a única razón para participar nun proxecto de código aberto, mais pode ser un factor.

Exemplo: http://www.openoffice.org (non, esta non é a única razón pola cal OpenOffice existe, mais este software é en parte unha resposta a Microsoft Office).

Mercadotecnia

Ter a túa empresa asociada cunha aplicación popular de código aberto pode ser unha boa xestión de marca.

Licenciamento dual

Licenciamento dual é a práctica baixo a cal se ofrece o software empregando unha licenza propietaria tradicional para clientes que desexen revendelo como parte doutra aplicación propietaria, e simultaneamente tamén se ofrece baixo unha licenza libre para aqueles que pretendan empregalo baixo os termos do software libre (consulta a sección chamada "Esquemas de licenciamento dual" no *Capítulo 9: Licenzas, Copyrights e Patentes*). Se a comunidade de desenvolvedores de software libre é activa, o programa recibe os beneficios do desenvolvemento e procura de erros de amplo espectro mentres a empresa segue obtendo beneficios polas regalías para manter algúns desenvolvedores a tempo completo.

Dous exemplos moi coñecidos son MySQL (http://www.mysql.com), creadores da base de datos co mesmo nome, e Sleepycat (http://www.sleepycat.com), que distribúe e dá soporte á base de datos Berkeley. Non é coincidencia que ambas sexan empresas de bases de datos; as bases de datos acostuman a estar integradas en aplicacións antes que seren vendidas directamente aos usuarios, o cal encaixa perfectamente cun modelo de licenciamento dual.

Doazóns

Un proxecto popular pode ás veces obter contribucións significativas tanto de individuos como de organizacións, simplemente tendo en liña un botón de doazóns ou ás veces vendendo produtos promocionais como cuncas de café, camisetas, tapetes de rato, etc. Precaución: se o teu proxecto acepta doazóns, planifica como vai ser empregado o diñeiro *antes* de que este chegue, e publica o plan na páxina web do proxecto. As discusións sobre a onde dirixir o diñeiro acostuman a ser máis suaves antes de telo; e de todos os xeitos, se houber desacordos significativos, é mellor descubrilos mentres a planificación do uso do diñeiro for teórica.

O modelo de negocio dun patrocinador non é o único factor co que este se relaciona cunha comunidade de código aberto. A relación histórica entre ambos tamén importa: iniciou a empresa o proxecto ou uniuse a posteriori? En ambos casos, o patrocinador terá que gañarse a credibilidade, mais (como era de

esperar) será preciso un maior esforzo no segundo caso. A organización debe ter uns obxectivos claros a respecto do proxecto. A empresa está tentando manter unha posición de liderado, ou simplemente está intentando ser unha voz dentro da comunidade para guiala mais sen necesariamente gobernar a dirección do proxecto? Ou só desexa ter un par de committers (desenvolvedores que teñen os permisos precisos para faceren mudanzas no repositorio de código fonte do proxecto) capaces de amañaren os problemas dos usuarios e introduciren as mudanzas na distribución pública sen moito rebumbio?

Mantede estas preguntas en mente mentres ledes as seguintes directrices. Están pensadas para a súa aplicación en calquera participación empresarial dentro dun proxecto de software libre, mais cada proxecto é un entorno humano, polo cal non hai dous que sexan iguais. Ata certo punto, será preciso improvisar, mais seguindo estes principios verás aumentadas as posibilidades de que as cousas saian como ti queres.

Contratos a longo prazo

Se estás xestionando desenvolvedores nun proxecto de código aberto, tenta mantelos o tempo suficiente para que adquiran experiencia tanto técnica como política; un par de anos, como mínimo. Por suposto que ningún proxecto nin de software libre nin de software privativo se beneficia do intercambio continuo de programadores. A necesidade dos chegados hai pouco de aprenderen todo desde cero pode ser perniciosa nalgúns entornos. Mais a penalización é maior nos proxectos de código aberto, porque os desenvolvedores que marchan levan con eles non só o coñecemento do código, senón tamén o seu status dentro da comunidade e mailas relacións humanas que fixeron nela.

A credibilidade acumulada por un desenvolvedor non pode ser transferida. O exemplo máis obvio está en que un desenvolvedor que acaba de incorporase non pode herdar o nivel acceso ao código de quen marcha (máis na sección chamada *O diñeiro non pode comprar o amor*), así que se o novo desenvolvedor non ten permisos para realizar mudanzas, deberá enviar parches ata conseguir estes permisos. Mais este nivel de acceso só é a manifestación máis cuantitativa da perda de influencia. Un desenvolvedor veterano tamén coñece os vellos temas que foron tratados unha e outra vez nas roldas de discusión. Un desenvolvedor novato, sen ter memoria destas conversacións, quizais tente retomar estes temas, levando a túa empresa a unha perda de credibilidade; outros pode que pensen: "Entón, esta xente non pode recordar nada?". Un novo desenvolvedor tampouco terá ningunha relación política coas personalidades do proxecto, e non poderá influenciar na dirección de desenvolvemento tan rápida ou suavemente como alguén que leve moito tempo no proxecto.

Adestra os novatos mediante un programa de incorporación supervisada. O novo desenvolvedor debe estar en contacto directo coa comunidade de desenvolvemento pública desde o primeiro día, comezando con correccións de erros e tarefas de limpeza, de xeito que poida aprender a base do código e adquirir unha reputación na comunidade, mais sen entraren en discusións de deseño. Durante este tempo, un ou máis desenvolvedores experimentados deben estar dispoñibles para resolver dúbidas, e deben ler cada mensaxe que o novo desenvolvedor envíe ás listas de correo de desenvolvemento, mesmo se se tratar de fíos aos que os desenvolvedores experimentados normalmente non lles prestan atención. Isto axudaralle ao grupo a ver potenciais dificultades antes de que o novo desenvolvedor caia nelas. Dar ánimos e orientación en privado, detrás do telón, pode axudar moito, especialmente se o novo desenvolvedor non estiver acostumado a que o seu código sexa paralela e masivamente revisado polos seus iguais.

Cando CollabNet contrata un novo desenvolvedor para traballar en Subversion, sentámonos xuntos e escollemos algúns erros abertos para que a nova incorporación se adestre. Discutimos os aspectos técnicos xerais da solución, e despois asignamos polo menos un desenvolvedor experimentado para que (publicamente) revise o remendo que ese novo desenvolvedor (publicamente) vai enviar. Normalmente nin tan sequera lle botamos unha ollada ao remendo antes de que a rolda de desenvolvemento principal o vexa, aínda que o facemos se houber algunha razón. O importante é que o novo desenvolvedor pase polo proceso de revisión pública, aprendendo a base do código mentres simultaneamente se habitúa a recibir críticas provenientes de completos descoñecidos. Aínda así, tentamos coordinar os prazos, de xeito que a nosas propias revisións cheguen inmediatamente despois do envío do remendo. Deste xeito a primeira revisión vista na rolda é a nosa, podendo axudar a establecer o ton do resto de revisións. Isto tamén contribúe á idea de que a nova persoa debe tomarse en serio: se outros ven que estamos empregando tempo en revisións detalladas, con profundas explicacións e referencias aos arquivos cando for apropiado, vanse decatar de que o estamos adestrando e que isto probablemente significa un investimento a longo prazo. Este feito pode posicionalos máis positivamente cara ao novo desenvolvedor, cando menos ata o punto de empregaren un pouco tempo extra en responder preguntas e revisar parches.

Parecer varios, non un

Os teus desenvolvedores deben tentar aparecer nos foros públicos do proxecto como participantes individuais, máis que como unha presencia corporativa monolítica. Isto non é así porque haxa unha connotación negativa inherente ás presencias corporativas monolíticas (ben, quizais si que as haxa, mais non é sobre o que trata este libro). en lugar diso, débese a que os individuos son o único tipo de entidade para a que os proxectos de código aberto están estruturalmente preparados. Un contribuínte individual pode ter discusións, enviar parches, adquirir credibilidade, votar, etc. Unha empresa non pode.

Máis aínda, ao comportarse dun xeito descentralizado, elimínase o estímulo de centralización da oposición. Deixa que os teus desenvolvedores teñan desacordos entre si nas listas de correo. Anímaos a revisaren o código dos compañeiros tan frecuente e publicamente como o farían co código dos demais. Disuádeos de votaren sempre como un bloco, porque se o fixeren, outros poden comezar a sentir que, dun xeito xeral, está habendo un esforzo organizado para mantelos aliñados.

Hai unha diferenza entre realmente estar descentralizado e simplemente parecer estalo. Baixo certas circunstancias, ter os teus desenvolvedores actuando concertadamente pode ser moi útil, e deberían estar preparados para coordinárense detrás dos bastidores cando for necesario. Por exemplo, cando se fai unha proposta, ter cedo varias persoas afirmando estaren de acordo pode darlle un pulo, dando a impresión dun consenso crecente. Os demais van sentir que a proposta ten potencial, e que se a refutaren, pararán ese potencial. Así, a xente só vai obxectar se tiver unha boa razón para facelo. Non hai nada malo en orquestrar o acordo deste xeito, sempre e cando as obxeccións sigan tomándose en serio. As manifestacións públicas dun acordo privado non son menos sinceras por teren sido coordinadas baixo a manga, e non son daniñas mentres non se empregaren para disolver argumentacións opostas. O seu propósito simplemente é inhibir o tipo de xente á que lle gusta obxectar só para manterse en forma; bótalle unha ollada á sección "*Canto máis suave for o tema, mais longo vai ser o debate*" no *Capítulo 6: Comunicacións* para saberes máis sobre eles.

Sé aberto sobre as túas motivacións

Sé tan aberto como poidas sobre os obxectivos da túa empresa, sen comprometeres os seus secretos comercias. Se queres que o proxecto adquira unha funcionalidade concreta porque se cadra os teus clientes a están a reclamar, dio directamente nas listas de correo. Se os clientes quixeren permanecer no anonimato, como pode ser o caso ás veces, polo menos pregúntalles se poden ser empregados como exemplo xenérico. Canto máis saiba a comunidade pública de desenvolvemento sobre *por que* ti queres o que queres, máis cómoda se vai sentir co que propuxeres.

Isto contradí o instinto -tan sinxelo de adquirir e tan complicado de deixar- de que o coñecemento é poder, e que canto máis saiban os demais sobre os teus obxectivos, máis control terán sobre ti. Mais ese instinto está errado neste caso. Ao avogares publicamente por unha funcionalidade (ou corrección de erro, ou o que for) ti *xa* puxeches as cartas enriba da mesa. A única cuestión agora é saber se terás éxito en guiar a comunidade a compartir o teu obxectivo. Se ti simplemente enuncias o que queres, mais non podes fornecer exemplos concretos do porqué, entón a túa argumentación é feble e a xente comezará a sospeitar dunha axenda secreta. Pola contra, se dás aínda que só sexa un pequeno conxunto de escenarios reais, amosando por que a funcionalidade proposta é importante, isto pode ter un efecto drástico no debate.

Para veres por que isto é así, considera a alternativa. Demasiado a miúdo, os debates sobre as novas funcionalidades ou direccións son longos e cansados. Os argumentos que a xente emprega a miúdo redúcense a "Eu persoalmente quero X" ou o sempre popular "Nos meus anos de experiencia como deseñador de software, X é extremadamente importante para os usuarios / unha parvada inútil que non agradecerá ninguén". Previsiblemente, a ausencia de datos sobre o uso no mundo real nunca axuda a acurtar estes debates, senón que permite divagar máis e máis sen ningunha base en experiencia real de usuario. Sen forzas de contraposición, o final resultante probablemente non sexa determinado pola alternativa mellor artellada, ou polo máis persistente, ou polo máis experimentado.

Como organización cunha chea de datos dispoñibles sobre os clientes, ti tes a oportunidade de forneceres esa forza de contraposición. Ti podes ser un canal para a información que doutro xeito non tería medios para chegar á comunidade de desenvolvemento. O feito de que esa información lle axude aos teus desexos non ten porque darche vergoña. A maioría dos desenvolvedores individualmente non teñen unha ampla experiencia sobre como se usa o software que escriben. Cada desenvolvedor emprega o software segundo a súa idiosincrasia; sobre os padróns de uso dos demais, dependen da súa intuición e capacidade de adiviñación, e no fondo sábeno. Fornecendo datos cribles sobre un número significativo de usuarios, estáslle dando á comunidade pública de desenvolvemento algo equivalente ao osíxeno. Sempre que os presentes dun xeito correcto, daranlles unha benvida entusiasta e impulsarán as cousas na dirección que ti queres que sigan.

A chave, por suposto, está en presentares ben as cousas. Nunca funcionará simplemente insistires en que tratas cun gran número de usuarios, e que porque eles precisan (ou pensan que precisan) unha funcionalidade dada, entón a túa solución debe ser implementada. En lugar diso, deberías enfocar as túas mensaxes iniciais no problema en lugar de nunha solución particular. Describe ao detalle as experiencias que os teus clientes están a atopar, fornece todas as análises que teñas dispoñibles e cada solución razoable na que poidas pensar. Cando a xente comezar a especular sobre a eficacia e eficiencia de varias solucións, ti podes basearte nos teus datos para defenderes ou refutares o que se está a dicir. Debes ter unha solución concreta en mente durante o proceso, mais non lle debes dar unha consideración especial ao comezo. Isto non é enganar, simplemente é o comportamento estándar dun

"operador de bolsa honesto". Despois de todo, o teu auténtico obxectivo é resolver o problema; unha solución só é un medio para ese fin. Se a solución que ti prefires realmente é superior , outros desenvolvedores recoñecerán este feito por si mesmos, e apoiarana porque así o decidiron, o cal é moito mellor que empurralos á súa implementación. (Tamén existe a posibilidade de que pensen nunha solución mellor).

Isto non quere dicir que nunca poidas interceder en favor dunha solución específica. Mais debes ser paciente para ver repetida na lista pública de desenvolvemento a análise que previamente xa fixestes dun xeito interno. Non envíes mensaxes dicindo "Si, nós xa o vimos, mais non funciona polas razóns A, B e C. Cando se chega ao fondo do problema, o único camiño para resolvelo é..." O problema non é o arrogante que isto soa, senón que dá a impresión de que vós *xa* investítedes, a porta pechada, unha cantidade descoñecida (aínda que a xente pensará que moi grande) de recursos analíticos no problema. Isto fai parecer que os esforzos de conceptualización, e pode que as decisións, se fixeron sen acceso por parte do público, e esta é unha receita para o resentimento.

Naturalmente, *ti* sabes canto esforzo investístedes internamente no problema, e ese coñecemento é, en certa maneira, unha desvantaxe, ao empurrares os teus desenvolvedores a un espazo mental diferente que o do resto da xente das listas de correo, reducindo a súa habilidade para veren as cousas desde o punto de vista daqueles que aínda non pensaron moito sobre o problema. Canto máis cedo poidas ter a todos os demais pensando sobre as cousas do mesmo xeito que vós, menor será o efecto deste distanciamento. Esta lóxica non só se aplica a situacións técnicas individuais, senón tamén ao gran mandamento de clarificares os teus obxectivos tanto como poidas. O descoñecido sempre é máis desestabilizante que o coñecido. Se a xente entende por que ti queres o que queres, sentirase cómoda falando contigo mesmo cando estiveren en desacordo. Se descoñecen o que che gusta, asumirán o peor, polo menos durante algún tempo.

Por suposto que non debes publicitar todo, nin a xente o espera de ti. Todas as organizacións teñen segredos,; quizais as organizacións comerciais teñan máis, mais as organizacións sen ánimo de lucro tamén os teñen. Se debes avogar por un camiño concreto, mais non podes revelar nada sobre o porqué, simplemente ofrece os mellores argumentos que poidas baixo esa limitación, e acepta o feito de que pode que non teñas tanta influencia na discusión como che gustaría. Este é un dos compromisos que adoptas para teres unha comunidade de desenvolvemento fóra da túa nómina.

O diñeiro non pode comprar o amor

Se es un desenvolvedor asalariado nun proxecto, entón establece axiña as normas sobre o que o diñeiro pode comprar e o que non. Isto non quere dicir que teñas que mandar dúas mensaxes ao día ás listas de correo reiterando a túa nobre e incorruptible natureza. Simplemente quere dicir que debes estar atento para desfacer as tensións que *poderían* ser creadas polo diñeiro. Non debes comezar asumindo que as tensións están aí; o preciso é demostrares que estás ao tanto de que potencialmente poden xurdir.

Un exemplo perfecto disto xurdiu no proxecto Subversion. Subversion arrancou en 2000 grazas a CollabNet (http://www.collab.net), o cal é o principal financeiro do proxecto desde a súa concepción, pagando os salarios de varios desenvolvedores (nota: eu son un deles). Pouco despois do comezo do proxecto, contratamos outro desenvolvedor, Mike Pilato, para unirse ao esforzo. Na altura xa se comezara a escribir código.
Embora Subversion aínda estaba nas súas etapas iniciais, xa había unha comunidade de

desenvolvemento cun conxunto de normas básicas.

A chegada de Mike fixo xurdir unha cuestión interesante. Subversion xa tiña unha política sobre como os novos desenvolvedores conseguían acceso para faceren commit (gravación de mudanzas no repositorio central do código fonte). Primeiro, debían enviar algúns parches ás listas de correo de desenvolvemento. Despois dun número suficiente de parches para que o resto de desenvolvedores puidesen ver que o novo contribuínte sabía o que estaba a facer, alguén propoñía que simplemente fixera commit directamente (esta proposta era privada, como se describe na sección "*Committers*"). Asumindo o acordo dos desenvolvedores con acceso a facer commit, un deles enviáballe un correo ao novo desenvolvedor e ofrecíalle acceso a facer commit no repositorio do proxecto.

CollabNet contratara a Mike especificamente para traballar en Subversion. Entre os que xa o coñecían, non había dúbida sobre as súas capacidades técnicas nin sobre a súa preparación para traballar no proxecto. Ademais, os desenvolvedores voluntarios tiñan unha moi boa relación cos empregados de CollabNet, e presumiblemente a maioría non presentaría obxeccións se lle tivermos dado a Mike acceso a facer commit o día que o contratamos. Mais sabiamos que estariamos sentando un precedente. Se lle tivermos dado acceso a facer commit a Mike por mandato, estariamos dicindo que CollabNet tiña o dereito de ignorar as directrices do proxecto, simplemente por ser o investidor primario. Mentres que o dano causado por isto aparentemente non necesariamente tería que ser inmediato, gradualmente provocaría un sentimento de non representatividade por parte dos desenvolvedores non asalariados: outras persoas teñen que gañar o seus acceso a facer commit -Collabnet simplemente cómprao.

Polo tanto Mike estivo de acordo en comezar o seu traballo en CollabNet como calquera outro desenvolvedor voluntario, sen acceso a facer commit. Enviou parches ás listas públicas de correo, onde os mesmos poderían ser, e eran, revisados por todo o mundo. Nós tamén dixemos na lista de correo que estabamos facendo estas cousas deliberadamente, para que non se perdera a perspectiva. Despois dun par de semanas de sólida actividade de Mike, alguén (non podo lembrar se foi un desenvolvedor de CollabNet ou non) propúxoo para ter acceso a facer commit, e foi aceptado, como sabiamos que ía ser.

Esta coherencia dáche unha credibilidade que o diñeiro non podería comprar nunca. E a credibilidade é unha moeda valiosa a ter nas discusións técnicas: inmunízate fronte a que cuestionen máis tarde os teus motivos. Na calor da argumentación, a xente ás veces busca camiños non técnicos para gañaren a batalla. O investidor principal do proxecto, debido á súa profunda participación e obvia incumbencia sobre as direccións que o proxecto toma, presenta uns obxectivos máis amplos que a maioría. Sendo escrupuloso na observación de todas as directrices do proxecto desde o comezo, o investidor adquire o mesmo tamaño que todos os demais.

(Ver tamén o blog de Danese Cooper en http://blogs.sun.com/roller/page/DaneseCooper/20040916 para ver unha historia similar sobre o acceso a facer commit. Cooper foi a "Open Source Diva" de Sun Microsystems -creo que ese era o seu título oficial- e na entrada do blog, describe como a comunidade de desenvolvemento de Tomcat fixo que Sun seguise para os seus desenvolvedores os mesmos estándares que para os desenvolvedores que non eran de Sun)

A necesidade dos investidores de xogaren coas mesmas normas que os demais quere dicir que o modelo de goberno da Ditadura Benevolente (ver a sección chamada "*Ditador benevolente*" no *Capítulo 4: Infraestrutura política e social*) é lixeiramente máis duro de conseguir ante a presencia de fondos económicos, sobre todo se o ditador traballa para o investidor principal. Dado que unha ditadura ten poucas normas, é difícil demostrar para o investidor que é respectuoso cos estándares da comunidade permanentemente, mesmo cando o for. Certamente non é imposible; só require un líder de proxecto

capaz de ver as cousas desde o punto de vista dos desenvolvedores externos, así como desde o do investidor, e de actuar en consecuencia. Mesmo así, probablemente sexa unha boa idea ter unha proposta de goberno non ditatorial preparada como Plan B, lista para ser presentada no momento que houber indicadores de insatisfacción na comunidade.

Contratación

O traballo contratado precisa ser manexado con coidado nos proxectos de software libre. Idealmente, ti queres que o traballo contratado sexa aceptado pola comunidade e incorporado á distribución pública. En teoría, non debería importar quen é o contratista, polo menos mentres cumprir coas directrices do proxecto. A teoría e a práctica ás veces coinciden: un completo descoñecido que se presenta con un bo remendo xeralmente *será capaz* de introducilo no software. O problema radica en que é moi difícil de producir un bo remendo para melloras non triviais ou producir unha nova funcionalidade cando realmente es un completo descoñecido; primeiramente debe discutirse co resto do proxecto. A duración desta discusión non pode ser calculada con precisión. Se ao contratista se lle pagar por horas, poderías acabar pagando máis do que esperas; se se lle pagar unha suma prefixada, el pode acabar facendo máis traballo do que pode permitirse.

Hai dous camiños para afrontar isto. O mellor xeito é facer un cálculo aproximado sobre a lonxitude do proceso de discusión, baseado en anteriores experiencias, engadíndolle algunhas marxes para contemplar a posibilidade de erros, e basear a contratación nisto. Este modo de operar tamén axuda a dividir o problema en varios máis pequenos, todo o independentes que for posible, para incrementar a previsibilidade de cada anaco. O outro xeito é contratar unicamente a entrega dun remendo, e tratar a aceptación do remendo no proxecto público como algo individual. Así é moito máis sinxelo escribir o contrato, mais ficas atrapado baixo a carga de manteres un remendo privado tanto tempo como dependeres do software, ou polo menos tanto tempo como demorares en conseguir ese remendo ou unha funcionalidade equivalente na rama principal. Por suposto, mesmo no mellor caso, o contrato en si mesmo non pode requirir que o remendo sexa aceptado no código, dado que isto requiriría vender algo que non está en venda. (Que pasaría se o resto do proxecto inesperadamente decidise non soportar a funcionalidade?) Non obstante, o contrato pode requirir un esforzo de *boa fe* para conseguir que as mudanzas sexan aceptadas pola comunidade, e que sexan introducidas no repositorio se a comunidade estiver de acordo con elas. Por exemplo, se o proxecto tiver estándares de escritura a respecto das mudanzas no código, o contrato pode referenciar estes estándares e especificar que o traballo debe respectalas. Na práctica, isto normalmente non funciona do xeito que todo o mundo espera.

A mellor táctica para unha contratación con éxito é contratar un dos desenvolvedores do proxecto, preferiblemente un committer. Esta pode parecer unha forma de comprar influencia, e si, o é. Mais non é tan corrupta como podería parecer. A influencia dun desenvolvedor no proxecto débese principalmente á calidade do seu código e ás súas interacción cos outros desenvolvedores. O feito de que teña un contrato para facer determinadas cousas non eleva o seu status, nin o diminúe, aínda que pode facer que a xente o escrute máis coidadosamente. Moitos desenvolvedores non arriscan a súa posición no proxecto por darlle cobertura a unha noa funcionalidade inapropiada ou fortemente degradada. De feito, parte do que obtés, ou poderías obter, cando contratas son consellos sobre que tipos de mudanzas probablemente sexan aceptadas pola comunidade. Ti tamén obtés unha leve elevación nas prioridades do proxecto. Debido á que a priorización non é máis que ver quen ten tempo para traballar en que, cando pagas polo tempo de alguén, provocas que o seu traballo suba un pouco na cola de prioridades. Este é un feito ben coñecido para os desenvolvedores de software libre

experimentados, e cando menos algúns deles vanlle dedicar atención ao traballo contratado simplemente porque parece que vai ser *feito*, dado que queren axudar a que se faga ben. Talvez non escriban nada do código, mais debaterán o deseño e revisarán o código, podendo ser moi útiles ambas cousas. Por todas estas razóns, o contratista é mellor visto desde as bancadas dos que xa están a participar no proxecto.

Isto lanza inmediatamente dúas preguntas: os contratos deben ser sempre privados? E cando non o foren, deberías preocuparte pola creación de tensións na comunidade derivadas do feito de tiveres contratado algúns desenvolvedores e non outros?

É mellor seres transparente sobre os contratos, cando puideres. Doutro xeito, o comportamento do contratado pode parecerlle raro a outros na comunidades -pode que de repente inexplicablemente lle dea alta prioridade a funcionalidades nas que nunca amosara interese no pasado. Cando a xente lle pregunta porque as valora agora, como pode responder convincentemente se non pode falar sobre o feito de que foi contratado para escribilas?

Ao mesmo tempo, nin ti nin o contratado deberiades actuar como se os demais debesen tratar o voso acordo como un gran trato. Demasiado frecuentemente teño visto contratados danzaren nas listas de correo de desenvolvemento coa actitude de que as súas mensaxes deberían ser consideradas máis en serio simplemente porque se lles estaba pagando. Este tipo de actitudes sinálalle ao resto do proxecto que o contratado considera máis importante o feito do contrato en lugar do resultado da codificación resultante do contrato. Mais desde o punto de vista do resto de desenvolvedores, só o código importa. En todo momento, o foco de atención debe manterse nas cuestión técnicas, e non nos detalles de quen paga a quen. Por exemplo, un dos desenvolvedores da comunidade de Subversion manexa a súa contratación dun xeito particularmente elegante. Mentres no IRC se discuten as súas mudanzas no código, el menciona aparte (normalmente dun xeito privado con mensaxes *privmsg*, do IRC) que o seu traballo nese erro ou funcionalidade en concreto está sendo pagado. mais sempre, e dun xeito coherente, dá a impresión de que de igual xeito querería estar traballando nesa mudanza, e de que está contento de que os cartos fagan posible que o poida facer. Pode ou non revelar a identidade do seu cliente, mais en todo caso, el non vive do contrato. Salienta que só é un adorno dos debates técnico sobre como facer as cousas.

Este exemplo mostra outra razón pola que é bo ser transparente sobre os contratos. Podería haber múltiples organizacións financiando contratos nun proxecto de código aberto dado, e se cada un coñece o que os demais están tratando de facer, poderían poñer en común os seus recursos. No caso anterior, o maior financiador do proxecto (CollabNet) non está involucrado de ningún xeito con estes contratos a desagregados, mais coñecer que algún outro está financiando determinadas correccións de erros permitiríalle a CollabNet encamiñar os seus recursos a outros erros, obtendo unha maior eficiencia para o conxunto do proxecto.

Vanse ofender outros desenvolvedores debido a que algúns cobren por traballar no proxecto? Xeralmente non, sobre todo cando os que cobran están ben posicionados, e xa eran membros respectados da comunidade. Ninguén espera que o traballo contratado sexa distribuído con ecuanimidade entre os committers. A xente comprende a importancia das relacións de longa duración: as incertezas involucradas na contratación son tales que unha vez atopares alguén que saibas que pode traballar contigo es relutante a trocalo por unha persoa diferente só para seres equitativo. Pénsao deste xeito: a primeira vez que contratas, non vai haber queixas, porque claramente tes que escoller *alguén*-non é culpa túa que non poidas contratar todo o mundo. Despois, cando contratas a mesma persoa por segunda vez, entón só é sentido común: ti xa o coñeces, a última vez tivestes éxito, entón por que

asumir riscos innecesarios? Deste modo, é perfectamente natural ter unha ou dúas persoas de man na comunidade, en lugar de repartir o traballo uniformemente.

Revisión e aceptación de mudanzas

A comunidade segue sendo importante para o éxito do traballo contratado. A súa participación no deseño e no proceso de revisión das mudanzas significativas non se pode facer a última hora. Debe ser considerada parte do traballo, e ser abrazada polo contratista. Non penses no escrutinio da comunidade como un obstáculo a ser evitado -pensa nel como un departamento de deseño e garantía de calidade. É beneficioso ser perseguido, e non simplemente soportado.

Caso de estudo: o protocolo de autenticación vía contrasinal de CVS

En 1995, fun a metade dunha sociedade que fornecía soporte e melloras a CVS (o "*Concurrent Versions System*"; mira http://www.cvshome.org). O meu socio Jim e máis eu eramos, informalmente, os mantedores de CVS. mais nós nunca pensaramos coidadosamente sobre como deberiamos dicirllo á comunidade de voluntarios de desenvolvemento de CVS. Nós simplemente asumimos que eles continuarían enviando parches, e nós aplicariámolos, e así sería como funcionaría.

Voltando ao tema, o traballo en rede sobre CVS só podía facerse mediante un programa de acceso remoto como *rsh*. Empregar a mesma seña tanto para o acceso ao CVS como para acceso ao login era un risco de seguridade obvio, e moitas organizacións adiaron o seu uso debido a isto. Un importante banco contratounos para engadir un novo mecanismo de autenticación, así poderían empregar de xeito seguro o traballo en rede sobre CVS nos lugares de traballo remotos.

Jim e máis eu asinamos o contrato e sentámonos a deseñar un novo sistema de autenticación. Chegamos a algo bastante simple (os Estados Unidos daquela tiñan controis de exportación sobre os códigos criptográficos, polo que os clientes entendían que non podiamos implementar unha autenticación forte), mais como non tiñamos experiencia no deseño de tales protocolos, fixemos unhas poucas chapuzadas que terían sido obvias para un experto. Estes erros poderían ter sido vistos facilmente se tiveramos tido o tempo para escribirmos unha proposta e enviarlla aos demais desenvolvedores para unha revisión. Mais nunca chegamos a facelo porque non se nos ocorreu pensar que a lista de desenvolvemento puidera ser empregada como un recurso. Nos sabiamos que a xente probablemente ía aceptar calquera cousa que nós mudaramos, e porque nós non sabiamos o que non sabiamos, non nos preocupabamos de facer o traballo dun xeito visible, por exemplo, enviando parches con frecuencia, facendo pequenos e sinxelos commits a unha rama especial, etc. O protocolo de autenticación resultante non foi moi bo, e por suposto, unha vez que se chegou a establecer, foi difícil de mellorar debido a cuestións de compatibilidade.

A raíz do problema non estivo na falta de experiencia; poderiamos ter aprendido dun xeito sinxelo aquilo que precisabamos saber. O problema estivo na nosa actitude cara a comunidade de voluntarios de desenvolvemento. Considerabamos a aceptación das mudanzas como un obstáculo a saltar, máis que como un proceso polo cal a calidade das mudanzas podería ser mellorada. Desde que confiamos en que case todo o que fixeramos sería aceptado (tal como fose), fixemos un esforzo menor para que os demais participasen.

Obviamente, cando vas escoller un contratista, ti queres alguén coas capacidades técnicas e experiencia axeitadas para o traballo. Mais tamén é importante escolleres algún con un historial de interacción construtiva na comunidade cos outros desenvolvedores. Deste xeito, vas a obter máis que unha simple persoa; vas obter un axente con permiso para liderar dentro dunha rede de experiencia, asegurando que o traballo se fai dun xeito robusto e que pode ser mantido.

Financiando actividades para alén da programación

A programación soamente é parte do traballo dun proxecto de código aberto. Desde o punto de vista dos voluntarios do proxecto, é a parte máis visible e glamurosa. Desafortunadamente, isto significa que as outras actividades, tales como documentación, testaxe formal, etc., ás veces poden desatenderse, polo menos comparadas coa cantidade de atención que acostuman a recibir no software propietario. As corporacións ás veces son capaces de completar isto, mediante a dedicación de parte da súa infraestrutura interna de desenvolvemento de software a proxectos de código aberto.

A chave para facelo con éxito é mediar entre os procesos internos da compañía e os da comunidade pública de desenvolvemento. Esta mediación require esforzo: habitualmente as dúas non son unha parella próxima, e as diferencias só poden ser salvadas mediante intervención humana. Por exemplo, a compañía pode empregar un sistema de notificación de erros distinto ao do proxecto público. Mesmo se ambas empregan o mesmo sistema de notificación de erros, os datos que se almacenan poden ser moi distintos, xa que as necesidades de notificación de erros dunha compañía son moi distintos das dunha comunidade de software libre. Unha peza de información iniciada nun sistema de notificación necesita ser reflectida no outro, cos anacos confidenciais borrados ou, na outra dirección, engadidos.

As seccións que seguen tratan o tema de como construír e manter estas pontes. O resultado final debería ser que o proxecto de código aberto avanza máis suavemente, a comunidade recoñece os investimentos e recursos da empresa, e mesmo non sente que a empresa está conducindo inapropiadamente as cousas cara aos seus propios obxectivos.

Garantía da calidade (por exemplo, testaxe profesional)

No desenvolvemento de software propietario, é normal ter equipos de xente dedicados unicamente á garantía da calidade: caza de erros, testaxe de rendemento e escalabilidade, revisión de interfaces e documentación, etc. Como regra, estas actividades non se perseguen tan vigorosamente pola comunidade de voluntarios dos proxectos de software libre. Isto parcialmente débese a que é difícil obter voluntarismo para traballo non glamuroso como a testaxe, en parte porque a xente tende a asumir que tendo unha ampla comunidade de usuarios obtés unha boa cobertura de testaxe; no caso das testaxes de rendemento e escalabilidade, en parte débese a que os voluntarios non soen ter acceso aos recursos hardware precisos.

A asunción de que ter moitos usuarios é equivalente a ter moitos testadores non se fai completamente sen base. Certamente, hai certo acerto asignando testadores a usuarios nas funcionalidades básicas dos entornos comúns: os erros serán rapidamente atopados polos usuarios no curso natural das cousas. Mais debido a que os usuarios só están facer o traballo, non exploran conscientemente os casos límite da funcionalidade do programa, polo que son propensos a deixaren certas clases de erros sen atopar. Máis aínda, cando atopan un erro cun rodeo sinxelo, acostuman a implementar en silencio o rodeo sen

importarlles a súa notificación. Máis insidiosamente, os padróns de uso dos teus clientes (a xente que dirixe *os teus* intereses no software) poden diferir dun xeito estatisticamente significativo dos padróns de uso do usuario medio da rúa.

Un equipo profesional de testaxe pode descubrir estes tipos de erros, e pode facelo dun xeito tan sinxelo co software libre como co software propietario. O reto é transferir os resultados do equipo de testaxe ao público dun xeito útil. Os departamentos de testaxe internos acostuman a ter un xeito propio de informar sobre os resultados das testaxes, envolvendo a xíria específica da empresa, ou coñecemento especializado sobre clientes particulares e os seus conxuntos de datos. Os devanditos informes poderían ser inapropiados para o sistema de notificación de erros público, tanto pola súa forma como pola cuestión da confidencialidade. Mesmo se o software de notificación de erros da túa empresa for o mesmo que se emprega no proxecto público, a xestión precisaría facer mudanzas nos comentarios e metadatos específicos da empresa debido ás anteriores cuestións (por exemplo para contemplar a cuestión da prioridade interna, ou planificar a resolución para un cliente particular). Normalmente estas notas son confidenciais; ás veces nin se lle mostran ao cliente. Mais mesmo cando non son confidenciais, non pertencen ao proxecto público, e por tanto o público non debería ser distraído con elas.

O núcleo da notificación de erros *é* importante en si mesmo para o público. De feito, unha notificación de erro que parte do teu departamento de testaxe é, desde múltiples puntos de vista, máis valiosa que unha que se reciba dos usuarios, dado que o departamento de testaxe proba cousas que o resto de usuarios non proban. Dado que é pouco probable que vaias recibir esa notificación de erro concreta de ningunha outra fonte, é importante que a conserves e a tornes dispoñible para o proxecto público.

Para facelo, ou ben o departamento de garantía de calidade arquiva estas cuestións directamente no sistema público de notificación (se se senten cómodos facéndoo) ou ben un intermediario (habitualmente un dos desenvolvedores) pode "traducir" os informes internos do departamento de testaxe en novas incidencias no sistema público de notificación de erros. A tradución só quere dicir describir o erro dun xeito que non faga referencia á información específica do cliente (o xeito de reprodución podería conter datos do cliente, asumindo que o cliente o aprobe, por suposto).

O preferible é ter o departamento de garantía de calidade arquivando directamente as incidencias no sistema de notificación público. Isto dálle ao público unha apreciación máis directa da participación da túa empresa no proxecto: as notificacións de erros útiles dánlle á túa empresa tanta credibilidade como calquera contribución técnica. Ademais, tamén lle dá aos desenvolvedores un canal de comunicación co equipo de testaxe. Por exemplo, se o equipo interno de garantía de calidade esta monitorizando o sistema público de incidencias, un desenvolvedor pode inserir unha solución para un erro de escalabilidade (para o cal o desenvolvedor podería non ter os recursos precisos por si mesmo para facer o test), e logo engadir unha nota á incidencia pedíndolle ao equipo de garantía de calidade que comprobe se a solución produce o efecto desexado. Conta con algunha resistencia por parte dalgúns desenvolvedores; os programadores teñen a tendencia de considerar a garantía de calidade como, no mellor caso, un mal necesario. O equipo de garantía de calidade pode vencer esta tendencia dun xeito sinxelo atopando erros significativos e arquivando informes comprensibles; por outra banda, se os seus informes non son polo menos tan bos como os que veñen dun usuario medio da comunidade, entón non ten sentido que interaxan directamente co equipo de desenvolvemento.

De calquera xeito, unha vez que existe unha incidencia pública, a incidencia interna orixinal podería simplemente referenciar á incidencia pública polo contido técnico. A xestión e os desenvolvedores en nómina deberían continuar comentando a incidencia interna empregando comentarios específicos da

empresa cando for necesario, mais deberían usar a incidencia pública para a información que debería estar dispoñible por todo o mundo.

Debes entrar neste proceso esperando unha sobrecarga extra. Manter dúas incidencias para un único erro é, naturalmente, máis traballo que manter unha soa incidencia. O beneficio está en que máis desenvolvedores van ver o informe de erros e van poder contribuír na súa solución.

Consellos legais e protección

As corporacións, con ou sen ánimo de lucro, son case as únicas entidades que sempre prestan atención ás complexas cuestións legais do software libre. Os desenvolvedores individuais habitualmente entenden os matices de varias licenzas de código aberto, mais xeralmente non teñen o tempo ou recursos para seguir en detalle as leis dos dereitos de copia, marcas comerciais e patentes. Se a túa empresa ten un departamento legal, pode colaborar cun proxecto examinando o status dos dereitos de copia do código, e botándolle unha man aos desenvolvedores para entenderen as posibles cuestións de patentes e marcas comerciais. As formas exactas que esta axuda pode adoptar discútense no *Capítulo 9 Licenzas Copyrights e Patentes*. O principal é asegurar que a comunicación entre o departamento legal e a comunidade de desenvolvemento, se a houber, ocorra cunha mutua apreciación dos moi diferentes universos dos que veñen ambas partes. En ocasións, estes dous grupos non se comunican entre si, cada parte asume que a outra posúe coñecementos específicos do dominio que en realidade non ten. Unha boa estratexia consiste en ter unha ligazón (normalmente un desenvolvedor, ou incluso un avogado con experiencia técnica) entre ambos que faga as traducións entre ambos cando se precisaren.

Documentación e usabilidade

A documentación e maila usabilidade son famosos puntos fracos nos proxectos de código aberto, aínda que creo que, polo menos no caso da documentación, a diferenza entre o software libre e o propietario é esaxerada con frecuencia. Non obstante, é empiricamente certo que moito software de código aberto padece da falta de documentación de primeira clase, e de investigación na usabilidade.

Se a túa organización quere axudar a cubrir estas deficiencias nun proxecto, probablemente a mellor cousa que poida facer sexa contratar xente que *non* sexa desenvolvedora habitual no proxecto, mais que sexa capaz de interaxir produtivamente cos desenvolvedores. A non contratación de desenvolvedores habituais do proxecto é boa por dúas razóns: primeiro, deste xeito non restas tempo de desenvolvemento no proxecto; segundo, a xente máis próxima ao proxecto normalmente é a xente equivocada para escribir a documentación ou investigar na usabilidade, xa que teñen dificultades para ver o software desde o punto de vista dun foráneo.

Porén, segue sendo necesario, para calquera que traballar nestes problemas, comunicarse cos desenvolvedores. Atopa xente suficientemente técnica para falar co equipo de codificación, mais non tan experta no software como para que non poida sentir empatía cos usuarios medios.

Un usuario de nivel medio probablemente sexa a persoa correcta para escribir unha boa documentación. De feito, despois de que a primeira edición deste libro foi publicada, recibín o seguinte correo electrónico dun desenvolvedor de software libre chamado Dirk Reiners:

> Un comentario sobre diñeiro, documentación e usabilidade: cando nós tivemos algúns cartos para gastar e decidimos que un titorial para principiantes era a peza máis fundamental que necesitabamos, contratamos un usuario de nivel medio para escribila. El recibira unha introdución ao sistema o bastante recente como para lembrar os problemas, e como os tiña sufrido sabía como describilos. Isto permitíalle escribir algo que necesitase só correccións menores polos desenvolvedores principais para as cousas que aínda non tiña comprendido ben.

O seu caso foi mesmo mellor, dado que iniciara un grupo de xente (estudantes) ao sistema, polo que combinou a experiencia de varias persoas, o cal é algo que só foi unha afortunada coincidencia e probablemente sexa difícil de conseguir na maioría dos casos.

Fornecendo aloxamento/largo de banda

Para un proxecto que non estiver a empregar aloxamento gratuíto (mira a sección chamada *"Escollendo un sitio preconfigurado"* no *Capítulo 3 Infraestrutura Técnica),* fornecer un servidor, conectividade de rede e, o máis importante, axuda coa administración de sistemas, pode ser de importancia capital. Mesmo se isto for todo o que a túa organización fai polo proxecto, pode ser un camiño moderadamente efectivo para obter bo karma nas relacións públicas, aínda que non che ofreza ningunha influencia sobre a dirección do proxecto.

Probablemente debas contar cun cartaz publicitario ou un recoñecemento na páxina de inicio do proxecto, agradecéndolle á túa empresa a provisión de aloxamento. Se configurares o aloxamento de xeito que o enderezo web do proxecto fique baixo o nome de dominio da túa empresa, vas obter algunha asociación adicional a través da URL. Isto provocará que moitos usuarios pensen no software como se tivese *algo* que ver coa túa empresa, mesmo se non contribuíres nada ao desenvolvemento. O problema está en que os desenvolvedores tamén son conscientes desta tendencia asociativa, e poden non sentirse moi cómodos con ter un proxecto baixo o teu dominio a non ser que vaias contribuír con máis recursos que unicamente o largo de banda. Despois de todo, hoxe en día existen unha morea de lugares nos que ter o aloxamento. A comunidade pode eventualmente sentir que a deslocalización de identidade implícita non é compensada polo aloxamento, e por tanto decidir mover o proxecto a outro lugar. Se queres fornecer aloxamento, faino, mais ou ben pensa participar máis a curto prazo, ou ben sé cauteloso sobre canta participación reivindicas.

Mercadotecnia

Embora a maioría dos desenvolvedores de software libre probablemente odien admitilo, a mercadotecnia funciona. Unha boa campaña de marketing *pode* crear ruxerruxe sobre dun produto de código aberto, mesmo ata o punto no que os desenvolvedores máis obstinados se atopen a si mesmos tendo pensamentos positivos sobre o software por razóns intanxibles. Non procede que eu aquí diseccione as dinámicas xerais da mercadotecnia. Calquera corporación involucrada no software libre poderá nunha altura determinada considerar como venderse a si mesma, o software ou a súa relación co software. O consello de abaixo versa sobre como evitar riscos neste esforzo; revisa tamén a sección "*Publicidade*" no *Capítulo 6 Comunicacións*.

Recorda que estás sendo observado

Polo ben de manter ao teu carón a comunidade de desenvolvedores voluntarios, é *moi* importante non dicires nada que non for certo. Revisa cada petición coidadosamente antes de facela, e dálle ao público os medios para revisar as túas peticións por si mesmos. A revisión independente dos feitos é unha parte fundamental do software libre, e envolve máis que unicamente o código fonte.

Naturalmente, de tódolos xeitos ninguén lle aconsellaría ás empresas faceren reivindicacións non verificables. Mais coas actividades de código aberto, hai unha cantidade de xente inusitadamente elevada con experiencia en verificar reivindicacións; así como probablemente xente con accesos a internet de moito largo de banda e cos contactos sociais adecuados para publicitar dun xeito daniño as cousas que poderían chegar a atopar, se así o decidisen. Cando as industrias "Global Megacorp Chemical" contaminan un regato, isto é verificable unicamente por científicos adestrados, os cales poden ser refutados polos científicos de "Global Megacorp", deixando o público rascando a cabeza e preguntándose que pensar. Pola outra banda, o teu comportamento no mundo do software libre non só é visible e gravado; tamén é facilmente revisable dun xeito sinxelo e independente polo resto da xente, permitíndolle chegar ás súas propias conclusións e espallalas. Estas redes de comunicación están ben, son a esencia de como opera o software libre, e poden ser empregadas para transmitir calquera tipo de información. A refutación soe ser difícil, cando non imposible, especialmente cando o que a xente está a dicir é certo.

Por exemplo, está ben referirte á túa organización como "investidora do proxecto X" se realmente o é. mais non te refiras a ti mesmo como "o creador de X" se a maioría do código foi escrito por xente externa. Á inversa, non reivindiques ter unha comunidade de voluntarios profundamente involucrada se calquera pode mirar no teu repositorio e ver que hai poucas ou ningunha mudanza no código provenientes de xente externa á túa organización.

Non hai demasiado tempo, vin o anuncio dunha empresa moi coñecida de informática dicindo que ían liberar un importante pacote de software baixo unha licenza de software libre. Cando saíu o anuncio inicial, deille unha revisión ao seu repositorio público e vin que contiña unicamente tres revisións. Noutras palabras, tiñan feita unha importación inicial do código fonte, mais realmente non acontecera nada desde aquela. Isto en si mesmo non era tan triste, mais xusto acababan de facer o anuncio, despois de todo. Non había ningunha razón para non esperar unha morea de actividade de desenvolvemento de aí en diante.

Algún tempo despois, fixeron outro anuncio. Aquí está o que dixeron, co nome e número de versión substituído por seudónimos:

> É para nós un pracer anunciar que seguindo rigorosas testaxes pola Comunidade Singer, Singer para Linux e Windows agora mesmo está listo para o seu uso en producións.

Curioso por saber que descubrira a comunidade na "testaxe rigorosa" volvín ao repositorio para ver a súa historia de mudanzas recentes. O proxecto aínda estaba na revisión 3. Aparentemente, non atoparan *ningún* erro ou solución antes da publicación! Pensando que os resultados da testaxe da comunidade tiñan que ter sido gravados, a continuación examinei o notificador de erros. Había exactamente seis incidencias abertas, catro das cales estiveran abertas durante varios meses.

Isto resta credibilidade, por suposto. Cando os testadores revisan por algún tempo unha peza de software grande e complexa, atopan erros. Mesmo se as reparacións destes erros non se fan na seguinte publicación, si que esperas algunha actividade de control de versións como resultado do proceso de testaxe, ou polo menos algunha incidencia nova. En base ás aparencias, non pasara nada entre o anuncio da licenza de software libre e a primeira publicación con código aberto.

A cuestión non é que a empresa estivera mentindo sobre a testaxe da comunidade. Non sei se o fixeron ou non. Mais omitían tantos datos a respecto da revisión como se estiveran mentindo. Dado que nin o sistema de control de versións nin o sistema de notificación de erros daban ningunha indicación de que a mencionada testaxe rigorosa tivera lugar, a empresa non debería ter feita a reivindicación en cuestión ou debería ter fornecido algunha ligazón clara a algún resultado tanxible da testaxe ("Atopamos 278 erros, preme aquí para veres os detalles"). A segunda opción permitiríalle a calquera obter rapidamente unha visión sobre o nivel de actividade da comunidade. Tal como vimos, só me levou uns poucos minutos comprobar que fose o que for esta comunidade de testaxe, non deixara ningunha traza en ningún dos lugares habituais. Non é moito traballo, e estou seguro que non son o único que se decatou do problema.

A transparencia e a verificabilidade tamén son unha parte importante dun correcto recoñecemento, por suposto. Revisa a sección "*Crédito*" no *Capítulo 8 Xestionando Voluntarios* para profundizar.

Non confrontes produtos de código aberto

Abstente de dar opinións negativas sobre produtos competidores de código aberto É perfectamente lexítimo amosar *feitos* -negativos, isto é, asercións facilmente confirmables do tipo habitualmente visible nas boas comparativas. Mais as caracterizacións negativas dunha natureza menos rigorosa é mellor eliminalas por dúas razóns. Primeiro, son responsables de comezar guerras incendiarias que desangran as discusións produtivas. Segundo, e máis importante, varios dos desenvolvedores voluntarios do *teu* proxecto tamén poden deixar de traballar no proxecto da competencia. Isto é máis probable do que pode parecer nunha primeira volta: os proxectos xa están no mesmo dominio (isto é polo que son competencia), e os desenvolvedores con experiencia nese dominio poden facer contribucións en calquera lugar no que a súa experiencia sexa aplicable. Mesmo cando non hai solapamento directo de desenvolvedores, é probable que os desenvolvedores do teu proxecto estean polo menos familiarizados cos desenvolvedores de proxectos relacionados. A súa habilidade para manteren ligaduras persoais produtivas pode verse dificultada por demasiadas mensaxes negativas provenientes do marketing.

A confrontación cos produtos de código pechado parece estar máis amplamente aceptada no mundo do software libre, especialmente cando estes produtos están feitos por Microsoft. Persoalmente, eu deploro esta tendencia (unha vez máis, non hai nada malo nas comparacións francas e que se cinguen aos feitos), non unicamente porque sexa de mala educación, senón tamén porque é perigoso para un proxecto comezar a crer a súa propia propaganda e dese modo ignorar os ámbitos nos cales a competencia pode realmente ser superior. En xeral, vixía o efecto que as declaracións de marketing poden provocar na túa propia comunidade de desenvolvemento. A xente podería excitarse tanto ao ter cobertura por parte do marketing que podería chegar a perder a obxectividade sobre os puntos fortes e fracos reais do software. É normal, e esperable, que os desenvolvedores dunha empresa exhiban un certo distanciamento a respecto das declaracións de marketing, mesmo nos foros públicos. Claramente, eles non deberían contradicir de xeito directo as mensaxes de marketing (a non ser que realmente sexan

falsas, aínda que é de esperar que ese tipo de cousas fosen descubertas máis cedo). Poderían rirse disto de cando en vez, como un xeito de tornar o resto da comunidade de desenvolvemento máis realista.

Capítulo 6. Comunicacións

A habilidade de escribir dun xeito claro quizais sexa a capacidade máis importante que unha persoa poida ter nun entorno de software libre. A longo prazo ten maior importancia que o talento programando. Un gran programador con habilidades comunicativas pobres só pode facer unha cousa á vez, e mesmo poder ter problemas para convencer a outros para que lle presten atención. Mais un programador mediocre con boas capacidades comunicativas pode coordinar e persuadir unha morea de xente para que faga distintas cousas, e dese xeito ter un efecto significativo na dirección do proxecto e no seu ímpeto.

Non parece haber moita correlación, en ningunha dirección, entre a capacidade de escribir bo código e mais a habilidade de comunicarse co resto dos compañeiros humanos. Si que hai correlación entre programar ben e describir tecnicamente ben as cousas, mais describir tecnicamente ben as cousas é unha parte ínfima das comunicacións nun proxecto. Moito máis importante é a capacidade de empatizar coa audiencia, de ver os correos e comentarios propios como os vería outra persoa, e facer que os demais vexan os seus propios correos cunha obxectividade similar. Igualmente importante é notificares cando un método de comunicacións non está funcionando ben, quizais porque non escale ben co incremento do número de usuarios, e tomáreste o tempo preciso para faceres algo ao respecto.

Todo isto na teoría é obvio. O que o fai difícil na práctica é que son moi diversas tanto no público como nos mecanismos de comunicación. Un pensamento debería ser expresado nunha mensaxe na lista de correo, como anotación no seguimento de erros, ou como comentario no código? Cando se responde unha pregunta nun foro público, canto debemos supor que sabe o lector, dado que o «lector» non vai ser só o que responda a pregunta en primeiro lugar, senón que tamén serán aqueles que vexan a túa resposta? Canto deben estar os programadores en contacto construtivo cos usuarios, sen seren inundados por solicitudes de funcións, informes de erros e charla xeral? como avisas cando un medio alcanzou o límite da súa capacidade e que fas?

As solucións a estes problemas normalmente son parciais, porque unha solución particular converterase en obsoleta cando medre o proxecto ou se dean mudanzas na estrutura deste. A miúdo tamén son ad hoc, porque se improvisan respostas a situacións dinámicas. Todos os participantes teñen que ser conscientes de cando e como as comunicacións poden chegar a estancarse e estar involucrados nas solucións. Axudar a xente a facer isto é unha grande parte de xestionar un proxecto fonte aberto. As seccións que seguen a continuación discuten sobre como conducir as túas propias comunicacións e como facer que os mecanismos de comunicación sexan unha prioridade para todos os que se atopan no proxecto[15].

Es o que escribes

15Sobre este tema tense realizado algunha investigación interesante; por exemplo, a publicación *Group Awareness in Distributed Software Development* de *Gutwin, Penner e Schneider*. Esta publicación estivo en liña por un tempo, logo non dispoñible e de novo en liña en http://www.st.cs.uni-sb.de/edu/empirical-se/2006/PDFs/gutwin04.pdf. Polo tanto usa esa ligazón primeiro, mais talvez terás que usar unha ferramenta de procura se o moveren de novo

Considera que a única cousa que todo o mundo sabe de ti en Internet virá polo que escribes o que os demais escriben sobre ti. Pode que sexas unha persoa brillante, perceptiva e carismática, mais se os teus emails son demasiado extensos e non teñen estrutura, a xente asumirá que ti tamén es así. Ou quizais si que es unha persoa que fala de máis e pouco estruturada, mais ninguén ten por que sabelo se as túas mensaxes son lúcidas e informativas.

Prestarlle atención ao que escribes paga a pena. O hacker Jim Blandy de software libre a longo prazo contaba a seguinte historia:

> «En 1993 estaba traballando para a Free Software Foundation e estabamos probando a versión 19 of GNU Emacs. Faciamos unha versión beta cada semana ou así, e a xente tiña que probala e mandarnos reportes de erros. Había un tipo que ninguén de nós coñecera en persoa mais que facía un gran traballo: os seus informes de erros eran sempre moi claros e contaba sempre directamente o problema, e cando nos achegaba el un arranxo, case sempre estaba no certo. Era moi bo.
>
> Agora, antes de que a Free Software Foundation puidera usar un código escrito por outra persoa, tiñamos que facer que asinasen algúns documentos legais para asinarlle o copyright a ese código para a FSF. Tomar un código dun descoñecido e deixalo é un indicio de catástrofe legal.
>
> Así que lle enviei ao tipo os formularios dicíndolle «precisamos ter listo este papelame, asina este documento, este que o asine o teu xefe e despois podemos comezar a pagarche. Moitas grazas».
>
> Entón respondeume «non teño xefe».
>
> Así que escribinlle «Dacordo, non te preocupes. Consegue a firma da universidade e será suficiente».
>
> Ao cabo dun momento, respondeume dicindo «É que en realidade… Teño trece anos e vivo con meus pais».

Como ese rapaz non escribía como alguén de trece anos, ninguén supuña que os tiña. A continuación temos algunhas formas para facer que a túa escrita dea unha boa impresión tamén.

Estrutura e formato

Non caias no erro de escribir todo coma se fose unha mensaxe ao móbil. Escribe frases completas, coa primeira letra de cada unha delas en maiúscula, e utiliza as parénteses onde foren necesarias. Isto é o máis importante na redacción de correos ou doutros textos. No Internet Relay Chat ou noutros foros deste tipo, xeralmente non é un erro esquecer as maiúsculas, utilizar contraccións ou expresións comúns… O que debes ter en conta é non trasladar estes hábitos a contornos formais e foros máis persistentes. Os correos, os documentos, os reportes de erros e outras formas de escrita están destinados

a durar para sempre, así que deberían ser escritos utilizando a gramática e a ortografía estándares, e ter unha estrutura narrativa coherente. Isto non é así non porque non sexa bo seguir as normas arbitrarias, senón porque estas normas non son arbitrarias: evolucionan á forma do presente porque fan que os textos sexan máis fáciles de ler, e deberías seguir isto por esta razón. É desexable que o texto sexa fácil de ler non só porque significa que a xente entenderá o que escribes, senón porque fará que parezas o tipo de persoa que toma o seu tempo para comunicarse claramente; é dicir, alguén a quen paga a pena prestar atención.

Para os correos, en particular, os expertos, os programadores, sentaron unha serie de convencións:

Envía só textos sen formato; nin HTML, nin textos enriquecidos ou outros formatos que poidan ser pouco claros para os lectores de correos. Dálles formato ás túas liñas sobre 72 columnas. Non excedas as 80 columnas, que se converteron no estándar de largura (pode que algunha xente utilice terminais máis largos, mais ninguén utilizará terminais máis estreitas). Se escribires as túas liñas en menos de 80 columnas, deixarás espazo para as citas que se engaden nas respostas, sen que haxa que axustar o texto.

Utiliza quebras de liñas reais. Algúns servizos de correo axustan o texto de forma errada, de forma que cando estás compoñendo un correo electrónico, visualízanse quebras de liña que en realidade non existen. Cando se envía o correo, pode que non teña quebras de liña que pensabas que tiña, e axustarase mal nas pantallas dos demais. Se pode que o teu servizo de correo use quebras de liña falsas, busca un configuración axeitada coa que poidas mostrar as quebras de liña verdadeiras tal e como ti as compós.

Cando inclúas saídas de pantalla, fragmentos de código, ou outro texto preformatado, compénsao de forma que mesmo un ollo vago poida ver claramente os límites entre a túa prosa e o material que citas. (Nunca pensei escribir este consello ao comezar este libro, mais ultimamente, nun número extenso de listas de correo fonte abertas, vin cantidade de mesturas de textos de distintas fontes sen que quedase claro cal era cal. O efecto é frustrante. Fai que as súas mensaxes sexan difíciles de entender, e francamente, fai que esa xente pareza desorganizada).

Cando cites o correo doutra persoa, insire as túas respostas onde for máis apropiado; en distintos lugares se for necesario, e corta as partes do correo que non fagan falta. Se escribes un comentario rápido que responde ao correo en conxunto, non está mal un top-post (isto é, situar a túa resposta sobre o texto citado do seu correo), doutra forma, deberías citar a parte relevante do texto orixinal, seguido da túa resposta.

Constrúe as liñas do asunto dos novos correos coidadosamente. É a liña máis importante do teu correo, porque permite que todas as persoas do proxecto decidan se queren ler máis ou non. O software de lectura moderno organiza grupos de mensaxes relacionadas en fíos, que poden estar definidos non só polo asunto común, senón por outros cabezallos (que normalmente non se mostran). É lóxico que se un fío comeza a desembocar nun novo tema, podes (e deberías) axustar o asunto de acordo coa resposta. A integridade dos fíos conservarase, debido aos outros cabezallos, mais o novo asunto axudaralle á xente a ter unha visión xeral do fío agora que o tema foi desviado. Así, se é que queres comezar un novo tema, faino cun novo correo, non respondendo un que xa existe mudándolle o asunto, senón, o teu correo estará clasificado no mesmo fío ao que respondes, e fará que a xente crea que trata sobre o mesmo, cando non é así. De novo, a consecuencia non só será a perda de tempo, senón que afectará levemente á túa credibilidade como persoa con habilidade en utilizar ferramentas de comunicación.

Contido

Os correos cun formato apropiado captan a atención do lector, mais é o contido a que a mantén. Non hai normas fixas que poidan garantir un bo contido, claro está, mais existen algúns principios que o fan máis probable.

Facilítalles a cousa aos lectores. Hai un montón de datos en cada proxecto fonte, e o lector non pode esperar que todo lle sexa familiar; de feito, non pode esperar saber como lle pode chegar a ser familiar. Dentro do posible, as túas mensaxes deberían proporcionarlle información ao lector da forma máis conveniente. Pasar dous minutos extra para investigar a URL para un fío particular nos arquivos da lista de correo para aforrarlle o traballo ao teu lector, paga a pena. Se tes que pasar de cinco a dez minutos resumindo a conclusión dun fío complexo para darlle á xente un contexto e que así entendan a túa mensaxe, entón faino. Pensa desta forma: canto máis éxito teña o proxecto, maior será o número de lectores en calquera foro. Se cada mensaxe que mandas é vista por N persoas, entón canto máis incremente N, máis pagará a pena calquera esforzo para aforrarlles o traballo a esa xente. E cando a xente ve que te estás impoñendo isto, traballarán para igualar as súas propias mensaxes. O resultado é, idealmente, un aumento na eficiencia mundial do proxecto: cando hai que elixir entre N persoas facendo un esforzo e unha persoa facéndoo, pois o mellor para o proxecto é este último caso.

Non utilices hipérboles. É común e esaxerar nas mensaxes en Internet. Por exemplo, pode que a unha persoa que envía un informe de erros non lle chame suficiente a atención, así que o describirá de forma seria e como un problema crítico que lle impide (e aos seus amigos/compañeiros de traballo/curmáns) utilizar o software produtivamente, cando en realidade non é máis ca un problema leve. Esaxerar non se limita aos usuarios; os programadores fan o mesmo acotío nos os debates técnicos, particularmente cando discuten sobre gusto máis que sobre corrección:

> «Ao facelo así, faría que o código fose ilexible. Sería un pesadelo de mantemento, comparado co propósito de J. Random...»

A mesma opinión faise máis forte cando se expresa con menos dureza:

> «Isto funciona, mais é menos ca un ideal no relativo a lexibilidade e mantemento, penso. A proposta de J. Random evitaría estes problemas porque…»

Non seredes quen de desfacervos das hipérboles completamente, e, en xeral, non é necesario facelo. Comparado con outras formas de erros de comunicación, a hipérbole non é moi grave; prexudica só ao que escribe a mensaxe. Os receptores poden compensala, mais o emisor perde cada vez máis credibilidade. Polo tanto, polo ben da túa influencia no proxecto, tenta tirar cara a moderación. Desa forma, cando *si* que precisares recalcar algo, a xente tomarate en serio.

Edita dúas veces. Le calquera mensaxe que ocupe máis de medio parágrafo de arriba a abaixo antes de envialo, mesmo despois de lelo unha primeira vez. Este consello elles familiar a aqueles que tiveron clase de redacción, mais é que especialmente importante na discusión en liña. Como o proceso da redacción en liña tende a ser altamente descontinuo (no curso no que escribes unha mensaxe pode que teñas que volver e ler outras mensaxes, visitar páxinas e escribir un comando para capturar unha saída de depuración, entre outras cousas), é especialmente sinxelo perder o fío. As mensaxes compostas de forma descontinua e que non se revisaron antes de envialas, son recoñecidas moitas veces, para o desgusto (ou polo menos é o que se espera) dos seus autores. Toma o tempo preciso para revisar o que envías. Canto máis xuntas estean as túas mensaxes estruturalmente, máis serán lidas.

Ton

Despois de escribires milleiros de mensaxes, o teu estilo converterase en cada vez máis seco. Isto parece se-la norma na maioría dos foros técnicos, e non hai nada malo en que así sexa. Un punto de brevidade e rudeza que sería inaceptable nas iteracións sociais cotiás é a norma para os hackers do software libre. Poño a continuación algunhas respostas que obtiven nunha rolda de correo sobre xestores libres de contidos, van entre aspas:

> Podes explicar máis exactamente que problemas estás sufrindo?
>
> Tamén:
>
> Que versión de Slash usas? Non puiden obtelo da mensaxe orixinal.
>
> Como tes construído o código do apache/mod_perl?
>
> Probaches o parche para Apache 2.0 que foi publicado en slashcode.com?
>
> -- Shane

Iso é sequidade! Sen un saúdo previo, sen firma máis alá do seu nome e a mensaxe mesmo sendo unicamente unha serie de preguntas tan compactas como sexa posible. Cada pregunta en si mesma era unha crítica implícita á miña orixinal mensaxe. Porén, alegroume recibir a mensaxe de Shane, e non tomei a resposta como un signo de mala educación, senón de que era unha persoa ocupada. O mero feito de que me estivese preguntando, en vez de ignorar a miña mensaxe, quería dicir que estaba disposto a dedicar un pouco do seu tempo no problema.

Reaccionarían todo tipo de persoas positivamente a este correo? Non necesariamente; dependerá da persoa e do contexto. Por exemplo, si alguén ten posteado recoñecendo que se equivocou (talvez escribiu un anaco de código con algún erro) e sabes por experiencias anteriores que esta persoa tende a ser un pouco insegura no código que envía, é posible que, aínda que escribas unha resposta lacónica, tamén desexes poñer algo que a faga sentir mellor. A resposta pode ser breve, unha análise técnica da situación, tan concisa como desexes. mais ó final, remata deixando claro que a concisión non se debe tomar como un signo de rudeza. Por exemplo, se tes escrito un consello de como debe resolve-lo bug, podes asinar o correo cun "Boa sorte, <o teu nome aquí>" para indicar que lle esperas o mellor dun modo positivo. Un emoticono colocado de modo estratéxico pode ser tamén suficiente.

Pode parecer raro poñer os sentimentos dos participantes ao mesmo nivel que a propia superficie do que escribe, mais debemos recoñecer que os sentimentos afectan a produtividade. Os sentimentos son importantes por outras razóns tamén, mais incluso levándoo ao argumentarao puramente utilitarista, é posible observar que xente infeliz escribe peor e menos software. Porén, dada a restrinxida natureza dos medios electrónicos, non é sinxelo habitualmente saber como se está a sentir unha persoa. Terás que facer unha estimación baseándote en a) como a maioría da xente se sentiría nesa situación, e b) que sabes desta persoa por pasadas iteracións con ela. Algunha xente prefire facelo doutro modo, e tratar

con todo o mundo sen entrar no terreo das emocións; que si unha persoa non di claramente como se sente, un non ten dereito a tratala como si o souber. Non estou de acordo. Por un par de razóns. Primeiro, a xente non se comporta dese modo na vida offline, entón por que debería facelo online? A segunda é que, debido a que a maioría das iteracións teñen lugar en foros públicos, a xente tende a restrinxir máis o ámbito das emocións que en privado. Por ser máis concreto, a xente está máis disposta a expresar emocións dirixidas a outros, como gratitude ou indignación, mais non as internas, como inseguridade ou orgullo. Porén, os seres humanos traballan mellor cando eles saben que os outros se preocupan por como están. Prestando atención aos pequenos detalles, podes adiviñar o que ocorre a maioría das veces, e motivar á xente para que se involucre en un grado maior do que faría doutro modo.

Desde logo, non pretendo dicir que o teu rol sexa de psicoterapeuta, axudando todo o mundo e sendo consciente en todo momento dos seus sentimentos. Mais prestando atención aos padróns de comportamento a longo prazo, poderás obter unha imaxe mental deles como individuos mesmo se xamais os viches cara a cara. Sendo precavido do teu propio ton ao escribires as mensaxes, podes ter unha cantidade incrible de influencia sobre os sentimentos dos outros, para maior beneficio do proxecto.

Recoñecer a rudeza

Unha das características definitorias da cultura do software libre é a noción do que é rudeza e do que non. Aínda que as convencións descritas a continuación non son únicas no mundo do software libre, nin sequera únicas do mundo do software en xeral -serán familiares a calquera que traballe no ámbito das matemáticas, ciencias duras ou enxeñerías- o mundo do software libre, coas súas fronteiras porosas e constante fluxo de xente nova, é un ambiente onde estas convencións van a ser vistas por xente non familiarizadas con elas.

Empecemos coas convencións do que *non* é rudeza:

Críticas técnicas, aínda cando foren directas e sen tacto, non é rudeza. Pode mesmo ser visto como un eloxio: implicitamente, o crítico está dicindo que se está tomando en serio o que dis, e vale a pena dedicarlle algún tempo. É dicir, o mellor tería sido simplemente ignorar esa mensaxe, tomarse un tempo para criticala ten o seu punto de eloxio (a menos que a crítica se converta nun ataque *ad hominen* ou algún outro modo de rudeza, desde logo).

Preguntas desproveitas de todo adorno, directas, tales como as anteriores de Shane, tampouco poden considerarse rudas. Preguntas que noutros contextos poden parecer frías, retóricas, ou incluso burlescas, habitualmente son formuladas seriamente, e non teñen maior intención que extraer información o máis rápido posible. A famosa frase de "está o seu ordenador encendido?" é un exemplo clásico disto. A persoa que traballa en soporte técnico precisa saber se a computadora está acesa, e despois dos primeiros días de traballo, está canso de engadir previamente á súa pregunta educados acompañamentos como "desculpe vostede, vou a facerlle unhas preguntas moi sinxelas para descartar certas posibilidades. Algunhas delas poden parecer moi básicas, mais teña un pouco de paciencia...". Nese momento, xa non lle preocupan as frases de recheo e vai directamente ó esencial. Preguntas dese estilo son realizadas todo o tempo nas listas de correo do movemento do software libre. Non pretenden insultar á persoa que enviou a mensaxe, senón descartar as máis obvias (e talvez máis comúns) explicacións. Os que comprendan esta lóxica e reaccionen de acordo con ela están gañando puntos respecto a aqueles que non. Aínda que a estes últimos tampouco se lles pode reprochar a súa actitude. O

conflito reside unicamente nun choque de culturas, non no erro de ninguén. Dun modo amigable debes comentar que a pregunta (ou crítica) non tiña significados ocultos, só desexaba obter a información o máis rápido posible.

Entón ... que é rudeza?

Polo mesmo principio baixo o cal unha crítica técnica detallada é un modo de eloxio, fornecer unha crítica mala pode ser considerado unha certa clase de insulto. Non quero dicir ignorar a contribución de alguén, for esta unha proposta, mudanza no código, petición de novas funcionalidades, etc. A xente asumirá que non tes tempo para dicir nada. Mais se reaccionares con algunha resposta, non escatimes: toma o tempo necesario para analizar as cousas, fornece exemplos concretos cando for apropiado, investiga nos arquivos para buscar post relacionados no pasado, etc. Ou se non tiveres tempo para poñer esa clase de esforzo, mais escribires unha breve resposta, entón comeza a túa mensaxe dicindo algo como "creo que existe algún comentario previo sobre isto, mais non tiven tempo a buscalo, síntoo". O principal consiste en recoñecer a existencia dunha norma cultural, cumpríndoa ou recoñecendo que esta vez non se puido cumprir. Calquera que for a opción escollida, a norma é fortalecida. Non cinguirse á norma implícita, e ó mesmo tempo non dicir *por que*, é como dicir que o asunto (e eses participando nel) non valen o teu tempo. Mellor amosar que o teu tempo é valioso sendo seco que folgazán.

Existen outras moitas formas de rudeza, obviamente. Mais a maioría delas non son específicas do mundo do software libre e o propio sentido común é unha guía máis que suficiente para evitala. Podes botarlle unha ollada á sección "*Evita a rudeza*" no *Capítulo 2 Primeiros Pasos*, se aínda non o fixeches.

Cara a cara

Hai unha rexión do cerebro humano deseñada especificamente para recoñecer caras. Coñécese informalmente como "área de recoñecemento de caras" e as súas capacidades son maiormente innatas, non aprendidas. Parece que o recoñecemento de persoas é unha tarefa crucial na supervivencia e que desenvolvemos hardware especializado para facelo.

A colaboración baseada en internet é tamén psicoloxicamente insólita, xa que ten que ver coa estreita cooperación entre seres humanos que case nunca se recoñecerán uns a outros polos métodos máis naturais, intuitivos: recoñecemento facial, voz, postura, etc. Para compensar todo isto, trata de usar un nome coherente *nick* en todos lados. Iso debería ser a primeira parte do teu correo (antes de @servidor.com), o nome que tes no IRC, o do repositorio de código, etc. Este nome é a túa cara "online": un curto identificador que ten o mesmo propósito que a túa cara real, aínda que desafortunadamente non estimula o mesmo hardware embebido no cerebro.

O alias podería ser algunha permutación intuitiva do teu nome real (o meu, por exemplo, é "kfogel"). Nalgunhas situacións irá acompañado do teu nome completo, por exemplo nas cabeceiras dos correos:

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

No anterior exemplo pódense observar 2 cousas. O alias encaixa dalgún modo intuitivo co nome real. Mais tamén, o nome real é *real*. É dicir, non é un nome como:

```
De: "Maravilloso hacker" <wonderhacer@whateverdomain.com>
```

Paul Steiner, nunha viñeta cómica publicada no *New Yorker* o 5 de xullo de 1993 mostra a un can usando unha computadora e dicíndolle a outro cunha pose conspiranoica: "na internet, ninguén sabe que es un can". Este tipo de crenzas descansan probablemente detrás dun feixe de auto-bombo e identidades en liña infladas que a xente se outorga a si mesmas -como se chamarse a un mesmo "Hacker Maravilloso" fixese que a xente o crese. Mais os feitos son os feitos: aínda que ninguén saiba que es un can, seguiralo sendo. Unha identidade en liña fantasiosa non impresiona os lectores, senón todo o contrario, failles crer que es máis imaxe que substancia ou que simplemente es inseguro. Usa o teu nome real para tódalas interaccións, ou, se por algunha razón precisares usar o anonimato, entón inventa un nome que soe como un real, e úsao con coherencia.

Ademais de usares a túa identidade dixital dun modo cohrente, hai algunhas cousas que poden facela máis atractiva. Si tiveres un título oficial (p. e. "doutor", "profesor", "director"), non te gabes, non o menciones a non ser que for relevante na conversa. O mundo hacker, e o do software libre en particular, tende a ver as insignias e títulos como signos de inseguridade. Non pasa nada si desexas usalo como unha sinatura estándar en cada correo-e que envíes, mais non o uses como un arma que sacar en cada conversa para defender a túa posición -pois terá o efecto contrario. Desexas que a comunidade te respecte a ti, non ó título.

Falando de sinaturas estándar: fai que sexan pequenas e con bo gusto, ou mellor, que non existan. Evita enviar longas cláusulas de exención de responsabilidade anexos a cada correo-e, especialmente cando expresaren sentimentos incompatibles coa participación nunha comunidade de software libre. Por exemplo, o seguinte clásico do xénero aparece ao fin de cada post que un usuario envía a unha lista de correo á que estou subscrito (Nota do tradutor: o seguinte texto reprodúcese no idioma orixinal xa que é un texto legal):

> IMPORTANT NOTICE
>
> If you have received this email in error or wish to read our email disclaimer statement and monitoring policy, please refer to the statement below or contact the sender.
>
> This communication is from Deloitte & Touche LLP. Deloitte & Touche LLP is a limited liability partnership registered in England and Wales with registered number OC303675. A list of members' names is available for inspection at Stonecutter Court, 1 Stonecutter Street, London EC4A 4TR, United Kingdom, the firm's principal place of business and registered office. Deloitte & Touche LLP is authorised and regulated by the Financial Services Authority.
>
> This communication and any attachments contain information which is confidential and may also be privileged. It is for the exclusive use of the intended recipient(s). If you are not the intended recipient(s) please note that any form of disclosure, distribution, copying or use of this communication or the information in it or in any attachments is strictly prohibited and may be unlawful. If you have received this communication in error, please return it with the title "received in error" to

> IT.SECURITY.UK@deloitte.co.uk then delete the email and destroy any copies of it.
>
> Email communications cannot be guaranteed to be secure or error free, as information could be intercepted, corrupted, amended, lost, destroyed, arrive late or incomplete, or contain viruses. We do not accept liability for any such matters or their consequences. Anyone who communicates with us by email is taken to accept the risks in doing so.
>
> When addressed to our clients, any opinions or advice contained in this email and any attachments are subject to the terms and conditions expressed in the governing Deloitte & Touche LLP client engagement letter.
>
> Opinions, conclusions and other information in this email and any attachments which do not relate to the official business of the firm are neither given nor endorsed by it.

Para alguén que simplemente envía algunha pregunta de vez en cando, a anterior cláusula de exención de responsabilidade parece un pouco parva mais probablemente non faga ningún mal. Porén, se a persoa desexase participar máis activamente no proxecto, ese molde legal comezaría a ter efectos insidiosos. Enviaría ao resto da comunidade polo menos 2 sinais: primeiro, que a persoa non ten un control total sobre as ferramentas que usa; está atrapado na maquinaria da compañía para a que traballa; e segundo, que ten pouco ou ningún soporte por parte da súa organización en canto ás súas actividades na comunidade. Si, non lle impiden participar en listas de correo públicas, mais con ese molde legal, fai que as súas mensaxes non sexan moi benvidas, enviando a mensaxe de que información confidencial debe imporse sobre calquera outra prioridade.

Si a túa organización insiste en usar tal final de firma para tódolos correo-e, podes considerar obter unha conta de correo-e gratuíta como por exemplo algunha de http://gmail.google.com www.hotmail.com http://www.yahoo.com usándoa para a túa participación no proxecto.

Evitando os obstáculos comúns

Non envíes correos sen un propósito

Un erro común que acostuma a xurdir durante a participación nun proxecto en liña é pensar que tes que responder a todo. Non o tes que facer. Primeiramente, acostuma a haber máis fíos activos dos que poder seguir en profundidade, cando menos despois dos primeiros meses. Segundo, mesmo nos fíos nos que decidiras participar, moito do que a xente di realmente non precisa dunha resposta. Os foros de desenvolvemento tenden a ser dominados por tres tipos de mensaxes:

1. Mensaxes propoñendo algo que non é trivial
2. Mensaxes expresando apoio ou oposición a algo que alguén dixo
3. Mensaxes de recapitulación ou resumo

Ningún deles, cando menos dun xeito *inherente*, require unha resposta, especialmente se estás bastante seguro (tras revisares o fío) de que alguén é probable que responda dicindo o que ti terías dito. (Se te preocupas de ser atrapado nun bucle de espera-espera porque todos os demais están empregando a mesma táctica, non te preocupes; sempre vai haber *alguén* aí fóra que entre ao trapo). Unha resposta debería estar motivada por un propósito definido. Antes de nada, pregúntate a ti mesmo: sabes que é o que queres conseguir? E en segundo lugar: non se vai conseguir salvo que ti digas algo?

Dúas boas razóns para engadires a túa voz a un fío son a) cando vires un defecto nunha proposta e sospeitares que es o único que o está a ver, e b) cando vires que está habendo falta de comunicación entre os demais, e souberes que a podes subsanar mediante unha mensaxe clarificadora. Tamén acostuma a estar xeralmente ben enviares mensaxes para darlle as grazas a alguén por agradecer algo, ou dicires "eu tamén", para que un lector diga que esas mensaxes non requiren ningunha resposta nin acción adicional, e polo tanto o esforzo mental demandado pola mensaxe remata limpamente cando o lector alcanza a última liña do correo electrónico. Mais mesmo nestes casos, pénsao dúas veces antes de dicires algo; sempre é mellor deixar que a xente desexe que envíes máis mensaxes, que deixar que a xente desexa que envíes menos (bótalle unha ollada á segunda parte de do *Apéndice C Porque me debería importar a cor do garaxe da bicicleta?* para veres máis ideas sobre como comportarse nunha lista de correo moi concorrida.

Fíos produtivos fronte a fíos improdutivos

Nunha lista de correo moi concorrida, hai dous imperativos: Un, obviamente, é comprenderes a que tes que prestarlle atención e que podes ignorar. O outro é comportáreste dun xeito que *evite a aparición* de ruído: ti non queres que unicamente as túas mensaxes teñan unha boa relación sinal/ruído, senón que tamén queres que sexan do tipo de mensaxes que estimulan *o resto da xente* a enviar mensaxes cunha relación similar de sinal/ruído, ou a non escribir nada.

Para ver como facer isto, imos considerar o contexto no que se fai. Cales son algúns dos sinais que identifican un fío improdutivo?

- Empézanse a repetir argumentos xa empregados con anterioridade, debido a que o que os emprega pensa que ninguén os escoitou a primeira vez.
- A medida que as ideas se fan máis e máis pequenas, aumenta o nivel de esaxeración e participación.
- A maior parte dos comentarios proveñen de xente que fai un traballo pequeno, ou mesmo que non fai ningún traballo, mentres que a xente que tende a facer as cousas fica calada.
- Discútense ideas que non teñen un propósito claro. (Por suposto, calquera idea interesante comeza cunha visión imprecisa; a cuestión importante está na dirección que colle a partir dese inicio. Parece que o fío comeza a converterse en algo máis concreto, ou pola contra está divagando con sub-visións, visións laterais ou disputas ontolóxicas?)

Só porque un fío pareza non produtivo nos seus primeiro pasos non quere dicir que sexa unha perda de tempo. De feito, pode tratar sobre un tema importante, en cuxo caso o feito de non producir ningún avance sería o máis molesto.

Conducir un fío cara á utilidade sen seres agresivo é toda unha arte. Amoestares a xente para que deixe

de perder o seu tempo non soe ser suficiente nin funcionar, do mesmo xeito que tampouco o é contestáreslle dicindo que non envíen mensaxes salvo que tiveren algo construtivo co que contribuíren. Ti podes, só faltaría, pensar esas cousas en privado, mais se as expresares abertamente en voz alta entón vas ser ofensivo. En lugar disto, debes propoñer condicións para promoveres o progreso -dálle á xente unha ruta, un vieiro que seguir para alcanzares o resultado que queres, e todo isto sen que pareza que estás ditando a conduta a seguir. A distinción reflíctese en gran medida no ton. Por exemplo, isto non está ben:

> *Esta discusión non vai a ningures. Podemos, por favor, abandonar este tema ata que alguén teña un parche que implemente unha desas propostas? Non hai razón para darlle máis voltas dicindo sempre o mesmo. O código fala máis alto que as palabras, rapaces.*

Mentres que esta si que está ben:

> *Neste fío estiveron convivindo varias propostas, mais ningunha delas suficientemente detallada, ou cando menos sen os detalles suficientes para facer unha votación. E a maiores tampouco estamos dicindo nada novo; simplemente estamos reiterando o xa dito con anterioridade. Así que o mellor, a partir deste punto, probablemente sexa que os correos subsecuentes conteñan ou ben unha especificación completa para a proposta, ou ben un parche. Deste xeito, cando menos contariamos con unha acción definitiva que acometermos (por exemplo, obtermos consenso na especificación, ou aplicarmos o parche).*

Compara a segunda aproximación coa primeira. A segunda proposta non debuxa unha liña entre ti e os demais, nin os acusa de levar a discusión a unha espiral sen fin. Fala sobre "nós", o cal é importante tanto se realmente participaches na discusión antes como se non o fixeches, dado que lle lembra a todo o mundo que mesmo aqueles que gardaron silencio tamén poden contribuír co seu gran de area ao fío. Ademais, describe o motivo polo que o fío está encamiñado cara ningures, mais faino dun xeito non pexorativo, nin xulgando a ninguén -simplemente, e sen paixón, presenta algúns feitos. E, máis importante aínda, ofrece un rumbo de acción positivo, polo que permite que en lugar de que a xente sinta que a discusión se está pechando (unha restrición atractiva contra a que rebelarse), sinta que a discusión se está levando a un nivel máis construtivo. Este é un estándar coa que a xente vai querer quedar.

Non sempre vas queres elevar un fío ao seguinte nivel de construtividade -algunhas veces vas queres liquidalo. Neste caso, o propósito da túa mensaxe debe ser ou ben facer unha cousa ou ben facer a outra. Se podes amosar o camiño polo que andou o fío de xeito que ninguén realmente *vaia* poder seguir os pasos que propós para reconducilo, entón a túa mensaxe vai (efectivamente) pechar o fío sen que pareza facelo. Por suposto, non hai ningún xeito infalible de pechar un fío, e aínda que o houbese, seguramente non quererías empregalo. Aínda así, responderlle aos participantes dicíndolles que ou ben fagan progresos visibles ou ben deixen de enviar mensaxes, pode ser unha acción perfectamente defendible, de facerse diplomaticamente. Porén, debes ser cauteloso, e non liquidar fíos prematuramente. Algunhas charlas especulativas poden ser produtivas, dependo do tema, e pedir que se resolvan con demasiada prontitude poder afogar o proceso creativo, para alén de facerte ver como un impaciente.

Non esperes que ningún fío remate en seco. Probablemente aínda haxa algunhas mensaxes posteriores á túa, tanto porque os correos se cruzaran na rede, como porque haxa xente que queira ter a última palabra. Tan só debes deixar que o fío se esfume, ou non, segundo for o caso. Non podes ter un control completo, mais por outra banda si que podes esperar ter un efecto estatisticamente significativo a través

dos varios fíos.

Canto máis suave for o tema, máis longo vai ser o debate

Aínda que as discusións poden serpear entremedias de calquera tema, a probabilidade de que xurdan aumenta a medida que a dificultade técnica no tema descende. Despois de todo, cando maior for a dificultade técnica, menos participantes poden realmente seguir o que está a pasar. Aqueles que xeralmente poden seguilas son os desenvolvedores máis experimentados, os mesmos que xa tomaron parte centos de veces nesas mesmas discusións, e coñecen os tipos de comportamento que levan a un consenso co que poida vivir todo o mundo.

Deste xeito, o consenso é máis difícil de alcanzar en cuestións técnicas sinxelas de entender e sobre as que é sinxelo ter unha opinión propia, así como en temas máis "suaves" como son a organización, a publicidade, o financiamento, etc. A xente pode participar nestas argumentacións para sempre, dado que non e preciso contar con unha cualificación determinada para poder facelo, non existen vieiros claros para decidir (mesmo a posteriori) se unha postura era correcta ou incorrecta, e simplemente porque discutir ás veces é unha táctica de éxito.

O principio de que a morea de discusión é inversamente proporcional á complexidade do tema ten estado dando voltas por aí desde hai moito tempo, e coñécese informalmente como o *efecto garaxe da bicicleta*. Aquí está a explicación do mesmo, achegada por Poul-Henning Kamp, proveniente dunha mensaxe famosa enviada aos desenvolvedores de BSD:

> É unha longa historia, ou máis ben unha vella historia, mais bastante curta na actualidade. C. Northcote Parkinson escribiu un libro nos comezos de 1960 titulado "A lei de Parkinson", o cal contiña moitos coñecementos sobre a dinámica da xestión.
>
> [...]
>
> No exemplo específico sobre o garaxe da bicicleta, a outra compoñente vital é unha central de enerxía atómica, supoño que para ilustrar a época do libro.
>
> Parkinson mostra como podes acudir ao consello de dirección e conseguir a aprobación para construíres unha central nuclear de varios millóns, ou mesmo miles de millóns, mais tamén como se quixeres construír un garaxe para unha bicicleta posiblemente te vexas envolto en discusións sen fin.
>
> Parkinson explica que isto se debe a que unha central nuclear é algo tan enorme, tan caro e tan complexo que a xente non ten coñecementos sobre a mesma, e en lugar de obtelos, van aferrarse á asunción de que algún terceiro revisará os detalles, antes de ir máis alá. Richard P. Feynmann dá no seu libro un par de exemplos moi interesantes e próximos a este tema, relacionados con Los Alamos.
>
> Pola contra, calquera pode construír un garaxe de bicicletas durante un fin de semana, e contar con tempo suficiente para ver a carreira de Fórmula 1 na televisión. Debido a isto non importa nin o ben preparado que esteas, nin como de razoable sexa a túa proposta, dado que seguramente alguén vai aproveitar a oportunidade para mostrar que está facendo o seu traballo,

> que está prestando atención, para demostrar (en definitiva) que está <emphasis>aí</emphasis>.
>
> En Dinamarca chamámoslle "deixar a túa pegada". Trata sobre o orgullo persoal e mailo prestixio, trata sobre ser capaz de sinalar algún lugar e dicir "Aí! <emphasis>Eu</emphasis> fixen iso". É un trazo con gran presencia nos políticos, mais tamén presente na maioría da xente que ten a oportunidade de poñelo en práctica. Simplemente, pensa nas pegadas no cemento húmido.

(A súa mensaxe completa é unha lectura de motivo valor. Podes vela no Apéndice C Por que me debería importar a cor do garaxe da bicicleta? bótalle tamén unha ollada a http://bikeshed.com)

Calquera que algunha vez participase de xeito frecuente nun grupo con capacidade de toma de decisións, pode recoñecer perfectamente isto do que fala Kamp. Porén, acostuma a ser imposible persuadir *todo o mundo* para que non pinte o garaxe da bicicleta. O mellor que podes facer é recoñeceres que este fenómeno existe, cando vires que está a suceder, e persuadir os desenvolvedores experimentados -a xente cuxas mensaxes cargan coa maior parte do peso- para que abandonen as súas brochas o máis axiña posible, e cando menos non contribúan ao ruído. As festas para pintar garaxes de bicicletas nunca desaparecen de todo, mais podes conseguir que sexan máis curtas e menos frecuentes mediante a concienciación sobre este fenómeno, e a súa contribución á cultura do proxecto.

Evitar as guerras santas

Unha *guerra santa* é unha disputa, normalmente sobre un tema non menor, que non é solucionable a través dos méritos dos argumentos, senón na que a xente se apaixona suficientemente como para seguir discutindo de todos os xeitos na esperanza de que a súa postura prevaleza. As guerras santas no son o mesmo que as sesións de pintado de garaxes de bicicletas. A xente que pinta garaxes de bicicletas muda frecuentemente de postura facilmente (dado que poden), mais sen se sentiren especialmente turbados por isto, e mesmo a veces poden expresar opinións incompatibles para mostraren que entenden todas as facetas da cuestión. Nunha guerra santa, pola outra banda, entender o resto de facetas é un sinal de debilidade. Nunha guerra santa, todo o mundo sabe que existe unha Única Resposta Correcta; simplemente non están de acordo en cal é.

Unha vez que unha guerra santa comezou, xeralmente non se pode resolver dun xeito satisfactorio para todo o mundo. Non é bo mostrar, no medio dunha guerra santa, que esta está tendo lugar. Isto xa é sabido por todo o mundo. Desafortunadamente, un padrón común nas guerras santas soe ser o desacordo sobre *se* a disputa é resoluble continuando a discusión. Visto desde fóra, está claro que ningunha parte está mudando a forma de pensar da outra parte. Visto desde dentro, a outra parte está sendo obtusa e non está a pensar con claridade, mais poden vir ao rego se son suficientemente intimidados. Iso si, *non* estou a dicir que non haxa unha parte con razón nunha guerra santa. Ás veces hainas, e as guerras santas nas que eu participei sempre estivo do meu lado, por suposto. Mais isto non importa, dado que non hai algoritmos para demostrar convincentemente que un lado ou o outro teñen a razón.

Un camiño común, mais insatisfactorio, que a xente emprega para resolver as guerras santas pasa por dicir "xa esbanxamos máis tempo é enerxía discutindo isto da que realmente nos pode fornecer! Por favor, podemos deixalo?" Hai dous problemas con isto. Primeiro, ese tempo e enerxía xa foron gastados e nunca van poder ser recuperados -a única cuestión agora é canto *máis* esforzo resta? Se

alguén sente que algo máis de discusión vai conseguir pechar a discusión, entón (desde o seu punto de vista) aínda se pode continuar.

O outro problema de pedir que a cuestión sexa abandonada reside en que, a miúdo, esta actitude considérase equivalente a unha vitoria de unha das partes, podendo declarar a vitoria por inacción da outra. E nalgúns casos, o status quo é inaceptable en calquera dos casos: todo o mundo está de acordo en que se debe tomar algunha decisión, e levar a cabo algunha acción. Deixar o asunto pode ser peor para todo o mundo que simplemente renderse. Mais dado que este dilema é aplicable a todas as partes por igual, aínda é posible continuar eternamente coa discusión sobre o que hai que facer.

Como deberías manexar unha guerra santa?

A primeira resposta é tentar configurar as cousas para que non se declaren guerras santas. Isto non é tan inútil como soa:

Podes anticiparte a determinadas guerras santas máis ou menos estándar: tenden a versar sobre linguaxes de programación, licenzas (bótalle unha ollada á sección "*GPL e Compatibilidade de Licenzas"* no *Capítulo 9 Licenzas, Copyrights e Patentes*), responderlle aos desconformes (ver a sección chamada *Capítulo 3 Infraestrutura Técnica*), e uns poucos temas máis. Ademais, cada proxecto acostuma a ter unha ou dúas guerras santas da súa propia colleita, coas que os desenvolvedores de longo percorrido vanse familiarizar con prontitude. As técnicas para parares guerras santas, ou para limitares os seus danos, son máis ou menos as mesmas en calquera lugar. Mesmo se estiveres seguro de que o teu lado é o que ten a razón, tenta atopar *algún* xeito de expresares simpatía e comprensión polas cuestións que o outro lado está a facer. Moitas veces o problema de base nunha guerra santa atópase en que cada parte construíu os seus muros o máis altos que puido, e deixou claro que calquera outra opinión é unha unha parvada. O acto de rendéreste ou mudares de idea tórnase en algo psicoloxicamente insoportable: podería ser un recoñecemento non só de estares equivocado, senón tamén de saberes que o estabas e non telo admitido. O xeito no que podes facer que esta admisión sexa aceptable para o outro bando é expresando algunha inseguridade en ti mesmo, precisamente explicando de que entendes algúns dos seus argumentos, e visualizando que os tes en consideración, se definitivamente non te persuadiron. Fai xestos que fornezan un espazo para que haxa outros xestos recíprocos, e normalmente a situación vai mellorar. Ti non vas estar nin mellor nin peor posicionado para alcanzares os resultados técnicos que buscas, mais cando menos vas poder eliminar danos colaterais (totalmente innecesarios) na moral do proxecto.

Cando unha guerra santa non pode ser evitada, decide o máis axiña que poidas canto che importa, e prepárate para renderte publicamente. Cando fagas isto, podes dicir que non estás apoiándoa porque a guerra santa non o merece, mais non expreses ningún rancor e *non* te aproveites da oportunidade para unha despedida disparando contra os argumentos da outra parte. Darse por vencido é efectivo só cando se fai con elegancia.

As guerras santas sobre linguaxes de programación son un caso especial, porque a miúdo son altamente técnicas, habendo moita xente que se sente cualificada para tomar parte nelas, e contando cun interese moi alto, xa que o resultado pode determinar en gran medida en que linguaxe se vai a desenvolver o proxecto. A mellor solución é escolleres a linguaxe logo, coa participación dos desenvolvedores iniciais influentes, e entón defenderes esta solución nos terreos nos que todos estiverdes cómodos escribindo nesa linguaxe, *non* nos terreos nos que estea linguaxe for mellor que outras linguaxes que poderían ser empregadas no seu lugar. Nunca deixes que a conversa dexenere nunha comparación académica de linguaxes de programación (isto parece ocorrer especialmente cando alguén menciona Perl, por

algunha razón); este é un tema morto no que simplemente debes evitar caer.

Para un maior coñecemento histórico sobre as guerras santas, mira http://catb.org/~esr/jargon/html/H/holy-wars.html, e mailo artigo de Danny Cohen, no que se popularizou o termo, http://www.ietf.org/rfc/ien/ien137.txt

O efecto "minoría ruidosa"

En calquera discusión nunha lista de correo, resúltalle sinxelo a unha minoría dar a impresión de que existe un gran acordo na disensión, mediante inundación da lista con numerosos e longos emails. Isto semella filibusterismo, excepto que a ilusión da difusión da disensión é mesmo máis poderosa, dado que está dividida entre un número arbitrario de mensaxes concretas e moita xente non se vai molestar en trazar quen dixo que, nin cando. Simplemente van ter a impresión instintiva de que o tema é moi controvertido, e van agardar ata que o escándalo diminúa.

O mellor xeito de contraatacar este efecto é dando conta dun xeito ben claro, e fornecendo probas, de que o número real de disensións, comparado cos acordos, é moi pequeno. Perseguindo o incremento da disparidade, poderías querer enquisar privadamente a xente que se ten mostrado silenciosa, mais que sospeitas que estarían de acordo coa maioría. Non digas nada que suxira que os desconformes están tentando inflar deliberadamente a impresión que están forxando. O único que precisas é mostrares os números reais nunha comparación un-a-un, e só con iso a xente xa se vai decatar de que a intuición que tiñan sobre a situación non encaixa coa realidade.

Este consello no só se aplica a temas cunha clara posición a favor ou en contra. Aplícase a calquera discusión onde houber un alboroto, mais non estiver claro que a maioría da xente considera ese tema en discusión como un problema real. Despois de todo, se estiveres de acordo en que o tema non é digno de acción, e podes ver o que fallou en atraer a atención (embora xerase moitos correos), podes facer pública esta observación. Se o efecto "minoría ruidosa" funcionou, a túa mensaxe vai parecer un sopro de aire fresco. A impresión da maioría da xente ata ese momento pode ser algo porquiña: "Uh, seguro que senten como se houbese un gran acordo aquí, porque seguramente hai unha morea de mensaxes, mais non podo ver que haxa ningún progreso claro." Explicar o ciclo de vida da discusión pode facer que pareza máis turbulenta do que realmente é, polo que ti retrospectivamente debes darlle unha nova forma, a través da cal a xente poida recapitular a súa comprensión do resultado.

Xente problemática

Os foros electrónicos non fan máis fácil tratar coa xente problemática ca en persoa. Por xente "problemática" non me refiro a "groseira". As persoas groseiras son molestas, mais non necesariamente problemáticas. Neste libro xa discutimos como tratar con elas: comentar as súas saídas de ton a primeira vez, e a partir de entón, ou ben ignoralos ou tratalos como a calquera outra persoa. Se continúan sendo maleducados, co tempo serán tan impopulares que non terán influencia algunha nos demais membros do proxecto, polo que realmente rematarán sendo un problema con auto-contención.

Os casos realmente problemáticos son os de xente que non é abertamente groseira, mais que manipula ou abusa dos procedementos do proxecto de modo que lle custa a outra xente tempo e esforzo, sen proporcionar beneficio algún ao proxecto. Estas persoas habitualmente buscan fisgoas nos

procedementos do proxecto para ter máis influencia da que normalmente terían. Isto é claramente máis insidioso que simplemente groseiro, porque nin o comportamento nin o dano causado son aparentes para observadores casuais. Un exemplo clásico é o filibusteiro, no cal alguén (sempre nun suposto razoable, claro está) mantén que un tema baixo discusión non se pode dar por pechado, e ofrece máis e máis posibles solucións, ou novos puntos de vista en solucións antigas, mentres que o que realmente persegue e que non se chegue a un consenso ou votación que non lle beneficia. Outro exemplo é cando hai un debate que non chegará a un consenso, mais o grupo tenta polo menos aclarar os puntos de desacordo e crear un sumario para que calquera poda consultar desde entón. O obstrucionista, que sabe que o sumario levará a un resultado que non desexa, tentará habitualmente de atrasar a súa creación, complicando continuamente a cuestión sobre que debería conter, ben obxectando de maneira razoable ou introducindo elementos novos inesperados.

Tratando con xente problemática

Para contrarrestar devandito comportamento, axuda coñece-la mentalidade de aqueles que o presentan. A xente normalmente non o fai de maneira consciente. Ninguén se esperta unha mañá dicindo "Hoxe vou manipular de maneira cínica os procedementos formais para converterme nun obstrucionista irritante". No seu lugar, ditas accións son habitualmente precedidas por un sentimento semi-paranoide de estar sendo vetado de interaccións no grupo e toma de decisións. A persoa sente que non se lle está tomando en serio, ou (en casos máis serios) de que existe unha conspiración na súa contra -que outros membros do proxecto decidiron formar un club exclusivo, do cal non é membro. Entón isto xustifica, na súa mente, o feito de tomar as regras ao pé da letra e involucrarse nunha manipulación formal dos procedementos do proxecto, para poder *facer* que todos o tomen en serio. En casos extremos, a persoa pode chegar a pensar que está loitando unha batalla para salva-lo proxecto de si mesmo.

Na natureza de ditos ataques ocorre que non todos se decatan del ao mesmo tempo, e algunha xente pode que non o vexa se non se lle presenta con probas moi claras. Isto significa que neutralizalo pode supor bastante traballo. Non é suficiente convencerte do que está ocorrendo; hai que reunir suficientes evidencias para convencer os demais, e entón distribuílas de maneira intelixente.

Sabendo que combatelos é tan complicado, moitas veces é mellor toleralos durante un tempo. Pensa nela coma nunha enfermidade parasitaria, mais leve; se non debilitar demasiado, o proxecto pode permitirse permanecer infectado, e a medicina podería ter efectos secundarios prexudiciais. Por outro lado, se se tornar en algo demasiado daniño, entón hai que pasar á acción. Comeza anotando os padróns que observares. Asegúrate de incluíres referencias aos arquivos públicos -esta é unha das razóns polas que o proxecto mantén rexistros, polo que fas ben en empregalos. Unha vez que tes un bo caso construído, comeza a ter conversacións privadas con outros participantes do proxecto. Non lles digas o que observaches; en lugar diso, pregunta primeiro que observaron eles. Esta pode ser a última oportunidade de obter información sen filtrar sobre como ven outros o comportamento do membro problemático; unha vez que comezas a falar de iso abertamente, a opinión estará polarizada e ninguén poderá recordar con exactitude o que pensaba antes sobre o tema.

Se as conversas privadas indicaren que polo menos algúns dos demais membros observaron tamén o problema, entón é hora de faceres algo. Entón é cando hai que ter *verdadeiro* coidado, porque é moi sinxelo para este tipo de persoas facer que pareza que os estás tratando de maneira inxusta. Fagas o que fagas, nunca os acuses de abusar maliciosamente dos procedementos do proxecto, ou de ser paranoicos, ou, en xeral, de calquera cousa que sospeites que é verdade. A túa estratexia debería ser semellar

razoable e preocupado polo ben do proxecto, co obxectivo de mudar o comportamento da persoa, ou facer que marchen definitivamente. Segundo os demais desenvolvedores, e a túa relación con eles, pode ser vantaxoso lograres aliados de maneira privada ao comezo. Ou talvez non; isto podería crear malestar detrás do escenario se a xente pensa que os estás metendo nunha campaña de rexoubas impropia.

Recorda que aínda que a outra persoa sexa a que ten un comportamento destrutivo es *ti* o que parecerá destrutivo se fixeres de maneira pública unha acusación que non podes soster. Asegúrate de ter multitude de exemplos para demostrar o que dis, e fala do modo máis xentil posible, sen deixares de ser directo. Pode que non convenzas a persoa en cuestión, mais iso non supón un gran problema mentres convenzas os demais.

Caso de estudo

Só recordo un caso, en máis de 10 anos de traballo con software libre, no que as cousas estaban tan mal que tivemos que pedirlle a alguén que parara totalmente de postear. Como acontece con frecuencia, non era maleducado, e a verdade é que simplemente quería axudar. O que pasaba era que non sabía cando postear e cando non. As nosas listas estaban abertas publicamente, e posteaba con tanta frecuencia, preguntando sobre temas moi diferentes, que comezaba a ser unha amoladura para a comunidade. Comentámoslle que investigase un pouco máis esas respostas antes de postear, mais non serviu de nada.

A estratexia que finalmente funcionou é un exemplo perfecto sobre como elaborar un caso sobre datos cuantitativos e neutrais. Un dos nosos desenvolvedores rebuscou nos arquivos e enviou o seguinte mensaxe privado a uns poucos desenvolvedores. A persoa problemática (o terceiro nome na lista inferior, indicado aquí como "X. Aleatorio") levaba pouco tempo no proxecto, e non tiña contribuído nin con código nin con documentación. Porén, era o terceiro membro máis activo nas listas de correo:

```
De: "Brian W. Fitzpatrick" <fitz@collab.net>
A: [... lista de receptores omitida por anonimato ...]
Asunto: O sumidoiro de enerxía de subversion
Data: Mércores, 12 Nov 2003 23:37:47 -0600

Nos últimos 25 días, os 6 membros máis activos na lista svn [dev|users]
foron:

    294  kfogel@collab.net
    236  "C. Michael Pilato" <cmpilato@collab.net>
    220  "X. Aleatorio" <xaleatorio@posteador-problematico.com>
    176  Branko &Ccaron;ibej <brane@xbc.nu>
    130  Philip Martin <philip@codematters.co.uk>
    126  Ben Collins-Sussman <sussman@collab.net>

Eu diría que 5 desas persoas están contribuíndo a que saía Subversion
1.0 nun futuro próximo.

Tamén diría que unha desas persoas está constantemente roubando tempo
e enerxía dos outros 5, sen mencionar toda a lista, e polo tanto
(aínda que de maneira non intencionada) freando o desenvolvemento de
Subversion.
Non fixen unha análise dos fíos, mais un vgrep da miña bandexa de correo de
```

```
Subversion mostra
que tódolos correos desta persoa foron respondidos polo menos una vez por
polo menos dúas das cinco persoas citadas enriba.

Penso que algún tipo de intervención radical é necesaria neste caso, mesmo
se escorrentarmos a esa persoa fora do proxecto. A amabilidade e as
suxestións xa mostraron que non teñen efecto.

dev@subversion é unha lista de correo para facilitar o desenvolvemento dun
sistema de control de versións, non para facer sesións de terapia de grupo.

-Fitz, tentando ler tres días de correo de svn amontoado
```

Pode que non o pareza ao principio, mais o comportamento de X. Aleatorio era un caso clásico de abuso dos procedementos dun proxecto. Non facía cousas evidentes como tentar obstruír unha votación, mais estaba sacando proveito das políticas das listas de correo de ser moderadas polos seus propios membros. Deixamos a xuízo de cada un o momento de postear e sobre que temas. Ademais, non tiñamos procedementos para tratar con alguén que non tivera ou non fixera uso dese xuízo. Non había regras que alguén puidera sinalar que dicir que outro as estaba violando, aínda que todos sabiamos que os seus mensaxes frecuentes comezaban a ser un problema serio.

Retrospectivamente, a estratexia de Fitz foi autoritaria. Reuniu probas claras cuantitativas, mais decidiu distribuílas de maneira discreta, mandándoas primeiro a aquelas persoas cuxo apoio sería clave nunha medida drástica. Aceptaron tomar ese tipo de decisión se for necesario, e ao final chamamos a X. Aleatorio ao teléfono, describímoslle o problema directamente e pedímoslle que simplemente cesara de postear. Realmente nunca comprendeu os motivos; se fose capaz de comprendelos, probablemente tivera feito uso dun xuízo apropiado en primeiro lugar. mais aceptou parar de postear, e as listas de correo volveron a poderse usar de novo. Parte do éxito desta estratexia foi, quizais, a ameaza implícita de restrinxir as súas mensaxes mediante o software de moderación normalmente empregado para previr o spam (ver a sección chamada “*Prevención de Spam*” no *Capítulo 3 Infraestrutura Técnica*). Mais o motivo de que tivéramos dita opción de reserva foi que Fitz conseguiu xa ao principio o apoio necesario de xente clave no proxecto.

Xestionando o crecemento

O prezo do éxito é forte no mundo do software libre. A medida que o teu software se torna máis popular, o número de persoas que xorden buscando información sobre o mesmo medra drasticamente, mentres que o número de persoas capacitadas para fornecer esa información aumenta moito máis lentamente. Máis aínda, mesmo se a relación estiver equilibrada, seguiría habendo un problema fundamental de escalabilidade no xeito no que os proxectos de software libre xestionan as comunicacións. Considera as listas de correo, por exemplo. A maioría dos proxectos teñen unha lista de correo para cuestións xerais dos usuarios -ás veces nomeada "users","discuss","help" ou algo así. Calquera que for o seu nome, o propósito da lista sempre é o mesmo: fornecer un lugar onde a xente poida ver resoltas as súas preguntas, mentres outros miran e (presumiblemente) absorben coñecementos mediante a observación destes trocos de información.

Estas listas de correo funcionan moi ben ata un poucos centos de usuarios e/ou un par de centos de mensaxes ao día. Mais para alén destes límites, o sistema comeza a fracturarse, dado que cada subscritor ve cada mensaxe; se o número de mensaxes á lista comeza a exceder o límite que cada lector

individual pode procesar nun día, a lista convértese nun peso morto para os seus membros. Imaxina, por exemplo, que Microsoft tivese unha lista deste xeito para Windows XP. Windows XP ten centos de millóns de usuarios; mesmo se un por cento deles tiver preguntas dun período dado de 24 horas, entón esta hipotética lista tería centos de miles de mensaxes cada día! Unha lista deste estilo nunca podería existir, por suposto, dado que ninguén estaría subscrito á mesma. Este problema non se limita ás listas de correo; a mesma lóxica exacta é válida para os canais IRC, os foros de discusión en liña, e por derivación para calquera sistema no cal un grupo escoite preguntas provenientes de persoas individuais. As implicacións son desafortunadas: o modelo de software libre habitual baseado nun soporte masivamente paralelizado simplemente non escala para os niveis precisos para a dominación mundial (ou cando menos da Terra Media, ou da Illa de Perdidos).

Obviamente non vai haber ningunha explosión cando os foros atinxiren o punto de fractura. Simplemente vai haber un silencioso efecto de retroalimentación negativa: xente que abandona as listas de correo, ou deixa os canais IRC, ou co ritmo que for deixan de se preocupar de faceren preguntas, dado que verán que non van ser escoitados no medio de todo ese barullo. A medida que máis e máis xente tome esta altamente racional decisión, vai parecer que a actividade do foro retorna a un nivel manexable. Mais será manexable precisamente porque a xente racional (ou cando menos experimentada) comezou a buscar información en calquera outro lugar -mentres que a xente sen experiencia segue aí e continúa enviando mensaxes. Noutras palabras, un efecto colateral de continuar empregando modelos de comunicacións non escalables cando o proxecto medra é que a calidade media tanto das preguntas como das respostas tende a reducirse, o que provoca que pareza que os novos usuarios son máis parvos do que o eran antes, cando de feito probablemente non o sexan. Trátase simplemente de que a relación beneficio/custo de empregar estes foros tan superpoboados diminúe, polo que naturalmente os usuarios experimentados comezarán a buscar respostas en primeira instancia noutros sitios. Axustar os mecanismos de comunicación para asimilar o crecemento do proxecto envolve dúas estratexias relacionadas:

1. Recoñecer cando partes concretas dun foro *non* están a sufrir dun crecemento excesivo, embora o foro en conxunto si que o estea a sufrir; e separar estas partes en foros novos e especializados (por exemplo, non deixes que paguen xustos por pecadores).

2. Asegurar que están accesibles varias fontes automatizadas de información, e que as mesmas están organizadas, actualizadas e facilmente accesibles.

A estratexia (1) non acostuma a ser demasiado dura. Moitos proxectos arrincan cun foro principal: unha lista de correo xeral de discusión, na cal tanto funcionalidades, como ideas, cuestións de desenvolvemento, problemas de implementación, etc poden ser todos eles discutidos. Todo o mundo que participa no proxecto está na lista. Despois dun tempo, con frecuencia tórnase evidente que a lista precisa evoluír en varias sublistas, relacionadas cada unha delas cun tópico específico. Por exemplo, algúns fíos teñen a ver claramente co desenvolvemento e co deseño; outros son de preguntas de usuarios do estilo "Como fago X?"; ás veces tamén existe unha terceira familia típica de fíos centrada no procesamento de notificacións de erros e peticións de melloras; e así. Un individuo concreto, como non pode ser doutro xeito, pode participar en fíos de moi diferente natureza, mais a importancia atópase en que con frecuencia non hai moito solapamento entre os distintos tipos. Debido ao anterior, pode facerse unha división en diferentes listas de correo sen iniciar perniciosos ataques de orcos, dado que os fíos en moi raras ocasións transgriden as fronteiras do tema no que se enmarcan.

Realmente, a execución desta división é un proceso de dous pasos. Por unha banda creas a nova lista de

correo (ou canal de IRC, ou o que for), e por outra invistes o tempo que for necesario para dun xeito educado ires instando a xente a *empregar* os novos foros dun xeito apropiado. O segundo paso pode levar semanas, mais normalmente a xente acaba captando a idea. Simplemente tes que facer ver, cando unha persoa envíe unha mensaxe a un destino equivocado, cal é o destino correcto, e facelo dun xeito visible e animando a xente a que axude neste proceso didáctico. Tamén pode ser útil contar cunha páxina web na que se forneza unha guía para todas as listas dispoñibles; deste xeito as túas respostas poden, simplemente, referenciar a esa páxina web e, como regalo, o receptor tamén vai aprender que debe pesquisar nas guías antes de preguntar.

A estratexia (2) é un proceso en curso, permanecendo en proceso durante toda a vida do proxecto e envolvendo moitos participantes. Por suposto, trátase en parte da recorrente cuestión de manter documentación actualizada (bótalle unha ollada a sección "*Documentación*" no *Capítulo 2 Primeiros Pasos*), mais tamén se trata se asegurarte de dirixir a xente cara onde esta se atopa, e en última instancia é moito máis que todo o anterior, tal como se vai discutir e amosar nas sección que seguen, nas cales se vai ver esta estratexia en detalle.

Uso intensivo dos arquivos

Tipicamente, todas as comunicacións nun proxecto de software libre (excepto, ás veces, as conversas por IRC) son arquivadas. Os arquivos son públicos e consultables, e teñen estabilidade referencial: isto é, unha vez que unha peza de información é gravada nun enderezo particular, permanece nese enderezo para sempre.

Emprega estes arquivos tanto como puideres, e dun xeito tan manifesto como che for posible. Embora saibas a resposta a unha cuestión, se pensas que existe unha referencia nos arquivos que conteña a resposta, emprega o tempo preciso para buscárela e emprégaa. Cada vez que fas isto dun xeito publicamente visible, algunha xente vai aprender para que valen estes arquivos e vaise decatar de que están aí á súa disposición, e que buscar as cousas neles pódelles dar respostas. Así, mesmo referenciares aos arquivos, en lugar de reescribires unha resposta dada con anterioridade, reforza a norma social de evitar a duplicación de información. Para que ter a mesma resposta en dous lugares diferentes? Cando o número de lugares nos que se pode atopar a resposta é mínimo, a xente que a atopase con anterioridade é máis propensa a lembrar que pesquisar para atopala novamente. As referencias ben situadas tamén contribúen á calidade dos resultados das procuras, dado que reforzan o ranking nas ferramentas de procura de Internet do recurso apuntado.

Porén, hai veces nas que duplicares a información si que ten sentido. Por exemplo, imaxina que existe unha resposta nos arquivos, non fornecida por ti, dicindo:

```
Parece que os índices do teu servidor Dharma están bloqueados.
Para desbloquealos, executa os seguintes comandos:

1. Apaga o servidor Dharma
2. Executa a aplicación 'deslocke' incluída no servidor Dharma
3. Acende o servidor
```

Entón, meses despois, ves outra mensaxe indicando que os índices de alguén sufriron un bloqueo. Fas unha procura nos arquivos e atopas a resposta anterior, mais decátaste de que na mesma non están

incluídos algúns pasos (quizais por erro, quizais porque o software mudou desde que a resposta primixenia pasou a formar parte dos arquivos). O xeito clásico de manexares esta situación é mediante o envío dunha nova mensaxe que inclúa un conxunto de instrucións máis completo, e colgarlle a etiqueta de "obsoleta" á resposta orixinal mediante unha mención ao estilo de:

```
Parece que os índices do teu servidor Dharma están bloqueados. Xa
vimos ese problema o xullo pasado, e D. Hume enviou unha solución,
que se pode ver en http://ghandalf-loves-lost.org/servidoresDharma
A continuación, expoñemos unha descrición máis completa sobre
como desbloqueares os teus índices, baseada nas instrucións de D. Hume,
mais ampliándoas un chisco:

1. Apaga o servidor Dharma
2. Rexístrate como usuario co que o servidor Dharma se executa habitualmente
3. Como ese usuario, executa a aplicación 'deslocke' sobre os índices
4. Arrinca manualmente o servidor Dharma para testares se os índices
funcionan
5. Reinicia o servidor
```

(Nun mundo ideal, sería posible engadirlle unha nota á antiga mensaxe dicindo que existe información actualizada e apuntando á nova mensaxe. Porén, non coñezo ningún software de arquivamento que ofreza unha funcionalidade ao estilo de "obsoletizado por", pode que debido a que sería moi dificultoso implementar isto sen violar a integridade dos arquivos. Esta é outra das razóns polas que crear páxinas web adicadas con respostas para as preguntas comúns é unha excelente idea).

Probablemente as consultas que se fagan con máis frecuencia sobre os arquivos sexan aquelas derivadas de cuestións técnicas, mais a súa importancia para o proxecto vai moito máis alá. Se as liñas e guías formais dun proxecto son o seu estatuto, os arquivos son a súa lei de usos e costumes: un rexistro de todas as decisións tomadas e de como se chegou ás mesmas. En calquera discusión recorrente, é case obrigatorio comezar cunha procura nos arquivos. Isto permite comezar a discusión cun resumo do estado actual das cousas, anticipar obxeccións, preparar refutacións, e posiblemente descubrir ángulos que doutro xeito non terías. Á súa vez, é lóxico que os outros participantes na discusión dean por suposto que fixeches esta procura. Mesmo se as discusións previas non chegaron a ningures, debe incluír referencias ás mesmas cando volvas a sacar o tema, para que a xente poida ver por si mesma a) que non se chegou a ningures, e b) que fixeches os teus deberes, e que probablemente se estea dicindo agora o que non se dixo anteriormente.

Trata todos os recursos como arquivos

Todos os consellos anteriores son válidos en máis circunstancias que o caso concreto das listas de correo. Ter pezas concretas de información en enderezos estables e fáciles de atopar é con frecuencia un principio básico de organización para a información de calquera proxecto. Imos tomar as FAQ (Preguntas Frecuentes) dun proxecto como caso de estudo:

Como emprega a xente unha FAQ?

1. Pesquisan palabras e frases específicas.
2. Queren poder navegala, gozando da información sen necesariamente buscar respostas a cuestións específicas.

3. Esperan que motores de procura como Google coñezan o contido da FAQ, de tal xeito que as procuras fornezan como resultado entradas na FAQ.
4. Queren poder referirse directamente a outra xente en temas específicos na FAQ.
5. Queren poder engadir novo material na FAQ, mais ten en conta que isto ocorre moito menos a miúdo que a procura de respostas -as FAQs son de lonxe moito máis lidas que escritas.

O punto 1 implica que as FAQ deberían estar dispoñibles nalgún tipo de formato de texto. Os puntos 2 e 3 implican que as FAQ deberían estar dispoñibles como unha páxina HTML, coa indicación adicional por parte do punto 2 de que o HTML debería estar deseñado para aumentar a lexibilidade (p.ex. vas querer ter algún control sobre o seu modo de visualización), e debería contar cunha táboa de contidos. O punto 4 indica que cada entrada individual da FAQ debería ter asignada unha etiqueta HTML *named anchor*, indicando que se lle permite a xente chegar a calquera localización concreta na páxina. O punto 5 significa que os ficheiros fonte para a FAQ deberían estar dispoñibles dunha maneira axeitada (ver a sección "*Versionar todo*" no *Capítulo 3 Infraestrutura Técnica*), nun formato que sexa sinxelo de editar.

Named Anchors (áncoras nomeadas) e atributos ID

Hai dúas maneiras para lograr que un navegador salte a un sitio específico dentro dunha páxina web: áncoras con nome e atributos id.

Unha *áncora con nome* é simplemente un elemento HTML de áncora (<a>...<a>), mais cun atributo "name":

...

Versións máis recentes de HTML soportan un *atributo id* xenérico, o cal pode achegarse conxuntamente con calquera elemento HTML, non só con <a>. Por exemplo:

<p id="hugo-reyes-label">...<p>

Ambos, tanto áncoras con nome como atributos id, empréganse do mesmo xeito. Un engade unha marca hash e etiquétaa a unha URL, para provocar que o navegador salte directo ao devandito punto da páxina:

http://ghandalf-loves-lost.org/faq.html#david-faraday-label

Virtualmente todos os navegadores Web teñen soporte para áncoras con nome; a maior parte dos navegadores modernos teñen soporte para atributos id. Para ir sobre seguro, recomendaría empregar tanto áncoras con nome en solitario, como áncoras con nome <emphasis>e</emphasis> atributos id conxuntamente (coa mesma etiqueta para ambos nun par concreto, por suposto). As áncoras con nome non poden ser autopechantes -embora non exista texto dentro do elemento, porén, debes escribilo na forma dupla:

...aínda que normalmente haberá algún texto, como o título dunha sección.

En calquera lugar no que empregues unha áncora con nome, ou un atributo id, ou ambos,

recorda que a etiqueta non vai ser visible por ninguén que navegue ata esa localización sen empregar a etiqueta. Mais unha persoa podería querer saber a etiqueta que identifica unha localización concreta, para poder enviar un correo coa URL da resposta na FAQ a un amigo, por exemplo. Para axudalo a facer isto, engade un *atributo título* ao mesmo elemento(s) ao que lle engadiches o "nome" e/ou o atributo "id", por exemplo:

```
<a name="kate-austen-label" title="#kate-austen-label">...</a>
```

Deste xeito, cando o rato se sitúe sobre o texto dentro do elemento co atributo de título, a maioría dos navegadores van mostrar unha pequena caixiña co título. Acostumo a incluír a sinatura hash, para lembrarlle ao usuario que isto é o que debe poñer ao final da URL para saltar directamente a esta localización a próxima vez.

Formatar a FAQ deste xeito só é un exemplo de como facer que un recurso sexa presentable. As mesmas propiedades -localización directa, dispoñibilidade para a maioría dos motores de procura en Internet, navegabilidade, estabilidade referencial, e (onde corresponder) editabilidade- correspóndense con outras páxinas web, coa árbore co código fonte, co bug tracker, etc. Debido a que a maioría do software de arquivamento de listas de correo xa desde hai moito tempo ten recoñecida a importancia destas propiedades, é polo que a listas de correo tenden a contar nativamente con estas funcionalidades, mentres que outros formatos poden requirir algún esforzo extra por parte do mantedor (o *Capítulo 8 Xestionando Voluntarios* analiza como repartir esta carga de mantemento entre varios voluntarios).

Tradición na codificación

A medida que un proxecto vai adquirindo historia e complexidade, a cantidade de datos que cada participante incorporado debe absorber vese incrementada. Aqueles que levan no proxecto desde hai moito tempo foron capaces de aprender, e inventar, as convencións do proxecto. A miúdo non son conscientes do enorme corpo de tradición que foron acumulando, e sorpréndense de cantos pequenos erros van cometendo os novatos. Por suposto, a cuestión non está en que as novas incorporación sexan de menor calidade que as anteriores; a cuestión está en que se teñen que enfrontar a unha maior cultura herdada, que aquela á que se enfrontaron os novatos de tempo atrás.

As tradicións que un proxecto vai acumulando atinxen sobre todo a como comunicar e preservar a información, estándares de codificación, e outras cuestións de índole técnica. Xa repasamos ambos tipos de estándares na sección "*Documentación de desenvolvemento*" no *Capítulo 2 Primeiros Pasos* e na sección "*Escribíndoo todo*" no *Capítulo 4 Infraestrutura política e social* respectivamente, mostrando exemplos. En lugar diso, esta sección trata sobre como manter estas guías e pautas actualizadas a medida que o proxecto evolúe, especialmente as pautas e guías sobre como xestionar as comunicacións, dado que estas son as que máis mudanzas sofren a medida que o proxecto medra en tamaño e complexidade.

Primeiro, busca padróns sobre como a xente se confunde. Se ves as mesmas situacións producíndose unha e outra vez, especialmente cos novos participantes, entón preséntase unha oportunidade en forma de guía ou pauta que necesita ser documentada mais aínda non o está. Segundo, non te canses de dicir as mesmas cousas unha e outra vez, e que non *adoita* como se estiveses cansado de repetilo. Ti e outros veteranos do proxecto teredes que repetírvolo entre vos a miúdo; este é un efecto colateral inevitable derivado da chegada de novos participantes.

Cada páxina web, cada mensaxe na lista de correo e cada canal IRC debería ser considerado como un espazo publicitario -non para anuncios comerciais, mais si para anuncios sobre os recursos do teu propio proxecto. O que poñas neses espazos debe depender da procedencia e cultura dos que os vaian ler. Un canal IRC para preguntas dos usuarios, por exemplo, parece o sitio ideal para obter xente que nunca antes tivera contacto co proxecto -habitualmente alguén que simplemente instalou o software e ten unha pregunta que precisa ser respondida o máis axiña posible (despois de todo, se puidese esperar, tería enviado a pregunta á lista de correo, o que probablemente lle custase menos tempo total, aínda que tardaría máis en recibir resposta). A xente normalmente non fai un investimento permanente no canal IRC; aparecen, lanzan a súa pregunta e vanse.

Deste xeito, o tópico e mailo tema do canal deberían dirixirse a xente que está a buscar por respostas técnicas sobre o software *o máis axiña posible*, en lugar de dirixirse a xente que podería querer participar no proxecto a longo prazo e para os cales unhas guías de interacción serían máis apropiadas. Aquí é onde un canal IRC realmente ocupado xestiona a cuestión (compáraa co exemplo anterior na sección "*IRC Sistemas de conversa en tempo real*" no *Capítulo 3 Infraestrutura Técnica*):

```
Estás falando en #linuxhelp

O tema para #linuxhelp é: Por favor, LE
http://www.catb.org/~esr/faqs/smart-questions.html &amp;amp;amp;
http://www.tldp.org/docs.html#howto ANTES DE faceres preguntas | As normas
do canal atópanse en http://www.nerdfest.org/lh_rules.html | Por favor,
consulta
http://kerneltrap.org/node/view/799 antes de preguntares sobre
actualizacións á versión
2.6.x do kernel | memory read possible: http://tinyurl.com/4s6mc -&gt;
update to 2.6.8.1 or 2.4.27 | hash algo disaster: http://tinyurl.com/6w8rf
| reiser4 out
```

Coas listas de correo, o "espazo publicitario" é un pequeno pé engadido a cada mensaxe. A maioría dos proxectos poñen as instrucións de subscrición e de-subscrición nese lugar, e ás veces unha indicación á páxina web do proxecto ou á páxina de FAQs. Poderías pensar que alguén subscrito á lista de correo sabería onde atopar estar cousas, e probablemente sexa así -mais moita máis xente que os propios subscritores conforma a audiencia destas mensaxes na lista de correo. Unha mensaxe arquivada pode enlazarse desde múltiples lugares; de feito, algunhas mensaxes poden tornarse tan famosas que poden chegar a ter máis lectores fóra da lista que dentro da mesma.

O formatado pode supor unha gran diferenza. Por exemplo, no proxecto Subversion, tivemos un éxito limitado empregando a técnica de filtraxe de erros descrita na sección "*Prefiltrado do sistema de notificación de erros*" no *Capítulo 3 Infraestrutura Técnica*. Moitos dos falsos informes de erros seguían sendo preenchidos por xente sen experiencia, e cada vez que isto pasaba, o usuario tiña que ser aprendido exactamente do mesmo xeito que as 500 persoas anteriores. Un día, despois de que a un dos nosos desenvolvedores finalmente se lle esgotara a paciencia e empezase a criticar e a meterse cun pobre usuario por non ler detidamente a guía de uso do notificador de erros, outro desenvolvedor decidiu que este padrón se reproducira demasiadas veces. Suxeriu que reformatáramos a páxina de portada do notificador de erros de xeito que o máis importante da mesma, a norma de discutir previamente os erros na lista de correo ou nos canais IRC antes de envialos polo notificador de erros, estivese formatado en letras moi grandes, en vermello resaltado sobre un fondo amarelo e resaltado claramente sobre todos os demais elementos da páxina. Así se fixo (podes ver os resultados en http://subversion.tigris.org/project_issues.html), e o resultado foi un descenso notable no número de

falsos informes notificados. Por suposto, aínda os temos hoxe en día, e sempre os teremos, mais o número descendeu notablemente, embora aumentase o número de usuarios. O resultado non se restrinxe unicamente a que a base de datos de erros teña menos lixo, senón que aqueles desenvolvedores que responden aos tickets de informes de erros fano con bo humor, e aumentou a probabilidade de manteren o bo humor, dado que teñen que lidar con poucos erros falsos. Isto mellora tanto a imaxe do proxecto como a saúde mental dos seus voluntarios.

A lección que nós tiramos foi que simplemente con escribir as normas non fas abondo. Tamén tivemos que poñelas onde fosen vistas por aqueles que máis as necesitan, e formatalas de tal xeito que o seus status de material introdutorio fose inmediatamente claro para a xente que non estaba familiarizada co proxecto.

As páxinas web estáticas non son o único lugar para publicitar os costumes do proxecto. Unha certa cantidade de políticas interactivas (no sentido de recordatorio amable, non no sentido de ataduras e sensación de encarceramento) tamén son precisas. Toda revisión entre iguais, mesmo as revisións de commits descritas na sección "*Practicando a revisión de código fonte*" no *Capítulo 2 Primeiros Pasos*, deberían incluír revisións sobre a conformidade ou non conformidade da xente coas normas do proxecto, especialmente en relación coas convencións relativas ás comunicacións.

Outro exemplo extraido do proxecto Subversion: establecemos nunha convención que "r12908" significaría "revisión 12908 no repositorio de control de versións". O prefixo en minúsculas "r" é sinxelo de escribir, e dado que ten a metade de altura que os díxitos, convérteo nun bloque de texto facilmente recoñecible ao combinalo con díxitos. Por suposto, establecelo deste xeito na convención non quixo dicir que todo o mundo comezara a empregalo con coherencia do xeito correcto. Así, cando un correo de commit ven cun log coma este:

```
------------------------------------------------------------------------
r12908 | dhume | 2010-05-24 08:05:06 -0600 (Mon, 24 May 2010) | 4 lines

Patch from James Ford (Sawyer) <sawyer@ghandalf-loves-lost.org>

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

...parte da revisión deste commit consiste en dicir "A partir de agora, por favor emprega 'r12828', en lugar de 'revisión 12828' cando te referires a mudanzas feitas no pasado." Isto non é pedantería; é importante tanto para o parseo automático como para os lectores humanos.

Seguindo o principio xeral de que debe haber métodos canónicos de referencia para entidades comúns, e que estes métodos de referencia deben empregarse con coherencia en todos os sitios, o proxecto en efecto exporta certos estándares. Estes estándares permiten que a xente escriba ferramentas que presenten as comunicacións do proxecto dun xeito máis útil -por exemplo, unha revisión formatada como "r12828" podería transformarse nunha ligazón viva dentro do sistema de navegación do repositorio. Isto sería moi difícil de facer se a revisión fose escrita como "revisión 12828", tanto porque esta segunda forma podería dividirse a través dun salto de liña, como porque é menos distintiva (a palabra "revisión", a miúdo aparecerá soa, e os grupos de números tamén aparecen sós, mentres que a combinación "r12828" é unívoca, tendo como significado un número de revisión). Consideracións similares valen tamén para temas con números, FAQs (truco: emprega unha URL cunha áncora con nome, como se describe na sección "*Áncoras nomeadas e atributos ID*"), etc.

Mesmo para entidades nas que non existe un modelo obvio de forma canónica, a xente debería estar igualmente disposta a fornecer pezas clave de información dun xeito coherente. Por exemplo, cando te referires a unha mensaxe nunha lista de correo, non fornezas unicamente a identidade da persoa que enviou o correo e mailo asunto do mesmo; fornece tamén a URL do arquivo *e* o cabezallo co *message-ID*. Isto último permite que a xente que teña a súa propia copia das listas de correo (hai xente que ás veces mantén copias offline, por exemplo para empregalas nun portátil mentres viaxa) poida identificar univocamente a mensaxe correcta embora non teñan acceso aos arquivos. O remitente e mailo asunto non serían suficientes, dado que a mesma persoa pode enviar varias mensaxes no mesmo fío, mesmo no mesmo día.

Canto máis medra un proxecto, máis importante se torna este tipo de coherencia. Coherencia quere dicir que todas as persoas observan os mesmos padróns e que estes son seguidos, o cal obriga a cada persoa individual a seguilos. Isto, ademais, reduce o número de cuestións que precisan ser preguntadas. A carga de ter un millón de lectores non é a mesma que a de ter un; os problemas de escalabilidade comezan a xurdir unicamente cando un determinado porcentaxe deses lectores fai preguntas. A medida que un proxecto medra, por tanto, debe reducir estas porcentaxes mediante o incremento da densidade e da accesibilidade da información, de tal xeito que a xente sexa máis propensa a atopar o que precisa sen ter a necesidade de preguntalo.

Prohibidas as conversas no notificador de erros (bug tracker)

En calquera proxecto que faga un uso activo do notificador de erros, sempre existe o risco de que o mesmo acabe tornándose nun foro de discusión en si mesmo, embora as listas de correo sexan un lugar mellor para facelo. Usualmente comeza dun xeito suficientemente inocente: alguén anota un informe de erro con, di el, unha proposta de solución, ou un parche parcial. Outra persoa o ve, observa que hai problemas coa solución, e inclúe outra anotación mostrando os problemas. A primeira persoa responde unha vez máis,... e así sucesivamente.

O problema con esta operativa reside, primeiramente, en que o notificador de erros é un lugar demasiado pesado e torpe para manter unha discusión, e en segundo lugar en que outra xente probablemente non estea prestando atención -despois de todo, esperan que as discusións de desenvolvemento se leven a cabo na lista de correo de desenvolvemento, polo que ese é o lugar ao que están mirando. Probablemente non estean subscritos á lista de notificacións, e mesmo se o estiveren, pode que non a sigan con demasiada atención.

Mais, exactamente en que punto do proceso se fixo algo incorrecto? Foi no momento no que a persoa orixinal engadiu a súa solución ao erro -tería que telo enviado á lista en lugar de engadilo? Ou foi cando a segunda persoa respondeu á incidencia, en lugar de facelo na lista de correo?

Non hai unha resposta correcta, mais si que hai un principio xeral: se simplemente estiveres engadindo datos a unha incidencia, entón faino no notificador de erros, mais se estiveres comezando unha *conversa*, entón faino na lista de correo. Non sempre poderás identificar cada caso, simplemente emprega o teu xuízo o mellor que poidas. Por exemplo, cando forneceres un parche cunha solución potencialmente controvertida, debes anticipar que posiblemente haxa xente que teña preguntas sobre a mesma. Así que aínda que normalmente fornezas o parche conxuntamente coa notificación, (asumindo que non queres, ou non podes, facer a mudanza directamente), neste caso deberías escoller enviar a

mensaxe á lista de correo en lugar de empregares o notificador de erros. De tódolos xeitos, pode haber casos nos que sexa difícil. En todo caso, eventualmente acontecerá un momento onde unha parte ou a outra indiquen que simplemente están engadindo datos á conversa actual -no exemplo que iniciaba esta sección, este sería o segundo actor, o cal responde, quen ao decatarse de que había problemas co parche, puido predicir que ía iniciarse unha conversa sobre el, e de aí que debera facerse no canal apropiado.

Empregando unha analoxía matemática, se a información parece como que vai ser rapidamente converxente, entón pona directamente no notificador de erros; se parece como que vai haber diverxencias, entón unha lista de correo ou un canal IRC serían un lugar mellor.

Isto non quere dicir que nunca deba haber ningún intercambio no notificador de erros. Preguntar por máis detalles sobre a reprodución do erro por parte do informador orixinal é con frecuencia un proceso moi produtivo, por exemplo. A resposta do informador é pouco probable que lance novas incidencias; simplemente aumentará a información previamente fornecida. Non hai necesidade de crear distracción na lista de correo con este proceso. En resumo, todo isto significa que debes ter coidado cos comentarios no notificador de erros. Do mesmo xeito, se estiveres completamente seguro de que o erro foi mal notificado (por exemplo, non é un erro), entón abonda con que o notifiques na propia incidencia. Mesmo notificar un problema menor cunha solución proposta é axeitado, asumindo que o problema non vai impedir toda a solución.

Porén, se estiveres lanzando incidencias filosóficas sobre o alcance do erro ou sobre o comportamento perfecto para o software, podes estar seguro de que outros desenvolvedores van querer participar nunha discusión. Probablemente a discusión diverxa durante un tempo, ata converxer, así que faino na lista de correo.

Enlaza sempre ao fío da lista de correo adicado á incidencia, cando escollas mandar a notificación á lista de correo. Segue sendo moi importante, para alguén que siga a incidencia, poder ser capaz de trazar a discusión, mesmo se a incidencia en si mesma non é o foro da discusión. A persoa que comeza o fío pode atopar este traballo como algo incómodo, mais o software libre baséase fundamentalmente nunha cultura de escritor responsable: é moito máis importante facer sinxelas as cousas para ducias ou centos de persoas que poidan ler o informe de erros que para tres ou cinco persoas que escriban sobre o mesmo.

É correcto coller as conclusións importantes e mailos resumos desde a lista de discusión e pegalos na incidencia no notificador de erros, se isto vai ser algo conveniente para os lectores. Un idioma común pode ser comezar nunha lista de discusión, poñer unha ligazón ao fío concreto na incidencia no notificador de erros, e cando a discusión rematar, pegar o resumo final na propia incidencia (conxuntamente cunha ligazón á mensaxe que conteña ese resumo), de tal xeito que calquera que navegue á incidencia poida dun xeito sinxelo ver que conclusión se alcanzou sen ter que premer en ningún outro lugar. Fíxate en que o problema habitual de duplicación proveniente da casuística das "dúas copias mestras" aquí non existe, dado que ambos arquivos e notificacións son habitualmente estáticos, co cal estamos a falar de datos inalterables.

Publicidade

No software libre existe unha barreira case inexistente entre as discusións puramente internas e mailas

públicas. Isto en parte débese a que a audiencia obxectivo sempre está pouco definida: dado que a maioría ou todos os correos son publicamente accesibles, o proxecto non ten un control efectivo sobre a impresión que del ten o resto do mundo. Alguén, digamos un editor de mancomun.org (http://mancomun.org), pode atraer a atención de milleiros de lectores a un correo que ninguén esperaba que fose visto fóra do proxecto. Esta é unha realidade coa que viven todos os proxectos de software libre, mais na práctica o risco é habitualmente pequeno. En xeral, os anuncios que o proxecto máis quere publicitar son con frecuencia os que son máis publicitados, asumindo que empregues os mecanismos axeitados para indicarllo ao resto do mundo.

Para grandes anuncios, adóitanse empregar catro ou cinco canais principais de distribución, nos cales os devanditos anuncios deberían facerse o máis simultaneamente posible:

1. A páxina de portada do teu proxecto probablemente sexa vista por máis xente que calquera outra parte do proxecto. Se tiveres un anuncio realmente grande que facer, fai propaganda aí. A propaganda debería estar composta por un breve resumo que enlace ao comunicado de prensa (ver a continuación) para máis información.
2. Ao mesmo tempo, deberías ter tamén unha área de "Novas" ou "Notas de prensa" no sitio web, onde o anuncio poida ser escrito en detalle. Parte do propósito dunha nota de prensa é fornecer un "obxecto de anuncio" simple e canónico que outros sitios web poidan enlazar, así que asegúrate de estruturalo correctamente: tanto unha páxina web por anuncio, como unha entrada concreta nun blog, como algún outro tipo de entidade que poida ser enlazada, mantenas distinguibles do resto de notas de prensa na mesma área.
3. Se o teu proxecto conta cun feed RSS (bótalle unha ollada á sección "*Feeds RSS*"), asegúrate que o anuncio tamén se publique nel. Isto debería facerse automaticamente cando creas a nota de prensa, dependendo de como teñas configuradas as cousas no teu sitio web.
4. Se o anuncio trata sobre o lanzamento dunha nova versión do software, entón actualiza a entrada do teu proxecto en http://freshmeat.net (ver a sección "*Anunciando*" para creares a entrada en primeiro lugar). Cada vez que actualizas unha entrada en Freshmeat, esta introdúcese na lista de mudanzas do día de Freshmeat. A lista de mudanzas actualízase non só en Freshmeat, senón tamén en varios portais (incluíndo http://slashdot.org) que son vistos con avidez por hordas de xente. Freshmeat tamén ofrece a mesma información por sindicación RSS, polo que a xente que non estiver subscrita ao RSS do teu proxecto podería igualmente ver o anuncio vía Freshmeat.
5. Envía un correo á lista de correo de anuncios do teu proxecto. O nome desta lista debería ser "announce", é dicir, *announce@dominio-do-teu-proxecto.org* xa que é unha convención estándar a día de hoxe; na definición da lista deberías deixar claro que é unha lista de moi baixo tráfego, reservada para anuncios importantes do proxecto. A maioría destes anuncios van ser sobre novas versións do software, mais ocasionalmente outros eventos, como os de colecta de fondos, o descubrimento dunha vulnerabilidade de seguridade (ver a sección "Anunciando vulnerabilidades de seguridade" máis adiante neste capítulo), ou unha mudanza importante na dirección do proxecto tamén poden ser enviados a esta lista. Dado que é de baixo tráfego e empregada unicamente para cousas importantes, a lista *announce* acostuma ser a de maior grao de subscrición de todo o proxecto (o cal, por suposto, significa que non debes abusar dela; así que pénsao coidadosamente antes de publicares na mesma). Para evitar que a xente aleatoriamente poida facer anuncios na mesma, ou peor, que chegue spam a través dela, a lista *announce* sempre debe estar moderada.

Tenta facer os anuncios en todos estes lugares ao mesmo tempo, o máis proximamente posible. A xente pode obter unha sensación de confusión se ven un anuncio na lista de correo mais non o ven reflectido

na páxina web do proxecto ou na súa área de notas de prensa. Se acumulares todas as fontes (emails, edición da páxina web, etc) e lanzares todo ao mesmo tempo, entón seguramente acadarás un nivel de incoherencia moi baixo.

Para un evento menos importante, podes prescindir dalgún dos medios explicados. O evento vai ficar igualmente notificado ao mundo exterior, mais vaino facer en proporción á súa importancia. Por exemplo, embora unha nova versión sexa un evento de importancia, simplemente marcar a data da seguinte versión, embora sexa aproximada, non é tan importante como o lanzamento en si mesmo. Marcar a data é merecedor dun correo electrónico ás listas de correo diarias (non á de anuncios) así como unha actualización da planificación temporal ou do status na páxina web, mais non é merecedora de máis.

Porén, podes chegar a ver a data en cuestión aparecendo en discusións por calquera lugar en Internet no que houber xente interesada no proxecto. A xente que simplemente é receptora nas listas de correo, só escoitando e nunca dicindo nada, non é necesariamente silenciosa en todos lados. O boca a boca fornece unha distribución moi grande; debes telo en conta e mesmo construíres os anuncios menores dun xeito que fomente a transmisión informal. Especificamente, os correos que esperas que sexan citados deberían contar cun anaco que diga "cita esta parte", como se estiveses escribindo unha nota de prensa formal. Por exemplo:

> *Actualización de progreso: estamos planificando a versión 2.0 de Ghandalf para mediados de Agosto do 2010. Sempre podes revisar http://www.ghandalf.org/status.html para ver actualización. A nova funcionalidade de maior calado vai ser as procuras mediante expresións regulares.*
>
> *Outras novas funcionalidades inclúen:... Para alén diso tamén vai haber varias correccións de erros, incluíndo:...*

O primeiro parágrafo é curto, fornece as dúas pezas máis importantes de información (a data de lanzamento e a nova funcionalidade de maior calado), e máis unha URL a visitar para ver máis novas. Se este parágrafo é a única cousa que vai cruzar pola pantalla de alguén, estariámolo facendo bastante ben. O resto do correo-e podería perderse sen afectarlle á esencia do correo. Por suposto, algunhas veces a xente enlazará o correo enteiro, mais en xeral o normal é que citen unicamente unha parte pequena. Dado que isto é unha posibilidade, de todos xeitos seguramente queiras facerllo fácil, e no mesmo pacote tamén contar con influencia sobre o que se citar.

Anunciando vulnerabilidades de seguridade

Manexar as vulnerabilidades de seguridade é diferente de manexar calquera outro tipo de informe de erros. No software libre, facer as cousas de xeito aberto e transparente acostuma a ser case un credo relixioso. Cada paso do proceso estándar de manexo de erros é visible para calquera que o quixer ver: a chegada da notificación inicial, a conseguinte discusión, e maila eventual solución.

Os erros de seguridade son diferentes. Poden comprometer os datos dos usuarios, ou mesmo os seus ordenadores dun xeito completo. Discutir ese problema dun xeito aberto podería amosar a súa existencia ao mundo enteiro -incluíndo todas as partes que poderían facer un uso malicioso do erro. Deste xeito, mesmo facer *commit* dunha solución pode anunciar a existencia do erro (hai potencias

atacantes que vixían dun xeito sistemático os rexistros públicos de *commits* dos proxectos, buscando mudanzas que indiquen problemas de seguridade no código previo á mudanza). A maioría dos proxectos software libre teñen establecido aproximadamente o mesmo conxunto de pasos para manexar o conflito entre o aperturismo e o secretismo, baseado nestas directrices básicas:

1. Non falar publicamente sobre o erro ata que houber unha solución dispoñible; nese momento fornecer a solución exactamente no mesmo momento no que anuncias o erro.
2. Acadar a solución o máis axiña que poidas;especialmente se alguén externo ao proxecto foi quen informou do erro, porque entón xa sabes que existe polo menos unha persoa allea ao proxecto que podería explotar a vulnerabilidade.

Na práctica, estes principios lévannos a unha serie estandarizada de pasos, os cales se describen nas seguintes seccións.

Recibir o informe

Obviamente, un proxecto precisa a habilidade de recibir informes de erro de seguridade enviados por calquera. Mais o enderezo habitual de recepción de erros non é o mellor lugar, xa que pode estar vixiado por calquera. Debido a isto, ten unha lista de correo separada para recibir informes sobre erros de seguridade. Esta lista de correo non debe ser accesible para o público xeral, e a subscrición á mesma debe estar estritamente controlada -só os desenvolvedores de confianza e con longa participación no proxecto poden estar nesta lista. Se precisares unha definición formal de "confianza", podes empregar "todo aquel que teña privilexios de *commit* desde hai dous anos ou máis" ou algo semellante, para evitares favoritismos. Este é o grupo que vai manexar os erros de seguridade.

Idealmente, a lista de seguridade non debería estar protexida contra spam nin moderada, dado que non queres que un informe importante sexa descartado ou atrasado simplemente porque non houbo moderadores en liña esa semana. Se empregares software de protección automática contra spam, trata de configuralo cos parámetros de maior tolerancia; é mellor deixar entrar un poucos correos de spam que perder un informe de erros. Para que a lista sexa efectiva por suposto que debes publicitar o seu enderezo; mais dado que non vai estar moderada, e máis importante, pobremente protexida contra spam, tenta non dar nunca o seu enderezo sen algún tipo de transformación que oculte o seu enderezo real, tal como se describe na sección "*Enderezo oculto en ficheiros*" no *Capítulo 3 Infraestrutura Técnica*. Afortunadamente, a ocultación de enderezo non fai que o enderezo sexa ilexible; bótalle unha ollada a http://subversion.tigris.org/security.html, e mira o HTML fonte da páxina, por exemplo.

Desenvolve silenciosamente a solución ao erro

Entón, que fai a lista de seguridade cando recibe un informe? A primeira tarefa é avaliar a severidade do problema e maila súa urxencia:

1. Como de seria é a vulnerabilidade? Permite que un atacante malicioso tome o control do ordenador de alguén que use o teu software? Ou simplemente pode haber algunha fuga de información sobre os tamaños dalgúns dos seus ficheiros?
2. Como de fácil é explotar a vulnerabilidade? Pode realizarse o ataque con *scripts*, ou require coñecemento circunstancial, suposicións e sorte?

3. *Quen* te informou do problema? A resposta desta pregunta por suposto que non muda a natureza da vulnerabilidade, mais dáche unha idea sobre que outra xente pode coñecelo. Se o informe provén dalgún desenvolvedor do proxecto, podes estar un pouco máis tranquilo (mais só un pouco) dado que podes confiar en que non llo teña contado a ninguén máis. Por outra banda, se o informe veu nun correo electrónico desde *anonymous14@globalhackerOrco.net* entón o mellor é que actúes o máis axiña que poidas. A persoa en cuestión fíxoche un favor informándote do problema, mais non tes nin idea de a canta outra xente llo puido contar, ou sobre cando tempo vai tardar antes de explotar a vulnerabilidade nas instalacións en produción.

Ten en conta que a diferenza sobre a que estamos falando é con frecuencia un estreito rango entre *urxente* e *extremadamente urxente*. Mesmo cando o informe provén dunha orixe coñecida e amistosa, pode haber outras persoas na Rede que descubriran o erro xa fai tempo e non o notificaran. O único caso no que as cousas non son urxentes é cando o erro inherentemente non compromete a seguridade dun xeito severo.

O exemplo *anonymous14@globalhackerOrco.net* non é de broma. Na realidade podes obter notificacións de erros enviadas por xente coa identidade encuberta, os cales tanto coas súas palabras como comportamento nunca chegan a deixar claro de todo se están do teu lado ou non. Realmente non importa: se te informaron da vulnerabilidade de seguridade, seguramente pensan que están facendo algo bo por ti, e debes responder en consecuencia. Agradécelles a notificación, dálles unha data estimada na que pensas publicar a solución do problema, e mantenos ao tanto. Algunhas veces son eles quen che dan a ti unha data; isto é, unha ameaza implícita de publicitar o erro en determinada data, esteas ti preparado ou non. Embora poida parecer un xogo de poder, realmente acostuma a ser unha acción preventiva derivada de anteriores decepcións con produtores irresponsables de software que non toman suficientemente en serio os erros de seguridade. De calquera xeito, non podes permitirte o luxo de non prestarlle atención a esta persoa. Despois de todo, se o erro é severo, el ten un coñecemento que lle podería causar grandes problemas aos teus usuarios. Trata ben estes informadores, e reza para que eles tamén te traten ben a ti.

Outro informador frecuente de erros de seguridade son os profesionais da seguridade, persoas que examinan código profesionalmente, e están ao tanto das últimas novas sobre vulnerabilidades no software. Esta xente acostuma a ter experiencia nos dous lados do valado -teñen recibido e enviado informes, probablemente en maior número que a maioría dos desenvolvedores do teu proxecto. Normalmente dánche unha data límite para reparares a vulnerabilidade, antes de facela pública. A data límite é con frecuencia negociable, mais depende do informador; as datas límite están consideradas polos profesionais da seguridade como o único xeito fiable para estimular as organizacións a solucionaren os problemas de seguridade con puntualidade. Debido a isto, non consideres que a data límite é unha imposición; é unha tradición honorable, motivada por razóns de peso.

Unha vez que coñezas a severidade e maila urxencia, podes comezar a traballar na solución. Ás veces xorde un conflito entre fornecer unha solución elegante e fornecela rápido; isto é polo que debes ter ben clara a urxencia antes de comezares. Mantén a discusión restrinxida unicamente aos membros da lista de seguridade e, por suposto, ao informador (se quixer participar), incorporando calquera desenvolvedor que for preciso incluír por motivos técnicos.

Non fagas commit da solución no repositorio. Mantena en formato parche ata a data da súa publicación. Se fixeres commit da mesma, mesmo engadindo unha mensaxe aparentemente inocente no rexistro de log, alguén podería decatarse e entender a mudanza. Nunca sabes quen está mirando o teu repositorio e os motivos polos que pode estar interesado. Deshabilitar os correos electrónicos de commit tampouco

axuda; primeiro de todo o intervalo diferencial na secuencia da lista de commits pode ser sospeitoso por si mesmo, e de todos xeitos, os datos seguirían estando no repositorio. O mellor é facer todo o desenvolvemento nun parche e manter este parche nun lugar privado, como un un repositorio separado e privado, coñecido unicamente pola xente que xa está informada sobre o erro. (Se empregares un sistema de control descentralizado como Arch ou SVK, poderás facer o traballo baixo un completo control de versións, e simplemente manter ese repositorio inaccesible para os que deba estalo).

Números CAN/CVE

Pode que teñas visto un *número CAN* ou un *número CVE* asociados con problemas de seguridade. Estes números preséntanse habitualmente do xeito "CAN-2004-0397" ou "CVE-2002-0092", por exemplo.

Ambos tipos de números representan o mesmo tipo de entidade: unha entrada na lista de "Exposicións e Vulnerabilidades Comúns", lista mantida en http://cve.mitre.org . O propósito da lista é fornecer nomes estándar para todos os problemas de seguridade, de tal xeito que cada un teña un número único e canónico que se poida empregar cando se discutir sobre o mesmo, así como un lugar centralizado a onde ir para atopar máis información. A única diferenza entre un número "CAN" e un número "CVE" está en que o primeiro representa unha entrada candidata, aínda non aprobada para a súa inclusión na lista oficial pola Xunta Editorial CVE, mentres que a segunda representa unha entrada efectivamente aprobada. Aínda así, ambos tipos de entradas son visibles para o público, e o número dunha entrada non muda cando a mesma é aprobada -o prefixo "CAN" simplemente substitúese por "CVE".

Unha entrada CAN/VCE non contén en si mesma unha descrición completa do erro e o xeito de protexerse del. En lugar diso, contén un breve resumo, e máis unha lista de ligazóns a recursos externos (como arquivos de listas de correo) onde a xente pode obter información máis detallada. O propósito verdadeiro de http://cve.mitre.org pasa por fornecer un espazo ben organizado no que cada vulnerabilidade poida obter un nome e unha ruta clara para máis información. Bótalle unha ollada a http://cve.mitre.org/cgi-bin/cvename.cgi?name=2002-0092 para veres un exemplo dunha entrada. Ten en conta que as ligazóns poden ser moi concisas, con fontes expostas como abreviaturas cifradas. A chave para interpretar estas abreviaturas atópase en http://cve.mitre.org/data/refs/refkey.html

Se a túa vulnerabilidade concordar cos criterios CVE, podes obter un número CAN para a mesma. O proceso para levar isto a cabo é deliberadamente tutelado: basicamente, tes que coñecer alguén, ou alguén que coñecer alguén. Isto non é tan tolo como poida parecer. Co obxectivo de evitar a saturación da Xunta Editorial CVE debido a solicitudes espúreas ou pobremente escritas, unicamente aceptan solicitudes provenientes de fontes que xa coñecen e nas que confían. Para conseguires que a túa vulnerabilidade se incorpore á lista, primeiramente debes atopar un camiño desde o teu proxecto ata a Xunta Editorial CVE. Pregúntalle aos teus desenvolvedores; probablemente algún dos mesmos coñeza a alguén que tivera percorrido o proceso CAN con anterioridade, ou que coñeza alguén que o fixera. A vantaxe de facelo deste xeito tamén se atopa en que ao longo da cadea pode que alguén teña o suficiente coñecemento para dicirche que a) non vai concibirse como unha vulnerabilidade ou risco baixo os criterios MITRE, polo que non ten sentido enviarlla, ou b)a vulnerabilidade *xa* ten un número CAN ou CVE. A última posibilidade pode xurdir se o erro de seguridade xa foi previamente publicada noutra lista de seguridade, como pode ser por exemplo http://www.cert.org, ou na lista de correo BugTraq dispoñible en http://www.securityfocus.com (Se isto acontecer sen que o teu proxecto o saiba, deberías preocuparte sobre que outras cousas poden saber outros e que ti non saibas).

Se finalmente acadas un número CAN/CVE, o normal é que desexes conseguilo nas primeiras fases de investigación do teu erro, de xeito que todo o resto da comunicación poida referirse a ese número. As entradas CAN están embargadas ata a data de publicación; a entrada unicamente existe como unha reserva vacía (para que non perdas o nome), mais sen capacidade de revelar ningunha información sobre a vulnerabilidade ata que se chegar á data na que se anuncia a solución do erro.

Máis información sobre o proceso CAN/CVE pode atoparse en http://cve.mitre.org/cve/identifiers/candidates_explained.html e unha exposición particularmente clara de como se empregan os números CNA/CVE nos proxectos de software libre está dispoñible en http://www.debian.org/security/cve-compatibility

Pre-notificación

Unha vez que o teu equipo de resposta ante problemas de seguridade (isto é, aqueles desenvolvedores incluídos na lista de correo de seguridade ou involucrados na resolución dun informe concreto) tiver a solución lista, debes decidir como distribuíla.

Se simplemente fas commit da solución no teu repositorio, ou se pola contra llo anuncias ao mundo, estás forzando calquera que usar o teu software a actualizarse inmediatamente baixo o risco de ser hackeado. Ás veces o máis axeitado, en liña co anterior, é levar a cabo unha *pre-notificación* no caso de certos usuarios importantes. Isto é particularmente preciso en software cliente/servidor, onde pode haber servidores ben coñecidos que son obxectivos tentadores para atacantes. Os administradores destes servidores van apreciar poder contar cun día extra ou dous para faceren a actualización, de tal xeito que xa estean protexidos unha vez que o *exploit* da vulnerabilidade sexa publicamente coñecido.

A pre-notificación quere dicir unicamente enviarlle correos electrónicos aos administradores anteriores antes da data de publicación da vulnerabilidade, facéndoos partícipes da existencia da vulnerabilidade e do xeito de solucionala. Debes facerlle estas pre-notificacións unicamente aquela xente na que confiares e sexa discreta coa información. Deste xeito, a cualificación para recibir pre-notificacións ten dúas compoñentes: o receptor debe administrar un servidor importante e grande onde un incidente de seguridade poida ser un tema serio, *e* o receptor debe ser recoñecido como alguén que non se vaia da lingua antes da publicación da vulnerabilidade.

Envía cada correo electrónico de pre-notificación individualmente (un de cada vez) a cada receptor. *Non* llo envíes á lista enteira de destinatarios dunha tacada, dado que nese caso verían o nome do resto de destinatarios -dando a entender que esencialmente están alertando cada receptor do feito de que *cada* destinatario pode ter un burato de seguridade no seu servidor. Enviárllelo mediante o uso do campo BCC (copia oculta) tampouco é unha boa solución, dado que algúns administradores protexen as súas bandexas de entrada con filtros anti-spam que con frecuencia bloquean ou reducen a prioridade dos correos electrónicos enviados en copia oculta, dado que moito spam envíase en copia oculta.

Aquí vai unha mostra de correo electrónico de pre-notificación:

> *De: Aquí vai o teu nome*
> *Para: admin@chachi-famoso-servidor.com*
> *Responder a: Aquí vai o teu nome (e non o enderezo da lista de seguridade)*
> *Asunto: Notificación Confidencial de vulnerabilidade en Ghandalf.*

Este correo electrónico é unha pre-notificación confidencial dunha alerta de seguridade no servidor Ghandalf.

*Por favor *non reenvíes* ningunha parte deste correo a ninguén.. O anuncio público non se vai facer ata o 23 de xaneiro, e gustaríanos manter a información oculta ata esa data.*

Recibiches este correo porque (pensamos que) empregas un servidor Ghandalf, e queremos que o remendes antes de que o burato de seguridade se faga público o o vindeiro 23 de xaneiro.

Referencias:
==========

CAN-2004-1771: Stack overflow de Ghandalf baixo consultas de Balrogs

Vulnerabilidade:
==========

O servidor pode executar comandos arbitrarios se o 'locale' do servidor está mal configurado e o cliente envía consultas mal formadas.

Severidade:
=========

Moi severa, pode involucrar execución de código arbitrario no servidor.

Rodeos:
==========

Establecer a opción de "procesado-de-linguaxe-natural" a 'apagado' no ficheiro ghandalf.conf péchalle a porta a esta vulnerabilidade.

parche:
======

O parche que se fornece a continuación pode empregarse en Ghandalf 3.0, 3.1, e 3.2.

Unha nova versión pública (Ghandalf 3.2.1) vai ser lanzada xusto despois do 23 de xaneiro, para que estea dispoñible ao mesmo tempo que a vulnerabilidade se fai pública. Podes remendar agora, ou esperar pola publicación. A única diferenza entre Ghandalf 3.2 e Ghandalf 3.2.1 vai ser este parche.

[...O parche anéxase aquí...]

Se tiveres un número CAN, achégao na pre-notificación (como se mostrou arriba) aínda que a información aínda estiver oculta e aínda que a páxina MITRE non mostrar nada. Incluír o número CAN permítelle ao destinatario coñecer con certeza que o erro que se lle pre-notificou é o mesmo do que máis tarde escoitou falar nos canais públicos, polo que non vai ter que preocuparse sobre se é preciso levar a cabo algunha acción máis ou non, dado que este é precisamente o obxectivo do números CAN/CVE.

Distribúe publicamente a solución

O último paso na xestión dun erro de seguridade é a distribución pública da solución. Deberías describir o problema nun anuncio sinxelo e comprensible, dando o número CAN/CVE se estiver dispoñible, e describindo como solucionar o erro tanto dun xeito temporal como dun xeito definitivo. Normalmente "solucionar" (fix) quere dicir levar a cabo unha actualización a unha nova versión do software, embora ás veces poida significar levar a cabo o parche, particularmente no caso de que o software normalmente se execute en forma de código fonte. Se fas unha nova versión, esta debería diferir da versión previa unicamente no parche de seguridade. Deste xeito os administradores conservadores poden levar a cabo a actualización sen se preocupar sobre que outras cousas poden resultar afectadas; a maiores tampouco terán que se preocupar no caso de futuras actualizacións, dado que a solución ao erro de seguridade vai estar incluída en todas as versión futuras como parte do seu ADN. (Os detalles dos procesos de publicación discútense na sección "*Releases de seguridade*" do *Capítulo 7 Empaquetado, versionado e desenvolvemento diario*)

Tanto se a solución publicada ao erro de seguridade leva aparellada unha nova versión do software, como se non, fai o anuncio coa mesma intensidade coa que farías a publicación dunha nova versión: envía un correo electrónico á lista *announce* do proxecto, fai unha nova nota de prensa, actualiza a entrada en Freshmeat, etc. Aínda que nunca deberías restarlle importancia, de cara ao exterior, á existencia dun erro de seguridade para manteres a reputación do proxecto, evidentemente debes empregar un ton e actitude durante o anuncio do problema de seguridade acorde coa severidade real do problema. Se o buraco de seguridade acarreta unicamente unha exposición menor de información, e non permite a existencia de exploits que permitan que o ordenador sexa controlado por terceiros, mellor evitar un montón de ruído. Mesmo podes decidir non distraer a lista *announce* con este erro menor. Despois de todo, se o proxecto constantemente di que ven o lobo, pode que os usuarios acaben pensando que o software é menos seguro do que realmente é, e mesmo que non te crean cando teñas que anunciar un problema realmente grande. Bótalle unha ollada a http://cve.mitre.org/about/terminology.html para unha boa introdución ao problema de asignar severidade.

En xeral, se non estiveres seguro de como tratar un problema de seguridade, atopa a alguén con experiencia e cóntallo. Avaliar e xestionar vulnerabilidades é unha destreza que se vai adquirindo, e é habitual cometer erros as primeiras veces.

Capítulo 7. Empacotamento, liberación e dia a dia no desenvolvemento

Este capítulo trata sobre como os proxectos de software libre empacotan e liberan o seu software e de

como os padróns de desenvolvemento globais se organizan en torno a eses obxectivos.

A maior diferenza entre os proxectos de software libre e os propietarios e a falta de control centralizado sobre o equipo de desenvolvemento. Cando unha nova versión está a ser preparada, esta diferenza é especialmente notable: unha corporación pode pedir a todo o seu equipo de desenvolvemento que se centre na versión que está a piques de ser lanzada, deixando de lado as novas funcionalidades e os erros que non son críticos ata que finalmente se lanzar a nova versión. Os grupos de voluntarios non son monolíticos. A xente traballa nun proxecto por moitas razóns diferentes, e aqueles que non están interesados en axudar no proceso de lanzamento da nova versión poden querer seguir colaborando no desenvolvemento mentres a nova versión está a piques de saír. Debido a que o desenvolvemento non para, o proceso de lanzamento de novas versión no software libre tende ser longo mais menos problemático que nos procesos comerciais. E similar a reparar unha autopista. Hai dúas maneiras de arranxar unha estrada, pódese pechar por completo de xeito que os operarios poidan traballar en toda ela por completo ata reparala toda ou pódese traballar nun par de faixas deixando outras abertas ao tránsito. A primeira forma é moi eficiente *para os operarios*, mais para ninguén máis; a estrada fica totalmente pechada ata a finalización dos traballos. A segunda forma require moito máis tempo e dificultades para os operarios (agora eles teñen que traballar con menos xente e menos equipamento, con estreiteces, con sinais para reducir a velocidade do tránsito e dirixilo, etc.), mais polo menos a estrada permanece aberta, embora non por completo.

Os proxectos de software libre tenden a traballar da segunda maneira. De feito, para un proxecto maduro con varias liñas diferentes de versións mantidas simultaneamente, o proxecto atópase permanentemente nun estado de reparacións menores na estrada. Hai sempre un par de faixas pechadas; un constante mais baixo nivel de inconvenientes é sempre soportado polo grupo de desenvolvemento coma unha molestia, para que a liberación das versións siga un ciclo regular.

O modelo que posibilita isto xeneralízase a algo máis que só a liberación de versións. É o principio de paralelización de tarefas que non son interdependentes; un principio que de ningún modo é exclusivo para o desenvolvemento de software libre, por suposto, mais que é implementado no software libre dun xeito particular. Non se pode asumir molestar demasiado os homes que traballan na estrada nin o tránsito, mais tampouco se pode permitir ter xente tras os conos laranxas dirixindo o tránsito con bandeiras. Así, gravitan en torno a procesos que teñen niveis planos e constantes de esforzo de administración en lugar de altos e baixos. Os voluntarios xeralmente están dispostos a traballar con cantidades de inconvenientes pequenas mais coherentes; a previsibilidade permítelles ir e vir sen se preocuparen de se as súas planificacións entran en conflito co que ocorre no proxecto. Mais se o proxecto estiver suxeito a unha planificación mestra na cal unhas actividades exclúen outras, o resultado sería desenvolvedores perdendo tempo; que sería, non só ineficiente, senón aburrido, e por tanto perigoso, xa que un desenvolvedor aburrido está preto de se converter nun ex-desenvolvedor.

O traballo de lanzamento de novas versións é xeralmente a tarefa de non desenvolvemento máis perceptible que sucede en paralelo co desenvolvemento, polo tanto os métodos que se describen nas seguintes seccións están concibidos maioritariamente para posibilitar as novas versións. Porén, nótese que estes deben ser aplicados paralelamente con outras tarefas, como a tradución e internacionalización, mudanzas claras na API feitos gradualmente a través do código base, etc.

Numeración de versións

Antes de falarmos sobre como facer unha versión, imos ver como nomear as versións, o cal require coñecer o que as versións queren dicir para os usuarios. Unha versión quere dicir que:

- Vellos erros foron resoltos. Isto é probablemente unha das cousas que os usuarios esperan en cada versión.
- Novos erros foron engadidos. Con isto sóese contar tamén, excepto algunhas veces no caso de versións de seguridade ou outras que só saen unha vez (ver a sección “security-releases" máis adiante neste capítulo).
- Novas funcionalidades puideron ser engadidas.
- Novas opcións de configuración puideron ser engadidas, ou o significado de vellas opcións puido mudar sutilmente. A instalación de procedementos puido mudar lixeiramente desde a última versión tamén, embora sempre se espere que non.
- Mudanzas incompatibles poden ser introducidas, por exemplo o formato dos datos usados por versións anteriores do software xa non son usables sen ningún tipo de conversión (posiblemente manual).

Como se pode ver, non todas estas cousas son boas. Isto é polo que os usuarios experimentados solicitan novas versións con algo de inquietude, especialmente cando o software é maduro e xa fai case todo o que eles queren (ou pensan que queren). Mesmo a chegada de novas funcionalidades é unha beizón a medias, xa que pode causar que o software se comporte agora de xeito inesperado.

O propósito da numeración de versións, polo tanto, é dupla: obviamente os números deben comunicar sen ambigüidade a orde das versións (por exemplo, ollando os números de dúas versións, un debe poder saber cal vén despois), mais ademais deben indicar do xeito máis compacto posible o nivel e natureza das mudanzas na versión.

Todo iso nun número? Ben, máis ou menos, si. As estratexias de numeración de versións son unha das vellas discusións (ver sección "Canto máis suave for o tema, máis longo vai ser o debate" no capítulo 6, "Comunicacións"), e é pouco probable que se atope unha solución sinxela, completa e estándar. Porén, existen unhas poucas boas estratexias, xunto cun principio universal: *ser coherente*. Escoller un esquema numérico, documentalo e mantelo. Os usuarios agradecerano.

Compoñentes numéricos da versión

Esta sección describe as convencións formais da numeración das versións en detalle, e asume moi poucos coñecementos previos. Está prevista principalmente coma unha referencia. Se xa estás familiarizado con estas convencións, podes saltar esta sección.

A numeración de versións son grupos de díxitos separados por puntos:

Scanley 2.3
Singer 5.11.4

...etc. Os puntos *non* representan decimais, son meros separadores; "5.3.9" iría seguido de "5.3.10". Uns poucos proxectos son etiquetados de forma diferente, o máis famoso é o kernel de Linux coas súas secuencias "0.95", "0.96"... "0.99" ata Linux 1.0, mais a convención de que eses puntos non son decimais é agora firmemente establecida e debería ser considerada estándar. Non hai límites sobre o

número de compoñentes (porción de díxitos que non conteñen puntos), mais a maioría dos proxectos non teñen máis de tres ou catro. A razón é para que sexan máis comprensibles.

Ademais dos compoñentes numéricos, os proxectos ás veces engaden unha etiqueta descritiva como "Alpha" ou "Beta" (ver "alpha-e-beta"), por exemplo:

Scanley 2.3.0 (Alpha)
Singer 5.11.4 (Beta)

Unha cualificación Alpha ou Beta quere dicir que esta versión *precede* a unha futura versión que terá o mesmo número mais sen o cualificador. De esta maneira, "2.3.0;(Alpha)" conduce finalmente a "2.3.0". Para permitir tanta cantidade de versións distintas, os cualificadores teñen eles mesmos meta-cualificadores. Por exemplo, aquí hai unha serie de versións que estarían dispoñibles para o público:

Scanley 2.3.0 (Alpha 1)
Scanley 2.3.0 (Alpha 2)
Scanley 2.3.0 (Beta 1)
Scanley 2.3.0 (Beta 2)
Scanley 2.3.0 (Beta 3)
Scanley 2.3.0

Nótese que cando ten o cualificador "Alpha", Scanley "2.3" escríbese como "2.3.0". Os dous números son equivalentes; os compoñentes que son ceros poden ser eliminados por brevidade; mais cando un cualificador está presente, a brevidade non serve para nada, polo que debe poñerse completo.

Outros cualificadores que soen ser empregados son "Estable", "Inestable", "Desenvolvemento", e "RC" (para "Versión Candidata"). Os máis comunmente empregados son "Alpha" e "Beta", con "RC" moi preto no terceiro posto, mais nótese que "RC" sempre inclúe un meta-cualificador numérico. Isto é, non liberas a versión "Scanley2.3.0(RC)" liberas "Scanley2.3.(RC 1)"), seguido da RC2, etc.

Esas tres etiquetas, "Alpha", "Beta", e "RC", son bastante máis coñecidas agora, e non recomendo usar ningunha outra, mesmo pensando que as outras poden parecer mellores a primeira vista porque son palabras normais, non xíria. Mais a xente que instala software desde versións está xa familiarizado coas tres, e non hai razón para facer as cousas de maneira distinta daquela que a xente coñece porque si.

Embora os puntos nos números da versión non sexan puntos decimais, a súa posición ten significado. Todas as versións "0.X.Y" preceden á "1.0" (a cal equivale a "1.0.0", por suposto). "3.14.158" inmediatamente preceden á "3.14.159", e non inmediatamente precede á "3.14.160" así como tamén á "3.15.algo", e así.

Unha política de numeración de versións coherente permite o usuario ver dous números de versión para a mesma peza de software e detectar, en base aos números, as importantes diferenzas entre esas dúas versións. Nun típico sistema de tres compoñentes, o primeiro é o número *maior*, o segundo e o número *menor*, e o terceiro é o número *micro*. Por exemplo, a versión "2.10.17" é a décimo sétima micro versión na décima liña de versión menor dentro da segunda serie de versións maiores. As palabras "liña" e "series" son empregadas informalmente aquí, mais teñen o significado que un esperaría. A serie maior é simplemente todas as versións que comparten o mesmo número maior, e a serie menor (ou liña menor) componse de todas as versións que comparten o número menor *e* o número maior. Isto é, "2.4.0" e "3.4.1" non están na mesma serie menor, embora compartan o "4" como número menor; por

outra banda, "2.4.0" e "2.4.2" están na mesma liña menor, pensando que non son adxacentes se a "2.4.1" foi lanzada entre elas.

O significados de estes números son exactamente o que se esperaría: un incremento do número maior indica que ocorreron mudanzas maiores; un incremento do número menor indica mudanzas menores; e un incremento do número micro indica mudanzas realmente triviais. Algúns proxectos engaden un cuarto compoñente, chamado habitualmente o *número de parche*, para un control moi especializado sobre as diferenzas entre as súas versións (confusamente, outros proxectos usan "patch" coma un sinónimo para "micro" nun sistema de tres compoñentes). Hai tamén proxectos que usan a última compoñente como *número de build*, incrementado cada vez que o software é compilado e sen representar ningunha outra mudanza. Isto axuda ao proxecto a enlazar calquera erro atopado cunha compilación específica, e é probablemente máis útil cando os pacotes binarios son o método por defecto de distribución.

Embora haxa moitas convencións diferentes sobre cantos compoñentes empregar, e sobre o que os compoñentes significan, as diferenzas tenden a ser pequenas podes ter un pouco de flexibilidade, mais non moita. As próximas dúas seccións tratan sobre as convencións máis empregadas.

A estratexia simple

A maioría dos proxectos teñen regras sobre que tipo de mudanzas son permitidas nunha versión se só se incrementa o número micro, regras diferentes para o número menor, e diferentes tamén para o número maior. Non hai un estándar para estas regras aínda, mais aquí farei unha descrición da política que foi adoptada con éxito por moitos proxectos. Ti podes querer simplemente adoptar esta política no teu propio proxecto, mais mesmo se non, é un bo exemplo de que tipo de información deberían dar os números de versións. Esta política está adaptada desde o sistema de numeración usado polo proxecto APR, ver versionamento (http://apr.apache.org/versioning.html).

Mudar o número micro só (isto é, mudanzas na liña menor) deben implicar compatibilidade tanto cara a adiante coma cara a atrás. Isto é, as mudanzas deberían resolver erros soamente, ou moi pequenas melloras en funcionalidades existentes. Novas funcionalidades non deberían ser introducidas nunha micro versión.

Mudanzas no número menor (isto é, dentro da mesma liña maior) deben ser compatibles cara a atrás, mais non necesariamente cara a adiante. É normal introducir novas funcionalidades nunha versión menor, mais xeralmente non moitas ao mesmo tempo.

Mudanzas no número maior marcan unha fronteira na compatibilidade. Unha versión maior pode ser incompatible cara a adiante e cara a atrás. Dunha versión maior espérase que teña novas funcionalidades, e pode mesmo ter un novo xogo de funcionalidades.

O que quere dicir ser *compatible cara a atrás* e *compatible cara a adiante* depende do que o software faga, mais en contexto xeralmente non están abertas a moita interpretación. Por exemplo, se o teu proxecto for unha aplicación tipo cliente/servidor, entón "compatibilidade cara a atrás" significa que actualizando o servidor á versión 2.6.0 non debería causar perda de funcionalidade ou comportamento diferente en clientes existentes 2.5.4 (excepto para os erros que fosen resoltos, por suposto). Por outra banda, actualizar un dos clientes á versión 2.6.0, xunto co servidor, poden ter *novas* funcionalidades

dispoñibles para ese cliente, funcionalidades das que os clientes 2.5.4 no saben como tirar partido. Se isto suceder, entón a actualización *non* é "compatible cara a adiante": claramente ti non podes agora desactualizar o cliente de novo á 2.5.4 e manter toda a funcionalidade que tiña coa 2.6.9, xa que algunhas desas funcionalidades eran novas na 2.6.0.

Isto é polo que as micro versións son esencialmente para arranxar erros. Deben manter a compatibilidade en ambas direccións: se actualizares da 2.5.3 á 2.5.4, despois mudares de opinión en voltares a 2.5.3, non deberías perder ningunha funcionalidade, agás no caso de que os erros restaurados non posibilitaren o uso de algunhas funcionalidades existentes.

Os protocolos cliente/servidor son un dos moitos posibles dominios. Outro é o formato dos datos: o software escribe os datos nun almacenamento permanente? Se o fixer, os formatos de lectura e escritura precisan seguir as liñas de compatibilidade prometidas pola política de numeración das versións. A versión 2.6.0 precisa ser capaz de ler os ficheiros escritos coa versión 2.5.4, mais pode silenciosamente actualizar o formato a algo que a 2.5.4 non pode ler, porque a habilidade de voltar a unha versión anterior non é requirida máis alá do límite que marca o número menor. Se o teu proxecto distribuír bibliotecas de código para que usen outros programas, entón as APIs serán un dominio compatible tamén: debes asegurarte de que as regras de compatibilidade de código e binaria son explicadas de forma que o usuario non precise preguntarse se a actualización vai ser segura ou non. Deberá ser capaz de ver os números de versións e sabelo ao instante.

Neste sistema, non tes unha oportunidade de partir de cero ata que incrementas o número maior. Isto pode a miúdo ser un inconveniente: pode haber funcionalidades que desexes engadir, ou protocolos que desexes redeseñar, que simplemente non poden ser feitos mentres se mantén a compatibilidade. Non hai unha solución máxica para isto, agás tentar deseñar as cousas de xeito extensible ao principio (un tema que merecería un libro, e certamente se escapa do ámbito deste). Mais publicar unha política de compatibilidade de versións, e adherirse a ela, é unha parte imprescindible da distribución de software. Unha desagradable sorpresa pode distanciar unha chea de usuarios. A política descrita é boa parcialmente porque está bastante estendida, mais tamén porque é fácil de explicar e recordar, mesmo para aqueles que aínda non estiveren familiarizados con ela.

Habitualmente asúmese que estas regras non se aplican as versións previas a 1.0 (aínda que a túa política de versións debería probablemente ser explícita, simplemente para ser clara). Un proxecto que está aínda nun desenvolvemento inicial pode ter versións 0.1, 0.2, 0.3, e así en secuencia, ata que estiver listo para a 1.0, e as diferenzas entre esas versións poden ser arbitrariamente grandes. Os micro números en versións anteriores a 1.0 son opcionais. Dependendo da natureza do proxecto e das diferenzas entre as versións, podes atopar útil ter 0.1.0, 0.1.1, etc, ou pode que non. As convencións para as versións anteriores a 1.0 están bastante perdidas, probablemente porque a xente entende que manter unha forte compatibilidade pode comprometer demasiado o desenvolvemento inicial do proxecto, e porque os que as adoptan tenden a esquecelas de calquera xeito.

Lembra que todas estas regras son aplicables só a este particular sistema do terceiro compoñente. O teu proxecto podería facilmente empregar un sistema diferente de tres compoñentes, ou mesmo decidir que non precisa tanta atomización e empregar un sistema con dous compoñentes. O importante é decidir cedo, publicar exactamente o que os compoñentes queren dicir, e manter a decisión.

A estratexia par/impar

Algúns proxectos usan a paridade do número menor para indicaren a estabilidade do software: par quere dicir estable, impar inestable. Isto aplícase só ao número menor, non ao maior e ao micro. Os incrementos no micro número indican aínda erros arranxados (non novas funcionalidades), e incrementos no número maior indican aínda grandes mudanzas, novos xogos de funcionalidades, etc.

As vantaxes do sistema par/impar, o cal é empregado polo kernel de Linux entre outros, ofrece unha forma de liberar versións con novas funcionalidades para testar sen que os usuarios teñan que empregar código inestable. A xente pode ver polos números que "2.4.21" está ben para instalar no seu servidor web, mais a "2.5.1" debería probablemente permanecer confinada para os seus experimentos locais. O equipo de desenvolvemento manexa os informes de erros que veñen das series inestables (as que ten o número menor impar), e cando as cousas empezan a calmarse despois de varias micro versións nesas series, incrementan o número menor, resetean o micro número a "0", e liberan a versión presumiblemente estable do pacote.

Este sistema preserva, ou polo menos, non entra en conflito con, as liñas de compatibilidade dadas anteriormente. Simplemente sobrecarga o número menor con algo de información extra. Isto forza a incrementar o número menor dúas veces tan a miúdo como por outra parte sería necesario, mais non hai maior dano en iso. O sistema par/impar é probablemente mellor para proxectos que teñen ciclos de versións moi longos, os cales pola súa natureza teñen unha alta proporción de usuarios fixos que valoran máis a estabilidade que as novas funcionalidades. Esta non é a única forma de probar novas funcionalidades durante o proceso, porén ver sección “estabilizar unha versión” máis adiante neste capitulo describe outra forma, quizais máis común, un método de liberar versións potencialmente inestables ao público, avisando a xente para que saiban o risco/beneficio inmediatamente ao veren o nome da versión.

Ramas de versións

Desde o punto de vista do desenvolvedor, un proxecto de software libre está sempre nun estado constante de liberación de novas versións. Os desenvolvedores normalmente executan sempre a última versión do código, porque queren descubrir erros, e porque seguen o proxecto o bastante de preto para seren quen de se manteren afastados das areas inestables. A miúdo actualizan a súa copia do software todos os días varias veces, e cando testan unha mudanza , esperan que todos os demais desenvolvedores a teñan en 24 horas.

Como debería entón o proxecto facer unha nova versión formal? Debería tirar unha fotografía da árbore nun determinado momento, empacotala, e entregarlla ao mundo, como, por exemplo, versión "3.5.0"? O sentido común di que non. Primeiro, a árbore de desenvolvemento pode non chegar a estar nunca limpa e preparada para ser liberada. Funcionalidades novas que se están desenvolvendo poden estar en diferentes estados de desenvolvemento. Alguén puido facer unha mudanza grande para arranxar un erro, mais a mudanza podería ser controvertida e estar baixo debate no momento de liberar a versión. Se for así, deberíase simplemente atrasar a liberación ata que rematar o debate, porque outro debate non relacionado podería comezar na mesma altura, e entón esperar a que *ese* debate rematase tamén. Este proceso pode non ter fin.

En calquera caso, empregar a árbore completa para as versións interferiría no proceso de desenvolvemento en marcha, mesmo se a árbore estiver nun punto adecuado para liberar a versión. Digamos que esta variante será a versión "3.5.0·; presumiblemente, a próxima sería a "3.5.1", e tería a maior parte dos erros atopados na 3.5.0 arranxados. Mais se ambas son estados da mesma árbore, que se supón que debería facer un desenvolvedor no período entre as dúas versións? Non poden engadir novas funcionalidades; as guías de compatibilidade o evitan. Mais non todo o mundo estará entusiasmado con arranxar erros no código da 3.5.0. Algunha xente pode ter novas funcionalidades que está tentando completar, e enfadaranse se os forzan a escoller entre non facer nada e traballar en cousas nas que non están interesados, só porque o proceso de lanzamento de versións do proxecto esixa que a árbore de desenvolvemento permaneza inactiva.

A solución a estes problemas é empregar sempre unha *rama para liberar as versións*. Unha rama para liberar as versións é simplemente unha rama no sistema de control de versións (ver "rama"), na cal o código destinado para esta versión pode estar isolado do desenvolvemento principal. O concepto dunha rama para as versións certamente non é orixinal do software libre; moitas organizacións comerciais de desenvolvemento a empregan tamén. Porén, en entornos comerciais, as ramas para as versións son as veces consideradas un luxo un tipo de "boa práctica" formal que pode, baixo presión por unha data límite, ser prescindible mentres todo o mundo do equipo está traballando para estabilizar a árbore principal.

As ramas para as versións son moito máis requiridas nos proxectos de software libre, porén, teño visto proxectos liberar versións sen elas, mais sempre hai algúns desenvolvedores que non fan nada mentres outros normalmente unha minoría traballa para liberar a versión. O resultado soe ser malo de varias maneiras. Primeiro, o desenvolvemento xeral é freado. Segundo, a versión ten menos calidade da que precisa ter, porque só un número pequeno de xente traballa nela, e están apurando para que o resto poida volver ao traballo. Terceiro, divide o equipo de desenvolvemento psicoloxicamente, creando una situación onde diferentes tipos de traballo interfiren con outros innecesariamente. Os desenvolvedores parados probablemente estarían máis contentos contribuíndo *con algo* da súa atención á rama da versión, sempre que for unha escolla acorde co seu propio horario e intereses. Mais sen a rama, a súa escolla convértese en: "participo no proxecto hoxe ou non?" no canto de: "traballo na nova versión hoxe ou nunha nova funcionalidade que estou a desenvolver no código principal?"

Mecanismos das ramas para as versións

Os mecanismos exactos para a creación da rama para as versións dependen do teu sistema de control de versións, por suposto, mais en xeral os conceptos son iguais na maioría dos sistemas. Unha rama, normalmente brota doutra rama ou do trunk. Tradicionalmente, o trunk é onde se fai o desenvolvemento principal, sen as limitacións da versión. A primeira rama para as versións, a que conduce á "1.0", brota do trunk. En CVS, o comando branch sería algo coma isto

```
$ cd trunk-working-copy
$ cvs tag -b RELEASE_1_0_X
```

ou en Subversion, así:

```
$ svn copy http://.../repos/trunk http://.../repos/branches/1.0.x
```

(Todos estes exemplos asumen un sistema de numeración das versións de tres compoñentes. Embora non poida mostrar os comandos exactos para cada sistema de control de versións, darei exemplos en

CVS e Subversion e espero que os correspondentes comandos en outros sistemas poidan ser deducidos destes dous.

Nótese que creamos a rama "1.0.x" (cun literal "x") no canto de "1.0.0". Isto é porque a mesma liña menor i.e., a mesma rama será empregada por todas as micro versións nesa liña. O proceso actual de estabilizar a rama para as versións esta cuberto en "Estabilizar una versión" máis adiante neste capítulo. Aquí preocuparémonos coa interacción entre o sistema de control de versións e o proceso de sacar a versión. Cando a rama das versións estiver estabilizada e preparada, é hora de etiquetar o estado da rama:

```
$ cd RELEASE_1_0_X-working-copy
$ cvs tag RELEASE_1_0_0
```

ou

```
$ svn copy http://.../repos/branches/1.0.x http://.../repos/tags/1.0.0
```

Esa etiqueta representa agora exactamente o estado da árbore de código do proxecto na versión 1.0.0 (isto é útil no caso de que alguén precise obter unha versión vella despois de que os pacotes e binarios fosen eliminados). A seguinte micro versión na mesma liña é tamén preparada na rama 1.0, e cando estiver preparada, farase unha etiqueta para a 1.0.1. Enxaboar, enxaugar, repetir para a 1.0.2, e continuar. Cando sexa tempo de empezar a pensar sobre a serie 1.1.x, crear unha nova rama desde o trunk:

```
$ cd trunk-working-copy
$ cvs tag -b RELEASE_1_1_X

$ svn copy http://.../repos/trunk http://.../repos/branches/1.1.x
```

O mantemento pode continuar en paralelo sobre a 1.0.x e a 1.1.x, e as versións poden ser feitas independentemente desde as dúas liñas. De feito, non é infrecuente publicar case simultaneamente desde dúas liñas diferentes. As vellas series están recomendadas para os administradores máis conservadores, os cales poden non querer dar o salto a 1.1 sen unha coidadosa preparación. Mentres tanto, a xente máis aventureira normalmente colle a versión máis recente da maior liña, para estar seguros de ter as últimas funcionalidades, mesmo co risco dunha maior inestabilidade.

Esta non é a única estratexia para facer ramas, por suposto. Nalgunhas circunstancias mesmo pode non ser a mellor, mais está funcionando bastante ben para proxectos nos que participei. Emprega calquera estratexia que pareza que funciona, mais recorda os puntos principais: o propósito das ramas é isolar a versión para liberar as mudanzas do desenvolvemento diario, e darlle ao proxecto unha entidade física coa que organizar os seus procesos de lanzamento de versións. Ese proceso é descrito con detalle na seguinte sección.

Estabilizando unha versión

Estabilización é o proceso de poñer a rama da nova versión nun estado axeitado para sacala; isto é, decidir que mudanzas irán na versión, cales non, e darlle forma ao contido da rama.

Hai unha chea de dor potencial contida nesa palabra, "decidir". As funcionalidades de último minuto son un fenómeno familiar en proxectos colaborativos de software: tan pronto como os desenvolvedores

ven que unha versión está a punto de sacarse, apresúranse a remataren as mudanzas nas que están a traballar, para non perderen o tren. Isto, por suposto, é exactamente o oposto ao que ti queres nese momento. E moito mellor para a xente traballar nas funcionalidades do xeito máis confortable posible, e non preocuparse se as súas mudanzas van a ir nesta versión ou na seguinte. Cantas máis mudanzas de última hora intentarmos meter na versión, máis se desestabiliza o código, e (xeralmente) máis erros serán introducidos.

A maioría dos enxeñeiros de software están de acordo en teoría con acordar os criterios sobre que mudanzas deberían ser permitidas dentro da liña da versión durante o período de estabilización. Obviamente, pódense incluír solucións a erros graves, especialmente a erros sen solucións parciais. Pódese actualizar a documentación, así como solucións a mensaxes de erro (excepto cando son consideradas parte da interface e deben permanecer estables). Moitos proxectos ademais permiten certos tipos de riscos baixos ou mudanzas que non sexan na parte central durante a estabilización, e poden ter pautas para minimizar o risco. Mais a cantidade de formalización non pode obviar a necesidade de xuízo humano. Sempre haberá casos onde o proxecto simplemente teña que tomar unha decisión sobre se unha mudanza debe ir na versión ou non. O perigo está en que cada persoa quere ver as súas mudanzas favoritas admitidas na versión, entón haberá milleiros de persoas motivadas para permitiren mudanzas, e non suficiente xente motivada para excluílas.

Por tanto, o proceso de estabilizar unha versión é máis sobre crear mecanismos para dicir "non". O truco para os proxectos de software libre, en particular, é atopar formas de dicir "non" sen que o resultado sexan moitos sentimentos feridos ou desenvolvedores decepcionados, e sen desmerecer mudanzas para a versión. Hai distintas formas de facer isto. É bastante sinxelo designar sistemas que satisfagan estes criterios, unha vez que o equipo está enfocado na importancia dos criterios. Aquí describirei brevemente dous dos sistemas máis populares, no estremo final do espectro, mais non permitas o desánimo no teu proxecto por ser creativo. Moitas outras alternativas son posibles; estas son só dúas que eu vin na practica.

Réxime ditatorial do mantedor

O grupo está de acordo con permitir unha persoa ser o *propietario da versión*. Esta persoa ten a palabra final sobre as mudanzas que van na versión. Por suposto, é normal que haxa discusións, mais ao final o grupo debe garantirlle ao propietario da versión suficiente autoridade para tomar as decisións finais. Para que este sistema funcione, é preciso elixir alguén o suficientemente competente tecnicamente para comprender tódalas mudanzas, e con habilidades sociais para navegar nas discusións e sacar a versión sen ferir demasiados sentimentos.

Un padrón común para o propietario da versión é dicir "Non penso que haxa nada malo con esta mudanza, mais non temos suficiente tempo para probala, así que non debería ir nesta versión". Axuda moito se o propietario da versión ten un bo coñecemento técnico do proxecto, e pode dar razóns de por que a mudanza podería ser potencialmente desestabilizadora (por exemplo, as súas interaccións con outras partes do software, ou portabilidades). A xente pedirá algunhas veces que se xustifiquen as decisións, ou argumentará que unha mudanza non é tan arriscada como parece. Estas conversacións precisan non ser confrontacións, sempre e cando o propietario da versión poida considerar todos os argumentos obxectivamente e non ser intransixente.

Nótese que o propietario da versión non ten por que ser a mesma persoa que o líder do proxecto (en

casos onde non hai líder do proxecto; ver “Dictador benovolente" no capítulo 4, Infraestrutura Social). De feito, algunhas veces é bo asegurarse que eles *non* son a mesma persoa. As habilidades que fan a un desenvolvedor líder non teñen porque ser necesariamente as mesmas que o fan o propietario da versión. En algo tan importante como o lanzamento dunha versión, pode ser prudente ter alguén que poña un contrapeso á opinión do líder.

Contrasta o rol do propietario da versión co de menos ditatorial descrito en “Manager das versións” mais adiante neste capítulo.

Mudar o voto

No extremo oposto da ditadura do propietario da versión, os desenvolvedores poden simplemente votar que mudanzas incluír na versión. Sen embargo, xa que a función máis importante da estabilización da versión é *excluír* mudanzas, é importante deseñar o sistema de votación de forma que poñer unha mudanza na versión implique accións positivas para moitos desenvolvedores. Incluír unha mudanza debería precisar máis dunha simple maioría (ver "Quen vota?" no capítulo 4, Infraestructura Social). Doutro xeito, un voto a favor e ningún en contra para unha mudanza sería suficiente para poñer unha mudanza na versión, e unha desafortunada dinámica podería ser creada polo cal cada desenvolvedor votaría polas súa propias mudanzas, porén, serían relutantes a votar en contra das mudanzas de outros desenvolvedores por medo a posibles represalias. Para evitar isto, o sistema debería organizar que subgrupos de desenvolvedores deben actuar en cooperación para poñer calquera mudanza na versión. Isto non so quere dicir que moita xente revise cada mudanza, ademais fai que calquera desenvolvedor dubide menos para votar en contra dunha mudanza, porque non coñece particularmente ningún dos que votaron porque tomaría o voto en contra coma unha afronta persoal. Canto maior for o número de xente involucrada, máis se discutirá sobre as mudanzas e menos sobre os individuos.

O sistema que empregamos no proxecto Subversion parece que acadou un bo balance, polo que o recomendarei aquí. Co fin de que unha mudanza sexa aplicada na rama da versión, polo menos tres desenvolvedores deben votar a favor, e ningún en contra. Un simple voto "en contra" é suficiente para deter a aplicación dunha mudanza; iso é, un voto "en contra" no contexto da versión equivale a un veto (ver "veto"). Naturalmente, calquera voto debe ir acompañado dunha xustificación, e en teoría o veto podería ser invalidado se suficiente xente sentise que é razoable e forzase un voto especial sobre iso. Na práctica, isto nunca sucede, e non espero que pase. A xente é conservadora a respecto das versións de calquera modo, e cando alguén sente fortemente a necesidade de vetar a inclusión dunha mudanza, normalmente hai unha boa razón para iso.

Debido a que o proceso de sacar unha versión está deliberadamente baseado no conservadorismo, as xustificación ofrecidas para os vetos son con máis frecuencia procedimentais que técnicas. Por exemplo, unha persoa pode sentir que unha mudanza está ben escrita e é pouco probable que cause novos erros, mais vota en contra da súa inclusión nunha micro versión porque é demasiado grande quizais engade unha nova funcionalidade, ou de algunha forma sutil falla ao cubrir as liñas de compatibilidade. Ocasionalmente teño visto desenvolvedores vetaren algo porque simplemente tiñan o presentimento de que a mudanza necesitaba máis testaxe, mesmo cando eles non puideron atopar erros nela inspeccionándoa. A xente refungou un pouco, mais os vetos resistiron e a mudanza non foi incluída na versión (Non lembro se foi atopado algún erro máis tarde testando ou non).

Manexando a estabilización das versións de xeito colaborativo

Se o teu proxecto elixe un sistema de voto sobre as mudanzas, é imperativo que o mecanismo físico de establecer votación e votos decisivos sexa o máis conveniente posible. Embora haxa moito software de código aberto para voto electrónico dispoñible, na practica o máis sinxelo e poñer un ficheiro de texto na rama da versión, chamado <filename>STATUS</filename> ou <filename>VOTES</filename> ou algo similar. Este ficheiro lista cada unha das mudanzas propostas calquera desenvolvedor pode propoñer unha mudanza para a súa inclusión xunto con todos os votos a favor e en contra, ademais de notas ou comentarios. (Propoñer unha mudanza non implica necesariamente votar para iso, por certo, aínda que as dúas cousas soen ir xuntas). Unha entrada neste ficheiro ten a seguinte forma:

```
* r2401 (issue #49)
  Prevent client/server handshake from happening twice.
  Justification:
    Avoids extra network turnaround; small change and easy to review.
  Notes:
    This was discussed in http://.../mailing-lists/message-7777.html
    and other messages in that thread.
  Votes:
    +1: jsmith, kimf
    -1: tmartin (breaks compatibility with some pre-1.0 servers;
                 admittedly, those servers are buggy, but why be
                 incompatible if we don't have to?)
```

Neste caso, a mudanza adquiriu dous votos positivos, mais foi vetada por tmartin, quen deu a razón para o seu veto nunha nota entre parénteses. O formato exacto da entrada non importa; sexa cal for o teu proxecto decidir está ben quizais a explicación de tmartin para o veto debería ir na sección "Notes:" ou quizais a descrición da mudanza debería ter un cabezallo "Description:" de acordo coas outras seccións. O importante é que toda a información necesaria para avaliar a mudanza estea dispoñible, e o mecanismo para votos decisivos sexa o máis lixeiro posible. A mudanza proposta está referenciada polo seu número de revisión no repositorio (neste caso unha simple revisión, r2401, embora unha mudanza proposta podería consistir en múltiples revisións). A revisión asúmese que está referida a unha mudanza feita no trunk; se a mudanza estivese na rama da versión, non habería que votar. Se o teu sistema de control de versións non ten unha sintaxe obvia para referirse a mudanzas individuais, o proxecto debería inventala. Para que o voto sexa práctico, cada mudanza baixo consideración debe ser identificable inequivocamente.

Esas proposicións ou votos para aprobar unha mudanza encárganse de asegurar de que se corresponde coa rama da versión, iso é, que se corresponde sen conflitos (ver "conflictos"). Se houber conflitos, entón a entrada debería apuntar a un parche que se corresponda limpamente, ou a unha rama temporal que teña a axustada versión da mudanza, por exemplo:

```
* r13222, r13223, r13232
  Rewrite libsvn_fs_fs's auto-merge algorithm
  Justification:
    unacceptable performance (>50 minutes for a small commit) in
    a repository with 300,000 revisions
  Branch:
    1.1.x-r13222@13517
  Votes:
    +1: epg, ghudson
```

Ese exemplo está tomado da vida real, vén do ficheiro *STATUS* no proceso de lanzamento da versión de Subversion 1.1.4. Nótese como se empregan as revisións orixinais coma manexadoras canónicas das mudanzas, mesmo hai unha rama cunha versión da mudanza axustada a conflitos (a rama ademais combina tres revisións do trunk nunha, r13517, para facer máis sinxela a integración das mudanzas na

versión, debería ser aprobada). As revisións orixinais aparecen porque se están a revisar aínda, xa que teñen as mensaxes de log orixinais. A rama temporal non tería esas mensaxes de log; para evitar a duplicación de información (ver sección “Singularidade da información” no capítulo “Infraestructura Técnica”), As mensaxes de log para a rama da revisión r13517 deberían dicir simplemente "Axustar r13222, r13223, r13232 para facer backport (aplicar parches a unha versión previa) da rama 1.1.x". Toda outra información acerca das mudanzas pode ser alcanzada nas súas revisións orixinais.

Manager das versións

O actual proceso de fusionar (ver “fusionar” no capítulo 3, “Infraestrutura Técnica”) as mudanzas aprobadas na rama da versión poden ser realizadas por calquera desenvolvedor. Aquí non se precisa unha persoa para facer o traballo de fusionar as mudanzas; se houber moitas mudanzas, pode ser mellor repartir a carga.

Porén, embora ámbolos dous, votación e fusionado, sucedan de forma descentralizada, na práctica normalmente hai unha ou dúas persoas conducindo o proceso de lanzamento da nova versión. Este rol e as veces bautizado como *release manager*, mais é un pouco diferente do propietario da versión (ver "Propietario da versión” máis atrás neste capítulo) quen ten a última palabra sobre as mudanzas. Os releases manager manteñen monitorizado cantas mudanzas están baixo consideración, cantas foron aprobadas, cantas parece que van ser aprobadas, etc. Se eles sentiren que algunhas mudanzas non están recibindo a atención precisa, e poden ficar fóra da versión por falta de votos, eles estarán encima de outros desenvolvedores para que voten. Cando unha manchea de mudanzas foren aprobadas, esta xente encargarase por si mesma de fusionalas na rama da versión; está ben se outros lles deixan esa tarefa a eles, sempre que todo o mundo entenda que eles non están obrigados a facer todo o traballo a non ser que eles explicitamente se prestasen a facelo. Cando a versión estiver a piques de saír (ver “Probas e publicacións das versións” máis adiante neste capitulo), os manager das versións preocuparanse tamén da loxística para a creación da versión, dos pacotes, de recoller as sinaturas dixitais, de subir os pacotes, e facer os anuncios públicos.

Empacotamento

A forma tradicional para a distribución do software libre é como código fonte. Isto é así dependendo de se o software normalmente se executa desde o código (por exemplo, podendo ser interpretado, coma Perl, Python, PHP, etc.) o precisa ser compilado primeiro (coma C, C++, Java, etc). Con software compilado, moitos usuarios probablemente non compilen as fontes eles mesmos, polo que precisarán instalalos desde pacotes binarios (ver “Pacotes binarios” máis adiante neste capitulo). Porén, eses pacotes binarios derivan das fontes da distribución. A cuestión é que as fontes son ambiguas para definir a versión. Cando o proxecto distribúe Scanley2.5.0", o que quere dicir, especificamente, é: "a árbore de ficheiros de código fonte, cando se compilaren (se for necesario) e instalaren, producen Scanley 2.5.0."

Hai un estándar bastante estrito sobre como deberían ser as versións das fontes. Ocasionalmente poden verse desviacións deste estándar, mais son a excepción, non a regra. A menos que houber una razón xustificada para facelo de outra forma, o teu proxecto debería seguir este estándar tamén.

Formato

O código fonte debería entregarse en formatos estándar para distribuír a árbore de directorios. Para Unix e sistemas operativos de tipo Unix, a convención é empregar o formato TAE, comprimido con `compress`, `gzip`, `bzip` ou `bzip2`. Para MS Windows, o método estándar para a distribución da árbore de directorios é o formato *zip*, o cal xa fai a compresión tamén, polo que non é necesario comprimir o ficheiro despois de crealo.

Ficheiros TAR

TAR abreviatura de "Tape ARchive", porque o formato tar representa unha árbore de directorios coma un xacto de datos lineais, o cal o torna ideal para gardar árbores de directorios en cinta. A mesma propiedade tórnao tamén o estándar para distribuír árbores de directorios nun ficheiro. Producir ficheiros tar comprimidos (ou *tarballs*) é bastante sinxelo. Nalgúns sistemas, o comando `tar` pode producir ficheiros comprimidos por si mesmo; noutros, hai que empregar outro programa para comprimir.

Nome e aspecto

O nome do pacote debería consistir no nome do software máis o número da versión, máis o formato de sufixos apropiado para o tipo de ficheiro. Por exemplo, Scanley 2.5.0, empacotado para Unix empregando GNU Zip (gzip) para comprimir, quedaría así:

scanley-2.5.0.tar.gz

ou para Windows usando compresión en zip:

scanley-2.5.0.zip

Algún destes ficheiros, cando son descomprimidos, deben crear unha nova árbore de directorios chamada *scanley-2.5.0* no directorio actual. Baixo o novo directorio, o código fonte debe estar preparado para a compilación (se a compilación for necesaria) e instalación. Na raíz do novo directorio, debería haber un ficheiro de texto plano *README* explicando o que fai o software e que versión é, e dar ligazóns a outros recursos, como a páxina web do proxecto, outros ficheiros interesantes, etc. Entre eses outros ficheiros debería haber un *INSTALL*, irmán do *README*, dando instrucións sobre como compilar e instalar o software para todos os sistemas operativos soportados. Como se mencionaba en “Como aplicar unha licenza ao teu código” no capítulo 2, “Primeiros pasos”, debería haber tamén un ficheiro *COPYING* coas cláusulas de distribución do software.

Debe haber tamén un ficheiro *CHANGES* (as veces chamado *NEWS*) coas novidades da versión. O ficheiro *CHANGES* acumula a lista de mudanzas para todas as versións, en orde cronolóxica inversa, para que a lista de mudanzas da versión apareza no principio do ficheiro. Completar esa lista é normalmente a última cousa que hai que facer para estabilizar a versión; algúns proxectos escriben a lista de vagar segundo se vai desenvolvendo, outros prefiren facelo ao final e ter una persoa para escribilos, recollendo información combinando os logs do control de versións. A lista é algo así:

```
Version 2.5.0
(20 December 2004, from /branches/2.5.x)
http://svn.scanley.org/repos/svn/tags/2.5.0/

 New features, enhancements:
    * Added regular expression queries (issue #53)
    * Added support for UTF-8 and UTF-16 documents
    * Documentation translated into Polish, Russian, Malagasy
    * ...

 Bugfixes:
    * fixed reindexing bug (issue #945)
    * fixed some query bugs (issues #815, #1007, #1008)
    * ...
```

A lista pode ser tan longa coma for necesario, mais non fai falta incluír cada pequeno erro resolto e funcionalidade. O seu propósito é simplemente darlle aos usuarios una visión sobre o que eles gañan actualizando á nova versión. De feito, a lista de mudanzas é incluída normalmente no email de anuncio (ver “Probas e publicacións de versións” máis adiante neste capitulo), polo tanto escríbea coa audiencia en mente.

CHANGES versus ChangeLog

Tradicionalmente, un ficheiro chamado *ChangeLog* que lista cada mudanza feita no proxecto isto é, cada revisión sobre a que se fixo commit no sistema de control de versións. Hai varios formatos para os ficheiros ChangeLog; os detalles dos formatos non son importantes aquí, xa que todos eles conteñen a mesma información: a data da mudanza, o autor, e un breve resumo (ou simplemente a mensaxe de log para esa mudanza).

O ficheiro *CHANGES* é diferente. Tamén é unha lista de mudanzas, mais só as que son importantes para que certa audiencia as vexa, e a miúdo con metadatos como a hora exacta e o autor. Para evitares confusións, non empregues termos intercambiables. Algúns proxectos empregan "NEWS" en lugar de "CHANGES"; aínda que isto evita a potencial confusión con "ChangeLog", é un termo un pouco errado xa que o ficheiro CHANGES ten información para todas as versións, e ten moitas noticias vellas ademais das novas.

Así e todo, os ficheiros ChangeLog poden estar desaparecendo. Eran de moita axuda cando CVS era a única opción de sistema de control de versións, porque a data non era sinxela de extraer. Porén, cos máis recentes sistemas de control de versións, a data que é empregada para manter no ChangeLog pode ser pedida desde o repositorio do sistema de control de versións en calquera momento, facendo que sea sinxelo para o proxecto manter un ficheiro estático que conteña a data de feito, o ChangeLog duplica as mensaxes de log que están no repositorio.

A distribución do código fonte dentro da árbore debería ser idéntica ou o máis similar posible á distribución do código fonte do proxecto que se obtería testando directamente desde o sistema de control de versións. Normalmente hai algunhas diferencias, por exemplo porque o pacote contén algúns ficheiros xerados necesarios para a configuración e compilación (ver “Compilación e instalación” máis adiante neste capitulo, ou porque inclúe software de terceiros non mantido polo proxecto, mais requirido e que os usuarios probablemente non teñen. Mais mesmo se a árbore de directorios coincide exactamente con algunha árbore de desenvolvemento no repositorio do sistema de control de versións, a distribución non debería ser unha copia de traballo (ver “copia de traballo” no capitulo 3,

“Infraestrutura Técnica”. A versión suponse que representa unha referencia estática en particular, unha configuración inmutable dos ficheiros fonte. Se for unha copia de traballo, o perigo sería que un usuario puidese actualizala,pensando que aínda ten a versión cando de feito ten algo distinto.

Recordar que o pacote é o mesmo embora o empacotamento. A versión iso é, a entidade precisa a que se remite cando alguén di "Scanley2.5.0" é a árbore de directorio creada pola descompresión do ficheiro zip ou tarball. Polo tanto o proxecto debe ofrecer todo isto para descarregar:

scanley-2.5.0.tar.bz2
scanley-2.5.0.tar.gz
scanley-2.5.0.zip

...mais a árbore de fontes creada polo desempacotamento debe ser a mesma. Esa árbore de fontes é a distribución; a forma na cal é descarregado é simplemente unha conveniencia. Certas triviais diferenzas entre os pacotes fontes son permisibles: por exemplo, no pacote de Windows, os ficheiros de texto deben ter liñas finais con CRLF (Carriage Return and Line Feed), mentres que os pacotes Unix deben empregar simplemente LF. As árbores poden ser ordenadas lixeiramente de forma diferente entre pacotes software destinados para diferentes sistemas operativos, tamén, se eses sistemas operativos requiriren diferentes ordenacións para a compilación. Porén, son transformación basicamente triviais. Os ficheiros fonte básicos deberían ser os mesmos en todos os pacotes dunha distribución.

Maiúsculas ou minúsculas?

Cando nos referimos a un proxecto polo nome, a xente normalmente o escribe en maiúsculas coma un nome propio, e escribe en maiúsculas os acrónimos se os houber: "MySQL5.0", "Scanley2.5.0", etc. Depende do proxecto que o nome do pacote se escriba tamén en maiúsculas. Tanto *Scanley-2.5.0.tar.gz* como *scanley-2.5.0.tar.gz* sería correcto, por exemplo (Eu persoalmente prefiro a última, porque non me gusta que a xente empregue a tecla shift, mais moitos proxectos escriben os pacotes en maiúsculas). O importante é que o directorio creado despois de desempacotar o tarball use a mesma grafía. Non debería haber sorpresas: o usuario debe ser capaz de predicir perfectamente o nome do directorio que será creado cando desempacote a distribución.

Pre-versións

Cando se saca unha pre-versión ou versión candidata, o cualificador é verdadeiramente parte do número de versión, polo tanto inclúese no nome do pacote. Por exemplo, a secuencia ordenada das versións alpha e beta vista antes en ”Compoñentes do número da versión” resultaría nos nomes de pacote:

scanley-2.3.0-alpha1.tar.gz
scanley-2.3.0-alpha2.tar.gz
scanley-2.3.0-beta1.tar.gz
scanley-2.3.0-beta2.tar.gz
scanley-2.3.0-beta3.tar.gz
scanley-2.3.0.tar.gz

O primeiro desempacotaríase nun directorio chamado *scanley-2.3.0-alpha1*, o segundo dentro *scanley-2.3-0-alpha2*, etc.

Compilación e instalación

Para software que require compilación ou instalación desde as fontes, hai normalmente varios procedementos estándar que usuarios experimentados esperan ser capaces de seguir. Por exemplo, para programas escritos en C, C++, ou outras certas linguaxes compiladas, o estándar baixo sistemas operativos tipo Unix é para o usuario dixitar:

```
$ ./configure
   $ make
   # make install
```

O primeiro comando tanto detecta o entorno como o pode preparar para o proceso de compilación, o segundo comando compila o software no sitio (mais non o instala), e o último comando instálao no sistema. Os dous primeiros comandos fanse como usuario normal, o terceiro como root. Para máis detalles a respecto da configuración deste sistema, le o excelente libro *GNU Autoconf, Automake, and Libtool* escrito por Vaughan, Eliston, Tromey e Taylor. Está publicado en forma de treeware por New Riders, e o seu contido está dispoñible gratis online en autobook (http://sources.redhat.com/autobook/).

Este non é o único estándar, mais é un dos máis estendidos. O sistema de compilación Ant (http://ant.apache.org) está gañando popularidade, especialmente nos proxectos escritos en Java, e ten os seus propios procesos de compilación e instalación. Ademais, certas linguaxes de programación como Perl e Python recomendan que ese mesmo método sexa empregado por máis programas escritos nesa linguaxe (por exemplo, os módulos Perl empregan o comando `perl Makefile.pl`). Se non é obvio para ti que estándar empregar no teu proxecto, pregunta a un desenvolvedor experimentado; podes asumir con seguridade que se aplica *algún* estándar, mesmo se non souberes cal é ao principio.

Sexa cal for o estándar apropiado para o teu proxecto, non te desvíes del a menos que debas facelo. Os procesos estándar de instalación son practicamente a columna vertebral para moitos administradores de sistemas. Se viren invocacións familiares documentadas no ficheiro *INSTALL* do teu proxecto, iso fará que instantaneamente incrementen a súa fe en que o teu proxecto segue as convencións, e que probablemente outras cousas tamén están ben. Ademais, como discutimos en “Descargas” no capitulo 2, “Primeiros pasos”, ter un proceso estándar de compilación comprace os potenciais desenvolvedores.

En Windows, os estándares para compilar e instalar están menos asentados. Para proxectos que requiren compilación, a convención xeral parece ser navegar por unha árbore que pode encaixar dentro do espazo de traballo/modelo do proxecto do entorno de desenvolvemento estándar de Microsoft (Developer Studio, Visual Studio, VS.NET, MSVC++, etc). Dependendo da natureza do teu software, pode ser posible ofrecer a opción dun sistema de compilación tipo Unix sobre Windows vía o entorno Cygwin (http://www.cygwin.com). E por suposto, se estás empregando unha linguaxe ou framework de programación que veña coa súa propia convención para compilar e instalar;por exemplo, Perl ou Python simplemente deberías usar o método estándar para ese framework, for en Windows, Unix, Mac OS X, ou calquera outro sistema operativo.

Anímate a faceres un esforzo extra para creares o teu proxecto conforme a un estándar de compilación ou instalación importante. A compilación e a instalación son un punto de entrada: é normal que as cousas que se tornen máis duras despois de iso, se deben, mais sería unha pena para os usuarios e

desenvolvedores que a primeira interacción co software requirise de pasos inesperados.

Pacotes binarios

Embora a liberación formal dunha versión sexa un pacote de código fonte, moitos usuarios instalarana desde pacotes binarios, fornecidos polo mecanismo de distribución de software do seu sistema operativo, ou obtido manualmente desde a páxina web do proxecto ou a través de terceiros. Aquí "binario" non necesariamente quere dicir "compilado"; simplemente quere dicir algunha forma de pacote que permite o usuario instalalo no computador sen pasar polo proceso de compilar e instalar as fontes. En RedHat GNU/Linux, é o sistema RPM; en Debian GNU/Linux, é o sistema APT (*.deb*); en MS Windows, xeralmente ficheiros *.MSI* ou auto-instalables *.exe*.

Independentemente de se estes pacotes binarios son empacotados por xente vinculada ao proxecto, ou por terceiros, os usuarios *trataranos* como se fosen oficiais, e avisarán de problemas no bug tracker do proxecto a respecto do comportamento deses pacotes binarios. Polo tanto, redunda no beneficio do proxecto fornecer guías claras para os empacotadores, e traballar con eles para ver se isto representa beneficios para o software.

O máis importante que os empacotadores precisan saber é que eles deben sempre basearse en versións oficiais para faceren os seus pacotes binarios. Algunhas veces os empacotadores teñen a tentación de facer mudanzas no código, ou incluír mudanzas sobre as que se fixo commit despois de estar feita a versión, para fornecer os usuarios de certas melloras. O empacotador pensa que lles está a facer un grande favor aos usuarios dándolles a última versión do código, mais de feito esta práctica causa unha grande confusión. Os proxectos están preparados para recibiren reportes de erros atopados en versións lanzadas, na rama principal e en recentes ramas (isto é, atopados por xente que deliberadamente executa este código). Cando un erro reportado vén destas fontes, o que responde a miúdo é capaz de confirmar se ese erro coñecido está presente nesa versión, e quizais xa foi arranxado e o usuario debería actualizar ou esperar a seguinte versión. Se for un erro descoñecido previamente, tendo a versión precisa fai máis sinxelo reproducilo e categorizalo.

Os proxectos non están preparados, porén, para recibir erros reportados baseados en versións intermedias non especificadas ou híbridas. Moitos erros poden ser difíciles de reproducir; ademais, poden ser debidos a inesperadas iteracións en mudanzas incorporadas máis tarde no desenvolvemento, e dese modo causar malos comportamentos dos que os desenvolvedores do proxecto poden non ter culpa. Eu teño visto con consternación perderse moito tempo por culpa dun erro *ausente* cando debería estar presente: alguén estaba executando unha versión remendada, baseada en (mais non idéntica) unha versión oficial, e cando o erro anunciado non sucedeu, todo o mundo tiña que escarvar arredor para adiviñar a causa.

Porén, haberá as veces circunstancias nas que un empacotador insistirá en que é necesario modificar o código da versión. Os empacotadores deberían ser animados a suxeriren isto aos desenvolvedores do proxecto e describiren os seus plans. Poden obter aprobación, mais se non, polo menos notificarán ao proxecto as súas intencións, polo que o proxecto pode observar informes de erros infrecuentes. Os desenvolvedores poden responder poñendo unha nota de descargo de responsabilidade na páxina web do proxecto, e pedirlle ao empacotador que faga o mesmo no lugar adecuado, para que os usuarios do pacote binario saiban que o que están obtendo non é exactamente o mesmo que a versión oficial do proxecto. Non hai que ter carraxe por esta situación, o que desafortunadamente pasa a miúdo. O que

pasa simplemente é que o empacotador ten obxectivos lixeiramente distintos aos desenvolvedores. Normalmente os empacotadores queren a mellor experiencia de serie posible para os seus usuarios. Os desenvolvedores queren iso tamén, por suposto, mais necesitan asegurarse tamén de que coñecen o que contén a versión para poderen recibir informes de erros coherentes e asegurar a compatibilidade. Algunhas veces os obxectivos entran en conflito. Cando isto pasa, é bo ter en mente que o proxecto non ten control sobre os empacotadores. É certo que o proxecto fai un favor aos empacotadores producindo o software. Mais os empacotadores están tamén facendo un favor ao proxecto, tornando o software dispoñible, a miúdo en ordes de magnitude maiores. Está ben estar en desacordo cos empacotadores, mais non entrar en conflito con eles, simplemente tenta facer as cousas o mellor que poidas.

Probas e publicación das versións

Unha vez que o tarball das fontes é producido desde a rama da versión estabilizada, comeza a parte pública do proceso de sacar a versión. Mais antes de que o tarball estea dispoñible para o mundo, debería ser testado e aprobado por un mínimo de desenvolvedores, normalmente tres ou máis. Aprobado non é simplemente inspeccionar a versión en busca de simples defectos; idealmente, os desenvolvedores baixan o tarball, compílano e instálano nun sistema limpo, lanzan os test de regresión (ver "Testaxe automatizada" no capitulo 8 “Xestionando voluntarios”, e fan algo de test manual. Asumindo que supera eses tests, como calquera outro que a versión pode ter a criterio do proxecto, os desenvolvedores asinan o tarball empregando GnuPG (http://www.gnupg.org/), PGP (http://www.pgpi.org/), ou algún outro programa que produza sinaturas compatibles con PGP.

En moitos proxectos, os desenvolvedores simplemente usan as súas sinaturas dixitais, no canto de compartiren a chave do proxecto, e moitos desenvolvedores queren poder asinar (por exemplo, hai un número mínimo, mais non máximo). Cantos máis desenvolvedores asinen, máis testaxe se fai na versión, e ademais máis grande a posibilidade de que un usuario preocupado pola seguridade pode atopar un camiño dixitalmente seguro entre el e o tarball.

Unha vez aprobada, a versión (isto é, todos os tarballs, ficheiros zip, e calquera outros formatos nos que for distribuída) deberían poñerse na zona de descargas dos proxectos, acompañada das sinaturas dixitais e o MD5/SHA1 checksums (ver http://en.wikipedia.org/wiki/Cryptographic_hash_function"). Hai varios estándares para facer isto. Unha forma é acompañar a cada pacote lanzado cun ficheiro que contén as correspondencias das sinaturas dixitais, e outro ficheiro co checksum. Por exemplo, se un dos pacotes lanzados for *scanley-2.5.0.tar.gz*, situaríase no mesmo directorio un ficheiro *scanley-2.5.0.tar.gz.asc* coa sinatura dixital para o tarball, outro ficheiro *scanley-2.5.0.tar.gz.mdz* co MD5 checksum, e opcionalmente outro, *scanley-2.5.0.tar.gz.sha1*, co SHA1 checksum. Unha forma diferente de facelo sería poñendo todas a sinaturas para os pacotes lanzados nun único ficheiro, *scanley-2.5.0.sigs*; o mesmo cos checksums.

Non importa realmente a forma de facelo. Simplemente segue un esquema simple, descrito claramente, e coherente de versión a versión. O propósito de todo estas sinaturas e checksumming é darlle aos usuarios unha forma de verificaren que a copia que reciben non ten nada malicioso. Os usuarios executan ese código na súa computadora se o código foi manipulado, un atacante podería ter unha porta de atrás para acceder a todos os datos. Ver "Versións seguras" máis adiante neste capítulo para máis información sobre paranoias.

Versións candidatas

Para versións importantes que conteñen moitas mudanzas, moitos proxectos prefiren sacar *versións candidatas* primeiro, p. ex., *scanley-2.5.0-beta1* antes de *scanley-2.5.0*. O propósito dunha candidata é probar ese código antes de darlle o status de versión oficial. Se se atopan problemas, son arranxados na rama da versión e unha nova versión candidata é estendida (*scanley-2.5.0-beta2*). O ciclo continúa ata que non fican erros inaceptables, neste punto a última versión candidata convértese na versión oficial iso é, a única diferenza entre a última versión candidata e a versión real é a falta do cualificador do número de versión.

En moitos outros aspectos, unha versión candidata debería ser tratada como a versión real. O cualificador *alpha*, *beta* ou *rc* é suficiente para advertir a usuarios conservadores que esperen ata a versión real, e por suposto o email de anuncio para versións candidatas debería apuntar que o seu propósito é solicitar feedback. Outra cousa, dá ás versións candidatas a mesma atención que ás regulares. Despois de todo, queres que a xente empregue as candidatas, porque é o mellor xeito de descubrir erros, e ademais porque nunca sabes que versión candidata se converterá na versión oficial.

Anunciando as versións

Anunciar unha versión é como anunciar calquera outro evento, e debería empregar os procesos descritos na sección “Publicidade2 no capitulo 6, “Comunicacións” . Hai algunhas cousas que facer especificamente para a versión.

Cando deas a URL para descarregar o tarball da versión, asegúrate tamén de dares o MD5/SHA1 checksums e indicacións sobre o ficheiro da sinatura dixital. Xa que o anuncio ten lugar en múltiples foros (listas de correo, páxinas de novas, etc.), os usuarios poden obter o checksum de varias fontes, o cal lles dá a máxima seguridade de que os checksums nos foron alterados. Dar a ligazón aos ficheiros de firma dixital múltiples veces non fai esas sinaturas máis seguras, mais tranquiliza a xente (especialmente aqueles que non seguen o proxecto de preto) de que o proxecto se toma a seguridade en serio.

Nese email de anuncio, e nas páxinas de noticias que conteñen máis que propaganda sobre a versión, asegúrate de incluíres a porción relevante do ficheiro CHANGES, para que a xente poida ver porque lles podería interesar a actualización. Isto é importante con versións candidatas e con versións finais; a presenza de erros arranxados e novas funcionalidades é importante e tentador para a xente para probaren a versión candidata.

Finalmente, non esquezas agradecer o equipo de desenvolvemento, os probadores, e toda a xente que empregou tempo para facer bos informes de erros. Non empregues nomes, a non ser que houber alguén individualmente responsable dalgunha peza grande de traballo. Simplemente sé cauteloso para non caeres na inflación do crédito (ver “credito” no capitulo 8, “Xestionando voluntarios”).

Mantendo múltiples liñas de versións

O proxectos máis maduros manteñen varias liñas de versións en paralelo. Por exemplo, despois de que

saia a 1.0.0, esa liña debería continuar con micro versións 1.0.1, 1.0.2, etc., ata que o proxecto explicitamente decida o fin da liña. Nótese que lanzar a 1.1.0 non é razón suficiente para finalizar a liña 1.0.x. Por exemplo, moitos usuarios manteñen a política de non actualizaren a primeira versión cun novo número menor ou maior na serie permiten que outros se sacudan os bugs, da 1.1.0, e esperar ata a 1.1.1. Isto non é necesariamente egoísta (recorda, están renunciando a erros amañados e novas funcionalidades tamén); é simplemente iso, por calquera razón, eles decidiron ser coidadosos coas actualizacións. Consecuentemente, se o proxecto se decatar dun erro grave na 1.0.3 xusto antes de sacar a versión 1.1.0, debería ser un pouco severo e poñer o amaño na 1.1.0 e dicirlle a tódolos usuarios da 1.0.x que deberían actualizar. Por que non sacar ambas 1.1.0 1.0.4, para que todo o mundo sexa feliz?

Despois de que a liña 1.1.x estea ben avanzada, podes declarar a 1.0.x como *finalizada*. Isto debería ser anunciado oficialmente. O anuncio podería ser independente ou podería ser mencionado como parte do anuncio da versión 1.1.x; porén, faino, os usuarios precisan saber que a liña vella está desfasada, para que eles poidan tomar a decisión de actualizar.

Algúns proxectos poñen un prazo durante o cal prometen dar soporte a liña previa da versión. No contexto do software libre, "soporte" quere dicir aceptar informes de erros sobre esa liña, e facer versións de mantemento cando se atoparen erros importantes. Outros proxectos non poñen un límite definitivo, mais seguen aceptando informes de erros para calcular canta xente está empregando a liña vella. Cando a porcentaxe atinxe certo punto, declaran a finalización desa liña e do soporte.

Para cada versión, asegúrate de teres unha *versión obxectivo* ou *fito* dispoñible no notificador de erros, para que a xente poida informar de erros nesa versión. Non esquezas ter tamén un obxectivo chamado "desenvolvemento" ou "última" para as fontes de desenvolvemento máis recentes, xa que algunha xente non só desenvolvedores activos; están atentos as versión oficias.

Versións de seguridade

A maioría dos detalles para manexar erros de seguridade están cubertos na sección “Seguridade” no capitulo 6, “Comunicacións”, mais hai algúns detalles especiais que discutir para facer versións de seguridade.

Unha *versión de seguridade* é unha versión feita para pechar unha vulnerabilidade de seguridade. O código que arranxa ese erro pode non facerse público ata que estiver dispoñible a versión. O cal significa que non só as solucións non poden ser engadidas ao repositorio ata o día da liberación, senón que non se pode publicar e ser testada antes de estar fóra. Obviamente, os desenvolvedores poden examinar o amaño entre eles, e testar a versión privadamente, mais sacala para ser testado publicamente non é posible.

Debido a esta falta de testaxe, unha versión de seguridade debería sempre consistir dunha versión existente máis os arranxos para o erro de seguridade, sen *ningunha outra mudanza*. Isto é porque cantas máis mudanzas metes sen testar, máis probabilidades hai de que algún cause un novo erro, quizais mesmo un novo erro de seguridade!. Este conservadorismo é práctico para os administradores que poidan precisar implementar o parche de seguridade, mais cuxa política de actualizacións pode recomendar que non se implementen máis mudanzas simultaneamente.

Facer versións de seguridade as veces supón algunha pequena decepción. Por exemplo, o proxecto

pode estar traballando na versión 1.1.3, con certos erros amañados da 1.1.2, cando un informe de seguridade aparece. Naturalmente, os desenvolvedores non poden falar do problema de seguridade ata que estiver amañado; teñen que seguir falando publicamente de que os seus plans son sacar a 1.1.3. Mais cando se vai a sacar a 1.1.3, só vai ser diferente da 1.1.2 no amaño de seguridade, a todos os demais erros amañados se deixan para a 1.1.4 (a que, por suposto terá *tamén* o amaño de seguridade, igual que o resto de futuras versións).

Podes engadir unha compoñente extra a unha versión existente para indicares que contén mudanzas só por seguridade. Por exemplo, a xente podería ser capaz de dicir só polos números que 1.1.2.1 é unha versión de seguridade e non 1.1.2, e saberían que calquera versión "maior" que esa (e.x., 1.1.3, 1.2.0, etc.) contén os mesmos amaños de seguridade. Para eles, este sistema expresa moita información. Por outra banda, para aqueles que non seguen de preto o proxecto, pode ser un pouco confuso ver unha terceira compoñente no número da versión con ocasionalmente un cuarto compoñente aparentemente aleatorio. Moitos proxectos simplemente empregan o seguinte numero programado para as versións de seguridade, mesmo cando implica mudar os plans da versión.

Versións e desenvolvemento diario

Manter versións paralelas simultaneamente ten implicacións para como se fai o desenvolvemento diario. En particular, torna practicamente obrigatoria unha disciplina que sería recomendable de calquera forma: ter cada commit como unha simple mudanza lóxica, e nunca mesturar mudanzas non relacionadas no mesmo commit.

Aquí hai un exemplo de commit mal concibido:

```
------------------------------------------------------------------------
r6228 | jrandom | 2004-06-30 22:13:07 -0500 (Wed, 30 Jun 2004) | 8 lines

Fix Issue #1729: Make indexing gracefully warn the user when a file
is changing as it is being indexed.

* ui/repl.py
  (ChangingFile): New exception class.
  (DoIndex): Handle new exception.

* indexer/index.py
  (FollowStream): Raise new exception if file changes during indexing.
  (BuildDir): Unrelatedly, remove some obsolete comments, reformat
  some code, and fix the error check when creating a directory.

Other unrelated cleanups:

* www/index.html: Fix some typos, set next release date.
------------------------------------------------------------------------
```

O problema aparece tan pronto como alguén necesita portar o arranxo para comprobar os erros na función *BuildDir* a outra rama para unha versión de mantemento. O portador non quere ningunha das outros mudanzas por exemplo, quizais o arranxo para o problema #1729 non foi de todo aprobado para a rama de mantemento, e o ficheiro *index.html* sería simplemente irrelevante alí. Mais non pode facilmente coller xusto a mudanza *BuildDir* a través da ferramenta de integración do sistema de control de versións xa que a mudanza está agrupada coas outras cousas non relacionadas. De feito, o problema aparecería mesmo antes da integración. Simplemente someter a mudanza a votación sería problemático: en lugar de dar o número de revisión, o que o propoño tería que facer un parche especial ou mudar a rama para isolar a porción do commit proposto. Iso sería un montón de traballo sufrido por

outros, e todo porque o commiter orixinal nun puido romper as cousas en grupos lóxicos.

De feito, ese commit realmente deberían ter sido *catro* commits separados: un para resolver o problema #1729, outro para eliminar comentarios obsoletos e reformatar o código en *BuildDir*, outro para arranxar o erro en *BuildDir*, e finalmente, un para o *index.html*. O terceiro deses commits seria o proposto para a rama de mantemento da versión.

Por suposto, estabilizar unha versión non é a única razón de por que é desexable que cada commit sexa unha mudanza lóxica. Psicoloxicamente, un commit semanticamente unificado é fácil de revisar, e fácil de reverter se for necesario (nalgúns sistemas de control de versións, reversión é realmente unha especie de fusión). Un pouco de disciplina por parte de todo o mundo pode evitarlle unha dor de cabeza ao proxecto a longo prazo.

Planeando as versións

Unha área onde os proxectos de software libre son diferentes historicamente dos propietarios é nos plans de lanzamento de versións. Os proxectos propietarios normalmente teñen datas límite firmes. Ás veces porque se lles prometeu aos clientes que unha actualización estaría dispoñible nunha data concreta, porque a nova versión precisa coordinarse con algún outro esforzo por propósitos de marketing, ou porque os ousados capitalistas que invisten en todo precisan ver algúns resultados antes de arriscaren os seus cartos para financiaren algo máis. Os proxectos de software libre, por outra banda, foron ata hai pouco máis motivados polo amadorismo no senso máis literal: foron escritos por amor á arte. Ninguén sentía a necesidade de facer os lanzamentos antes de todas as funcionalidades estiveren listas, e por que deberían? Non era como se o traballo de alguén estivese en xogo.

Hoxe en día, a maioría dos proxectos de software libre son fundados por corporacións, e cada vez están máis influenciados pola cultura das datas límite. Isto é en certa forma algo bo, mais pode causar conflitos entre as prioridades dos desenvolvedores que son pagados e quen o fai como voluntario. Estes conflitos a miúdo suceden cando se trata de deseñar o esquema de lanzamento de versións. Os desenvolvedores pagados que son os que están baixo presión queren establecer unha data para lanzar a versión, e que teñen as actividades de todo o mundo se preguen á súa vontade. Mais os voluntarios poden ter outras axendas quizais funcionalidades que queren completar, ou algún test que queren realizar eles senten que o lanzamento da versión debería esperar.

Non hai unha solución xeral para este problema excepto a discusión e o compromiso, por suposto. Mais non podes minimizar a frecuencia e o nivel de fricción causado, polo desacoplamento da proposta *existente* dunha versión dada desde a data cando sairía. Isto é, tratar de levar a discusión cara a que versións do proxecto se farán nun futuro próximo, e que funcionalidades irán en cada unha, sen ningunha mención sobre datas, excepto con grandes marxes de erro[16]. E sobre o uso do tempo baseado en procesos de lanzamento de versións, coma contraposición ás funcionalidades, e moitos proxectos de software. Michlmayr ademais deu unha charla en Google sobre o tema, dispoñible a través de Google Video en http://video.google.com/videoplay?docid=-5503858974016723264 . Dando logo un set de funcionalidades, reduces a complexidade da discusión centrada nunha versión individual, e por tanto melloras a previsibilidade. Isto ademais crea unha certa inercia contra alguén que propoñer expandir a

16Para unha aproximación alternativa, podes ler a tese de Martin Michlmayr's Ph.D. Quality Improvement in Volunteer Free and Open Source Software Projects: Exploring the Impact of Release Management (http://www.cyrius.com/publications/michlmayr-phd.html)

definición da versión engadindo novas funcionalidades ou complicacións. Se os contidos da versión están máis ou menos ben definidos, a responsabilidade de xustificar a expansión vai ser de quen a propón, mesmo se a data de lanzamento non foi posta aínda.

Na biografía de Thomas Jefferson, de varios volumes, *Jefferson and His Time*, Dumas Malone conta a historia de como Jefferson xestionou a primeira reunión que tivo para decidir como organizar o futuro da Universidade de Virxinia. A Universidade estivera no primeiro lugar na mente de Jefferson, mais (como pasa en todas partes, non só nos proxectos de software libre) moitos outros grupos subiron ao carro rapidamente, cada un cos seus propios intereses e axendas. Cando se reuniron por primeira vez para determinaren as cousas, Jefferson asegurouse de mostrar con meticulosidade planos arquitectónicos preparados, orzamentos detallados para construción e xestión, unha proposta de curriculum, e de nomes de facultades específicas que quería importar de Europa. Ninguén máis na sala estaba remotamente preparado; o grupo esencialmente tivo que capitular ante a visión de Jefferson, e a Universidade foi máis ou menos fundada segundo os seus plans. Os feito de que a construción ficara por riba do orzamento, e que moitas ideas non se fixeran, por varias razóns, eran cousas que probablemente Jefferson sabía que ían acontecer. O seu propósito era estratéxico: mostrar na reunión algo tan detallado que ninguén puidera propoñer modificacións, e que polo tanto as liñas, e o esquema, do proxecto fora coma el quería.

No caso dun proxecto de software libre, non hai unha soa "reunión", no canto diso hai unha serie de pequenas propostas polo seguimento do problema. Mais se tiveres certa credibilidade no proxecto para empezar, e comezares asignando varias funcionalidades, melloras, e erros ás versións obxectivo, de acordo co anunciado no plan xeral, a maioría da xente estará contigo. Unha vez tiveres as cousas ordenadas máis ou menos como quixeres, as conversas sobre as *datas* da versión actual serán máis suaves.

É crucial, por suposto, non presentar nunca unha decisión individual como se estiver escrita en pedra. Nos comentarios asociados a cada asignación de temas a unha versión futura, invita á discusión, a disentir, e a estar xenuinamente disposto a ser persuadido na medida do posible. Nunca exerzas control simplemente co fin de exerceres control: Canto máis participen os demais no plan de lanzamento das versións (ver sección “Xestión compartida” no capitulo 8, “Xestionando voluntarios”), máis sinxelo será persuadilos para compartiren as túas prioridades nos asuntos que realmente teñen importancia para ti.

A outra forma de conseguir que haxa menos tensión arredor do proceso de lanzamento de versións é facer versións a miúdo. Cando pasa moito tempo entre versións, a importancia de cada versión individual é magnificada pola mente de todos; a xente indígnase máis cando o seu código non vai nunha versión, xa que saben que pasará bastante tempo ata que teñan outra oportunidade. Dependendo da complexidade do proceso de lanzamento de versións e a natureza do proxecto, un período entre 3 e 6 meses soe ser o adecuado entre versións, ademais as liñas de mantemento poden facer micro versións máis rápido, se houber demanda.

Capítulo 8. Xestionando voluntarios

Conseguir que a xente se poña de acordo sobre o que un proxecto precisa, e que traballe xunta para acadalo, require de máis que unha atmosfera xenial e sen disfuncións obvias. Require unha, ou varias, persoas que xestionen concienciudamente toda a xente involucrada. Xestionar voluntarios pode non ser unha arte técnica no mesmo senso que a programación, mais é unha arte no senso de que pode

mellorarse a través do estudo e da práctica.

Este capítulo é un caixón de xastre con técnicas específicas para xestionar voluntarios. Aliméntase como caso de estudo, quizais dun xeito aínda máis forte que en capítulos anteriores, do proxecto Subversion, en parte porque estiven traballando nese proxecto mentres escribía este libro (o que facía que tivera as fontes primarias preto da man), e tamén en parte porque é máis aceptable tirar pedras ao propio tellado que non ao dos demais. Aínda así. tamén teño visto noutros proxectos os beneficios de aplicar -e as consecuencias de non aplicar- as recomendacións que seguen. Sempre que for politicamente correcto fornecer exemplos derivados destes outros proxectos, fareino.

Falando de política, este é un momento tan bo como calquera outro para inspeccionarmos a devandita palabra maligna. A moitos enxeñeiros gústalles pensar que a política é algo no que está envolvida outra xente, ao xeito de "*Eu* estou avogando polo mellor camiño para o proxecto, mais *Ela* pon atrancos por razóns políticas." Creo que este distanciamento derivado da política (ou derivado do que se pensa que é a política) é especialmente forte nos enxeñeiros debido a que os enxeñeiros edúcanse coa idea de que algunhas solucións son obxectivamente superiores a outras. Por tanto, cando alguén actúa dun xeito que parece motivado por consideracións externas -o mantemento da súa posición de influencia, minar a influencia dun terceiro, "vender motos", ou evitar ferir os sentimentos de alguén- outros integrantes do proxecto poden anoxarse. Por suposto, isto raramente evita que se comporten do mesmo xeito cando son os seus propios intereses os que están en xogo.

Se consideras que "política" é unha palabra porca, e tes esperanzas de manter o teu proxecto libre dela, abandona agora mesmo. A política é algo que aparece inevitablemente onde a xente teña que xestionar cooperativamente un recurso compartido. É absolutamente racional que unha das consideracións que se teñan en conta durante o proceso de toma de decisións de cada persoa sexa a cuestión de como unha acción dada pode afectarlle á súa futura influencia no proxecto. Despois de todo, se cres no teu propio xuízo e capacidades, e moitos programadores fano, a potencial perda futura de influencia ten que ser considerada como unha cuestión técnica, en certo senso. Un razoamento similar é aplicable a outros comportamentos que poden ser vistos, superficialmente, como "pura" política. De feito, non hai nada como a tal pura política: é precisamente debido a que as accións teñen múltiples consecuencias no mundo real polo que a xente chega a ser politicamente consciente nun primeiro momento. A política é simplemente, ao final, un recoñecemento de que *todas* as consecuencias das decisións deben ser tidas en conta. Se unha decisión concreta conduce a un resultado que a maioría dos participantes atopan tecnicamente satisfactorio, mais implica unha mudanza nas relacións de poder que provoca o isolamento de xente clave, entón debemos telo en conta e darlle importancia a ambas consecuencias. Ignoralo non só sería pouco xuizoso, senón tamén curto de miras.

Así, mentres leas os consellos que seguen, e mentres traballes no teu propio proxecto, lembra que *ninguén* está por enriba da política. Amosarse como alguén por enriba da política é só unha mera estratexia política, sendo moi útil ás veces, mais nunca é a realidade. A política é simplemente o que pasa cando a xente está en desacordo, e os proxectos exitosos son aqueles que desenvolven os mecanismos políticos para xestionar construtivamente o desacordo.

Conseguir o máximo de voluntarios

Por que os voluntarios traballan en proxectos de software libre[17]?

17 Esta cuestión foi estudada en detalle, con resultados interesantes, nun estudo de Karim Lakhani e

Cando lles preguntas, moitos expoñen que o fan porque queren producir bo software, ou queren participar persoalmente na reparación dos erros que lles afectan. Mais estas razóns son habitualmente só unha parte da historia. Despois de todo, podes imaxinar un voluntario traballando nun proxecto mesmo se nunca ninguén lle dirixiu unha palabra de aprecio sobre o seu traballo no proxecto, ou escoitou a súa opinión nunha discusión? Claro que non. Claramente, a xente emprega tempo no software libre por razóns que van para alén do desexo abstracto de producir bo código. Comprender as motivacións reais dos voluntarios vaiche axudar a arranxar as cousas de xeito que che permita atraelos e mantelos. O desexo de producir bo software pode ir no medio desas motivacións, da man do desafío e do valor educativo de traballar en problemas difíciles. Mais os humanos tamén temos un desexo intrínseco de traballar con outros humanos, e de dar e gañar respecto a través de actividades cooperativas. Os grupos involucrados en actividades cooperativas deben desenvolver normas de comportamento de xeito que o status sexa adquirido e mantido a través de accións que axuden a obter os obxectivos do grupo.

Estas normas non sempre xorden por si soas. Por exemplo, nalgúns proxectos -os desenvolvedores experimentados de software libre probablemente poidan nomear varios de memoria- a xente aparentemente sente que o status é adquirido mediante o envío de mensaxes frecuentes e longas. Habitualmente non chegan a esta conclusión dun xeito accidental, senón que chegan a ela porque son recompensados con respecto por faceren longas e intricadas argumentacións, axúdenlle ou non ao proxecto. A continuación amósanse algunhas técnicas apropiadas para xerar unha atmosfera na que as accións que permitan adquirir status tamén sexan construtivas.

Delegación

Delegar non é simplemente un xeito de repartir a carga de traballo; tamén é unha ferramenta política e social. Considera todos os efectos cando lle pides a alguén que faga algo. O efecto máis obvio é que, se acepta, el fai a tarefa e ti non. Mais outro efecto é que se decata de que ti confías nel para manexar a tarefa. Máis aínda, se fixeres a petición nun foro público, entón tamén saberá que outras persoas do grupo tamén se decataron desa confianza. Tamén é posible que se sinta presionado para aceptar, o que quere dicir que debes facerlle a petición de xeito que lle permitas declinala dun xeito apropiado se realmente non quere ese traballo. Se a tarefa require coordinación con outra xente do proxecto, entón tamén lle estás pedindo que participe dun xeito maior no proxecto, creando lazos que doutro xeito non se terían formado, e quizais que se converta nunha fonte de autoridade nalgún subdominio do proxecto. A participación adicional pode ser asoballante, ou pode conducilo a participar doutros xeitos a maiores, grazas a un renovado sentimento de compromiso.

Debido a todos estes efectos, a miúdo ten sentido pedirlle a alguén que faga algo aínda que saibas que ti poderías facelo máis rápido ou mellor. Por suposto, ás veces de tódolos xeitos hai un argumento de estrita eficiencia económica: pode que o custe de oportunidade de facelo por ti mesmo sexa demasiado alto -pode haber algo mesmo máis importante que poidas facer con ese tempo que aforras. Mais mesmo cando o argumento do custe de oportunidade non for relevante, *aínda así* podes queres pedirlle a alguén que faga unha tarefa concreta debido a que no longo prazo queres que esa persoa participe máis

---

Robert G. Wolf titulado *Why Hackers Do What They Do: Understanding Motivation and Effort in Free/Open Source Software Projects* (Por que os Hackers fan o que fan: entendendo a motivación e o esforzo en proxectos de software libre e de código aberto). Bótalle unha ollada a http://freesoftware.mit.edu/papers/lakhaniwolf.pdf

profundamente no proxecto, mesmo se iso significa empregar un tempo extra en supervisar o seu traballo ao comezo. A técnica oposta tamén funciona: se ocasionalmente fas voluntariamente o traballo que outra persoa non quere ou non ten tempo para facer, vas gañar a súa simpatía e respecto. A delegación e a substitución non teñen a ver unicamente con conseguir que tarefas individuais sexan levadas a cabo; tamén teñen a ver con aumentar o grao de compromiso da xente co proxecto.

Distingue claramente entre solicitar e asignar

En ocasións é lóxico ter a expectativa de que unha persoa vaia aceptar unha tarefa concreta. Por exemplo, se alguén insire un erro dentro do código, ou fai *commit* de código que incumpre dun xeito obvio as directrices do proxecto, entón debería ser suficiente con sinalar o problema para asumir que a persoa implicada se fará cargo do mesmo. Mais hai outras situacións nas que non está claro que as expectativas sexan correctas. A persoa pode facer o que lle pides, ou non. Como a ninguén lle gusta que se dea por feito que vai obedecer, debes ter a sensibilidade de distinguires entre estes dous tipos de situacións, e formular a túa petición dun xeito acorde á situación que se presentar.

Algo que con frecuencia senta mal é que lle requiras a alguén facer algo dun xeito que implique que pensas que claramente é a súa responsabilidade facelo, cando el non o pensa do mesmo xeito. Por exemplo, a asignación de incidencias entrantes é frecuentemente un terreo fértil para este tipo de cabreos. Os participantes dun proxecto normalmente saben quen é experto en que áreas, de tal xeito que cando se recibe unha notificación de erro, haberá unha ou dúas persoas que todo o mundo saiba que probablemente poidan solucionalo rapidamente. Aínda así, se asignares a incidencia a unha destas persoas sen o seu previo consentimento, podería sentirse posta nunha situación incómoda, dado que sente a presión das expectativas e tamén podería sentirse penalizada polo seu coñecemento e experiencia. Despois de todo, o xeito no que un adquire experiencia é mediante a resolución de erros, polo que quizais algún outro membro do grupo podería resolver este! (Ten en conta que os sistemas de notificación de erros que automaticamente asignan incidencias a persoas concretas, en base á información presente no sistema, acostuma a ser menos ofensivos debido a que todo o mundo sabe que a asignación foi feita por un proceso automático, e non é un indicativo de expectativas humanas).

Aínda que o desexable sexa repartir a carga o máis equitativamente posible, hai algúns momentos concretos nos que simplemente queres envolver a persoa que pode arranxar o erro o máis axiña posible. Dado que non podes afrontar un diálogo en quendas para cada asignación ("Importaríache botarlle unha ollada a este erro?" "Si." "De acordo. Daquela vou asignarche a incidencia" "De acordo"), o máis sinxelo é que fagas a asignación na forma dunha solicitude, sen exercer presión. Virtualmente todos os sistemas de notificación de erros permiten asociar un comentario coa asignación das incidencias. Nese comentario, podes dicir algo como o seguinte:

> *Tarefa asignada para ti, X, debido a que es o que está máis familiarizado con este código. Séntete libre de devolveres a asignación se non tiveres tempo de revisala. (E faime saber se prefires non recibir este tipo de solicitudes no futuro).*

Así distínguese claramente a *solicitude* da tarefa da *aceptación* da mesma. A audiencia nestes procesos non é unicamente o delegado, senón todo o mundo; o grupo enteiro observa unha confirmación pública do coñecemento e experiencia do delegado, mais a mensaxe tamén deixa claro que o delegado é libre de aceptar ou declinar a responsabilidade.

Persevera no delegado

Cando lle pedires a alguén que faga algo, lembra que o fixeches, e persevera pase o que pasar. A maioría das solicitudes son feitas en foros públicos, máis ou menos do seguinte xeito: "Poderías encargarte de X? Dinos algo en calquera caso; non hai problema se non podes, só fáinolo saber". Podes recibir ou non unha resposta. Se a recibires, e a resposta é negativa, o círculo péchase -precisarás intentar algunha outra estratexia para emparellalo con X. Se houber unha resposta positiva, entón non perdas de vista o progreso da incidencia, e comenta o progreso que vexas ou que non vexas (todo o mundo traballa mellor cando sabe que alguén aprecia o seu traballo). Se non houber resposta despois dun poucos días, volve a preguntar, ou envía unha mensaxe dicindo que no recibiches resposta e que estades a procurar algún outro que o faga. Ou simplemente faino ti mesmo. Mais asegúrate de dicires que non recibiches resposta á solicitude inicial.

O motivo de amosares publicamente a falta de resposta *non* é humillar á persoa á que se lle fixo a petición, e cando comentes o tema debes facelo de xeito que isto quede claro. O motivo é simplemente amosar que segues a pista daquilo que pides, e que notificas as reaccións que obtés. Esta forma de traballo torna máis probable que alguén che diga si a seguinte vez, porque van decatarse (aínda que só sexa de xeito inconsciente) de que vas apreciar o traballo que fagan, dado que prestas atención a detalles moito menos visibles como que alguén non responda unha solicitude.

Fíxate no que lle interesa á xente

Outra cousa que fai feliz á xente é que te fixes nos seus intereses -en xeral, cantos máis aspectos da personalidade de alguén teñas presentes e lembres, máis cómodo vai estar este alguén, e máis vai querer traballar en grupos dos que ti formes parte.

Por exemplo, había unha nítida diferenza no proxecto Subversion entre a xente que quería chegar a unha versión 1.0 (cousa que finalmente fixemos), e a xente que principalmente quería engadir novas funcionalidades e traballar en problemas interesantes sen importarlles moito cando iamos lanzar a versión 1.0. Ningunha destas dúas posturas é mellor ou peor que a outra; simplemente son dous tipos distintos de desenvolvedores, e ambos tipos levan a cabo moito traballo no proxecto. Mais aprendemos rapidamente que era moi importante *non* asumir que o interese por ter unha versión 1.0 era compartido por todos. As comunicacións electrónicas poden enganar; podes pensar que hai unha atmosfera de propósito común cando, realmente, dito propósito só é compartido pola xente coa que ti tes falado, mentres outras persoas teñen prioridades completamente distintas.

Canto máis consciente fores do que a xente quere obter do proxecto, máis eficiente e efectivamente poderás facerlles solicitudes. Simplemente demostrares comprensión do que queren, sen faceres ningunha solicitude asociada, xa é útil, dado que lle confirma a cada persoa que non é simplemente unha partícula máis nunha masa amorfa.

Louvanzas e críticas

Louvanzas e críticas non son antónimos; de feito son moi similares en varias facetas. Ambos son formas de atención primarias, e son máis efectivos cando son específicos que cando son xenéricos.

Ambos deben ser empregados con obxectivos concretos na mente. Ambos poden diluírse por uso en exceso: louva demasiado ou demasiado pouco e restaraslle forza ás túas louvanzas; o mesmo para o caso das críticas, aínda que na práctica a crítica acostuma a ser reactiva e por tanto un pouco máis resistente a perder forza.

Un cuestión interesante da cultura tecnolóxica é que a detallada e desapaixonada crítica a miúdo tómase como una especie de louvanza (como se viu na sección "*Recoñecendo a rudeza*" no *Capítulo 6 Comunicacións*), isto débese á implicación subxacente de que o traballo que se está criticando vale suficientemente a pena como para ser analizado polo miúdo. En calquera caso, ambos aspectos; *detallada* e *desapaixonada* deben cumprirse para que esta afirmación sexa verdade. Por exemplo, se alguén fixer algunha mudanza chapuceira no código, é inútil (e de feito é contraproducente) comentares o asunto simplemente dicindo "Fixeches unha chapuza". Ser un chapuzas é, ao final, unha característica da *persoa*, non do seu traballo, e é importante acoutar a túa atención e opinión ao traballo feito. É moito máis eficiente describires todas as cousas mal feitas que se introduciron ao facer a mudanza, e hai que facelo con tacto e sen malicia. Se esta for a terceira ou cuarta mudanza descoidada da mesma persoa, entón o máis apropiado sería mencionares o feito, despois da crítica sobre o traballo realizado, e de novo sen ningún signo de enfado, para así deixares claro que dito padrón de comportamento foi advertido.

Se alguén non mellorar tras as críticas, a solución non é máis, ou peor crítica. A solución é que o grupo elimine esa persoa do traballo non que é incompetente, nun xeito no que se minimice tanto como for posible ferir os seus sentimentos; na sección "*Transicións*" ao final deste capítulo podes ver uns exemplos. Este é habitualmente un caso raro. A maioría da xente responde bastante ben á crítica específica, detallada e que contén unha clara (mesmo aínda que non sexa explícita) esperanza de mellora.

As louvanzas non feren os sentimentos de ninguén, por suposto, mais iso non significa que deban ser empregadas con menor coidado que as críticas. A louvanzas son unha ferramenta: antes de usala, pregúntate a ti mesmo *por que* queres usala. Como regra, non é unha boa idea louvar a xente por faceren aquelo que acostuman a facer, ou por accións que son normais ou parte da súa participación no grupo. Se fixeses isto, sería difícil saber cando parar: deberías gabar a *todo o mundo* por facer as cousas habituais? Despois de todo, se deixas algunha xente fóra, preguntaranse por que. É moito mellor expresar gabanzas e gratitude ocasionalmente en resposta a esforzos non usuarios ou inesperados, coa intención de fomentar máis deses esforzos. Cando un participante parece terse movido a un estado permanente de alta produtividade, axusta as túas gabanzas consecuentemente. As gabanzas repetidas por comportamentos normais acaban gradualmente perdendo o seu significado. En lugar diso, esa persoa debe sentir que o seu alto nivel de produtividade agora é considerado normal e natural, e só o traballo que vaia máis alá debe ser especialmente considerado.

Isto non quere dicir que a contribución desta persoa non deba ser recoñecida, por suposto. Mais recorda que se o proxecto estiver ben configurado, todo o que esa persoa fixer xa é visible de tódolos xeitos, e por tanto o grupo verá (e esta persoa saberá que o resto do grupo sabe) todo o que ela fai. Tamén hai outros xeitos de recoñecer o traballo de alguén mediante outros medios distintos da gabanza directa. Poderías mencionar de pasada, cando se discuta un tema relacionado, que esa persoa fixo unha cantidade enorme de traballo nesa área e por tanto é o experto na mesma; poderías consultarlle publicamente algunhas cuestións sobre o código;ou quizais sexa máis eficaz empregar o seu traballo de xeito ostensible para que a persoa poida ver que os demais se senten cómodos cos resultados do seu traballo. Probablemente non sexa necesario facer estas cousas dun xeito calculado. Alguén que regularmente faga grandes contribucións nun proxecto saberao, e ocupará por defecto unha posición de

influencia. Normalmente non é necesario dar pasos explícitos para asegurar isto, a non ser que sintas que, pola razón que sexa, un contribuínte está sendo infravalorado.

Evita a territorialidade

Ten coidado cos participantes que intentan apropiarse en exclusividade da propiedade de certas área do proxecto, e que parece que queren facer todo o traballo nesas áreas, ata o punto de apropiárense do traballo que outros comezaron. Este comportamento pode parecer saudable ao comezo, despois de todo superficialmente parece que a persoa está collendo máis responsabilidade e amosando un incremento da actividade dentro dunha área dada, mais a longo prazo é destrutivo. Cando a xente ve un "prohibido o paso", apártase. Isto provoca unha redución da supervisión nesa área, e unha fraxilidade maior, dado que se depende da dispoñibilidade dun único desenvolvedor. Aínda peor, esta actitude rompe o espírito cooperativo e igualitario do proxecto. En teoría calquera desenvolvedor é benvido para axudar en calquera tarefa en calquera momento, na práctica por suposto as cousas funcionan dun xeito algo diferente: a xente ten áreas nas que é máis ou menos influínte, e os non expertos acostuman a deixar que os expertos manexen certos dominios do proxecto. Mais a chave atópase en que todo isto é voluntario: a autoridade informal é obtida en base á competencia demostrada, mais nunca debería ser activamente *conquistada*. Mesmo se a persoa que desexa a autoridade realmente é competente, segue sendo crucial que atesoure esa autoridade dun xeito informal, a través do consenso do grupo, e que a autoridade nunca provoque que exclúa os demais do traballo nesa área.

Rexeitar ou editar o traballo de alguén por motivos técnicos é unha cuestión totalmente diferente, por suposto. Aquí o factor decisivo é o contido do traballo, non o que sucede ao actuar como gardameta. Podería suceder que a mesma persoa faga a maior parte da revisión para unha área dada, mais mentres non tente evitar que os demais poidan facer tamén ese traballo, probablemente as cousas vaian ben.

Para combater unha incipiente territorialidade, ou mesmo a súa aparición, moitos proxectos teñen tomada a decisión de prohibir a inclusión dos nomes dos autores, ou dos mantedores designados, non ficheiros fonte. Eu estou de acordo de todo corazón con esta práctica: nós seguímola no proxecto Subversion, e é unha directiva máis ou menos oficial na Apache Software Foundation. Sander Striker, membro da ASF, explícao deste xeito:

> *Na Apache Software Foundation desalentamos o uso de etiquetas de autor no código fonte. Hai varias razóns para isto, a parte da ramificacións legais. O desenvolvemento colaborativo trata sobre o traballo como un grupo nos proxectos, coidándoos como un grupo. Dar recoñecemento é bo, e debería facerse, mais dun xeito que non permita falsa atribución, nin sequera por implicación. Non hai unha liña clara que marque cando engadir ou quitar unha etiqueta de autor. Engades o teu nome cando mudas un comentario? Cando forneces unha solución dunha liña? Quitas a etiqueta de autor doutro contribuínte cando refactorizas o seu código nun 95%? Que fas coa xente que toca cada ficheiro, mudando unicamente o necesario para facerse o autor virtual segundo as etiquetas de autor, de xeito que o seu nome estea en todos lados?*
>
> *Hai mellores xeitos de dar crédito, e nós preferimos empregar estoutros. Desde un punto de vista técnico, as etiquetas de autor son innecesarias; se quixeres saber quen escribiu un anaco de código en particular, o sistema de control de versións pode ser consultado para averigualo. As etiquetas de autor tenden a desactualizarse. Realmente queres que contacten contigo dun xeito privado sobre unha peza de código que escribiches hai cinco anos e da que serías feliz de*

poder esquecerte?

Os ficheiros fonte dun proxecto de software son o núcleo da súa identidade. Deberían reflectir o feito de que a comunidade de desenvolvemento no seu conxunto é a responsable del, e non estar dividido en pequena leiras.

A xente ás veces argumenta a favor das etiquetas de autor ou de mantedor nos ficheiros fonte naquelas áreas que fixeron a maior parte do traballo e nas que lles habería que dar un crédito visible. Hai dous problemas con este argumento: Primeiro, as etiquetas inevitablemente sempre fan xurdir a fea cuestión de canto traballo debe facer un para ter o seu propio nome incluído na lista. Segundo, mesturan a cuestión do crédito coa da autoría: ter feito traballo no pasado non implica a propiedade da área na que o traballo foi feito, mais é difícil, senón imposible, obviar esta implicación cando nomes individuais son listados no cabezallo dos ficheiros fonte. En calquera caso, a información de crédito tamén pode ser obtida dos logs do control de versión e doutros mecanismos como os arquivos das listas de correo, polo que non se perde ningunha información como consecuencia da prohibición de explicitala nos ficheiros fonte[18].

Se o teu proxecto decide prohibir os nomes individuais nos ficheiros fonte, asegúrate de non meteres a zoca. Por exemplo, moitos proxectos teñen unha área de *contrib/* (contribucións) onde se manteñen pequenas ferramentas e scripts de axuda, habitualmente escritos por xente que doutro xeito non estaría asociada co proxecto. É bo que estes ficheiros si conteñan nomes de autor, porque realmente non están sendo mantidos polo conxunto do proxecto. Por outra banda, se unha ferramenta así contribuída comeza a recibir contribucións e hacks por parte de outra xente do proxecto, pode que queiras movela a un lugar menos isolado, asumindo a aprobación do autor orixinal, e eliminar o nome do autor, para que o código se mostre do mesmo xeito que calquera outro recurso mantido pola comunidade. Se o autor for sensible respecto a isto, hai solucións de compromiso aceptables como, por exemplo:

```
# indexclean.py: Borra datos antigos orixinados por un índice.
#
# Autor Orixinal: K. Maru &lt;kobayashi@yetanotheremailservice.com&gt;
# Agora Mantido Por: O Proxecto Scanley &lt;http://www.scanley.org/&gt;
#                    e K. Maru.
#
# ...
```

Mais é mellor evitares estes compromisos, na medida que for posible, e a maioría dos autores están dispostos a seren persuadidos, porque se senten felices de que a súa contribución se converta nun parte máis integral do proxecto.

O importante é lembrares que hai unha continuidade entre o núcleo e a periferia de calquera proxecto. Os ficheiros fonte principais para o software son claramente parte do núcleo, e debería considerarse que son mantidos pola comunidade. Por outra banda, as ferramentas de acompañamento ou as pezas de documentación poden ser o traballo de individuos isolados, os cales os manteñen en solitario, aínda cando ese traballo poida asociarse, ou mesmo distribuírse, co proxecto. Non hai necesidade de aplicar a mesma regra a cada ficheiro, sempre e cando se cumprir o principio de que os recursos mantidos pola comunidade non poidan converterse no coto privado de ninguén.

18 Mais podes ver o fío titulado "*Having authors names in .py files*" en http://groups.google.com/group/sage-devel/browse_thread/thread/e207ce2206f0beee para unha boa contraargumentación, en particular na mensaxe de William Stein. A chave, nese caso, penso, é que moito autores veñen dunha cultura (a comunidade académica matemática) onde o crédito inserido directamente nas fontes é a norma e é altamente considerado. Nestas circunstancias, pode ser preferible incluír os nomes dos autores nos ficheiros fonte, incluíndo unha precisa descrición do que cada autor fixo, dado que a maioría dos autores van esperar un recoñecemento deste estilo

A relación de automatización

Tenta evitar que os humanos fagan aquelo que as máquinas poden facer no seu lugar. Como norma xeral, automatizar unha tarefa común supón dez veces menos traballo do que perdería un desenvolvedor facéndoa manualmente de cada vez. Para tarefas moi frecuentes ou complexas, a relación pode facilmente dispararse a vinte ou máis.

Pensar en ti mesmo como un "xefe de proxecto" máis que como simplemente outro desenvolvedor pode ser unha actitude moi positiva neste particular. Ás veces os desenvolvedores individuais están demasiado metidos no traballo de baixo nivel e non son capaces de veren o cadro ao completo e decatarse de que alguén está gastando unha morea de esforzo levando a cabo manualmente tarefas que se poderían automatizar. Mesmo aqueles que si que se decatan poden non empregar o tempo preciso para resolveren o problema: dado que o rendemento e custo individual de cada tarefa non se presentan como demasiado altos no conxunto, nunca ninguén se anoxa tanto como para facer algo ao respecto. O que fai atractiva a automatización é que esa pequena carga multiplícase polo número de veces que cada desenvolvedor debe afrontala, e *ese* número aínda hai que multiplicalo polo número de desenvolvedores.

Aquí estou a empregar o termo "automatización" nun senso amplo, para dar a entender non unicamente tarefas repetitivas onde unha ou dúas variables mudan de cada vez, senón tamén calquera tipo de infraestrutura técnica que axude os humanos. A automatización mínima estándar precisa para levar a cabo un proxecto a día de hoxe describiuse no *Capítulo 3 Infraestrutura Técnica*, mais cada proxecto pode ter a súa problemática específica, por suposto. Por exemplo, un grupo traballando en documentación podería querer ter un sitio web no que se mostren as versións máis actualizadas dos documentos en cada momento. Dado que a documentación se escribe frecuentemente en linguaxes de marcado estilo XML, pode haber un paso de compilación, ás veces bastante intricado, involucrado na creación dos documentos visualizables e descarregables. Desenvolver unha páxina web onde esta compilación se faga de xeito automático ante cada commit pode ser complexo e bastante esixente en tempo -mais é rendible, mesmo se a súa configuración che custa un día ou máis. O beneficio xeral de ter páxinas actualizadas dispoñibles en todo momento é enorme, mesmo se o custo de *non* telas puidese parecerlle un tema menor nun momento concreto a algún desenvolvedor concreto.

Dar estes pasos non só elimina gastos de tempo, senón que tamén elimina as obsesións e frustracións que aparecen nas persoas cando cometen erros (cousa que inevitablemente vai suceder) ao tentar executaren manualmente procedementos complexos. As operacións deterministas de varios pasos é precisamente para o que se inventaron os computadores; deixa que os humanos fagan cousas máis interesantes.

Probas automáticas

As execucións automáticas de probas son útiles para calquera proxecto de software, mais aínda o son máis no caso dos proxectos de software libre, dado que a testaxe automatizada (especialmente a testaxe de regresión) permite que os desenvolvedores se sintan cómodos mudando código en áreas que non lles son familiares, e mesmo fomenta o desenvolvemento exploratorio. Como é moi difícil detectar fallos a man -tes que adiviñar onde puideches romper algo, e probar varios experimentos para comprobares se o

fixeches ou non- ter mecanismos automatizados para detectar estes erros afórralle ao proxecto *unha chea* de tempo. A maiores, tamén consegue que a xente se sinta máis relaxada á hora de refactorizar grandes conxuntos de código, e polo tanto contribúe a manter a longo prazo o software.

Probas de regresión

> As *probas de regresión* son as comprobacións que se fan para detectar a reaparición de erros previamente reparados. A finalidade das probas de regresión é reducir a probabilidade de que mudanzas no código rompan o software de xeito inesperado. Do mesmo xeito que un proxecto de software vai aumentando de tamaño e complexidade, as probabilidades destes efectos colaterais aumentan da man. Un bo deseño pode reducir a relación de incremento destas probabilidades, mais non pode eliminar o problema por completo.
>
> Como resposta a esta problemática, moitos proxectos teñen unha *colección de probas*, un programa separado que invoca o software do proxecto en casuísticas nas que se sabe que no pasado se producían erros. Se a colección de probas fai que aparezan algúns deste erros, entón este feito coñécese como *regresión*, o que quere dicir que as mudanzas feitas por alguén reabriron un erro previamente reparado.
>
> Bótalle unha ollada a http://en.wikipedia.org/wiki/Regression_testing

As probas de regresión non son a panacea. Ten en conta que funcionan mellor con programas con interfaces ao estilo dos procesos por lotes. O software que se opera principalmente a través de interfaces de usuario é moito máis complicado de manexar programaticamente. Outro problema é que o marco das coleccións de proba para comprobacións de regresión habitualmente é bastante complexo, cunha curva de aprendizaxe e de mantemento nada desprezables. Reducir esta complexidade é unha das cousas máis útiles que podes facer, mesmo se implica a inversión dunha morea considerable de tempo. Canto máis sinxelo for engadir novas comprobacións á colección, máis desenvolvedores as engadirán, e así a taxa de supervivencia dos erros será moito menor. Calquera esforzo dirixido a facer que as comprobacións sexan máis sinxelas de compoñer será recompensado con xuros abundantes ao longo do ciclo de vida do proxecto.

Moitos proxectos observan con decisión a regra de "*Non rompas o que funciona!*", o que significa: non fagas mudanzas que deixen o software nun estado no que non compile ou non execute. Ser a persoa que rompeu a compilación ou execución causa con frecuencia certa vergoña e burla. Os proxectos que teñen coleccións de probas de regresión habitualmente teñen como norma un corolario da anterior: non fagas ningunha mudanza que provoque que as comprobacións fallen. Estes fallos son sinxelos de identificar se hai execucións nocturnas automáticas das coleccións de proba nas que se envíen os resultados por correo-e á lista de desenvolvemento ou a listas dedicadas aos resultados das probas; este é outro exemplo de automatización que vale a pena.

A maioría dos desenvolvedores voluntarios están dispostos a empregaren tempo extra en escribir probas de regresión, sempre e cando o sistema de comprobacións for comprensible e fácil traballar con el. O acompañamento das mudanzas coas comprobacións habitualmente enténdese como unha cuestión de responsabilidade, e a maiores tamén é unha boa oportunidade para que xurda a colaboración: habitualmente dous desenvolvedores divídense o traballo preciso para reparar un erro, cun deles escribindo a corrección en si mesma, e co outro escribindo a comprobación. O segundo desenvolvedor pode ter ao final máis traballo, e a maiores escribir a comprobación é menos satisfactorio que escribir a

corrección do erro, polo que se torna imperativo que a experiencia de usuario coa colección de comprobacións non sexa máis dolorosa do imprescindible.

Algúns proxectos aínda van alén disto, requirindo que unha nova comprobación acompañe *cada* solución de erros ou nova funcionalidade. Pode ser unha boa idea ou non en función de varios factores: a natureza do software, a predisposición do equipo de desenvolvemento, e a dificultade de escribir novas comprobacións. O proxecto CVS (http://www.cvshome.org/) hai tempo que tivo esta norma. En teoría é unha boa política, dado que CVS é un software de control de versións e por tanto debe ter unha alta aversión polos riscos e posibilidades de perder ou corromper os datos dos usuarios. Na práctica o problema está en que a colección de probas de regresión de CVS é un solitario script para consola de comandos (apropiadamente chamado *sanity.sh*) moi difícil de ler, modificar ou estender. A dificultade de engadir novos casos de proba, combinada co requisito de que os parches deben estar acompañados de novas comprobacións, provoca que efectivamente CVS free o envío de parches. Cando traballaba en CVS, ás veces vin xente comezar ou mesmo rematar un parche para CVS, mais renderse e abandonar cando lles mencionaba o requisito de que tiñan que engadir un novo caso de proba ao ficheiro *sanity.sh*

É normal empregares máis tempo escribindo un novo caso de proba de regresión que en solucionares o erro orixinal. Mais CVS levou este fenómeno ás súas últimas consecuencias: podías perder horas tentando deseñar convenientemente un novo caso de proba, e mesmo así non conseguilo debido a que había demasiadas complexidades imprevisibles á hora de mudar un ficheiro de liña de comandos composto por 35.000 liñas de código. Mesmo os desenvolvedores que participaban desde había moito tempo rosmaban cando tiñan que engadir un novo caso de proba.

Esta situación debíase a un fallo de parte a parte na consideración da relación de automatización. É certo que facer a mudanza a un marco de comprobacións real -tanto fose personalizado como empregado así como saía da caixa- tería sido un esforzo moi grande[19]. Mais negarse a facelo tivo un custo moito maior para o proxecto, ao longo dos anos. Cantas correccións de erros e novas funcionalidades *non* están incluídas en CVS hoxe en día debido ás dificultades que producía unha colección de probas pouco axeitada? Non podemos saber o número exacto, mais seguramente sexa varias veces maior que o número de correccións de erros e novas funcionalidades que se perderían polo gasto do tempo preciso para desenvolver un novo sistema de comprobación (ou integrar un xa desenvolvido). Esta tarefa consumiría unicamente unha cantidade de tempo finita, mentres que a penalización derivada do uso do sistema de comprobacións que se emprega prolongarase para sempre se non se fai nada ao respecto.

A cuestión non está en que ter requisitos estritos de escritura de comprobacións sexa malo, nin peor que escribir o teu sistema de comprobacións como un script por liña de comandos. Todo isto podería funcionar ben, dependendo de como o deseñes e do que necesites comprobar. A cuestión redúcese a que cando o sistema de comprobacións se tornar nun impedimento significativo para o desenvolvemento, deberás facer algo. O mesmo pensamento pode estenderse para calquera proceso rutineiro que se torna nunha barreira ou gargalo.

Trata a cada usuario como a un potencial voluntario

19 Ten en conta que non había a necesidade de converter todas as probas preexistentes ao novo marco/ferramenta de comprobación; os dous marcos/ferramentas poderían convivir, convertendo unicamente aquelas comprobacións que precisaran ser actualizadas

Todo contacto cun usuario é unha oportunidade para captares un novo voluntario. Cando un usuario se molesta en enviar unha mensaxe a unha lista de correo ou cubrir un erro no notificador de erros, está amosando máis potencial de colaboración que a maioría de outros usuarios dos que nunca se saberá nada. Fomenta ese potencial: se enviar un erro, agradécelle o traballo e pregúntalle se desexa corrixilo. Se apuntou que á FAQ lle falta algunha cuestión importante ou que a documentación é mellorable en certo punto, entón recoñece abertamente o problema (se en verdade existir) e pregúntalle se está interesado en resolvelo el mesmo. Evidentemente, algúns porán obxeccións e non o farán. Mais non custa nada preguntar, e cada vez que o fas, estás recordándolle aos outros usuarios que participar no proxecto é algo que todo o mundo pode facer.

Non limites os teus obxectivos para conseguires novos desenvolvedores e persoas que documenten. Por exemplo, formar xente para escribir bos informes de erros paga a pena a longo prazo, se non empregares *demasiado* tempo por persoa e se eles van a enviar máis informes no futuro -o que é probable que fagan se teñen unha relación construtiva co proxecto desde o inicio. Unha relación construtiva non quere dicir que o erro se resolva, aínda que iso é o ideal; senón que pode ser simplemente unha solicitude de máis información ou unha confirmación de que *si* é un erro. A xente desexa que a escoiten. Ademais, desexa que os erros sexan resoltos. Aínda que non se poida resolver o erro de inmediato, sempre se pode escoitar os colaboradores.

Un corolario disto é que os desenvolvedores non deberían expresar enfado ante a xente que envía informes ben intencionados aínda que vagos. Esta é a miña maior queixa; vexo a desenvolvedores facelo en todo momento en varias listas de correo de proxectos de software libre e o dano que iso fai é palpable. Algún usuario novo non informático pode escribir un informe non inútil, como o que segue:

> Ei, non son capaz de arrincar Scanley. Cada vez que arrinco, só aparecen erros. Sofre alguén este mesmo problema?

Algún desenvolvedor -que ten visto este tipo de informes máis dunha vez e non se parou a pensar que o novato non- responderá algo como isto:

> Que pensas que podemos facer con tan pouca información? A ver. Indícanos algúns detalles como a versión de Scanley, que sistema operativo usas e os erros que amosa.

Este desenvolvedor non se parou a observar o problema desde o punto de vista do usuario e tampouco ten considerado o efecto nocivo que pode ter unha resposta dese estilo en toda a *outra* xente vendo o intercambio de correos. Evidentemente, un usuario sen experiencia en programación nin en informes de erros, non saberá como enviar un. O mellor modo de xestionar a súa contribución é aprenderlle. E facelo dun modo que volvan por máis.

> Sinto que esteas tendo problemas. Precisaremos máis información para averiguarmos que está a ocorrer no teu caso. Poderías dicirnos que versión de Scanley usas, o sistema operativo e o texto exacto do erro? O mellor que podes facer é enviarnos os comandos que estás tratando de executar e a saída que producen. Bótalle un ollo a http://www.scanley.org/how_to_report_a_bug.html para máis información.

Esta resposta é moito máis efectiva xa que está escrita desde o punto de vista do usuario. Punto un, expresa empatía: *tes un problema e nós sentímolo*. (Isto non é preciso en cada resposta; depende da severidade do problema e como de frustrado se mostre o usuario). Punto dous, en vez de botarlle en cara que non sabe como enviar un informe de erro, a resposta dille como facelo, con suficiente detalle

para que sexa útil -por exemplo, moitos usuarios non se decatan de que "móstranos o erro" significa "móstranos o texto exacto do erro". A primeira vez que respondes a un usuario así, é preciso ser específico. Finalmente, a resposta ofrece unha orientación a instrucións máis completas e detalladas para enviar informes. Se coa resposta tes creado confianza co usuario é altamente probable que se moleste en botarlle un ollo ao documento e facer o que alí se indica. Isto quere dicir que é preciso ter o documento escrito por adiantado, o cal debe indicar claramente a información que o equipo de desenvolvemento precisa para resolver o erro. Idealmente, o documento vai evoluír co tempo en resposta ás omisións e equivocacións máis habituais dos usuarios.

As instrucións para enviar erros do proxecto Subversion son un bo exemplo (ver o *Apéndice D Exemplo de instrucións para informar de erros*). Nótese como se finaliza cunha invitación para resolver o erro. Isto non se fai porque leve a mellores relacións de informes/parches enviados -a maioría dos usuarios que saben arranxar os erros, tamén saben que os parches son benvidos. O verdadeiro propósito da invitación é que os novos participantes no proxecto o saiban tamén, que o proxecto se nutre das contribucións de voluntarios. Dalgún xeito, os actuais desenvolvedores non son máis responsables de resolveren o erro do que a persoa que o reportou. É este un punto importante co que non todo o mundo estará familiarizado. Facéndoo explícito estamos favorecendo que os usuarios axuden a arranxalo: se non é contribuíndo con código, pode ser ofrecéndose a testar parches que outros envían, etc. A chave reside en facer explícito que non hai diferenza *innata* entre o usuario que informa do erro e a persoas que colaboran no proxecto -a diferenza radica en canto tempo e esforzo un pon no proxecto, non en quen é.

A política de non responder duramente non se aplica a usuarios maleducados. De cando en vez, a xente postea informes de erro ou quéixase con ton burlón de que o proxecto falla nalgún punto. Habitualmente esa xente trata de insultar e louvar, como a persoa que posteou a seguinte mensaxe na lista de correo do proxecto Subversion:

> Por que despois de 6 días non están aínda subidos os binarios para Windows? Cada vez ocorre o mesmo e é moi frustrante. Por que non está iso automatizado para que poida estar dispoñible de inmediato tras cada actualización?Cando se sobe unha "RC" creo que a idea é que os usuarios a proben para comprobaren erros, porén, non se dispón dunha maneira para que os usuarios poidan facelo. Para que ofrecer un período de proba se non se ofrece ningún método para probalo?

A resposta inicial a esta mensaxe foi sorprendentemente contida: a xente indicou que o proxecto tiña establecido unha política de non ofrecer binarios oficiais, e dito isto con diferentes graos de irritación, invitábase a este usuario a producir por si mesmo os binarios se eran tan importantes para el. Créase ou non, a seguinte mensaxe deste usuario comezou así:

> Antes de nada, dicir que creo que o proxecto Subversion é un gran proxecto e realmente aprecio os esforzos de todo o mundo que colabora nel. [...]

... e entón voltou *de novo* a reprender a xente do proxecto por non proporcionar os binarios, aínda que non se propuxo voluntario para facer nada. Despois diso, unhas 50 persoas se abalanzaron sobre el, e non podo dicir que me preocupase demasiado. A política de "tolerancia cero" contra a mala educación predicada anteriormente na sección "*Tolerancia cero coa mala educación*" no *Capítulo 2 Primeiros pasos* aplícase ás persoas coas que o proxecto ten (o debería ter) unha interacción sostida no tempo, mais cando alguén se postula como unha fonte de malos modais desde o comezo, non paga a pena facer que se sinta como na casa.

Afortunadamente, tales situacións son raras e son aínda moito máis infrecuentes naqueles proxectos que fan o esforzo de integrar os novos participantes de modo construtivo e cortesmente desde o inicio.

Comparte as tarefas de xestión así como as técnicas

Comparte tanto o traballo de xestión así como o técnico do día a día do proxecto. A medida que un proxecto se fai máis complexo, cada vez máis traballo ten que ver con xestión da xente e fluxo de información. Non hai razón para non compartir esas tarefas, e por compartir non quero dicir necesariamente unha xerarquía top-down -na práctica dáse máis unha topoloxía de redes igualitarias antes que unha estrutura de estilo militar.

Algunhas veces, os roles de xestión están formalizados e outros emerxen de modo espontáneo. No proxecto Subversion, temos un xestor de parches, outro de traducións, de documentación, xestión de incidencias (aínda que non oficial) e un xestor das publicacións do software. Algúns deses roles saíron dunha decisión consciente, outros simplemente emerxeron por si mesmos; a medida que o proxecto medre, supoño que máis roles aparecerán. A continuación examinaranse ese roles e outros en detalle (a excepción do xestor de publicación, o do que xa se falou na sección “xestión de publicacións” e na sección “ditadura do propietario das publicacións” con anterioridade neste capítulo).

A medida que vexas as descrición dos roles, observa como ningún deles require control exclusivo sobre o dominio en cuestión. O xestor de incidencias non evita que outra xente faga mudanzas na base de datos de incidencias, a persoa encargada da FAQ non pretende ser a única persoa en editar a FAQ, etc. Eses roles versan sobre responsabilidades, non sobre monopolio. Unha parte importante sobre o traballo de cada un dos xestores é observar cando a xente está a traballar nese dominio e axudalos para que fagan mellor as cousas, de cara a que os múltiples esforzos se vexan multiplicados e non entren en conflito. Os xestores deberían tamén documentar os procesos mediante os que fan o seu traballo, para que cando un queira deixalo, algún outro poida retomar o seu traballo.

A veces existe un conflito: dúas ou máis persoas desexan o mesmo rol. Non hai un modo correcto de xestionar isto. Pódese suxerir que cada voluntario propoña unha candidatura e tódalas persoas con acceso ao repositorio voten sobre cal lles parece a mellor. Mais esta técnica é potencialmente nociva. Unha mellor é falar cos múltiples candidatos para que entre eles cheguen a un consenso. Habitualmente chegarán a el, e esta solución sentirase como máis xusta que se viñer imposta por alguén de fóra.

Xestor de parches

Nun proxecto de software libre que recibe moitos parches, facer seguimento de cada un deles pode ser un pesadelo, especialmente se se fixer dun modo descentralizado. A maioría dos parches son enviados como mensaxes á lista de correo (aínda que algún deles pode ser enviado ao xestor de incidencias, ou ter aparecido nalgún sitio web), e existen varios camiños que poden seguir desde ese momento.

Ás veces, alguén revisa o parches atopa algún problema e encamíñao ao autor orixinal para que corrixa os erros. Este proceso habitualmente leva a un proceso iterativo -sempre visible a través da lista de correo- no que o autor orixinal envía versións sucesivas do parche ata que ninguén ten nada máis que criticar. Non sempre é sinxelo saber cando remata o proceso: se o revisor fixer commit do parche,

claramente o proceso está completo. Mais se non for así, podo ser porque non tivo tempo, ou non ten acceso de commit e non puido falar con outros desenvolvedores para que o fixesen.

Outra resposta frecuente é un debate espontáneo, non necesariamente sobre o parche, senón sobre se o concepto detrás do parche é adecuado. Por exemplo, o parche pode ter un erro, mais o proxecto prefire arranxalo doutro modo, como parte dunha estratexia para resolver unha clase máis xeral de problemas. Habitualmente, isto non se sabe por adiantado, é o mesmo parche o que fai o descubrimento.

Ocasionalmente, o parche atópase con silencio absoluto. Isto soe acontecer porque ningún desenvolvedor ten tempo *nese preciso instante* para revisalo, e cada un deles espera que outro o revise. Debido a que non hai un límite específico sobre canto unha persoa pode esperar para que alguén responda e que poden chegar novas prioridades ao proxecto, é moi sinxelo que un parche pase desapercibido. O proxecto pode estar perdendo un bo parche, mais existen outros efectos colaterais peores que este: é desalentador para o autor -quen ten invertido traballo no parche- e transmite que o proxecto non está ó día, sobre todo con esa xente que escribe e envía parches.

O traballo do xestor de parches é asegurarse de que os parches non se perden no vacuo. Isto faise seguindo a evolución de cada parche con algunha clase de táboa de estado. O xestor de parches vixía cada lista de correo na que se envían parches. Se o envío do parche finaliza en commit, non fai nada. Se entra nun proceso interactivo de revisión, tendo unha versión do parche definitiva mais non se fai commit, preocúpase de cubrir unha incidencia que apunte a esa versión final do parche e á lista de correo na que sucedeu, para que sexa sinxelo que os desenvolvedores o revisen máis tarde. Se o parche ten a ver cunha incidencia concreta, anota toda esa información baixo a incidencia existente.

Cando un parche non xera ningunha reacción, o xestor de parches espera uns poucos días, e entón pregunta se alguén vai revisalo. En xeral, isto xera algunha reacción: un desenvolvedor pode explicar que o parche non debería ser aplicado ou pode revisalo el mesmo, dando lugar a un dos camiños expostos anteriormente. Se non houber resposta, o xestor de parche pode (ou non) cubrir unha incidencia para o parche segundo mellor vexa, mais polo menos o autor orixinal obtén *algún* feedback.

Para o proxecto Subversion, ter un xestor de parches supuxo un aforro de tempo e enerxía mental. Sen unha persoa especialmente dedicada a estes temas, cada desenvolvedor tería que preocuparse constantemente sobre a xestión do mesmo: "Se non teño tempo de respondelo agora, podo confiar en alguén para que o faga? Debería botarlle un ollo? E se outra xente xa o estiver revisando? Entón estaríamos duplicando esforzos de modo innecesario". O xestor de parches elimina este tipo de dúbidas e cuestións. Cada desenvolvedor toma a decisión que mellor lle veña no mesmo momento en que ve o parche: se desexa seguilo e revisalo, pode facelo. Se vai ignoralo, iso tamén está ben, xa que o xestor do parche fará que non fique no esquecemento.

Este sistema funciona só se a xente pode confiar en que o xestor de parche vai estar aí sempre, o rol debe ser formalmente declarado. En Subversion, publicitámolo nas listas de correo de usuarios e desenvolvedores, obtendo bastantes voluntarios e escollendo ó primeiro que respondeu. Cando esta persoa tivo que deixalo (ver a sección "*Transicións*" máis adiante neste capítulo), fixemos o mesmo de novo. Nunca se tentou ter varias persoas co mesmo rol, debido á sobrecarga en comunicacións que requiriría entre ambos, mais talvez en proxectos con moita carga de parches, un equipo de xestores de parches teña sentido.

Xestor das traducións

Nos proxectos de software, "tradución" pode referirse a cousas moi diferentes. Por un lado, pode ter que ver coa tradución da documentación a outras linguaxes, polo outro, pode referirse á tradución da aplicación -é dicir, facer que as mensaxes que emite o programa estean dispoñibles na linguaxe preferida polo usuario. Ambas son tarefas complexas, mais unha vez a infraestrutura estiver lista, son tarefas ben diferenciadas de outras de desenvolvemento. Xa que son similares entre elas, pode ter sentido (dependendo do proxecto) ter un xestor de traducións que manexe as dúas, ou polo contrario pode ter sentido ter dous diferentes.

No proxecto Subversion, temos a unha única persoa xestionando ambas. Esta persoa non escribe as traducións por si mesmo, desde logo -pode botar unha man nunha ou dúas, mais precisaría falar 10 linguas (12 dialectos) para traballar en todas elas! O seu labor ten que ver coa xestión de voluntarios e equipos de tradución: axúdaos a coordinárense entre eles, e esta persoa é o punto común entre todos os equipos e o resto do proxecto.

Unha das razóns principais polas que o xestor de tradución é necesario, é que en xeral, os tradutores son bastante diferentes dos desenvolvedores. Ás veces, teñen pouca ou ningunha experiencia traballando con sistemas de control de versións, ou mesmo de traballo dentro dun grupo distribuído de voluntarios. Mais noutros aspectos son os mellores voluntarios: xente con coñecemento do dominio que viu unha necesidade e desexa participar na súa resolución. Habitualmente están dispostos a aprenderen todo o necesario e son entusiastas do seu traballo. Todo o que precisan é que alguén os guíe. O xestor de traducións asegúrase de que as mesmas evolúan correctamente sen interferiren no desenvolvemento regular do proxecto. Tamén exerce de representante dos tradutores como un grupo en si mesmo, cando se comunica con desenvolvedores de cara a informalos de mudanzas técnicas necesarias para mellorar o traballo de tradución.

Sendo así, as capacidades máis valiosas para este posto son diplomáticas, non técnicas. Por exemplo, no proxecto Subversion temos a política de que tódalas traducións deben ter polo menos 2 persoas traballando nelas, debido a que doutro xeito o texto non será revisado. Cando un novo voluntario se ofrece a traducir o proxecto a, por exemplo, Malagasy, o xestor de tradución ten que presentarlle a alguén quen informou 6 meses atrás de que quería traducir o proxecto a Malagasy ou amablemente indicarlle que busque *outro* tradutor. Unha vez que se consegue a xente suficiente, o xestor ocúpase de todo o relativo a darlle acceso de commiter, informalos sobre as convencións do proxecto (tales como escribir mensaxes de log) e fai o seguimento oportuno para que sigan esas convencións.

As conversas entre os desenvolvedores e o xestor de traducións, ou entre este e os equipos de tradución, mantéñense na lingua franca do proxecto -é dicir, na lingua orixe desde a que se fan tódalas traducións. Para a maioría de proxectos de software libre, esta lingua é o inglés, mais non importa cal for sempre e cando o proxecto tiver un acordo sobre a mesma (o inglés é probablemente mellor para proxectos que desexen atraer unha comunidade de desenvolvedores internacional).

As conversas *dentro* dun equipo de tradución, habitualmente teñen lugar na súa lingua compartida. Unha das tarefas que debe levar a cabo o xestor de traducións é encargarse de que o equipo teña unha lista de correo exclusiva para eles. Deste modo, os tradutores poden debater o seu traballo libremente, sen distraeren outras persoas que están noutras listas de correo, a maioría das cales non entenderán a lingua.

Internacionalización fronte a localización

> *Internacionalización* (*I18N*) e *localización* (*L10N*) refírense ao proceso de adaptación dun programa a outros entornos lingüísticos e culturais diferentes do orixinal onde foi creado. Embora sexa trocado habitualmente o uso dos termos, non se refiren ao mesmo. Como http://en.wikipedia.org/wiki/G11n describe:
>
> > A distinción entre eles é sutil aínda que importante: a internacionalización é a adaptación de produtos para o seu *potencial* uso en todas partes, mentres a localización ten a ver con engadir funcionalidades *específicas* para o seu uso en determinadas contornas.
>
> Por exemplo, o feito de mudar o teu programa para que use Unicode (http://en.wikipedia.org/wiki/Unicode) como formato de codificación de texto é un movemento de internacionalización, xa que non se refire a unha linguaxe particular senón a facilitar o procesamento de texto en múltiples linguaxes. Por outra banda, facer que o programa imprima tódalas mensaxes de erro en esloveno, cando se detecta que está sobre un entorno configurado para tal linguaxe, é un movemento de localización.
>
> Así, as tarefas do xestor de traducións refírense principalmente á localización, non á internacionalización.

Xestor de documentación

O mantemento da documentación é unha tarefa que nunca remata. Cada nova funcionalidade ou mellora introducida no código pode esixir unha mudanza na documentación. Ademais, cando a documentación dun proxecto atinxir un certo nivel de completitude, verás que moitos dos parches que a xente envía son para a documentación, non para o código. Isto débese a que hai máis xente competente para resolver erros na prosa que no código: tódolos usuarios son lectores, mais só uns poucos son programadores.

Os parches de documentación son máis sinxelos de revisar e aplicar que os de código. Hai pouco que revisar ou testar, e a calidade da mudanza pode ser avaliada rapidamente mediante a revisión do texto. Xa que a cantidade é alta e a carga de revisión é baixa, a relación de carga administrativa en relación ao traballo produtivo é maior para parches de documentación que para os de código. Por outra banda, a maioría dos parches probablemente precisarán de algún axuste, de cara a manter unha voz única ao longo de toda a documentación. En moitos casos, os parches estarán solapados ou afectarán a outros parches, e precisarán ser axustados antes de facerse commit con eles.

Dadas as esixencias de xestión dos parches de documentación, e o feito de que o código precisa ser monitorizado constantemente para que a documentación estea actualizada, ten sentido que unha persoa ou un equipo se dedique á tarefa. Eles poden facer seguimento de onde e como a documentación vai atrasada con respecto do código, ademais de que poden ter desenvolvido procedementos eficientes para manexar grandes cantidades de parches dun modo integrado.

Evidentemente, isto non impide que outras persoas poidan enviar e aplicar mudanzas na documentación, especialmente os pequenos. O mesmo xestor de parches (ver a sección "*Xestor de Parches*" anteriormente neste mesmo capítulo) pode facer seguimento tanto dos parches de código

como dos de documentación, cubríndoos onde os equipos de desenvolvemento e documentación estiveren habituados a recibilos. (Se a cantidade de parches en total excede o que unha persoa pode seguir, dividir o traballo entre un xestor de parches de código e outro de documentación é probablemente un bo primeiro paso). A vantaxe de ter un equipo de documentación é ter persoas que teñan en mente a tarefa de manter a documentación organizada, actualizada e coherente. Na práctica, isto quere dicir coñecer a documentación en profundidade, revisar o código, ver tamén as mudanzas que *outros* fan sobre a documentación, revisar parches á documentación e usar toda esas fontes de información para facer o que for necesario para manter a documentación nun bo estado.

Xestor de incidencias

O número de incidencias no sistema de xestión de erros do proxecto crece proporcionalmente co número de xente que usa o programa. Embora arranxes erros e distribúas unha aplicación cada vez máis robusta, deberías esperar que o número de erros abertos medre. Tamén se incrementará a frecuencia de erros duplicados, así como a de aqueles incompletos ou pobremente descritos.

Os xestores de erros axudan a aliviar eses problemas mediante a revisión de todo o que se vai introducindo na base de datos, e de forma periódica buscando problemas específicos. A tarefa máis común probablemente sexa a de revisar tódolos erros novos e comprobar que non son duplicados de outros ou que os campos están correctamente preenchidos. Obviamente, canto máis familiarizada estiver esta persoa coa base de datos do proxecto, máis eficientemente será capaz de detectar duplicados -o que é unha das principais vantaxes de ter unha pouca xente especializada, en vez de ter a todo o mundo tratando de facelo *ad hoc*. Cando un grupo trata de facelo dun modo descentralizado, ninguén adquire un coñecemento profundo do contido da base de datos.

Os xestores de erros poden tamén axudar tamén a diminuír a fenda existente entre usuarios e desenvolvedores. Cando se notifican moitos erros, non tódolos desenvolvedores poden seguir con igual atención a lista de correo. Porén, se alguén próximo ao equipo de desenvolvemento revisa tódolos erros entrantes, pode chamar a atención dun desenvolvedor sobre un problema concreto que a este se lle pasou. Desde logo, isto ten que facerse coa sensibilidade adecuada. É unha boa práctica que os xestores de erros sexan á súa vez desenvolvedores.

En función de como o proxecto use o sistema de envío de erros, os coordinadores poden tamén darlle forma á base de datos para reflectir as prioridades actuais do proxecto. Por exemplo, no proxecto Subversion programabamos certas funcionalidades/corrección de erros para certas versións futuras do proxecto. Seguindo esta política, cando alguén pregunta "Cando será resolto este erro?" podemos dicir "dentro de 2 versións", mesmo se non for posible dar unha data exacta. As versións estaban representadas no sistema como milestones (marcos), un campo dispoñible na aplicación que nos usabamos IssueZilla[20]. Como norma, cada nova versión de Subversion ten unha nova funcionalidade e un conxunto de erros resoltos. Nós asignamos o milestone adecuado a tódalas incidencias planeadas para esa versión (incluíndo a nova funcionalidade -nós considerámola tamén como unha incidencia), así a xente pode navegar na base de datos a través das diferentes versións sabendo que mudanzas incluirán. Porén, eses obxectivos raramente se manteñen estáticos. A medida que chegan novos erros, as prioridades teñen que ser repensadas e as incidencias deben ser re-planeadas dun milestone a outro, de cara a que a nova versión sexa manexable. Este traballo, é mellor que o faga xente que teña unha visión global do que hai na base de datos e de como as diferentes incidencias se relacionan entre si.

20 IssueZilla é un sistema xestor de erros descendente de BugZilla

Outra das tarefas que leva a cabo un coordinador é ver cando unha incidencia é obsoleta. A veces, un erro é corrixido accidentalmente como parte dunha mudanza no programa, ou a veces a xente implicada muda de opinión sobre se un comportamento é erróneo. Atopar incidencias obsoletas non é sinxelo: o único modo de facelo é rastreando sistematicamente a base de datos do proxecto. Os rastrexos completos son menos prácticos a medida que aumenta o número de incidencias. Chegado a certo punto, a única aproximación para manter unha lista de erros adecuada é usar a estratexia de divide e vencerás: categorizar as incidencias a medida que cheguen e dirixilas directamente ao desenvolvedor ou equipo adecuado. Esa persoa ou grupo de persoas faise cargo desa incidencia desde entón, ficando obrigado a resolvela ou encamiñala cando for necesario. Cando a base de datos é moi grande, o xestor de incidencias parécese máis a un coordinador, invertendo menos tempo revisando cada incidencia e máis en encamiñala á persoa adecuada.

Xestor da FAQ (Preguntas Frecuentes)

Contra todo prognóstico, o mantemento da FAQ é un problema complexo. A diferenza da maioría dos documentos dun proxecto, cuxos contidos son planeados por adiantado polos propios autores, a FAQ é un documento reactivo (ver a sección "*Mantendo unha FAQ*"). Non importa como de grande for, é imposible coñecer cal será o seguinte engadido. E debido a que vai crecendo por pequenas mudanzas, é sinxelo que se converta nun documento desorganizado e incoherente, mesmo contendo duplicados ou semi-duplicados. Mesmo cando non existan obvios problemas como eses, haberá interdependencias entre os temas -ligazóns que deberían estar mais non están- debido a que os elementos relacionados foron engadidos un ano atrás.

O rol do xestor da FAQ ten dúas caras. A primeira, consiste en manter a calidade global da FAQ e estar familiarizado con tódolos temas que hai en ela, para que cando a xente engada novos temas que son duplicados de, ou relacionados con outros, se poidan levar a cabo os axustes necesarios. A segunda, consiste en revisar a lista de correo do proxecto e outros foros buscando preguntas recorrentes, escribindo novas entradas na FAQ baseadas nesta información. Esta última tarefa pode ser bastante complexa: un debe ser capaz de seguir un fío de correos, recoñecer as cuestións clave, enviar para debate unha entrada na FAQ que explique o anterior, incorporar os comentarios doutras persoas (xa que é imposible para o xestor da FAQ ser un experto en cada tema cuberto pola FAQ) e saber cando o proceso de debate finaliza para poder engadir o novo ítem á FAQ.

Esta persoa, habitualmente convértese no experto por defecto no formato da FAQ. Hai un montón de pequenos detalles ao redor de manter unha FAQ (ver a sección "*Trata todos os recursos coma ficheiros*" no *Capítulo 6 Comunicacións*); cando xente aleatoria a edita, algunhas veces van esquecer certos detalles. Non hai problema por iso, sempre e cando o xestor da FAQ estiver aí para resolver o problema.

Existen varias ferramentas de software libre para manter a FAQ. Paga a pena usalas sempre e cando non comprometa a calidade da FAQ nin supoña unha sobre-automatización. Algúns proxectos tratan de automatizar o proceso por completo, permitindo que todo o mundo contribúa e a edite dun modo similar a un wiki (ver a sección "*Wikis*" no *Capítulo 3 Infraestrutura Técnica*). Particularmente, isto ocorre con Faq-O-Matic (http://faqomatic.sourceforge.net/), aínda que é posible que os casos que teño visto foran simplemente abusos que foron máis alá das intencións orixinais para o que Faq-O-Matic fora pensado. En calquera caso, aínda que a descentralización da edición e mantemento da FAQ reduce

a carga de traballo do proxecto, iso tamén produce unha FAQ de peor calidade. Non hai ninguén cunha visión completa do documento, ninguén para observar cando certos ítems son obsoletos, duplicados ou precisan actualización, non hai ninguén que se encargue de ver as interdependencias entre os elementos. Como resultado obtense un documento que non satisfai as necesidades do usuario, e no peor caso, o confunde. Usa as ferramentas que consideres necesarias para manter o documento, mais nunca te deixes cegar polas mesmas comprometendo a calidade da FAQ.

Bótalle un ollo ao artigo de Michael Kerner *A FAQ das FAQs*, en http://osdir.com/Article1722.phtml para obteres descricións e avaliacións detalladas de ferramentas de mantemento de FAQs con software libre.

Transicións

A veces ocorre que, un voluntario nunha posición de responsabilidade (por exemplo: xestor de parches, de tradución, etc) será incapaz de levar a cabo as tarefas requiridas. Isto pode deberse a que son maiores que as que se anticiparan, ou por outro tipo de factores externos: mudanza do estado civil, situacións familiares novas como ter un fillo, un novo emprego, etc.

Cando isto ocorre, o propio voluntario non se decata de inmediato. A mudanza prodúcese de modo progresivo, cunha redución das tarefas que leva a cabo e talvez non sexa consciente de que xa non é capaz de satisfacer tódolos requirimentos que o propio rol acarreta. Por outra banda, o resto do proxecto simplemente non ten noticias del nun tempo. Soe ocorrer que entón o proxecto ve un pico repentino de actividade, que soe vir dado porque a persoa se sente culpable por deixar de lado o traballo no proxecto e se senta un día a facer tarefas e poñerse ao día. Despois diso. é habitual que non se saiba nada del durante un tempo máis longo que o anterior, para ter de novo -ou talvez non- outro pico de actividade. Raramente acontece unha rexeizón voluntaria do posto. O voluntario estaba a facer o traballo no seu tempo libre, polo que rexeitar significaría recoñecer abertamente a si mesmo que ten menos tempo libre do que pensaba. Á xente habitualmente lle costa asumir iso.

Polo tanto, depende totalmente de ti e das outras persoas no proxecto decatarvos do que está a ocorrer -ou do que non está a ocorrer- e preguntarlle ao voluntario o que está pasando. A pregunta debe ser totalmente amigable e sen que implique un sentimento de culpa para el. O propósito é saber o que acontece, non facer que a persoa se sinta mal. En xeral, é mellor que isto se faga publicamente, mais é perfectamente válido que se faga en privado se existir algunha razón para tal cousa. A razón de facelo usando os canais públicos reside en que se o voluntario contesta que non pode levar a cabo as tarefas encomendadas, existe un contexto xa para a túa *seguinte* mensaxe pública: unha petición de novos voluntarios para que ocupen ese rol.

Ás veces, o voluntario non é capaz de levar a cabo as tarefas encomendadas, mais tampouco está disposto ou é capaz de recoñecer ese feito. Evidentemente, calquera pode ter problemas ao principio, especialmente se a responsabilidade for complexa. Porén, se a cousa simplemente non funcionar, mesmo despois de recibir toda a axuda posible, só existe unha solución apropiada: que o voluntario deixe paso a outra persoa. E se el non o ve por si mesmo, necesitará que alguén llo diga. Creo que só hai un modo de manexar isto, mais é un proceso de múltiples pasos, onde cada un deles é importante.

Primeiro, asegúrate de que non está errado. Fala en privado con outras persoas do proxecto para ver se eles concordan contigo e o problema é tan serio como ti pensas que é. Embora xa esteas seguro, isto

serve para facerlle ver aos demais que estás a considerar pedirlle á persoa que dea o relevo do rol. Habitualmente ninguén terá obxeccións -a xente estará contenta de que deas a cara para facer unha tarefa ingrata, e así eles non teñen que facelo!

A continuación, contacta *en privado* coa persoa e coméntalle o que pensas, directa, mais amablemente. Sé específico, dando tantos exemplos como for posible. Asegúrate de facer explícito como o resto da xente fixo o posible por axudar, mais que o problema continúa. É probable que este correo che tome un tempo escribilo, e se non estiveres seguro do que estás a dicir, non deberías dicilo. Unha vez o fixeres, explica que che gustaría atopar un novo voluntario para o rol, mais tamén deixa claro que existen outras maneiras de contribuír ao proxecto. Nesta etapa, non digas que falaches con outra xente; a ninguén lle gusta ouvir que outros estiveron conspirando ás súas costas.

Isto pode dar lugar a diferentes respostas pola súa parte. A máis probable é que estea de acordo contigo ou que non desexe discutir, polo que está disposto a dar o relevo. Nese caso, o mellor sería que el fixese o anuncio por si mesmo, e despois diso podes facer unha petición para atopares un novo voluntario.

Outra delas, sería que estea de acordo en que non puido levar a cabo o traballo encomendado, mais que pida un pouco máis de tempo (ou unha nova oportunidade, para roles con tarefas discretas como por exemplo o xestor da publicación). A reposta a isto depende do teu xuízo, mais decidas o que decidires, non o fagas unicamente porque sentes que non podes rexeitar unha petición tan razoable como esa. Ese enfoque prolongaría a agonía, non a eliminaría. Unha boa razón para rexeitares a petición é que xa existiron un bo número de oportunidades, as que vos levaron ao punto no que estades agora. A continuación podes ver o correo-e que lle enviei a alguén que estaba ocupando o posto de xestor da publicación, mais que en verdade non estaba feito para el:

> *> Se desexas substituírme por algunha outra persoa, enténdoo e*
> *> farei o relevo o mellor que poida. Porén, teño unha petición,*
> *> que penso que é razoable. Gustaríame facer unha publicación máis*
> *> para poñerme a proba.*
>
> *Comprendo perfectamente o teu desexo (pasoume a min mesmo!), mais creo que*
> *neste caso non deberíamos intentalo de novo.*
>
> *Esta non é a primeira ou segunda publicación do programa, é a sexta ou sétima ...*
> *e para todas elas, sei que tamén ti non estás satisfeito cos resultados*
> *(temos falado diso antes). Pódese dicir que xa o temos intentado*
> *unha vez máis.*
> *E un deses intentos ten que ser o último. Creo que*
> *[a pasada publicación] debería ter sido este último intento.*

No peor caso, a persoa pode estar en desacordo. Tes que aceptar entón que as cousas van ser incómodas e tirar para diante. É momento de dicir que tes falado xa con outra xente (mais non dicir con quen a non ser que tiveres o seu permiso, xa que esas conversas foron confidenciais), e non cres que sexa bo para o proxecto que as cousas continúen dese modo. Sé insistente, mais non ameazador. Ten en conta que para a maioría dos roles, a transición ocorre no momento en que alguén novo ocupa o rol, *non* cando a antiga persoa deixa de facelo. Por exemplo, se a discusión versar sobre, digamos, o xestor das incidencias, en calquera momento ti e outras persoas do proxecto podedes facer explícito o teu desexo de teres outra persoa para o posto. Non é necesario obrigatoriamente que a persoa que estaba facendo o traballo deixe de facelo, sempre e cando non sabote (deliberadamente ou doutro modo) os esforzos do

novo voluntario.

O que nos leva a unha idea tentadora: por que en lugar de pedirlle á persoa que abandone o posto, non o enfocamos como simplemente unha cuestión de darlle máis axuda? Por que non ter dous xestores de incidencias ou de parches, ou calquera que for o rol?

Aínda que esa idea poida soar tentadora a primeira vista, xeralmente non é unha boa idea. O que fai que rol de coordinador funcione -o que o fai útil, de feito- é a súa centralización. As cousas que poden facerse dun modo descentralizado xa son feitas dese modo. Ter dúas persoas facendo un traballo que pode facer unha, introduce unha sobrecarga comunicativa entre ambos, ademais do potencial desentendemento da responsabilidade (un polo outro e a casa sen varrer). desde logo, hai excepcións. Ás veces, dúas persoas traballan moi ben xuntas, ou pode que que a propia natureza do rol pode ser distribuída entre varios dun modo sinxelo. Mais non soe ocorrer iso cando ves alguén fracasar nun rol para o que non ten as capacidades adecuadas. Se o voluntario tiver visto o problema por si mesmo, tería pedido axuda por si mesmo. En calquera caso, sería unha falta de respecto deixar que alguén esbanxase o seu tempo nun traballo que ninguén vai ter en consideración.

O factor máis importante de pedirlle a alguén que dea o relevo é a privacidade: darlle espazo para que tome unha decisión sen que sinta que os demais o están mirando ou esperado. Unha vez, eu cometín o erro -un erro bastante obvio, en retrospectiva- de enviarlle un correo ás tres partes implicadas para pedir que o xestor da publicación de Subversion dese o relevo en favor de outras dúas persoas. Xa tiña falado cos dous novos voluntarios e estaban dispostos a asumir a responsabilidade. Dun modo inocente, crin que aforraría algún tempo se lle enviaba o email aos tres á vez para que iniciasen a transición. Asumín que o actual coordinador era plenamente consciente do problema e vería razoable a miña petición.

Estaba equivocado. O xestor da publicación ofendeuse moito, e con razón. Unha cousa é facerlle ver que ten que deixar o seu rol; outra moi distinta é facelo -diante- da xente que dará o relevo. Unha vez me decatei de por que estaba ofendido, pedín desculpas. El deu o relevo dun modo elegante para todos e a día de hoxe segue a participar no proxecto. Mais os seus sentimentos foron feridos, e é necesario dicir, que tampouco foi o mellor dos comezos que imaxinaran os novos voluntarios.

Committers

Debido a que formalmente son a única clase de voluntarios que atopamos en calquera proxecto de software libre, os *commiters* merecen unha especial atención aquí. Os commiters son unha inevitable concesión discriminatoria, nun sistema que por outra banda é o menos discriminatorio dos posibles. Mais non tomes "discriminación" no senso pexorativo. A función que teñen os commiters é necesaria e non creo que un proxecto puidese sobrevivir sen eles. Evidentemente o control de calidade require control. Hai moita xente que se sente competente para facer mudanzas a un programa, e algúns poucos que realmente o son. O proxecto non pode depender do propio xuízo da xente; deben impoñerse estándares e conceder unicamente acceso só a aqueles que os cumpriren[21]. Por outro lado, conseguir

21 Obsérvese que ter acceso de commiter ten un significado diferente en sistemas de control de versións descentralizados, onde calquera pode configurar un repositorio que enlace ao proxecto e outorgarse a si mesmo acceso de commiter nese repositorio. Porén, o *concepto* de commit ten validez mesmo neses entornos: "acceso de commit" é un modo resumido de dicir "o dereito de facer mudanzas no programa que se empacotará na seguinte versión do mesmo". En sistemas de control de versións centralizados, isto quere dicir ter acceso de commit directo; en sistemas descentralizados, quere dicir que as mudanzas que un fai son integradas directamente no repositorio principal. En ambos casos é a mesma idea, a

que a xente que ten acceso de commiter traballe ombro con ombro coa xente que non o ten outórgalle ao proxecto unha dinámica moi boa de traballo. E debe ser xestionada para que non dane o proxecto.

Na sección "*Quen vota?*" do *Capítulo 4 Infraestrutura Social e Política* xa se ten debatido a mecánica de incluír novos commiters. A continuación botaráselle un ollo aos estándares polos cales o potenciais commiters deben ser xulgados e como este proceso debe ser presentado ante a comunidade que rodea o proxecto.

Escollendo os commiters

No proxecto Subversion escollemos os committers principalmente empregando o Principio Hipocrático: *primeiro, non fagas dano*. O noso criterio principal non son nin as capacidades técnicas nin o coñecemento do código, simplemente que o committer amose un bo criterio. Criterio pode simplemente significar saber o que non asumir. Unha persoa podería enviar soamente pequenos parches, arranxando pequenos problemas no código; mais se os parches se aplican limpamente, non conteñen erros, e están en harmonía co rexistro de mensaxes e as guías de estilo do proxecto, e hai parches de abondo para consideralo un padrón evidente, entón un commiter existente habitualmente proporá esa persoa para darlle permiso de commit. Se un mínimo de tres persoas din que si, e ninguén se opón, faise a oferta. Certo, pode que non teñamos ningunha proba de que a persoa está capacitada para resolver problemas complexos en tódalas áreas do código, mais iso non importa: a persoa deixou claro que polo menos é quen de xulgar as súas propias habilidades. As capacidades técnicas pódense aprender (e ensinar), mais o criterio, as máis das veces, non. Polo tanto, esta é a única cousa da que te tes que asegurar que unha persoa ten antes de darlle permiso de commit.

Cando unha proposta nova de committer provoca unha discusión, non acostuma a ser sobre as capacidades técnicas, senón máis ben sobre o comportamento desa persoa nas listas de correo ou no IRC. Ás veces, unha persoa amosa capacidades técnicas e habilidades para traballar conforme á filosofía do proxecto, mais tamén se amosa constantemente belixerante e non cooperativa nos foros públicos. Este é un asunto importante: se a persoa non se adaptar ao longo do tempo, mesmo como resposta a indirectas, entón non imos engadilo como commiter independentemente do capacitado que estiver. Nun grupo de voluntarios, as habilidades sociais, ou a habilidade de "xogar limpo na area", son tan importantes como as capacidades técnicas en estado puro. Xa que todo está baixo control de versións, as consecuencias de ter engadido a un commiter que non deberías non son tanto os problemas que pode causar no código (a revisión sacaríaos á luz) como que pode forzar finalmente a anular o permiso de committer desa persoa; unha medida que non é nunca agradable e pode ás veces causar confrontacións.

Moitos proxectos insisten en que un commiter potencial ten que demostrar un certo nivel de experiencia técnica e persistencia, enviando un certo número de parches non triviais; é dicir, estes proxectos non se conforman con saber que a persoa non fará dano, senón que queren saber que probablemente traballará ben ao longo do código. Isto está ben, mais hai que ter coidado de que non converta a pertenza ó grupo de committers nunha cuestión de pertenza a un grupo exclusivo. A pregunta que todo o mundo debería ter na cabeza é "Que traerá os mellores resultados para o código?" e non "Imos empeorar o status social asociado coa pertenza ao grupo de committers admitindo a esta persoa?" A idea do permiso de commit non é reforzar a autoestima da xente, senón permitir que as boas mudanzas entren no código cun mínimo esforzo. Se tiveres 100 committers, 10 dos cales fan grandes

mecánica de como se fai non é realmente importante

mudanzas por norma xeral, e os outros 90 só arranxan erros de escritura e pequenos erros poucas veces ao ano, isto é mellor que ter so ós 10 primeiros.

Revocación do permiso de commit

A primeira cousa que se debe dicir sobre revocar o permiso de commit é: tenta non chegar a esta situación, en primeiro lugar. Dependendo de a quen se lle retirar o permiso, e por que, as discusións en torno a está situación poden chegar a dividir. Mesmo se non chegaren a dividir, serán unha perda de tempo que nos distraerán do traballo produtivo.

Porén, se tiveres que facelo, a discusión debería manterse en privado entre as mesmas persoas que estarían en posición de votar para concederlle o permiso a esa persoa, independentemente do tipo de permiso de commit que tiver. A persoa involucrada non debería estar incluída nesta discusión. Isto refuta o mandato habitual en contra do segredo, mais neste caso é preciso. Primeiro, ninguén sería quen de falar libremente no caso contrario. Segundo, se a moción fracasar, pode non interesarche que a persoa saiba que foi considerada, porque isto podería provocar que lle xurdisen preguntas. (Quen estivo do meu lado? Quen estivo na miña contra?) que conducirán ao peor dos sectarismos. En certas circunstancias infrecuentes, o grupo pode querer que a persoa coñeza que a revocación do permiso de commit se considera ou se considerou, como un aviso, mais isto debe ser unha decisión do grupo. Ninguén debería, por iniciativa propia, revelar información relativa a unha discusión e votación que os demais supoñen secreta.

Unha vez que o permiso for revocado, debe facerse público inevitablemente (ver a sección "*Evitar o misterio*" máis adiante neste capítulo), así que tenta ser o máis diplomático posible en canto á forma en que o anuncias ao mundo exterior.

Permiso de commit parcial

Certos proxectos ofrecen graos de permiso de commit. Por exemplo, podería haber contribuíntes con permisos que lles proporcionen vía libre na documentación, mais que non poidan facer commits sobre o código. Áreas habituais para permisos de commit parciais son documentación, traducións, ligazóns de código con outras linguaxes de programación, especificación para o empacotado dos ficheiros (p.ex. especificación RPM de RedHat), e outras áreas onde un erro non causará un problema no núcleo do proxecto.

Xa que o permiso de commit non implica só facer commits, senón tamén formar parte dun electorado (ver a sección "*Quen vota?*" no *Capítulo 4 Infraestrutura Social e Política*), a pregunta naturalmente xorde: en que poden votar os committers parciais? Non hai resposta correcta; depende do tipo de dominios de commit parcial do proxecto. No proxecto Subversion mantemos as cousas moi simples: un commiter parcial pode votar en temas relacionados exclusivamente co seu dominio de commit, e en nada máis. É importante destacar que temos mecanismos para emitirmois votos de asesoramento (basicamente o committer escribe "+0" ou "+1 (non vinculante)" en vez de "+1"). Non hai razón para silenciar a xente simplemente porque o seu voto non é formalmente vinculante.

Os committers con permiso total poden votar sobre calquera cuestión, do mesmo xeito que poden facer commits en calquera parte do código, e só eles poden votar para engadir novos committers de calquera

tipo. Na práctica, porén, a capacidade para engadir novos commiter con permisos parciais é delegada. Calquera commiter total pode "patrocinar" un commiter parcial, e os commiters parciais poden ás veces escoller novos commiters para o seu dominio (isto axuda considerablemente a axilizar o traballo de tradución).

O teu proxecto pode precisar dunha filosofía diferente, dependendo da natureza do traballo, mais estes mesmos principios xerais son válidos para tódolos proxectos. Cada committer debe poder votar nos temas que caen dentro do seu dominio de commit, e non en temas que caen fóra del, e os votos sobre cuestións relativas ao procedemento deben ser por defecto dos committers totais, a menos que existir algunha razón (decidida polos committers totais) para ampliar o electorado.

Con respecto á aplicación de permiso de commit parcial: con frecuencia é mellor *non* empregares o sistema de control de versións para aplicar dominios de commit parcial, mesmo se for posible. Mira a sección "*Autorización*" no *Capítulo 3 Infraestrutura Técnica* para saberes por que.

Committers inactivos

Algúns proxectos automaticamente revogan o permiso para facer commits da xente se pasaren un determinado período de tempo (por exemplo, un ano) sen faceren ningún commit. Penso que isto é normalmente erróneo e mesmo contraproducente, por dúas razóns.

Primeiro, pode tentar a xente a faceren commits de mudanzas aceptables, mais innecesarias, simplemente para evitaren que o permiso para facer commits expire. En segundo lugar, non serve realmente para ningún propósito. Se o principal motivo para outorgar permiso de commit é o bo xuízo, entón por que asumir que o xuízo sobre alguén se deteriora simplemente porque se afasta do proxecto por un tempo? Mesmo se desaparecer completamente por anos, sen atender ao código ou ás discusións dos desenvolvedores, cando reaparecer quererá *saber* como de desconectado está realmente, e actuar en consecuencia. Confiaches nel anteriormente, así que por que non confiar nel sempre? Se os diplomas do instituto non caducan, é evidente que o acceso de commit tampouco debería.

Ás veces un committer pode pedir ser eliminado, ou ser marcado explicitamente como inactivo, na lista de committers (véxase a sección "*Evitar o Misterio*" máis adiante para coñecer mellor esa lista). Nestes casos, o proxecto debería acceder aos desexos da persoa, evidentemente.

Evitar o misterio

Embora as discusión acerca de aceptar novos comitters deban ser confidenciais, as propias regras e procedementos non deben selo. De feito, é mellor publicalos, de modo que a xente comprenda que os committers non son ningunha Cámara Celestial pechada aos meros mortais, senón que calquera pode unirse simplemente posteando bos parches e sabendo como manexarse na comunidade. No proxecto Subversion poñemos esta información directamente no documento coas pautas para os desenvolvedores, xa que a xente máis interesada en coñecer como se outorgan permisos para commits é aquela que pensa en contribuír co proxecto.

Para alén de publicares os procedementos, publica a *lista* completa e actualizada de committers. O lugar onde se fai habitualmente é o arquivo *MAINTAINERS*, ou *COMMITTERS* no nivel máis alto da

árbore de código fonte do proxecto. Debería ser unha lista que inclúa en primeiro lugar os committers globais, seguidos dos diferentes dominios de commit parciais e os membros de cada un deles. Cada persoa debe ser listada co nome e o enderezo de email, embora o enderezo se poida cifrar para evitar spam (véxase a sección "*Ocultamento de enderezo en ficheiros*" no *Capítulo 3 Infraestrutura Técnica*) se a persoa o preferir.

Xa que a distinción entre acceso como committer global e parcial é obvia e ben definida, é aconsellable indicala na lista tamén. Para alén disto, a lista non debe tratar de indicar as diferenzas informais que inevitablemente xorden en cada proxecto, como quen son as persoas máis influíntes, e de que modo. Trátase dun rexistro público, non dun ficheiro de recoñecementos. Os committers lístanse en orde alfabética ou na orde en que se uniron ao proxecto.

Recoñecemento

O recoñecemento é a principal moeda no mundo do software libre. Embora todo o que a xente diga sobre as súas motivacións para traballar nun proxecto, non coñezo ningún desenvolvedor que se sentise feliz facendo o seu traballo de maneira anónima ou co nome doutra persoa. Existen razóns tanxibles para isto: a reputación de cada un nun proxecto determina claramente a súa influencia, e a participación nun proxecto de software libre tamén pode indirectamente proporcionar beneficios económicos, porque actualmente algúns empresarios os buscan nos currículums. Hai tamén razóns intanxibles, quizais máis poderosas: a xente simplemente quere ser apreciada, e instintivamente buscan sinais de que o seu traballo é recoñecido por outros. A promesa do recoñecemento está aí e constitúe unha das maiores motivacións que un proxecto ten. Cando as pequenas contribución son recoñecidas, a xente volta para facer máis.

Unha das características máis importantes do desenvolvemento colaborativo de software (véxase o *Capítulo 3 Infraestrutura Técnica*) é que conserva rexistros precisos sobre quen fixo que e cando. Sempre que for posible, débense empregar estes mecanismos existentes para asegurar que o mérito é distribuído correctamente, sendo específicos na natureza da contribución. Non escribir simplemente "Gracias a X. Aleatorio <xaleatorio@exemplo.gl>" se podemos poñer "Gracias a X. Aleatorio <xaleatorio@exemplo.gl> polo informe do erro e o método para repetilo" nunha mensaxe de log.

En Subversion temos unha política informal mais coherente de acreditación de informes de erros, ben nun ficheiro de problemas, se existir, ou na mensaxe de log do commit que soluciona o erro. Unha inspección rápida do log de commits de Subversion ata o número 14525 mostra que preto do 10% dos commits acreditan alguén polo nome e o enderezo de correo electrónico, normalmente a persoa que informou ou analizou o erro solucionado nese commit. Nótese que esta persoa é diferente do desenvolvedor que realmente fixo o commit, cuxo nome xa é rexistrado automaticamente no sistema de control de versións. Dos oitenta e tantos committers globais e parciais, cincuenta e cinco foron acreditados en logs de commit (normalmente múltiples veces) antes de seren committers. Isto non demostra, evidentemente, que ter sido acreditado fose un factor na súa colaboración continua no proxecto, mais polo menos crea unha atmosfera na cal a xente sabe que pode contar con que as súas contribucións sexan recoñecidas.

É importante distinguir entre a rutina de recoñecemento e os agradecementos especiais. Cando se discute un fragmento particular de código, ou calquera outra contribución, sempre é correcto recoñecer o traballo feito. Por exemplo, dicir "as mudanzas recentes de Xan sobre o código delta queren dicir que

agora podemos implementar a funcionalidade X" simultaneamente axudan á xente a identificar de que mudanza se está a falar e recoñece o traballo de Xan. Por outra banda, darlle as gracias nun post individual a Xan non ten ningunha utilidade inmediata. Non engade información, xa que o sistema de control de versións e outros mecanismos xa rexistraron o feito de que el fixo as mudanzas. Agradecer a todos por todo sería unha distracción e carecería de toda información, debido a que o agradecemento resulta efectivo na medida en que se aparta dos comentarios favorables que se ven habitualmente. Isto non quere dicir, por suposto, que nunca se deba agradecer á xente. Só se debe estar seguro de que non se fai de maneira que conduza ao exceso de recoñecemento. As seguintes liñas xerais son de axuda niso:

- Canto máis efémero sexa o foro, con máis liberdade se pode agradecer nel. Por exemplo, agradecer alguén por arranxar un erro durante unha conversa de IRC é correcto, así como facelo nun correo centrado principalmente noutros temas. Mais non se debe mandar un correo unicamente para agradecer alguén, se non for por algunha labor realmente inédita. Do mesmo xeito, non se deben encher as páxinas do proxecto con expresións de gratitude. Unha vez que se comezar, nunca estará claro onde ou cando deterse. E nunca pór agradecementos no código; iso sería unicamente unha distracción sobre o obxectivo principal dos comentarios, que é axudar o lector a comprender o código.

- Canto menos participar alguén nun proxecto, máis apropiado vai ser agradecerlle o traballo feito. Isto pode parecer un contrasentido, mais encaixa coa actitude de que expresar agradecemento é algo que se fai cando alguén contribúe máis do que era de esperar. Polo tanto, agradecer continuamente a xente que contribúe habitualmente por un traballo que xa farían en condicións normais ía querer dicir que esperamos deles menos que eles mesmos, e está claro que se algo nos interesa é o efecto contrario.

  Existen algunhas excepción a esta norma. Resulta aceptable agradecer alguén por cumprir o seu rol cando ese traballo require esforzos importantes de cando en vez. O exemplo por excelencia é o xestor das versións, quen realiza un traballo importante cando se libera cada versión, mentres que o resto do tempo permanece inactivo (inactivo como xestor das versións, xa que por outra banda pode ser un desenvolvedor activo, mais iso é outro asunto).

- Igual que as críticas e o recoñecemento, o agradecemento debe ser específico. Non se debe agradecer a xente simplemente por seren excelentes, mesmo se realmente o foren. Os agradecementos faranse por algo fóra do ordinario, e por puntos adicionais, aclarando por que o traballo feito foi tan destacable.

En xeral, sempre hai tensión entre asegurarse de que as contribucións individuais da xente son recoñecidas e que o proxecto resulta máis un esforzo de grupo que una colección de logros individuais. Simplemente hai que ser conscientes desta tensión e tratar de inclinar a balanza cara ao lado do grupo, e dese modo estará baixo control.

Forks (escisións)

Na sección "*Forkability*" no *Capítulo 4 Infraestrutura Social e Política* vimos como o *potencial* de crear escisións ten importantes efectos en como se gobernan os proxectos. Mais, ¿que ocorre cando realmente se crea unha nova rama? ¿Como se debe manexar, e que efectos podemos esperar dela? No

estremo contrario, cando se debe *iniciar* unha nova rama/escisión?

A resposta depende da clase de rama que sexa. Algunhas débense a desacordos amigables mais irreconciliables sobre a dirección do proxecto; quizais os máis debidos tanto a diverxencias técnicas como a conflitos persoais. Por suposto, non sempre é posible diferenciar ambos, xa que os argumentos técnicos poden tamén implicar elementos persoais. O que tódalas ramas teñen en común é que un grupo de desenvolvedores (nalgunhas ocasión mesmo un só deles) decidiu que o custe de traballar con algúns ou con tódolos demais supera os beneficios.

Unha vez que un proxecto se ramifica, non hai unha resposta segura sobre a pregunta de cal é o proxecto "verdadeiro" ou "orixinal". A xente falará de maneira coloquial acerca de unha rama R que sae dun proxecto P, de maneira que P segue o seu rumbo natural mentres que R penetra nun novo territorio, mais isto é, realmente, unha declaración de como ese observador en concreto percibe a situación. Trátase fundamentalmente dun asunto de percepción: cando un porcentaxe suficientemente grande de observadores está de acordo, devandita afirmación comeza a ser certa. Non se trata de que exista unha verdade obxectiva, unha que simplemente non somos capaces de percibir desde o principio, senón de que as percepcións *son* a verdade obxectiva, simplemente porque un proxecto -ou unha rama- é unha entidade abstracta na nosa mente.

Se a xente que crea unha nova rama sente que están indo nun camiño diferente ao do proxecto principal, a cuestión queda resolta inmediata e sinxelamente. Todos, tanto desenvolvedores coma usuarios, tratarán á rama coma un novo proxecto, cun novo nome (pode que baseado no antigo, mais facilmente distinguible del), un sitio web diferente e unha filosofía de traballo ou un obxectivo distinto. en lugar, as cousas complícanse cando ambos lados cren ser os gardiáns lexítimos do proxecto orixinal e por tanto teñen dereito a continuaren co antigo nome. Se existir algunha organización con dereitos sobre o nome, ou control legal sobre o dominio ou páxinas web, o problema resólvese habitualmente segundo o seu criterio: a organización decidirá quen é o proxecto e quen a rama, porque ten tódalas de gañar nunha guerra de relacións públicas. Naturalmente, rara vez as cousas chegan tan lonxe: como todos coñecen as dinámicas do poder, evitan unha guerra cuxo resultado xa saben de antemán, e prefiren ir directamente á solución.

Por sorte, na maioría dos casos hai poucas dúbidas sobre cal é o proxecto e cal a rama, porque a rama é, en esencia, un voto de confianza. Se máis da metade dos desenvolvedores estiveren a favor de calquera rumbo que a rama propón tomar, habitualmente non hai motivos para ramificar; o proxecto simplemente vai ir por ese camiño por si mesmo, a non ser que estiver controlado por un ditador especialmente teimudo. Por outra banda, se estiveren a favor menos da metade dos desenvolvedores, a rama é claramente unha rebelión minoritaria, e tanto a cortesía coma o sentido común dinnos que eles mesmos deberían considerarse unha rama diverxente mais que a tendencia principal.

Manexar unha escisión/fork

Se alguén ameazar con crear unha rama no teu proxecto, mantén a calma e recorda os teus obxectivos a longo prazo. A simple *existencia* da rama non prexudica o proxecto, senón a perda de desenvolvedores e usuarios. Polo tanto o teu verdadeiro obxectivo non é esmagar esa rama, senón minimizar eses efectos daniños. Podes enfadarte, pensar que a rama é inxusta e ninguén a pedía, mais expresárelo en público só servirá para alienares a desenvolvedores indecisos. En lugar diso, non forces a xente a tomaren decisións excluentes, e sé tan cooperativo coa rama como for posible. Para comezar, non revoques

permisos de commit no teu proxecto a alguén simplemente porque decidiu traballar na rama. Traballar nela non significa que esa persoa perdera de repente a súa competencia para traballar no proxecto orixinal; os committers anteriores deberían seguir séndoo. Aparte disto, deberías expresar o teu desexo de teres toda a compatibilidade posible coa rama, e afirmar que desexas que os desenvolvedores adaptarán mudanzas entre ambos sempre que for apropiado. Se tiveres acceso como administrador aos servidores do proxecto, ofrece publicamente as túas infraestruturas para os seus inicios. Por exemplo, ofrécelles unha copia completa do repositorio de control de versións, se non tiveren outra maneira de conseguilo, de modo que non teñan que comezar sen ningún dato histórico (isto pode non ser necesario segundo o sistema de control de versións empregado). Pregúntalles se precisan algunha outra cousa, e proporciónalla na medida do posible. Móstrate cordial e amistoso con eles, sen estorbarlles no seu camiño, demostrando que queres que a rama teña un futuro baseado nos seus propios méritos, nin máis nin menos.

A razón de faceres todo isto -e de facérelo publicamente- non é axudares realmente á rama, senón persuadires aos desenvolvedores de que o teu lado é unha aposta segura, semellando o menos vingativo posible. Na guerra ás veces ten sentido (sentido estratéxico, non ético) forzares a xente a escolleren bando, mais no software libre rara vez é así. De feito, tras unha rama é habitual que algúns desenvolvedores colaboren publicamente en ambos proxectos e fagan todo o posible por mantelos compatibles. Estes desenvolvedores axudan a manter as liñas de comunicación abertas despois de se crear a rama. Tamén permiten que o teu proxecto se beneficie de novas características interesantes na rama (si, é posible que a rama teña cousas que che interesen) e incrementan as posibilidades dunha fusión máis adiante.

Nalgunhas ocasións, unha rama ten tanto éxito que, mesmo se foi considerada polos seus creadores como unha rama diverxente, tórnase a versión preferida por todos, e chega un momento en que supera a orixinal en demanda popular. O exemplo por excelencia disto é a rama GCC/EGCS. A *GNU Compiler Collection* (*GCC*, antigamente o *GNU C Compiler*) é o compilador de código fonte libre máis popular, e tamén un dos compiladores máis portables do mundo Debido a desacordos entre os mantedores oficiais de GCC e Cygnus Software[22], un dos grupos de desenvolvedores de GCC máis activos, Cygnus creou unha rama de GCC chamada *EGCS*. A rama foi deliberadamente amistosa: os desenvolvedores de EGCS non tentaron, en ningún momento, implantar a súa versión de GCC como unha nova versión oficial. No seu lugar, concentráronse en facer EGCS o mellor posible, incorporando parches a un maior ritmo que os mantedores oficiais de GCC. EGCS gañou popularidade, e finalmente algunhas distribucións decidiron substituír GCC por EGCS como o seu compilador por defecto. Chegado este punto, fíxose evidente para os mantedores de GCC que manter o nome "GCC" mentres que todo o mundo migraba a EGCS simplemente serviría para que todo o mundo sufrira unha mudanza de nome sen motivo algún. Por iso GCC adoptou o código base de EGCS, e así temos de novo un único GCC, mais inmensamente mellorado debido á rama.

Este exemplo demostra por que non debemos considerar sempre unha rama como algo inevitablemente malo. Unha rama pode ser dolorosa e molesta en certo momento, mais nunca podemos saber se vai ter éxito. Polo tanto debemos manter, xunto co resto do proxecto, un ollo sobre ela, e estarmos preparados non só para absorbermos funcionalidades e código cando for posible, senón para no caso máis extremo unírmonos á rama se se convirte no referente do obxectivo do proxecto. Por suposto, é habitual podermos predicir as posibilidades de éxito dunha rama segundo quen se unir a ela. Se for comezada polo membro máis molesto do proxecto, e seguida por un bo número de desenvolvedores anoxados que xa non tiñan unha actitude construtiva, en esencia resolveron un dos nosos problemas, e probablemente

22 Agora parte de RedHat (http://www.redhat.com/)

non nos teñamos que preocupar de se a rama supera o proxecto orixinal. Mais se vemos como influentes e respectados desenvolvedores apoian a rama, deberiamos preguntarnos o motivo. Pode que o proxecto estea sendo demasiado restritivo, e a mellor solución pasa por adoptarmos na liña principal algunha ou tódalas accións contempladas na rama -en esencia, evitarmos que a rama se produza como tal.

Comezar unha rama/escisión

Tódolos consellos aquí asumen que pensamos en crear unha rama como último recurso, fartos doutras posibilidades xa tentadas. Crear unha rama case sempre implica perder desenvolvedores, con só unha pequena esperanza de gañar outros novos máis adiante. Tamén quere dicir comezar de novo con competencia por captar a atención do usuario: todo o que estiver a punto de descargar o software preguntarase "Hmmmm, prefiro este ou este outro?". En calquera caso, a situación é confusa, porque se introduce unha nova pregunta. Algunha xente sostén que as ramas son saudables para o ecosistema do software en conxunto, por un argumento de selección natural estándar: o máis capacitado sobrevive, o que significa que, ao final, todos conseguen un mellor software. Pode que isto sexa certo desde o punto de vista do ecosistema, mais non desde o punto de vista de calquera proxecto individual. A maioría das ramas non teñen éxito, e a maioría dos proxectos non se alegran de sufriren ramas.

Un corolario disto é que non se debe empregar a ameaza dunha rama como una técnica extremista de debate -"Ou fas as cousas á miña maneira ou ramifico o proxecto!"- porque todos son conscientes de que unha rama que non logra atraer desenvolvedores do proxecto orixinal ten poucas probabilidades de sobrevivir. Tódolos observadores -non soamente desenvolvedores, senón tamén usuarios e empacotadores de sistemas operativos- farán o seu xuízo sobre que lado escoller. Polo tanto debemos ter moitas reticencias coa creación de ramas, de maneira que cando o fagamos sexa porque non vemos outra solución.

Non debemos descoidarnos á hora de termos en conta *tódolos* factores cando avaliamos o éxito potencial da nosa rama. Por exemplo, se moitos desenvolvedores do proxecto teñen o mesmo xefe, mesmo se están molestos e persoalmente a favor da rama, será difícil que o demostren se saben que o seu xefe se mostra en contra dela. A moitos programadores de software libre gústalles pensar que ter unha licenza de software libre no código significa que ningunha compañía pode dominar no desenvolvemento. É verdade que a licenza é, en última instancia, unha garantía da liberdade -se outros teñen o suficiente interese en crear unha rama, e os recursos necesarios, poden facelo. Mais na práctica, algúns equipos de desenvolvedores de proxectos están fundados na súa maioría por unha entidade, e non podemos pretender que o seu soporte non importa. Se for contrario á rama, os desenvolvedores raramente tomarán parte, mesmo se en secreto así o quixeren.

Porén, se decidimos crear unha rama, en primeiro lugar buscaremos apoio de maneira privada, e logo o anunciaremos nun ton nada hostil, aínda que estivermos molestos ou decepcionados cos mantedores actuais. Simplemente explicaremos con desilusión os motivos que nos levaron a ramificar e que non desexamos ningún mal ao proxecto orixinal. Asumindo que realmente o consideramos unha rama (en contraposición a preservación de emerxencia do proxecto orixinal), faremos énfase en que estamos ramificando o código e non o nome, e escolleremos un novo que non sexa conflitivo co orixinal. Podemos empregar un que teña referencias ao nome orixinal mentres non leve a confusión á hora de identificalos. Por suposto podemos explicar na páxina web da rama que descende do programa orixinal, e mesmo que tenta substituílo. Soamente tentaremos non facer a vida dos usuarios máis difícil

introducíndoos nunha disputa de identidades.

Finalmente, podemos comezar as cousas con bo pé outorgando permisos de commit na rama a tódolos committers do proxecto orixinal, incluídos aqueles que estiveren manifestamente en contra da ramificación. Mesmo se nunca fixeren uso do seu acceso, a nosa mensaxe é clara: pode que haxa desacordos, mais non inimigos, e agradecemos as contribucións de código de calquera colaborador competente.

Capítulo 9. Licenzas, dereitos de autor (*copyright*) e patentes

A licenza que escolleres probablemente non vai ter un impacto significativo na adopción do teu proxecto, sempre e cando a licenza for de software libre. Os usuarios normalmente escollen o software baseándose na calidade e nas funcionalidades, non baseándose nos detalles da licenza. Porén, segues precisando ter un coñecemento básico relativo ao licenciamento do software libre, tanto para aseguráreste de que a licenza do proxecto é compatible cos seus obxectivos, como para estares capacitado para discutires as decisións de licenciamento co resto da xente. Por favor, ten en conta que eu non son avogado, e que nada neste capítulo deber ser tido en conta como consellos legais. Para elo, deberás contratar un avogado, ou selo.

Terminoloxía

En calquera discusión sobre licenzas de software libre, o primeiro que sae á superficie é que parece haber varias palabras diferentes para falar da mesma cousa: *software libre*, *código aberto*, *FOSS*, *F/OSS* e *FLOSS*. Comecemos por traballarmos estes termos, ademais de algúns outros.

Software libre (free software)

> Software que pode ser libremente compartido e modificado, incluíndo o código fonte. O termo foi coinado por primeira vez por Richard Stallman, quen o inseriu na GNU Public License (GPL), e quen fundou a Free Software Foundation (http://www.fsf.org) para promover o concepto.
>
> Embora "software libre" cubra case exactamente o mesmo tipo de software que "software de código aberto" (open source software), a FSF (Free Software Foundation), entre outros, prefire o primeiro termo debido a que fai fincapé na idea da liberdade, así como no concepto de que o software redistribuíble de xeito libre é principalmente un movemento social máis que algo técnico. A FSF é consciente de que o termo, en inglés, é ambiguo -"free" pode significar "gratuíto", en lugar de "liberdade"- mais aínda así considera que segue sendo o mellor termo, e que o resto de posibilidades, en inglés, teñen as súas ambigüidades. (Ao longo deste libro, "free"/"libre" é empregado co significado de "liberdade" e non co significado de "gratuíto").

Software de código aberto (open source)

> Software libre baixo outro nome. Mais a diferenza de nome reflicte unha diferenza filosófica importante: o nome "software de código aberto (open source)" foi coinado pola Open Source Initiative (http://www.opensource.org) como unha alternativa deliberada a "software libre (free software)", co fin de tornar este software máis atractivo para as corporacións, presentándoo

como unha metodoloxía de desenvolvemento máis que como un movemento político. Tamén pretendían superar outro estigma: o gratuíto ten pouca calidade.

Aínda que calquera licenza que sexa libre tamén o é de código aberto, e viceversa (cunhas poucas excepcións), a xente tende a escoller un termo e aferrarse a el. En xeral, os que prefiren "software libre" acostuman a ter unha postura máis filosófica ou moral sobre esta cuestión, mentres que os que prefiren "software de código aberto" non ven este tema como unha cuestión de liberdade, ou non están interesados en publicitar este punto de vista. Bótalle unha ollada a á sección "*Libre contra fontes abertas*" no *Capítulo 1 Introdución* para veres unha historia máis detallada sobre este cisma.

A Free Software Foundation ten unha excelente -carente de obxectividade, mais matizada e moi xusta- exexese de ambos termos, en http://www.fsf.org/licensing/essays/free-software-for-freedom.html. A visión da Open Source Initiative a respecto disto (ou polo menos a que tiñan en 2002) móstrase en dúas páxinas: http://web.archive.org/web/20021204155022/ e http://www.opensource.org/advocacy/case_for_hackers.php#marketing

FOSS, F/OSS, FLOSS

Onde caben dous, caben tres, e isto é exactamente o que está a suceder cos termos que enuncian o software libre. O mundo académico, quizais buscando precisión e inclusión máis que elegancia, parece que se decidiu pola forma "FOSS", ou ás veces "F/OSS", co significado "Free / Open Source Software". Outra variante que está gañando adeptos é "FLOSS", que significa "Free / Libre Open Source Software" (*libre* é común en varios idiomas e non está suxeita ás ambigüidades de "free"; bótalle unha ollada a http://en.wikipedia.org/wiki/FLOSS para máis detalles).

Todos estes termos significan esencialmente o mesmo: software que pode ser modificado e redistribuído por calquera, algunhas veces -mais non sempre- coa condición de que os traballos derivados teñan que ser libremente redistribuíbles baixo as mesmas cláusulas.

Conforme coas DFSG

Conforme coas "Directrices de Software Libre de Debian" (http://www.debian.org/social_contract#guidelines). Esta é unha proba amplamente empregada para comprobar se unha licenza dada é realmente software libre (de código aberto, etc). A misión do Proxecto Debian é manter un sistema operativo completamente libre, de xeito que ninguén que o instale nunca teña dúbidas sobre se ten o dereito de modificar e redistribuír calquera parte do sistema, ou mesmo o sistema enteiro. As Directrices de Software Libre de Debian son os requisitos que a licenza dun pacote de software ten que cumprir para poder sen incluído en Debian. Debido a que o Proxecto Debian empregou unha morea de tempo en pensar como construír este test, as directrices que xurdiron demostraron ser moi robustas (véxase http://en.wikipedia.org/wiki/DFSG) e ata onde eu sei, non se fixo ningunha obxección seria ás mesmas nin desde a Free Software Foundation nin desde a Open Source Initiative. Se sabes que unha licenza dada é conforme ás DFSG, entón sabes que garante todas as liberdades importantes (tales como a posibilidade dun *fork* mesmo contra os desexos do autor orixinal) requiridas para soster as dinámicas dun proxecto software libre. Todas as licenzas tratadas neste capítulo son conformes coas DFSG.

Aprobadas pola OSI

Aprobadas pola Open Source Initiative. Esta é outra proba amplamente empregada para saber se unha licenza permite todas as liberdades necesarias. A definición de software de código aberto da OSI está baseada nas Directrices de Software Libre de Debian, e calquera licenza que concorda con unha das definicións, case sempre concorda coa outra. Ten habido unhas poucas excepcións ao longo do tempo, mais eran só licenzas de nicho sen relevancia. Ao contrario que o proxecto Debian, a OSI mantén unha lista de todas as licenzas que aprobou, en http://www.opensource.org/licenses/ , co cal ser "aprobada por OSI" é un estado non ambiguo: unha licenza ou está ou non está na lista.

A Free Software Foundation tamén mantén unha lista de licenzas en http://www.fsf.org/licensing/licenses/license-list.html . A FSF clasifica as licenzas non só segundo sexan libres ou non, senón tamén segundo sexan compatibles coa GNU General Public License. A compatibilidade coa GPL é un asunto importante, cuberto na sección "*A GPL e a compatibilidade de licenzas*" máis adiante neste capítulo.

Propietario, *código pechado*

O oposto de software libre ou de código aberto. Fai referencia a software distribuído baixo licenzas tradicionais baseadas en regalías, onde os usuarios pagan por copia, ou baixo outros termos suficientemente restritivos para evitar as dinámicas operativas do software libre. Mesmo software distribuído sen cargo pode ser propietario, no caso de que a súa licenza non permita a libre redistribución e modificación.

Xeralmente "propietario" e "código pechado" son sinónimos. Aínda así, "código pechado" implica adicionalmente que o código fonte non pode nin sequera ser visto. Dado que o código fonte non pode ser visto na maior parte do software propietario, normalmente é máis unha distinción que non unha diferenza. Porén, ocasionalmente alguén publica software propietario baixo unha licenza que lle permite os demais veren o código fonte. Confusamente, estes autores ás veces chámanlle "software de código aberto" ou "próximo ao software de código aberto", etc., mais isto é enganoso. A *visibilidade* do código fonte non é a cuestión; a cuestión importante é aquilo que tes permitido facer co código fonte. Así, a diferenza entre o software propietario e o de código pechado é en gran parte irrelevante, e ambos termos poden ser tratados como sinónimos.

Ás veces *comercial* é empregado como sinónimo de "propietario", mais falando con propiedade, non é o mesmo. O software libre pode ser software comercial. Despois de todo, o software libre pode ser vendido, se os compradores non tiveren restrinxida a distribución de copias. Ademais, tamén pode ser comercializado doutros xeitos como, por exemplo, mediante a venta de soporte, servizos e certificación. Hoxe en día hai empresas multimillonarias baseadas no software libre, polo que claramente non é nin anticomercial nin antiempresarial. Por outra banda, si que *é* antipropietario por natureza, e esta é a característica que o diferenza dos modelos tradicionais de licenza por copia.

Dominio público

Que non ten dereitos de autor, quere dicir, que non hai ninguén co dereito de restrinxir a copia do traballo. Estar no dominio público non é o mesmo que non ter autor. Todo ten un autor, e

mesmo se o autor ou autores dun traballo escollen poñelo no dominio público, isto non muda o feito de que eles escribiron o traballo.

Cando un traballo está no dominio público, o contido do mesmo pode incorporarse nun traballo con dereitos de autor (un traballo con copyright), e a continuación *esa copia* do contido está cuberta baixo os mesmos dereitos de autor que o traballo completo. Mais isto non lle afecta á dispoñibilidade do traballo orixinal, o cal segue estando no dominio público. Así, publicar algo no dominio público é tecnicamente o mesmo que que facelo "libre" de acordo coas directrices da maioría das organizacións de certificación do software libre. Porén, normalmente hai boas razóns para empregar unha licenza en lugar de simplemente publicar no dominio público: mesmo co software libre, algunhas restricións concretas poden ser útiles, non unicamente para o posuidor dos dereitos de autor, senón tamén para os beneficiarios, como se clarifica na seguinte seccións.

Copyleft ("esquerdo de copia")

Unha licenza que emprega as leis de cobertura dos dereitos de autor para alcanzar o resultado oposto ao resultado tradicional dos dereitos de autor. Dependendo a quen lle preguntes, quere dicir tanto licenzas que permiten as liberdades que estamos a narrar aquí, ou, máis estritamente, licenzas que non só permiten esas liberdades, senón que as *impoñen*, estipulando que as liberdades deben acompañar o traballo. A Free Software Foundation emprega exclusivamente a segunda definición; noutros lugares é un complemento: moita xente emprega o termo do mesmo xeito que a FSF, mais outra -incluíndo xente que escribe nos principais medios de comunicación- tende a empregar a primeira definición. Non está claro que toda a xente que emprega o termo é consciente de que hai unha distinción que debe ser feita.

O exemplo clásico da definición máis estrita é a GNU General Public License, a cal estipula que calquera traballo derivado debe ser licenciado baixo a GPL; véxase a sección "*A GPL e a compatibilidade de licenzas*" máis adiante neste capítulo para máis detalles.

Características das licenzas

Embora haxa dispoñibles moitas licenzas de software libre distintas, nos temas importantes todas elas din as mesmas cousas: que calquera pode modificar o código, que calquera pode redistribuílo tanto na súa forma orixinal como modificada, e que os titulares dos dereitos de autor así como os autores non fornecen garantía ningunha (evitar as responsabilidades é especialmente importante debido a que a xente pode empregar versións modificadas mesmo sen sabelo). As diferenzas entre licenzas redúcense a unhas poucas cuestións recorrentes.

Compatibilidade con licenzas propietarias

Algunhas licenzas libres permiten que o código que cobren sexa empregado en programas propietarios. Isto non prexudica as cláusulas de licenza do programa propietario: segue sendo tan propietario como sempre, só que contén algún código de fontes non propietarias. A licenza Apache, X Consortium License, as licenzas estilo BSD e as licenzas estilo MIT son todas elas exemplo de licenzas compatibles con licenzas propietarias.

Compatibilidade con outras licenzas libres

A maioría das licenzas libres son compatibles co resto, quere dicir, o código baixo unha destas licenzas pode ser combinado con código baixo outra delas, e o resultado pode ser distribuído baixo calquera das licenzas se non houber violación dos termos da outra. A principal excepción é a GNU General Public License, que require que calquera traballo que empregue código baixo a licenza GPL debe á súa vez ser distribuído baixo a licenza GPL, e sen engadir ningunha outra restrición para alén das que require a GPL. A licenza GPL é compatible con algunhas licenzas libres, mais non con outras. Isto analízase en maior profundidade na sección "*A GPL e a compatibilidade de licenzas*" máis adiante neste capítulo.

A importancia da acreditación

Algunhas licenzas libres estipulan que calquera uso do código que cubriren debe acompañarse dun aviso, cuxo emprazamento e contido normalmente está especificado, dándolle crédito aos autores ou aos titulares dos dereitos de autor do código. Estas licenzas habitualmente son compatibles coas propietarias: non requiren necesariamente que o traballo derivado sexa libre, simplemente requiren que se lle dea crédito ao código libre.

Protección da marca comercial

Unha variante da obriga de acreditación. As licenzas de protección das marcas comerciais especifican que o nome do software orixinal (ou dos titulares dos dereitos de copia, ou da súa institución, etc.) *non* deben ser empregados nos traballos derivados sen o previo permiso por escrito. Aínda que a obriga de acreditar insista en que se empregue un nome concreto, e a protección de marcas comerciais insista en que non sexa empregado, ambas son expresións do mesmo desexo: que a reputación do código orixinal sexa preservada e transmitida, mais que non se desvirtúe por asociación.

Loita contra as patentes

Tanto a GNU General Public License versión 3 como a Apache Licese version 2 conteñen cláusulas deseñadas para previr que haxa xente que poida empregar a lexislación de patentes para eliminar os dereitos (baixo a lei de dereitos de autor) asegurados pola licenza. Ambas requiren que os contribuíntes outorguen automaticamente licenzas de patente sobre toda a súa contribución, cubrindo as mesmas calquera patente licenciable polo contribuínte que puidera ser infrinxida pola súa contribución (ou pola incorporación da súa contribución no conxunto do traballo completo). Pero sobre esta base, aínda foron máis lonxe: se calquera persoa que empregue software baixo estas licenzas comeza un xuízo contra outra parte, denunciando que o traballo cuberto infrinxe algunha patente do denunciante, automaticamente o denunciante *perde* todos os dereitos de patente fornecidos pola licenza para ese traballo, e no caso da GPLv3 incluso perde os seus dereitos de distribución baixo a licenza.

Protección da "integridade artística"

Algunhas licenzas (a Licenza Artística, empregada pola implementación máis popular da linguaxe de programación Perl, e a Licenza Tex de Donald Knuth, por exemplo) requiren que a modificación e redistribución sexa feita de tal xeito que distinga claramente entre a versión orixinal do código e calquera modificación. Estas licenzas permiten esencialmente as mesmas liberdades que o resto de licenzas libres, mais impoñen certos requisitos que permitan verificar

de xeito sinxelo a integridade do código orixinal. Estas licenzas non despertaron moito interese para alén de nos programas para os cales foron feitas, e non serán tratadas neste capítulo; mencionáronse aquí só para non deixarmos cabos soltos.

A maioría destas disposicións non son mutuamente excluentes, e algunhas licenzas inclúen varias. O fío común entre elas é a imposición de regras sobre o receptor a mudanza do dereito do receptor para empregar e/ou redistribuír o código. Por exemplo, algúns proxectos queren que o seu nome e reputación sexan transmitidos conxuntamente co código, o que provoca a imposición das cláusulas de crédito ou protección da marca comercial; dependendo da súa onerosidade, esta carga adicional pode provocar que algúns usuarios escollan un pacote con unha licenza menos esixente.

A GPL e a compatibilidade entre licenzas

Con diferenza, a maior liña divisoria no licenciamento é a que separa as licenzas compatibles co software propietario das licenzas incompatibles co software propietario, é dicir, entre a GNU Public License e todas as demais. Debido a que o obxectivo principal dos autores da GPL é a promoción do software libre, deliberadamente crearon este licenza de xeito que fose imposible mesturar código baixo a GPL dentro dos programas propietarios. Especificamente, dentro dos requirimentos da GPL (en http://www.fsf.org/licensing/licenses/gpl.html podes ver o texto completo) están estes dous:

1. Calquera traballo derivado -isto é, calquera traballo que conteña unha cantidade non trivial de código baixo a licenza GPL- debe á súa vez ser distribuído baixo a licenza GPL.
2. Na redistribución, tanto do traballo orixinal como dun traballo derivado, non se pode engadir ningunha restrición adicional. (a frase literal é: "non pode impoñer ningunha restrición adicional no exercicio dos dereitos concedidos ou afirmados baixo esta licenza").

Con estas condicións, a licenza GPL ten éxito en facer contaxiosa a liberdade. Unha vez que un programa se publica baixo a GPL, as súas cláusulas de redistribución son *virais* -transfírense a calquera cousa á que for incorporado ese código, facendo que sexa efectivamente imposible empregar código GPL en programas de código pechado.
Porén, estas mesmas cláusulas provocan que a GPL sexa incompatible con algunhas outras licenzas de software libre. O xeito habitual no que isto sucede é cando a outra licenza impón un requisito -por exemplo, unha cláusula de acreditación dos autores orixinais a mencionar dalgún xeito- que sexa incompatible co requisito propio da GPL ("vostede non pode impoñer ningunha outra restrición..."). Desde o punto de vista da Free Software Foundation, estes efectos colaterais son desexables, ou cando menos non son prexudiciais. A GPL non só mantén libre o teu software, senón que tamén torna o teu software nun axente no impulso de que *outro* software á súa vez impoña a liberdade.

A cuestión sobre se este é ou non un bo camiño para promover o software libre é unha das guerras santas máis persistentes en Internet (bótalle unha ollada á sección "*Evitando Guerras Santas*" no *Capítulo 6 Comunicacións*), e non imos investigala aquí. O importante para os nosos propósitos é que a compatibilidade coa GPL é un tema importante á hora de escollermos unha licenza. A GPL é, con diferenza, a licenza de software libre máis popular; en http://freshmeat.net/stats/#license , emprégase en 68% do software, e a seguinte licenza máis empregada ten unha porcentaxe de uso de 6%. Se quixeres que o teu código sexa mesturable con código GPL -e hai unha morea de código GPL aí fóra- entón deberías escoller unha licenza compatible coa GPL. A maioría das licenzas libres compatibles coa GPL tamén son compatibles coas licenzas propietarias; isto é, o código baixo estas licenzas poder

empregarse tanto nun programa GPL como nun programa propietario. Por suposto, o *resultado* destas misturas non será compatible entre elas, dado que un estaría baixo a GPL e o outro baixo unha licenza de código pechado. Mais esta preocupación ten a ver unicamente con traballos derivados, non co código que ti distribúas.

Afortunadamente, a Free Software Foundation mantén unha lista amosando que licenzas son compatibles coa GPL e cales non, en http://www.gnu.org/licenses/license-list.html . Todas as licenzas tratadas neste capítulo están presentes na devandita lista, nun lado ou no outro.

Escollendo unha licenza

Cando escolleres unha licenza para o teu proxecto, se for posible emprega unha licenza xa existente en lugar de creares unha nova. Hai dúas razóns polas que as licenzas existentes son mellores:

- Familiaridade. Se empregares unha das tres ou catro licenzas máis populares, a xente non vai percibir a necesidade de ler os textos legais para poder empregar o teu código, xa que ese estudo legal para a licenza xa foi feito moito tempo atrás.
- Calidade. A non ser que tiveres un equipo de avogados á túa disposición, é pouco probable que vaias crear unha licenza legalmente sólida. As licenzas mencionadas neste capítulo son o produto de moito traballo e experiencia; salvo que o teu proxecto teña necesidades realmente infrecuentes, non é probable que o vaias facer mellor.

Para aplicares unha destas licenzas ao teu proxecto, bótalle unha ollada á sección "*Como aplicar unha licenza ao teu software*" no *Capítulo 2 Primeiros Pasos*

A MIT / X Window System License

Se o teu obxectivo é que o teu código sexa accesible para a maior cantidade posible de desenvolvedores e de traballos derivados, e non che importa que o código vaia ser empregado en programas propietarios, entón escolle a MIT / X Window System license (así chamada debido a que é a licenza baixo a que o Instituto de Tecnoloxía de Massachusetts publicou o código do sistema X Windows orixinal). A mensaxe básica desta licenza é: "Es libre para empregares este código como quixeres". Esta licenza é incompatible coa GNU GPL, e é curta, sinxela e fácil de entender (NOTA: a continuación a licenza reprodúcese no seu idioma orixinal, dado que para a mesma non existe tradución legalmente válida a outros idiomas):

> Copyright (c) <year> <copyright holders>
>
> Permission is hereby granted, free of charge, to any person obtaining
> a copy of this software and associated documentation files (the
> "Software"), to deal in the Software without restriction, including
> without limitation the rights to use, copy, modify, merge, publish,
> distribute, sublicense, and/or sell copies of the Software, and to
> permit persons to whom the Software is furnished to do so, subject to
> the following conditions:
>
> The above copyright notice and this permission notice shall be

included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

(Tomada de http://www.opensource.org/licenses/mit-license.php)

A GNU General Public License

Se preferires que o código do teu proxecto non sexa empregado en programas propietarios, ou se cando menos non che importa que se poida ou non empregar en programas propietarios, escolle a GNU General Public License (http://www.fsf.org/licensing/licenses/gpl.html). A GPL probablemente sexa a licenza de software libre máis amplamente empregada no mundo hoxe en día; o feito de ser tan coñecida é en si mesmo unha das maiores vantaxes da GPL.

Cando programares unha biblioteca de código destinada principalmente a ser usada como parte doutros programas, pensa coidadosamente se as restricións impostas pola GPL están aliñadas cos obxectivos do teu proxecto. Nalgúns casos -por exemplo cando tentares desbancar unha biblioteca propietaria que fai o mesmo- é estratexicamente máis asisado licenciares o teu código de tal xeito que poida ser mesturado en programas propietarios, embora esta non sexa a situación que desexaras. A Free Software Foundation preparou unha alternativa á GPL precisamente para estas circunstancias: a *GNU Library GPL*, máis tarde rebautizada como *GNU Lesser GPL* (de todas maneiras, a maioría da xente simplemente emprega o acrónimo *LGPL*. A LGPL ten unhas restricións menores que a GPL, e pode mesturarse dun xeito máis sinxelo con código non libre. Porén, trátase dunha licenza complexa e leva algún tempo entendela, polo que se non vas empregar a GPL, recoméndoche empregares licenzas estilo MIT/X.

A GNU Affero GPL; unha versión da GNU GPL para código executado en servidor

En 2007, a Free Software Foundation publicou unha variante da GPL chamada *GNU Affero GPL* http://www.fsf.org/licensing/licenses/agpl.html [23]. O seu obxectivo é fortalecer a

23 A historia desta licenza e do seu nome é un pouco complicada. A primeira versión da licenza foi orixinalmente publicada pola empresa Affero Inc. baseándose na versión 2 da GNU GPL. Coñecíase habitualmente como a AGPL. Máis tarde, a Free Software Foundation decidiu adoptar a idea, mais naquela altura tiñan liberada a versión 3 da súa licenza GNU GPL, polo que basearon a súa licenza "Afferizada" na GPL v3 e chamáronlle "GNU AGPL". A vella licenza Affero está máis ou menos obsoleta a día de hoxe. Se quixeres garantías e seguridades ao estilo das que che ofrece Affero, deberías empregar a versión GNU. Para eliminares a ambigüidade, chámalle "AGPLv3", "GNU AGPL", ou con máxima precisión "GNU AGPLv3".

seguridade e garantía de que se comparte o código neste entorno actual dun crecente número de empresas que ofrecen os seus servizos desde servidor -é dicir, software que corre nos seus servidores, co que os usuarios interaxen unicamente sobre a rede, e que nunca se distribúe directamente aos usuarios nin como código executable nin código fonte. Un número enorme de servizos estiveron empregando software GPL, habitualmente con modificacións, sen ter que compartir as súas mudanzas co mundo debido a que non estaban distribuíndo ningún código (xa que o código en cuestión se executa nos seus servidores e non nos equipos dos usuarios).

A solución achegada a este respecto pola GNU AGPLv3 foi coller a GPL e engadirlle unha cláusula de "interacción remota por rede" propoñendo "*...se ti modificares o Programa, a túa versión modificada deberá ofrecerlle claramente a todos os usuarios que interaxiren con el remotamente a través dunha rede de computadores... a posibilidade de recibiren o código fonte correspondente á túa versión... sen custo, a través dalgún medio estándar ou personalizado para facilitar a copia de software*". Esta cláusula expande os poderes de potenciación da GPL no novo mundo dos fornecedores de servizos e aplicacións pola rede.

Ten en conta que a AGPLv3 non é directamente compatible coa GPLv2 (embora sexa compatible coa GPLv3, por suposto). Porén, a maioría do software licenciado baixo a GPLv2 inclúe a cláusula "ou calquera versión posterior", co cal podes simplemente promocionalo a GPLv3 cando precisares mesturalo con código AGPLv3. Porén, se precisares mesturar código con programas licenciados estritamente baixo a GPLv2 (isto é, sen a cláusula "ou calquera versión posterior"), entón a AGPLv3 non che vai valer.

Aínda que a historia da AGPLv3 é un pouco complicada, a licenza en si mesma é sinxela: é simplemente unha GPLv3 con unha cláusula extra que trata a interacción a través da rede. O artigo da Wikipedia sobre a AGPLv3 é excelente: http://en.wikipedia.org/wiki/Affero_General_Public_License

É a GPL unha licenza libre?

Unha consecuencia de escoller a GPL é a posibilidade -pequena, mais non infinitesimal- de atopáreste a ti mesmo ou o teu proxecto inmerso nunha disputa sobre se a GPL é verdadeiramente "libre" ou non, dado que establece algunhas restricións sobre o que podes facer co código -nomeadamente, a restrición de que o código non poida ser distribuído baixo ningunha outra licenza. Para algunha xente, a existencia desta restrición quere dicir que a GPL é "menos libre" que outras licenzas máis permisivas como as licenzas tipo MIT/X. Esta argumentación normalmente apunta, por suposto, a que "máis libre" debe ser mellor que "menos libre" (despois de todo, quen non está a favor da liberdade?), ao que lle segue que esas outras licenzas son mellores que a GPL.

Este debate é outra popular guerra santa (bótalle unha ollada á sección *"Evitando Guerras Santas"* no *Capítulo 6 Comunicacións*). Evita participares nela, cando menos nos foros do proxecto. Non intentes demostrar que a GPL é menos libre, tan libre ou máis libre que outras licenzas. En lugar diso, fai énfase nas razóns específicas polas que o teu proxecto escolleu a GPL. Se a facilidade para recoñecer a licenza foi unha razón, dio. Se a obriga de licenciamento libre dos traballos derivados tamén foi unha razón, dio, mais evita entrar en disputas sobre se isto torna o código máis ou menos "libre". A liberdade é un tema complexo, e é unha perda de tempo falar dela se a terminoloxía vai ser o cabalo de batalla principal.

Embora sexa isto un libro e non un fío dunha lista de correo, teño que admitir que nunca entendín o argumento "a GPL non é libre". A única restrición que a GPL impón é evitar que a xente impoña *maiores* restricións. Dicir que isto quere dicir ter menos liberdade sempre me pareceu o mesmo que dicir que a abolición da escravitude reduciu a liberdade porque evitou que algunha xente puidera ter escravos.

(E se te vires no medio dun debate sobre isto, non subas a aposta facendo analoxías incendiarias).

Que tal a licenza BSD?

Unha gran cantidade de software libre é distribuído baixo a *licenza BSD* (ou algunhas veces baixo unha *licenza estilo BSD*). A licenza BSD orixinal foi empregada pola Berkeley Software Distribution, na que a Universidade de California publicou partes importantes dunha implementación de Unix. Esta licenza (cuxo texto exacto pode verse na sección 2.2.2 de http://www.xfree86.org/3.3.6/COPYRIGHT2.html#6) era similar en esencia á licenza MIT/X, excepto por unha cláusula (da que a continuación se fai unha tradución libre sen validez legal ningunha):

> *Todo material publicitado que mencionar características ou empregar este software deberá mostrar a seguinte advertencia: "Este produto contén software desenvolvido pola Universidade de California, Lawrence Berkeley Laboratory*

A presenza desta cláusula no só fai que a BSD sexa incompatible coa GPL, senón que tamén senta un perigoso precedente: mentres outras organizacións poñan cláusulas publicitarias similares no *seu* software libre -substituíndo o seu propio nome en lugar de "a Universidade de California, Lawrence Berkeley Laboratory- os redistribuidores do software enfróntanse a unha crecente carga en canto ao que se lles require mostrar. Afortunadamente, moitos dos proxectos que empregaron esta licenza decatáronse do problema, e simplemente eliminaron esta cláusula. En 1999, ,mesmo a Universidade de California fixo iso.
O resultado é a licenza BSD revisada, que simplemente é a licenza BSD orixinal sen a cláusula publicitaria. Porén, esta historia fai que a expresión "licenza BSD" sexa un pouco ambigua: refírese á orixinal, ou á versión revisada? Isto é polo que prefiro a licenza MIT/X, a cal é equivalente en esencia, e non está suxeita a ningunha ambigüidade. Porén, pode que si que haxa unha razón para preferir a BSD revisada fronte á licenza MIT/X, que é que a BSD inclúe esta cláusula:

> *Nin o nome da <ORGANIZACIÓN> nin os nomes dos seus contribuíntes deben empregarse para apoiar ou promocionar produtos derivados deste software sen permiso previo explícito por escrito.*

Non está claro que sen unha cláusula coma esta un receptor do software puidese ter o dereito de empregar o nome do autor, mais a cláusula borra calquera posible dúbida. Para organizacións preocupadas polo control das marcas comerciais, polo tanto, a licenza BSD revisada pode ser lixeiramente preferible á MIT/X. En xeral, porén, unha licenza liberal de cara aos dereitos de autor non implica que os receptores teñan ningún dereito ao uso ou dilución das túas marcas comerciais -a lexislación dos dereitos de autor e das marcas comerciais son dúas lexislacións distintas.

Se quixeres empregar a licenza BSD revisada, tes dispoñible un molde en

http://www.opensource.org/licenses/bsd-license.php

Asignación e propiedade dos dereitos de autor (copyright)

Hai tres xeitos de manexar a propiedade dos dereitos de autor para o código e documentación libres contribuídos por varias persoas. O primeiro é ignorando complemente a cuestión dos dereitos de autor (non o recomendo), O segundo xeito é recollendo un *acordo de licenza do contribuínte* (*CLA* en inglés, xa que provén do *contributor license agreement*) por parte de cada persoa que traballa no proxecto, no cal se garanta explicitamente o dereito do proxecto a empregar a contribución desa persoa. Isto normalmente é suficiente para a maioría dos proxectos, e o positivo é que en varias xurisdicións, os CLAs poden enviarse por correo electrónico. O terceiro xeito é obter asignacións de dereitos de autor por parte dos contribuíntes, de forma que o proxecto (por exemplo, unha entidade legal, habitualmente sen ánimo de lucro) sexa o propietario dos dereitos de autor. Este é o xeito legalmente máis hermético, mais tamén é o xeito máis traballoso/gravoso para os contribuíntes; só uns poucos proxectos insisten neste punto[24].

Hai que ter presente que mesmo baixo unha propiedade centralizada dos dereitos de autor, o código[25] permanece libre, xa que as licenzas de software libre non lle dan ao titular dos dereitos de autor o dereito de apropiarse retroactivamente de todas as copias do código. Debido a isto, mesmo se o proxecto, como entidade legal, de repente dese un xiro e comezase a distribuír todo o código baixo unha licenza restritiva, isto non lle causaría un problema á comunidade pública. Os outros desenvolvedores comezarían unha rama (fork) baseándose na última copia libre do código, e continuarían como se nada pasase. Como os desenvolvedores saben que poden facelo, a maioría dos contribuíntes cooperan cando se lles pide que asinen un CLA ou que fagan unha transferencia dos dereitos de autor.

Non facer nada

Moitos proxectos nunca recollen CLAs ou transferencias de dereitos de autor dos seus contribuíntes. En lugar disto, simplemente aceptan código cando parece razoablemente claro que o contribuínte ten a intención de que se inclúa no proxecto.

Baixo circunstancias normais, isto é suficiente. Mais de cando en vez, alguén pode decidir demandar por infrinximento dos seus dereitos de autor, alegando que é o auténtico propietario do código en cuestión, e que nunca estivo de acordo en que fose distribuído polo proxecto baixo unha licenza de software libre. Por exemplo, o Grupo SCO fixolle algo similar ao proxecto Linux, bótalle unha ollada a http://en.wikipedia.org/wiki/SCO-Linux_controversies para saberes os detalles. Se acontecer isto, o proxecto non vai ter documentación na que se observe que o contribuínte formalmente concedeu o dereito de uso do código, o que vai dificultar a defensa legal.

---

24 Hai que ter en conta que a transferencia dos dereitos de autor está suxeita ás leis nacionais, e que as licenzas deseñadas para os Estados Unidos poden atopar problemas noutros lugares (por exemplo, en Alemaña, onde aparentemente non é posible transferir os dereitos de autor)

25 Vou empregar "código" para referirme tanto ao código fonte como á documentación, a partir de agora

Acordos de licenza do contribuínte (Contributor License Agreements)

Os CLAs probablemente ofrezan o mellor compromiso entre seguridade e conveniencia. Un CLA é un formulario electrónico que un desenvolvedor preenche e lle envía ao proxecto. En moitas xurisdicións, o envío vía correo electrónico é suficiente. Unha sinatura dixital segura pode ou non ser precisa; consulta a un avogado para atopares que método é o mellor para o teu proxecto.

Moitos proxectos empregan dous tipos lixeiramente distintos de CLAs, un para individuos e outro para contribuíntes corporativos. Aínda así, en ambos casos a linguaxe central é a mesma: o contribuínte garántelle ao proxecto <emphasis>"unha licenza de copyright perpetua, global, non exclusiva, sen cargo, libre de regalías por dereitos de autor e irrevogable para reproducir, facer traballos derivados, mostrar publicamente, executar publicamente, licenciar a terceiros, e distribuír as contribucións e os traballos derivados."</emphasis>. Unha vez máis, debes ter un avogado para aprobares calquera CLA, mais se introducires todos estes adxectivos no mesmo, probablemente vaia ben.

Cando solicitares CLAs por parte dos teus contribuíntes, asegúrate de resaltar que ti *non* estás solicitando unha transferencia/asignación de dereitos de autor. De feito, moitos CLAs comezan recordándolle ao lector o seguinte:

> *Isto só é un acordo de licenza; non transfire a propiedade dos dereitos de autor e non muda os teus dereitos de empregares a túa propia contribución para calquera propósito.*

Aquí van algúns exemplos:

- CLA para contribuínte individual:
  - http://apache.org/licenses/icla.txt
  - http://code.google.com/legal/individual-cla-v1.0.html

- CLA para contribuínte corporativo:
  - http://apache.org/licenses/cla-corporate.txt
  - http://code.google.com/legal/corporate-cla-v1.0.html

Transferencia dos dereitos de autor

A transferencia dos dereitos de autor quere dicir que o contribuínte lle asigna ao proxecto a titularidade e propiedade dos dereitos de autor sobre as súas contribucións. Idealmente, isto faise en papel e envíaselle por fax ou correo postal ao proxecto.

Algúns proxectos insisten na asignación completa debido a que ter unha única entidade legal como propietaria dos dereitos de autor de todo o código fonte pode ser útil se as cláusulas da licenza de software libre algunha vez deben facerse valer nun xulgado. Se non existir unha entidade única que tiver o dereito de facelo, todos os contribuíntes deberían cooperar, mais algúns poderían non ter o tempo preciso, ou ser imposible contactar con eles cando xurdir a cuestión.

Distintas organizacións aplican un grao de rigor diferente na tarefa de recolección das asignacións. Algunhas simplemente obteñen un manifesto informal do contribuínte nunha lista de correo pública

-algo na liña de "Pola presente asígnolle os dereitos de autor deste código ao proxecto, para que sexa licenciado baixo as mesmas condicións que o resto do código". Cando menos un dos avogados cos que falei dixo que isto é suficiente, presumiblemente porque sucede nun contexto no que as asignacións dos dereitos de autor son normais e esperadas, así como porque representa un esforzo de boa fe por parte do proxecto para averiguar as intencións reais do desenvolvedor. Por outra banda, a Free Software Foundation está no outro estremo: requírelle aos contribuíntes a sinatura e envío físico por correo postal dun anaco de papel que conteña unha manifestación formal da transferencia dos dereitos de autor, ás veces só por unha contribución, ás veces para todas as contribucións presentes e futuras. Se o desenvolvedor é empregado seu, a FSF pídelle que o asine igualmente.

A paranoia da FSF é comprensible. Se alguén violar os termos da GPL incorporando algún do seu software nun programa propietario, a FSF vai ter que ir a xuízo, e van querer ter os seus dereitos de autor tan blindados como sexa posible cando isto suceda.

Esquemas de licenciamento dual

Algúns proxectos tentan financiarse empregando un esquema de licenciamento dual no cal os traballos derivados propietarios deben pagarlle ao propietario dos dereitos de autor para teren o dereito de usaren o código, mais o código segue sendo libre para o seu uso por proxectos software libre. Isto tende a funcionar mellor con bibliotecas de código que con aplicacións de escritorio, naturalmente. Os termos exactos difiren entre os distintos casos. A GNU GPL acostuma a ser a licenza para o lado libre, dado que evita que terceiros poidan incorporar o código cuberto dentro de produtos propietarios sen o permiso do titular dos dereitos de autor, mais ás veces emprégase unha licenza personalizada que ten o mesmo efecto. Un exemplo do primeiro é a licenza de MySQL, descrita en http://www.mysql.com/company/legal/licensing/ un exemplo do segundo é a estratexia de licenciamento do software de Sleepycat, descrita en http://www.oracle.com/technology/software/products/berkeley-db/htdocs/licensing.html

Estaraste preguntando: como pode o titular dos dereitos de autor ofrecer licenzas propietarias en troca dun pagamento obrigatorio se os termos da GNU GPL estipulan que o código debe estar dispoñible baixo termos menos restritivos? A resposta é que os termos da GPL son algo que o titular dos dereitos de autor lle impón ao resto da xente; polo cal o propietario é libre de decidir *non* aplicarse eses termos a si mesmo. Un bo xeito de pensalo é imaxinando que o titular dos dereitos de autor ten un número infinito de copias do software gardadas nunha artesa. Cada vez que saca unha da artesa para darlla ao mundo, pode decidir que licenza lle pon: GPL, propietaria ou calquera outra. O seu dereito a facelo non está restrinxido pola GPL ou por ningunha outra licenza de software libre: simplemente é un poder que lle vén garantido pola lei de dereitos de autor.

O atractivo do licenciamento dual é que, no mellor caso, fornece un medio para obter un caudal de ingresos fiable para un proxecto de software libre. Infelizmente, á súa vez tamén pode interferir coas dinámicas normais dos proxectos de software libre. O problema está en que calquera voluntario que faga unha contribución de código está a contribuír con dúas entidades distintas: a versión libre do código e maila versión propietaria. Mentres que o contribuínte se vai sentir cómodo contribuíndo á versión libre, pode non sentirse cómodo ao contribuír á fonte de ingresos semi-propietaria dun terceiro. A incomodidade exacérbase polo feito de que no licenciamento dual, o titular dos dereitos de autor realmente precisa obter formalmente unha transferencia asinada de dereitos de autor por parte de tódolos contribuíntes, co fin de protexerse a si mesmo dun contribuínte desgustado reclamando unha

porcentaxe das regalías obtidas da fonte de ingresos propietaria. O proceso de recollida destas transferencias implica que os contribuíntes se enfronten ao feito de que están a facer traballo que redunda en diñeiro para un terceiro alleo a eles mesmos.

Non todos os voluntarios se van sentir molestos por isto; despois de todo, as súas contribucións tamén se incorporan na edición software libre, e aí é onde reside o seu principal interese. Non obstante, o licenciamento dual é un exemplo do titular dos dereitos de autor asignándose a si mesmo un dereito especial que o resto dos integrantes do proxecto non teñen, e isto pode provocar a aparición de tensións nalgúns momentos, cando menos con algúns voluntarios.

O que parece ocorrer na práctica é que as empresas baseadas en software con licenciamento dual realmente non teñen comunidades de desenvolvemento igualitarias. Obteñen correccións de erros a pequena escala e remendos de limpeza desde recursos externos, mais acaban facendo a maior parte do traballo duro mediante recursos internos. Por exemplo, Zack Urlocker, vicepresidente de marketing en MySQL, díxome que a empresa de todos xeitos xeralmente acaba por contratar os voluntarios máis activos. Embora o produto en si mesmo sexa software libre, licenciado baixo a GPL, o seu desenvolvemento é máis ou menos controlado desde a empresa, aínda que coa posibilidade (extremadamente pouco probable) de que alguén realmente insatisfeito coa xestión da empresa decida facer unha rama (fork) do proxecto. Descoñécese como as políticas da empresa se adaptan a esta ameaza, mais en calquera caso, MySQL non parece ter problemas de aceptación nin no mundo do software libre nin para alén deste.

Patentes

As patentes do software son a vareada do momento no lombo do software libre, porque supoñen a única ameaza contra a que a comunidade do software libre non se pode defender por si mesma. Os problemas dos dereitos de autor e das marcas comerciais sempre poden ser evitados .Se unha parte do teu código parece que pode infrinxir os dereitos de autor dun terceiro, ti podes simplemente reescribir esa parte. Se resulta que alguén ten rexistrado como marca comercial o nome do teu proxecto, no peor caso podes simplemente renomear o proxecto. Embora a mudanza de nome poida supoñer temporalmente un contratempo, non vai importar a longo prazo, dado que o código en si mesmo vai seguir facendo o que sempre fixo.

Mais unha patente é un mandamento global contra a implementación de determinada ideas. Non importa quen escribise o código, nin que linguaxe de programación fose empregada. Unha vez que alguén teña acusado un proxecto de software libre da infrinxencia dunha patente, o proxecto debe ou ben parar de implementar esa característica particular, ou ben enfrontarse a un caro e longo proceso xudicial. Dado que os instigadores destes xuízos son habitualmente corporacións con grandes petos -precisamente son eles quen teñen os recursos e afinidades precisas para patentaren nun primeiro lugar- a maioría dos proxectos de software libre non poden afrontar a segunda opción, polo que se renden inmediatamente aínda que pensen que a patente podería inutilizarse nun xuízo. Para evitaren chegar a esta situación nun primeiro lugar, os proxectos de software libre están comezando a codificar defensivamente, eliminando algoritmos patentados xa de primeiras, mesmo cando son a mellor ou única solución para un problema de programación[26].

26 Sun Microsystems e IBM polo menos teñen feito un xesto cara este problema desde a outra dirección, mediante a liberación dun gran número de patentes de software -1600 e 500 respectivamente- para o uso pola comunidade de software libre. Non son avogado e por iso non podo

Tanto os inquéritos, como a evidencia anecdótica, mostran que non só a inmensa maioría dos programadores de software libre, senón que a maioría de *todos* os programadores pensan que as patentes de software deberían ser totalmente abolidas[27]. Os programadores de software libre están moi convencidos disto, e poderían rexeitar traballar en proxectos que están demasiado proximamente asociados coa recolección ou fomento das patentes do software. Se a túa organización recoller patentes de software, fai público dun xeito irrevogable que as patentes nunca se empregarían contra proxectos de software libre, e que unicamente se empregarían como defensa en caso de que outra parte comezara un proceso de infrinxencia de patentes contra a túa organización. Isto non é soamente o máis correcto que podes facer, tamén é algo bo para as túas relacións co software libre[28].

Infelizmente, a recolección de patentes con propósitos defensivos é unha acción racional. O sistema de patentes actual, cando menos nos Estados Unidos, é por natureza unha carreira armamentística: se a túa competencia adquiriu unha morea de patentes, entón a túa mellor defensa é que ti tamén adquiras outra morea de patentes, de xeito que se ten acusaren de infrinxires unha patente, ti poidas responder con unha ameaza similar -neste caso as dúas partes acostuman a sentarse e traballar nun acordo de cruce de patentes para que ningunha das dúas partes teña que pagar nada, excepto aos seus avogados especialistas en propiedade intelectual, por suposto.

O dano feito ao software libre polas patentes do software é máis insidioso que unicamente as ameazas directas ao desenvolvemento de software. As patentes de software fomentan unha atmosfera de secretismo entre os deseñadores de *firmware*, os cales están (xustificadamente) preocupados con publicar detalles das súas interfaces que lle dean axuda técnica á súa competencia na súa procura de golpealos con xuízos de infrinxencia de patentes. Isto non é só un perigo teórico: aparentemente está sucedendo desde hai tempo na industria das tarxetas gráficas, por exemplo. Moitos fabricantes de tarxetas gráficas son reticentes a publicaren as especificacións programáticas precisas para produciren controladores de código aberto de alto rendemento para as tarxetas, provocando que sexa imposible para os sistemas operativos libres darlle soporte a esas tarxetas para despregar completamente o seu potencial. Por que farían isto os fabricantes? Para eles non ten sentido traballar *contra* o soporte software; despois de todo, a compatibilidade con máis sistemas operativos só pode supor unha maior venda de tarxetas. Mais resulta que tras da porta de deseño, estes fabricantes están violando patentes de outros, algunhas veces deliberadamente e outras veces accidentalmente. As patentes son tan imprevisibles e potencialmente tan extensas que ningún fabricante de tarxetas nunca pode estar seguro, mesmo despois de ter feita unha procura de patentes. Debido a isto, os fabricantes non se atreven a publicar a especificación completa de interfaces, dado que podería facerlle máis sinxelo á competencia supor se algunha patente está sendo infrinxida. (Por suposto, a natureza desta situación é tal que non vas atopar ningunha declaración escrita de ningunha fonte primaria acerca do que está a ocorrer: isto seino a través de comunicacións persoais).

Algunhas licenzas de software libre teñen cláusulas especiais para combater, ou cando menos frear, as

---

avaliar a utilidade real destas concesións, mais mesmo se todas elas foren patentes importantes, e as condicións das concesións permitiren que realmente sexan de uso libre por calquera proxecto de software libre, non deixaría de ser unha gota no océano.

27Bótalle unha ollada a http://groups.csail.mit.edu/mac/projects/lpf/Whatsnew/survey.html para veres unha enquisa deste estilo

28 Por exemplo, Red Hat prometeu que os proxectos de software libre están seguros fronte ás súas patentes, tal como podes ver en http://www.redhat.com/legal/patent_policy.html

patentes de software. A GNU GPL, por exemplo, contén o seguinte (ten en conta que a continuación faise unha tradución NON oficial da "GNU General Public License" ao galego. Esta tradución non foi publicada po-la "FSF Free Software Foundation", e non apoia legalmente os termos de distribución do software que empregue a "GNU GPL", só o fai o texto orixinal en inglés. Porén, esperamos que esta tradución lle axude ás persoas galegofalantes a entenderen mellor a " GNU GPL"):

> 7. Se como consecuencia dun veredicto dun xulgado ou pola alegación de infrinxir unha patente ou por calquera outra razón (non limitada unicamente a cuestións de patentes) foren impostas condicións sobre vostede que contradiciren as cláusulas desta Licenza, estas non o eximen das cláusulas aquí descritas.
> Se vostede non puider distribuír o produto cumprindo totalmente coas obrigas relativas á resolución oficial e ao mesmo tempo coas obrigas que se describen neste contrato de Licenza, entón non poderá distribuír máis este produto. Por exemplo, se unha licenza de patente non permitir a distribución do Programa de forma libre de regalías (sen pagamento de regalías) por parte de quen as recibiren directa o indirectamente, entón a única forma de cumprir con ambas obrigas é renunciar á distribución do mesmo.
>
> [...]
>
> A intención desta sección non é a de inducilo a infrinxir ningunha lei de patentes, nin tampouco infrinxir ningún reclamo de dereitos, nin discutir a validez de tales reclamos; esta sección ten o único propósito de protexer a integridade do sistema de distribución do software libre, que está implementado por prácticas de licenza pública. Moita xente ten feito xenerosas contribucións á ampla gama de software distribuído baixo este sistema favorecendo así a constante aplicación deste sistema de distribución; é decisión do autor/doador se o seu Programa será distribuído empregando este ou outro sistema de distribución, e a persoa que recibir o software non pode obrigalo a facer ningunha escolla en particular.

A Apache License, Versión 2.0 (http://www.apache.org/licenses/LICENSE-2.0) tamén contén requisitos anti-patentes. Primeiro, estipula que calquera que distribúa código baixo esta licenza debe implicitamente incluír unha licenza libre de regalías para calquera patente que puidese ostentar e que se puidera aplicar ao código. Segundo, e máis enxeñoso, castiga a calquera que comezar un proceso por infrinxencia de patentes sobre o código cuberto, finalizando automaticamente a súa licenza de patente implícita no momento no que se fixer a devandita reclamación (a continuación transcríbese en idioma orixinal a cláusula en cuestión, dado que é a única versión con validez legal):

> 3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to

> make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

Aínda que é útil, tanto legal como politicamente, construír defensas contras as patentes para as licenzas de software libre, ao final estes pasos non son suficientes para anular os efectos nocivos que a ameaza das patentes ten sobre o software libre. Só mudanzas na substancia ou interpretación da lei internacional de patentes poderían facer isto. Para aprenderes máis sobre este problema, e como se está combatendo, diríxete a http://www.nosoftwarepatents.com . O artigo da Wikipedia inglesa http://en.wikipedia.org/wiki/Software_patent , así como os respectivos artigos noutros idiomas como o galego, tamén teñen moita información útil sobre patentes de software. Escribín unha entrada no meu blog resumindo os argumentos en contra das patentes de software en http://www.rants.org/2007/05/01/how-to-tell-that-software-patents-are-a-bad-idea/

Recursos adicionais

Este capítulo só foi unha introdución ás cuestións do licenciamento do software libre. Porén, espero que conteña información de abondo para que poidas comezar o teu propio proxecto de software libre. Calquera investigación seria sobre cuestións de licenciamento pode facilmente aumentar a información que este libro pode fornecer. Aquí vai unha lista de recursos adicionais sobre licenciamento de software libre:

- *Understanding Open Source and Free Software Licensing* por Andrew M. St. Laurent. Publicado por O'Reilly Media, primeira edición en Agosto de 2004, ISBN: 0-596-00581-4.

  Este é un libro completo sobre licenciamento de software libre en toda a súa complexidade, incluíndo moitos temas omitidos neste capítulo. Bótalle unha ollada a <ulink url="http://www.oreilly.com/catalog/osfreesoft/"></ulink> para máis detalles.

- *Make Your Open Source Software GPL-Compatible. Or Else.*por David A. Wheeler, en http://www.dwheeler.com/essays/gpl-compatible.html

  Este é un artigo detallado e ben escrito sobre a importancia de empregar unha licenza GPL compatible mesmo se non empregares a GPL en si mesma. O artigo tamén toca moitas outras cuestións relativas ao licenciamento, e ten unha alta densidade de excelentes ligazóns.

- http://creativecommons.org

Creative Commons é unha organización que fomenta unha familia de dereitos de autor máis flexibles e liberais que os potenciados polos dereitos de autor tradicionais. Ofrecen licenzas non só para software, senón tamén para texto, arte, e música, todas accesibles mediante selectores amigables de licenza; algunhas das licenzas son copyleft, algunhas non son copyleft mais son libres, outras son simplemente licenzas tradicionais con algunhas restricións aminoradas. O sitio web de Creative Commons dá unhas explicacións extremadamente claras sobre o que tratan. Se tiver que elixir un sitio para demostrar as extensas implicacións filosóficas do movemento do software libre, escollería este.

Apéndice A. Sistemas de control de versións libres

Estes son todos os sistemas de control de versións libres dos que tiña coñecemento ata a metade de 2007. O único que emprego de xeito habitual é Subversion. Teño pouca ou ningunha experiencia coa maioría destes sistemas, excepto con Subversion e CVS; a información que aquí mostro procede dos seus sitios web. Bótalle unha ollada tamén a http://en.wikipedia.org/wiki/List_of_revision_control_software

CVS - http://www.nongnu.org/cvs/

CVS existe desde hai moito tempo, e moitos desenvolvedores están familiarizados con el. No seu día supuxo unha revolución: foi o primeiro sistema de control de versións de código aberto con acceso de rede para desenvolvedores desde redes de área extensa (ata onde eu sei), e tamén o primeiro en ofrecer "checkouts" (descargas) anónimos só de lectura, funcionalidades que lle permitían aos novos desenvolvedores participaren con facilidade en proxectos. CVS só versiona ficheiros, non directorios; ofrece ramificacións ("branching"), etiquetaxe ("tagging"), e bo rendemento no lado cliente, mais non manexa demasiado ben ficheiros grandes nin binarios. Para alén disto, tampouco soporta commits atómicos. *[Nota: participei no desenvolvemento de CVS durante cinco anos, antes de axudar a botar a rolar o proxecto Subversion, que o ía substituír.]*

Subversion - http://subversion.tigris.org/

Subversion foi escrito ante todo para ser o substituto de CVS -é dicir, para aproximarse ao control de versións do mesmo xeito que o fai CVS, mais sen os problemas e omisións de funcionalidades que anoxaban con frecuencia os usuarios de CVS. Un dos obxectivos de Subversion é fornecer unha transición suave para a xente habituada a CVS. Non hai espazo aquí para entrar no detalle das funcionalidades de Subversion; bótalle unha ollada ao seu sitio web para obteres máis información. *[Nota: Estou implicado no desenvolvemento de Subversion, e é o único destes sistemas que acostumo a empregar.]*

SVK - http://svk.elixus.org/

Embora estea construído sobre a base de Subversion, SVK probablemente se pareza máis a algúns dos sistemas descentralizados posteriores que ao propio Subversion. SVK soporta desenvolvemento distribuído, commits locais, mesturas sofisticadas de mudanzas, e a habilidade de facer espellos de árbores de desenvolvemento con orixe en sistemas de control de versións que non sexan o propio SVK.

Bótalle unha ollada ao seu sitio web para máis detalles.

Mercurial - http://www.selenic.com/mercurial/

Mercurial é un sistema distribuído de control de versións que ofrece, entre outras cousas, "unha completa indexación cruzada de ficheiros e conxuntos de mudanzas; protocolos de sincronización HTTP e SSH eficientes desde o punto de vista do largo de banda e da CPU; fusión arbitraria entre ramas de desenvolvedores; unha interface web integrada autónoma; [portabilidade a] UNIX, MacOS X e Windows" e máis (a lista de funcionalidades precedente foi elaborada a partir do contido do sitio web de Mercurial).

GIT - http://git.or.cz/

GIT é un proxecto que iniciou Linus Torvalds para xestionar a árbore de desenvolvemento do código fonte do Kernel de Linux. No comezo GIT estaba claramente enfocado ás necesidades do desenvolvemento do kernel, mais expandiuse máis alá e agora mesmo é empregado por outros proxectos distintos do kernel de Linux. A súa páxina web di que está "...deseñado para manexar proxectos moi grandes con velocidade e eficiencia; está a ser empregado principalmente por varios proxectos de software libre, entre os que salienta o kernel de Linux. GIT atópase na categoría das ferramentas de xestión distribuída do código fonte, da mesma maneira que por exemplo GNU Arch ou Monotone (ou BitKeeper no mundo propietario). Cada directorio de traballo de GIT é un repositorio completo con capacidades completas de xestión de revisións, que non depende de acceso á rede ou a un servidor central."

Bazaar - http://bazaar.canonical.com/

Bazaar (ou bzr) é un sistema distribuído de control de versións que se centra na facilidade de uso e nun modelo de datos flexible. É un proxecto oficial de GNU, e é o sistema de control de versións nativo para o sitio de aloxamento de proxectos de software libre "Launchpad.net". Bazaar fornece control de versións totalmente distribuído: todo o traballo ten lugar en "branches" (ramas), e tipicamente cada desenvolvedor ten unha copia completa da historia da rama. As ramas poden fusionarse dentro doutras de xeito descentralizado, mais Bazaar tamén pode configurarse para traballar de xeito centralizado. Bazaar botou a andar como unha división de GNU Arch, mais foi reescrito desde cero e agora non ten relación directa con GNU Arch.

Darcs - http://darcs.net

"O Sistema Avanzado de Control de Revisións de David (David's Advanced Revision Control System) é outro substituto de CVS. Está escrito en Haskell, e ten sido usado en Linux, MacOX, FreeBSD, Open BSD e Microsoft Windows. Darcs inclúe un ficheiro de comandos cgi, que podes empregar para veres os contidos de teu repositorio."

Arch - http://www.gnu.org/software/gnu-arch/

GNU Arch soporta tanto desenvolvemento distribuído como centralizado. Os desenvolvedores fan commit das súas mudanzas a un "ficheiro", que pode ser local, e as mudanzas poden ser inseridas ou removidas noutros ficheiros segundo os administradores deses ficheiros o consideren conveniente. Como esta metodoloxía implica, Arch ten un soporte de mesturas máis sofisticado que CVS. Arch tamén permite crear ramas dun xeito sinxelo para aqueles ficheiros dos que careces de acceso para faceres commit. Este é só un breve resumo; bótalle unha ollada ao sitio web de Arch para máis detalles.

Monotone - http://www.venge.net/monotone/

"monotone é un sistema libre distribuído de control de versións. Fornece un sinxelo almacenamento transaccional de versións a nivel de ficheiros individuais, con total operatividade baixo desconexión, e cun protocolo eficiente de sincronización entre pares. Entende a fusión sensible ao historial de mudanzas, ramas lixeiras, revisión integrada de código e testaxe por parte de terceiros. Emprega nomenclatura cifrada de versións e certificados RSA no lado cliente. Posúe un bo soporte de internacionalización, non ten dependencias externas, corre en Linux, Solaris, OSX e Windows, e está publicado baixo licenza GNU/GPL."

Codeville - http://codeville.org/

"Por que outro sistema de control de versións? Todos os demais sistemas de control de versións esixen que manteñas unha xestión coidadosa das relacións entre as ramas para non mesturares constantemente os mesmos conflitos. Codeville é moito máis anárquico. Permite actualizar ou facer commits a calquera repositorio en calquera momento sen re-mesturas innecesarias".

"Codeville funciona creando un identificador para cada mudanza feita, e recordando a lista de todos os mudanzas que foron aplicadas a cada ficheiro e a última mudanza que modificou cada liña en cada ficheiro. Cando hai un conflito, comproba se unha das dúas partes ten sido xa aplicada á outra, e se for así fai que esa parte automaticamente sexa a gañadora. Cando hai un conflito de versións real que non pode ser mesturado automaticamente, Codeville compórtase case igual que CVS."

Vesta - http://www.vestasys.org/

"Vesta é un sistema portable de SCM [Xestión de Configuración de Software ou "Software Configuration Management" en inglés] orientado a dar soporte ao desenvolvemento de sistemas de software de case calquera tamaño, desde bastante pequenos (menos de 10.000 liñas de código fonte) ata moi grandes (10.000.000 de liñas de código fonte)."

"Vesta é un sistema maduro. É o resultado de máis de 10 anos de investigación e desenvolvemento no Centro de Desenvolvemento de Sistemas de Compaq/Digital, e foi empregado en produción polo grupo de microprocesadores Alpha de Compaq durante máis de dous anos e medio. O grupo Alpha tivo máis de 150 desenvolvedores activos en dous lugares a miles de quilómetros de distancia entre si, na costa leste e na costa oeste dos Estados Unidos. O grupo empregou Vesta para xestionar "builds" con 130 MB de datos fonte como máximo; cada un deles producía 1,5 GB de datos derivados. Os "builds" feitos na parte oeste nun día normal producían 10-15 GB de datos derivados, todos administrados por Vesta. Embora Vesta fose deseñado de cara ao desenvolvemento de software, o grupo Alpha demostrou a flexibilidade do sistema usándoo para desenvolvemento de hardware, comprobando os seus ficheiros

de linguaxe de descrición de hardware dentro da funcionalidade de control de código fonte de Vesta e construíndo simuladores e outros obxectos derivados co "construtor" de Vesta. Os membros do antigo grupo Alpha, agora parte de Intel, continúan a empregar aínda hoxe Vesta nun proxecto dun novo microprocesador."

Aegis - http://aegis.sourceforge.net/

"Aegis é un sistema de xestión da configuración de software baseado en transaccións. Proporciona un marco dentro do que un equipo de desenvolvedores pode traballar independentemente en moitas mudanzas dun programa, e Aegis coordina a integración desas mudanzas dentro do código fonte mestre do programa, co menor trastorno posible."

CVSNT - http://cvsnt.org/

"CVSNT é un sistema avanzado e multiplataforma de control de versións. Compatible co protocolo estándar da industria CVS, agora soporta moitas máis funcionalidades... CVSNT é Código Aberto, Software Libre adherido á Licenza Pública Xeral GNU." A súa lista de funcionalidades inclúe autenticación mediante todos os protocolos estándar de CVS, máis os SSPI e Active Directory específicos de Windows; soporte de transporte seguro, vía sserver ou SSPI cifrada; multiplataforma (corre en entornos Windows ou Unix); a versión NT está totalmente integrada co sistema Win32; o procesamento MergePoint permite non ter máis etiquetaxe a fusionar nas mesturas; está en desenvolvemento activo.

META-CVS http://common-lisp.net/project/meta-cvs/

"Meta-CVS é un sistema de control de versións construído sobre CVS. Embora conserve a maioría das funcionalidades de CVS, incluíndo todo o soporte ao traballo en rede, ten maior capacidade que CVS, e é máis fácil de empregar." As funcionalidades listadas no sitio web de META-CVS inclúen: versionamento da estrutura de directorios, manexo de tipos de ficheiro mellorado, mesturado ("merging") e ramificado ("branching") máis simple e máis amigable, soporte para ligazóns simbólicas, listas de propiedade anexadas aos datos versionados, importación mellorada de datos de terceiros, e unha sinxela actualización desde CVS.

OpenCM - http://www.opencm.org/

"OpenCM está deseñado como un substituto seguro e de alta integridade de CVS. Pódese atopar una lista das súas funcionalidades chave na páxina de características. Embora non sexa tan 'rico en funcionalidades' como CVS, soporta algunhas cousas útiles das que carece CVS. En poucas palabras, OpenCM fornece soporte de primeira clase para renomenclatura e configuración, autenticación criptográfica e control de acceso, e ramas de primeira clase."

PRCS - http://prcs.sourceforge.net/

"PRCS, o Sistema de Control de Revisión de Proxecto (Project Revision Control System), é a interface

dun conxunto de ferramentas que (como CVS) fornece un xeito de tratar con conxuntos de ficheiros e directorios como unha unidade, preservando versións coherentes de todo o conxunto... O seu propósito é similar ao de SCCS, RCS, e CVS, mais (cando menos segundo os seus autores), é moito máis sinxelo que calquera deses sistemas."

ArX - http://www.nongnu.org/arx/

ArX é un sistema de control de versións distribuído que ofrece funcionalidades de ramificación e mestura, verificación criptográfica da integridade dos datos, e a capacidade de publicar ficheiros con facilidade en calquera servidor HTTP.

SourceJammer - http://www.sourcejammer.org/

"SourceJammer é un sistema de versionamento e control de código fonte escrito en Java. Consiste nun compoñente na parte servidor que mantén os ficheiros e maila historia das versións, e manexa subidas, descargas, etc. e outros comandos; e nun compoñente na parte cliente que fai peticións ao servidor e xestiona os ficheiros no sistema de ficheiros do lado cliente."

FastCST - http://www.zedshaw.com/projects/fastcst/index.html

"Un sistema 'moderno' que emprega conxuntos de mudanzas sobre revisións dos ficheiros, e operabilidade distribuída fronte a control centralizado. Se tiveres un enderezo de correo electrónico, podes empregar FastCST. Para unha distribución maior só precisas un servidor FTP e/ou un servidor HTTP ou empregares o comando embutido 'serve' para ofrecer directamente o teu material. Todos os conxuntos de mudanzas son únicos e teñen toneladas de metadatos, polo que podes rexeitar calquera material que non quixeres antes de probalo. O mesturado faise comparando un conxunto mesturado de mudanzas cos contidos actuais do directorio, en lugar de tentar mesturalo con outro conxunto de mudanzas."

Superversion - http://www.superversion.org/

"Superversion é un sistema distribuído de control de versións e multiusuario baseado en conxuntos de mudanzas. Pretende ser unha alternativa de software libre con calidade industrial ás solucións privativas, con idéntica facilidade de uso (ou aínda máis sinxelo) e de potencia similar. De feito, intuitividade e eficiencia na usabilidade foron as principais prioridades no desenvolvemento de Superversion desde a súa orixe."

Apéndice B. Sistemas libres de xestión de erros (Bug Trackers)

Non importa que sistema de xestión de erros ("bug tracker") empregar un proxecto, sempre hai desenvolvedores que teñen queixa deles. Isto é máis notorio nos sistemas de xestión de erros que en calquera outra ferramenta habitual de desenvolvemento. Na miña opinión é debido a que os "bug trackers" son tan visuais e interactivos que é fácil imaxinar as melloras que a un mesmo lle gustaría

facer (se tiver tempo) e describir esas melloras abertamente. Acepta as inevitables queixas con filosofía, moitos dos sistemas que van ser presentados a seguir son moi bos.

Ao longo das seguintes listaxes a palabra "problema" ("*issue*" no orixinal) é empregada para referirse aos elementos que o sistema de notificación de erros rexistra. Mais recorda que cada sistema pode ter a súa propia terminoloxía sendo sinónimos "artefacto" ou "erro" ("artifact" ou "bug" no orixinal) entre outros (como pode ser "incidencia").

Bugzilla - http://www.bugzilla.org/

Bugzilla é moi popular, conta con soporte activo e os seus usuarios parecen estar satisfeitos. Eu traballei cunha versión modificada durante catro anos e gústame. Non é demasiado personalizable, o que en certa maneira pode considerarse unha das súas características; as instalacións de Bugzilla acostuman a ser moi semellantes en todas partes, polo que moitos desenvolvedores están xa afeitos á súa interface e vanse sentir nun terreo coñecido.

GNATS http://www.gnu.org/software/gnats/

GNU GNATS é un dos sistemas máis vellos de xestión de erros e é empregado amplamente. Os seus puntos fortes son a diversidade de interfaces (pode ser usado non só mediante un navegador web, senón tamén a través do correo electrónico ou de ferramentas de liña de comandos) e a almacenaxe dos erros en texto plano. Ter todos os erros en texto plano no disco facilita a tarefa de escribir ferramentas adaptadas para puxar e parsear os datos (por exemplo, para xerar informes estatísticos). GNATS pode tamén acceder a mensaxes de correo electrónico de diversas maneiras e anexalas aos temas correspondentes en función de padróns existentes nos cabezallos das mensaxes; isto facilita en gran medida o rexistro das conversacións entre usuarios e desenvolvedores.

RequestTracker (RT) - http://www.bestpractical.com/rt/

A páxina web de RT di o seguinte: "RT é un sistema de tickets de nivel profesional que permite a un grupo de persoas xerir con eficacia e habilidade tarefas, problemas e peticións enviadas por unha comunidade de usuarios" e, resumindo, iso é o que fai. RT posúe unha interface web bastante pulida e parece ter unha ampla base de instalacións. O seu aspecto visual é algo complexo, mais tórnase menos molesto cando estás afeito a empregalo. RT, está rexistrado baixo licenza GNU GPL (por algunha razón, o sitio web non aclara este punto).

Trac - http://trac.edgewall.com/

Trac é máis que un sistema de reporte de erros, en realidade é un sistema que integra wiki e un "bug tracker". Emprega wiki links para ligar temas, arquivos, grupos de mudanzas de control de versións e páxinas wiki simples. É bastante sinxelo de configurar e pódese integrar con Subversion (ver a sección *"Sistemas Libres de Control de Versións"* no *Apéndice A*).

Roundup - http://roundup.sourceforge.net/

Roundup é bastante sinxelo de instalar (só é necesario un Python 2.1 o superior) e fácil de usar. Ten tanto interface web, coma interface email e en liña de comandos. O molde ("template") de datos dos erros e a interface web son personalizables, así como parte da lóxica de transición de estados.

Mantis - http://www.mantisbt.org/

Mantis é un sistema de xestión de erros en web, escrito en PHP e que emprega MySQL como base de datos de almacenaxe. Presenta as características que calquera esperaría deste tipo de programas. Persoalmente, acho a súa interface web atractiva, elegante e intuitiva.

Flyspray - http://www.flyspray.org/

Flyspray é un sistema web de xestión de erros escrito en PHP. Segundo a súa páxina web é un sistema "pouco complicado" que inclúe as seguintes funcionalidades: soporte para diferentes bases de datos (nesta altura MySQL e PGSQL), xestión de múltiples proxectos, monitoramento de tarefas con notificacións de mudanzas (vía email ou jabber), histórico de tarefas exhaustivo, deseño CSS, ficheiros adxuntos, ferramentas avanzadas de procura (mais sinxelas de manexar), sindicación vía RSS/Atom, admite entradas en wiki ou texto plano, mecanismos de votacións e gráficos de dependencia.

Scarab - http://scarab.tigris.org/

A idea principal de Scarab é operar como un sistema completo de xestión de erros, altamente adaptable, que ofrece, máis ou menos, a unión das ferramentas que ofrecen outros sistemas de este tipo: entrada de datos, consultas, informes, notificacións ás partes interesadas, almacenaxe de comentarios colaborativos e rexistro de dependencias.

Pode ser personalizado mediante páxinas web de administración. Permite ter activos, nunha única instalación de Scarab, varios "módulos" (proxectos) simultaneamente. Dentro de cada módulo é posible crear novos tipos de elementos (soporte de peticións, defectos, melloras, tarefas, etc.) e engadir outros atributos arbitrarios. Todo isto permite adaptar o sistema aos requisitos determinados de cada proxecto.

A finais de 2004 estaba a punto de ser publicada a versión 1.0 de Scarab.

Debian Bug Tracking System (DBTS) - http://www.chiark.greenend.org.uk/~ian/debbugs/

O Debian Bug Tracking System é inusitado en tanto que toda a introdución e manipulación de erros se fai vía email: cada erro recibe o seu propio enderezo de email. O DBTS escala bastante ben: por exemplo. http://bugs.debian.org ten 277.741 erros.

Dado que a interacción é feita mediante clientes de email, un entorno facilmente accesible e ao que están afeitos moitos usuarios, o DBTS é bo para manexar, clasificar e responder rapidamente a un gran volume de notificacións de erro. Por suposto, ten desvantaxes tamén. Os desenvolvedores deben investir tempo para aprenderen o sistema de comandos de email e os usuarios deben escribir os informes de erro sen un formulario que os guíe na información que hai que incluír. Existen ferramentas para axudar os usuarios a escribiren mellores informes de erros tales como un programa en liña de

comandos *reportbug* ou o pacote *debbugs-el* para Emacs. A maioría da xente non usa estas ferramentas xa que prefire escribir os emails manualmente, e pode que cumpran ou non as regras sobre notificación do proxecto.

O DBTS ten unha interface web exclusivamente de lectura para visualizar e consultar erros.

Sistemas de rexistro de tickets de problemas (*Trouble-Ticket Trackers*)

Estes sistemas están máis orientados á xestión de tickets de axuda que á xestión de erros en programas de software. Embora un sistema "bugtracker" normal poida ser o máis adecuado, estes sistemas inclúense nesta obra en prol dun traballo o máis completo posible. Porén, en algúns proxectos poden ser máis axeitados que os tradicionais sistemas de xestión de erros.

- **WebCall -** http://myrapid.com/webcall/

Bluetail Ticket Tracker (BTT) - http://btt.sourceforge.net/

BTT é unha mestura entre un sistema de rexistro de tickets e un xestor de erros. Ofrece mecanismos de privacidade que non son frecuentes entre os xestores de erros. Os usuarios do sistema son clasificados en categorías: empregado (staff), amigo (friend), cliente (customer) ou anónimo. Esta clasificación define o nivel de acceso a máis ou menos datos en función da categoría de cada usuario. Ofrece unha certa integración co correo electrónico, unha interface de liña de comandos e mecanismos para a conversión de mensaxes de correo electrónico en tickets. Tamén pode xestionar certa información non vinculada directamente a un ticket específico, como documentación interna ou preguntas frecuentes.

Apéndice C. Por que me debería preocupar pola cor do garaxe da bicicleta?

O certo é que non deberías. Seguro que tes mellores cousas coas que perder o tempo.

O famoso artigo de Poul Henning Kamp sobre "o garaxe da bicicleta" (un resumo do mesmo pode atoparse no *Capítulo 6 Comunicacións*) é unha excelente disquisición sobre o que acostuma a ir mal nas conversas en listas de correo. O texto completo é reproducido aquí co seu permiso. A URL orixinal é http://www.freebsd.org/cgi/getmsg.cgi?fetch=506636+517178+/usr/local/www/db/text/1999/freebsd-hackers/19991003.freebsd-hackers (para unha ligazón máis sinxela, tamén o podes atopar en http://bikeshed.com)

```
Asunto: A cor do garaxe da bicicleta (calquera que for a súa cor) na herba
máis verde...
De: Poul-Henning Kamp <phk@freebsd.org>
Data: sab, 02 out 1999 16:14:10 +0200
ID da mensaxe: <18238.938873650@critter.freebsd.dk>
Remitente: phk@critter.freebsd.dk
Copia oculta: Blind Distribution List: ;
MIME-Version: 1.0
```

[bcc'ed to committers, hackers]

O meu último panfleto tivo unha acollida o bastante boa como para non ter medo de enviar outro, e hoxe teño tempo e motivación para facelo

Non foi fácil decidir a quen enviar este tipo de mensaxes, nesta ocasión vai con copia oculta para committers e hackers, que é probablemente o mellor que podo facer... Eu non estou subscrito a hackers mais xa falarei sobre isto despois.

O que me provocou nesta ocasión foi o fío "debería sleep(1) admitir segundos fraccionarios", que xa leva amolando tempo de abondo, probablemente un par semanas xa, mais nin sequera me vou molestar en comprobalo.

Para aqueles que non seguísedes ese fío en particular: parabéns

Era unha proposta para que sleep(1) "fixera o correcto" se se lle pasaba un argumento non enteiro que prendese a mecha deste problema. Non vou dicir nada máis ao respecto, porque é un asunto moito menos importante do que un podería esperarse dada a lonxitude do fío e xa recibiu moita máis atención que algúns dos *problemas* que temos por aquí.

A saga do "sleep"(1) é o exemplo máis patente dunha típica "discusión sobre o garaxe da bicicleta" que vin en FreeBSD. A proposta estaba ben pensada, gañariamos compatibilidade con OpenBSD e NetBSD e, porén, seguiriamos tendo compatibilidade con calquera código que xa estivese escrito.

Mais foron formuladas e enviadas tantas obxeccións, propostas e mudanzas que un podería pensar que a mudanza acabaría cos buracos do queixo suízo, ou que mudaría o sabor da Coca Cola ou que atinxiría algo de igual magnitude.

"De que vai esta discusión sobre a cor do garaxe?", preguntástesme algúns.

É unha longa historia, ou mellor dito, unha vella historia que en realidade é bastante curta. C. Northcote Parkinson escribiu un libro a principios dos anos 60 chamado "As leis de Parkinson" que afonda nas dinámicas da xestión.

Podedes atopalo en Amazon, e talvez na biblioteca dos vosos pais. Está ben de prezo e lese rápido. Se che gusta Dilbert, gustarache Parkinson.

Alguén díxome hai pouco que o leu e que se decatou de que soamente máis ou menos 50% era aplicable hoxe en día. Iso é asombroso, diría eu, a maioría dos libros de xestión modernos teñen porcentaxes de acerto moi por debaixo diso, e este ten máis de 35 anos.

No exemplo concreto referido ao garaxe da bicicleta, o outro compoñente vital é una central nuclear, supoño que isto dá unha idea da antigüidade do libro.

Parkinson mostra como podes falar cun consello directivo e conseguir que aproben a construción dunha central nuclear de centos, mesmo de miles de millóns, mais se o que queres é construír un garaxe veraste enredado nunha discusión sen fin.

Parkinson explica que isto sucede porque unha central nuclear é tan enorme, cara e complicada que fica fóra da capacidade de comprensión da xente, e en lugar de tentaren entender o asunto dan por sentado que alguén xa revisou todos os detalles con anterioridade.
Richard P. Feynman dá unha serie de exemplos referidos a "Los Álamos" nos

seus libros, exemplos máis interesantes e máis relacionados co tema.

Por outra banda temos o garaxe da bicicleta. Calquera pode construír un destes nun fin de semana e aínda vai ter tempo para ver o partido na TV. Xa que logo, non importa se a túa proposta é razoable ou está ben preparada, alguén aproveitará a oportunidade para demostrar que está facendo o seu traballo, que está prestando atención, que está *presente*.

En Dinamarca chamámolo "deixar a túa pegada". É unha cuestión de orgullo persoal e prestixio, de poder sinalar algo e dicir, "Ei, *eu* fixen iso" É un trazo moi marcado nos políticos, e presente na maioría da xente cando aparece a oportunidade. Non tedes máis que pensar nas pegadas no cemento fresco.

Sinto moito respecto por quen fixo a proposta inicial, porque se mantivo firme embora todo o que choveu sobre el, conseguindo que mudanza sexa hoxe unha realidade no noso código.

E isto lévame, como prometín antes, a explicar por que non estou subscrito a hackers:

Deime de baixa de hackers hai varios anos porque o número de correos non me permitía manterme ao día. Desde entón deixei varias listas máis pola mesma razón.

E, porén, recibo unha morea de correo. Unha gran parte é encamiñado directamente a /dev/null por medio de filtros: Xente como [omitido] nunca terá acceso á miña pantalla, así como commits a documentos en linguaxes que non entendo, ou commits a ports que non me interesan. Todo isto, e máis, vai parar ao fondo do caixón sen que chegue a saber que existen.

Mais a pesar desta trituradora sigo a recibir demasiadas mensaxes na miña caixa de correo.

E aquí é onde comeza o que quería dicir.

Oxalá puideramos diminuír o ruído nas nosas listas, e oxalá puideramos deixar a xente construíren un garaxe de cando en vez, e non me importaría de que cor o pintasen.

O primeiro destes desexos é sermos cívicos, sensibles e intelixentes no uso do correo electrónico.

Se eu puidese, de forma concisa e precisa, establecer un conxunto de criterios de cando se debe e cando non se debe responder a un correo electrónico que todos acordaran e respectaran, sería feliz, mais son demasiado intelixente sequera para intentalo.

Mais permitídeme suxerir algunhas xanelas emerxentes que me gustaría que os clientes de correo lanzasen sempre que alguén enviara ou respondera un email a unha lista de correo á que estiver subscrito.

```
+-----------------------------------------------------------+
| O seu email está a punto de ser enviado a centos de       |
| miles de persoas, que terán que perder polo menos 10      |
| segundos para decidir se é . Polo menos dúas semanas de   |
| traballo vanse perder en ler o seu correo. Moitos dos     |
| receptores terán que pagar para descarregar o seu correo  |
|                                                           |
| Está absolutamente seguro de que o seu correo electrónico é|
```

```
| suficientemente importante como para molestar a todas esas |
| persoas?                                                   |
|                                                            |
|               [Si] [Revisar] [Cancelar]                    |
+------------------------------------------------------------+


+------------------------------------------------------------+
| Aviso: aínda non leu todos os correos deste fío. Pode que  |
| alguén xa dixera o que vostede está a piques de            |
| escribir. Pór favor, lea todo o fío antes de responder a   |
| unha mensaxe do mesmo                                      |
|                      [Cancelar]                            |
+------------------------------------------------------------+


+------------------------------------------------------------+
| Aviso: O seu cliente de correo aínda non mostrou a mensaxe |
| enteira. Loxicamente, pódese deducir que probablemente     |
| vostede non o leu de principio a fin e o comprendeu        |
|                                                            |
| Non é de boa educación responder a un correo ata que non o |
| lea enteiro e reflexione                                   |
|                                                            |
| Un temporizador vai impedir que  responda a calquera       |
| correo deste fío durante a próxima hora                    |
|                                                            |
|                      [Cancelar]                            |
+------------------------------------------------------------+


+------------------------------------------------------------+
| Escribiu este correo a un ritmo superior a N.NN ppm        |
| Xeralmente non é posible pensar e teclear a un ritmo       |
| superior a A.AA ppm, e, polo tanto, é probable que a súa   |
| resposta sexa incoherente, estea mal pensada e/ou          |
| sexa emocional                                             |
|                                                            |
| Un temporizador impediralle o envío de calquera            |
| correo durante a seguinte hora                             |
|                                                            |
|                   [Cancelar]                               |
+------------------------------------------------------------+
```

A segunda parte do meu desexo é máis emocional. Obviamente, nunca fixemos uso das capacidades que mostramos no desagradable debate sobre sleep(1) para tomarmos esta pequena iniciativa, a pesar dos moitos anos que leva no proxecto. Entón, por que se enfadan de súpeto cando alguén con moita menos experiencia trata agora de solucionalo?

Ogallá o soubera

Sei que o meu razoamento non poderá impedir este "conservadorismo reaccionario". Talvez esas persoas estean frustradas porque non fixeron contribucións tanxibles recentemente, ou pode tratarse dun caso grave de"estamos vellos e de mal humor, NÓS sabemos como se ten que comportar a xuventude".

De calquera xeito, é algo moi improdutivo para o proxecto, mais non teño

suxestións de como paralo. O mellor que podo suxerir é que vos absteñades de alimentar os monstros que espreitan nas listas de correo: ignorádeos, non lles respondades, esquecede que están aí.

Espero que poidamos ter unha base máis forte e máis ampla de colaboradores en FreeBSD, e espero que xuntos poidamos impedir que os vellos rosmóns e os [omitido]do mundo lles dean tralla e os asusten antes que poidan pór un pe no chan.

Para as persoas que estiveron á espreita, temorosos de colaboraren por culpa das gárgolas: soamente podo pedirvos desculpas e animarvos a participardes de todas maneiras. Este non é o ambiente que quero para o proxecto.

Poul-Henning

Apéndice D. Exemplo de instrucións para informar da presenza de erros ("bugs")

A presente é unha copia lixeiramente modificada das instrucións en liña para novos usuarios do proxecto Subversion sobre como informar da presenza de erros. Mira a sección "*Trata a cada usuario como a un potencial voluntario*" do *Capítulo 8 Xestionando voluntarios* para entender por que é importante que un proxecto teña tales instrucións. O documento orixinal atópase en http://svn.collab.net/repos/svn/trunk/www/bugs.html

Como informar da presencia de erros en Subversion

Este documento conta como e onde informar da presenza dun erro. (Non se trata dunha lista dos erros pendentes — podes ver esa lista aquí.)

Onde informar da presenza dun erro
---

* Se o erro estiver localizado no propio Subversion, envía un correo a users@subversion.tigris.org. unha vez for confirmado que se trata dun erro, unha persoa, polo xeral ti, pode introducilo no sistema de notificación de erros. (Se estiveres moi seguro de que é un erro, envíao directamente á nosa lista de desenvolvemento dev@subversion.tigris.org. Mais, se non estiveres seguro, é mellor que escribas a users@ en primeiro lugar; algún dos usuarios pode dicirche se o comportamento que atopaches é o esperado ou non.

* Se o erro estiver localizado na biblioteca APR, por favor, informa sobre el nas seguintes listas de correo: dev@apr.apache.org, dev@subversion.tigris.org.

* Se o erro estiver localizado na biblioteca Neon-HTTP, por favor, informa sobre o mesmo en: neon@webdav.org, dev@subversion.tigris.org.

* Se o erro estiver localizado en Apache HTTPD 2.0, por favor informa sobre el nas seguintes dúas listas de correo: dev@httpd.apache.org, dev@subversion.tigris.org. A lista de desenvolvedores Apache httpd ten moito tráfego, polo que existe a posibilidade de que o teu informe de erro non sexa atendido. Tamén tes a posibilidade de preencheres un informe de erro en http://httpd.apache.org/bug_report.html.

* Se o erro estiver no teu tellado, colle un par de tellas e ponte a reparalo[29]

Como informar da presenza dun erro

En primeiro lugar, asegúrate de que realmente é un erro. Se Subversion non se comportar do modo que esperas, procura na documentación e nos arquivos da lista de correo probas de que debería comportarse do xeito que supós. Por suposto, se for unha cousa de senso común coma que Subversion destruíu os teus datos e fixo saír fume da túa pantalla, entón, podes fiarte do teu xuízo. Mais se non estiveres seguro, pregunta primeiro na lista de correo de usuarios users@subversion.tigris.org ou no IRC, irc.freenode.net, channel #svn.

Unha vez decidiches que é un erro, o máis importante é propoñeres unha descrición simple e indicacións de como reproducilo. Por exemplo, se o erro tal como o atopaches inicialmente implica cinco ficheiros sobre dez *commits*, tenta reproducilo con soamente un ficheiro e un *commit*. Canto máis sinxela for a "receita" para reproducilo, máis probable será que un desenvolvedor reproduza o erro e o corrixa con éxito.

Cando escribires a receita para reproducilo, non escribas unha descrición prosaica do que fixeches para para que acontecera o erro. Pola contra, fornece unha transcrición literal da serie exacta de comandos que executaches e a súa saída. Emprega cortar-e-pegar para facelo. Se houber ficheiros implicados, asegúrate de incluíres os seus nomes e mesmo o seu contido, se pensares que pode ser relevante. O mellor de todo sería que empacotases a túa receita para reproducir o erro coma un script. Iso axúdanos moitísimo.

Comprobación rápida de integridade: Estás a executar a versión máis recente de Subversion, verdade? :-) Poida que o erro xa fose corrixido; deberías probar a súa receita para reproducir o erro sobre a versión máis recente da árbore de desenvolvemento

29 Nota do tradutor: tradución libre do orixinal en inglés "*if the bug is in your rug, please give it a hug and keep it snug*", sen equivalente exacto en galego por tratarse dunha rima popular en lingua inglesa

de Subversion.

Ademais da receita para reproducilo, tamén necesitaremos unha completa descrición do entorno onde se reproduciu o erro. Isto é:

* O teu sistema operativo
* A versión publicada e/ou revisión de Subversion
* O compilador e as opcións de configuración que usaches para construír Subversion
* Calquera modificación persoal que fixeras ao teu Subversion
* A versión da BD de Berkeley coa que estás a usar Subversion, se é que a usar
* Calquera outra cousa que poida ser relevante. Mellor ter información de mais que de menos.

Unha vez tiveres todo isto, xa estarás preparado para redactar o informe. Comeza cunha descrición clara do erro en si. É dicir, di como esperabas que se comportara Subversion e contrástao con como se comportou realmente. Aínda que o erro poida parecerche obvio, pode que non sexa tan obvio para outra persoa, polo tanto é mellor evitar xogar ás adiviñas. Continúa coa descrición do entorno e a receita de reprodución do erro. Se tamén quixeres incluír algunha especulación sobre a causa e mesmo un remendo/parche para resolver o problema, aínda mellor.— Consulta http://svn.collab.net/repos/svn/trunk/www/hacking.html#patches para obteres instrucións de como mandar remendos.

Envía toda esta información a dev@subversion.tigris.org ou se xa o fixeches e che pediron que preencheses unha notificación, vai ao sistema de notificación de erros e segue as instrucións.

Agradecémoscho. Sabemos que preencher un bo informe de erros dá moito traballo, mais un bo informe pode aforrarlle moitas horas ao desenvolvedor e fai que o erro teña moitas máis probabilidades de ser corrixido.

Apéndice E. Copyright[30]

30 Nota do tradutor: O texto orixinal da licenza atópase en http://creativecommons.org/licenses/by-sa/3.0/ Esta tradución engádese soamente a título informativo.

fornecido debaixo, seguido do texto legal completo. Se desexar distribuír a parte ou a todo este traballo baixo diferentes termos legais, por favor contacte co autor, Karl Fogel <kfogel@red-bean.com>

-*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*-

Vostede é libre de:

* copiar, distribuír e comunicar publicamente a obra
* facer obras derivadas

Baixo ás condicións seguintes:

* Recoñecemento — Debe recoñecer os créditos da obra do xeito especificado polo autor ou polo licenciador (mais non de xeito que suxira que ten o seu apoio ou apoian o uso que fan da súa obra).

* Compartir baixo a mesma licenza. — Se transformar ou modificar esta obra para crear unha obra derivada, só pode distribuír a obra resultante baixo a mesma licenza, unha similar ou unha compatíbel.

* Ao reutilizar ou distribuír a obra, ten que deixar ben claro os termos da licenza desta obra. O mellor xeito de facer isto é incluír unha ligazón a esta páxina web

* Calquera das condicións anteriores pode ser alterada se vostede obtiver permiso do posuidor do copyright

* Nada nesta licenza restrinxe ou dana os dereitos morais do autor.

-*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*-

Código Legal de Creative Commons: Recoñecemento-CompartirIgual 3.0 España

CREATIVE COMMONS CORPORATION NON É UN GABINETE DE AVOGADOS E NON PROPORCIONA SERVIZOS XURÍDICOS. A DISTRIBUCIÓN DESTA LICENZA NON CREA UNHA RELACIÓN AVOGADO-CLIENTE. CREATIVE COMMONS PROPORCIONA ESTA INFORMACIÓN SEN MUDANZAS (ON AN "AS-IS" BASIS). CREATIVE COMMONS NON CONFIRE NINGUNHA GARANTÍA A RESPECTO DA INFORMACIÓN PROPORCIONADA, NIN ASUME NINGUNHA RESPONSABILIDADE POR DANOS PRODUCIDOS COMO CONSECUENCIA DO SEU USO.

Licenza

A OBRA OU A PRESTACIÓN (SEGUNDO SE DEFINE MÁIS ADIANTE) PROPORCIONASE BAIXO OS TERMOS DESTA LICENZA PÚBLICA DE CREATIVE COMMONS (CCPL OU LICENZA). A OBRA OU A PRESTACIÓN ESTÁ PROTEXIDA POLA LEI ESPAÑOLA DE PROPIEDADE INTELECTUAL E/OU CALQUERA OUTRA NORMA QUE FOR APLICABLE. ESTÁ PROHIBIDO CALQUERA USO DA OBRA OU PRESTACIÓN DIFERENTE AO AUTORIZADO POR ESTA LICENZA OU AO DISPOSTO NA LEI DE PROPIEDADE INTELECTUAL.

POR MEDIO DO EXERCICIO DE CALQUERA DEREITO SOBRE A OBRA OU A PRESTACIÓN, VOSTEDE ACEPTA E CONSENTE AS LIMITACIÓNS E OBRIGAS DESTA LICENZA, SEN PREXUÍZO DA NECESIDADE DE CONSENTIMENTO EXPRESO EN CASO DE VIOLACIÓN PREVIA DOS SEUS TERMOS. O LICENCIADOR CÉDELLE OS DEREITOS CONTIDOS NESTA LICENZA, SEMPRE QUE VOSTEDE ACEPTAR AS PRESENTES CLÁUSULAS.

1. Definicións

1. A obra é a creación literaria, artística ou científica ofrecida nos termos desta licenza.
2. Nesta licenza considérase unha prestación calquera interpretación, execución, fonograma, gravación audiovisual, emisión ou transmisión, simple fotografía ou outros obxectos protexidos pola lexislación de propiedade intelectual vixente aplicábel.
3. A aplicación desta licenza a unha colección (definida máis adiante) afectaralle unicamente á súa estrutura en canto forma de expresión da selección ou disposición dos seus contidos, mais non é extensiva a estes. Neste caso a colección terá a consideración de obra para os efectos desta licenza.
4. O titular orixinario é:
    1. No caso dunha obra literaria, artística ou científica, a persoa natural ou grupo de persoas que creou a obra.
    2. No caso dunha obra colectiva, a persoa que a editar e divulgar baixo o seu nome, salvo pacto contrario.
    3. No caso dunha interpretación ou execución, o actor, cantante, músico, ou calquera outra persoa que representar, cantar, ler, recitar, interpretar ou executar de calquera forma unha obra.
    4. No caso dun fonograma, o produtor fonográfico, é dicir, a persoa natural ou xurídica baixo a iniciativa e responsabilidade da cal se realiza por primeira vez unha fixación exclusivamente sonora da execución dunha obra ou doutros sons.
    5. No caso dunha gravación audiovisual, o produtor da gravación, é dicir, a persoa natural ou xurídica que teña a iniciativa e asuma a responsabilidade das fixacións dun plano ou secuencia de imaxes, con ou sen son.

6. No caso dunha emisión ou dunha transmisión, a entidade de radiodifusión.

7. No caso dunha simple fotografía, aquela persoa que a realizou.

8. No caso doutros obxectos protexidos pola lexislación de propiedade intelectual vixente, a persoa que esta indicar.

5. Consideraranse obras derivadas aquelas obras creadas a partir da licenciada, como por exemplo: as traducións e adaptacións; as revisións, actualizacións e anotacións; os compendios, resumos e extractos; os arranxos musicais e, en xeral, calquera transformación dunha obra literaria, artística ou científica. Para evitar a dúbida, se a obra consistir nunha composición musical ou gravación de sons, a sincronización temporal da obra cunha imaxe en movemento (synching) será considerada como unha obra derivada para os efectos desta licenza.

6. Terán a consideración de colección a compilación de obras alleas, de datos ou doutros elementos independentes como as antoloxías e as bases de datos que pola selección ou disposición dos seus contidos constitúan creacións intelectuais. A simple incorporación dunha obra nunha colección non dará lugar a unha derivada para os efectos desta licenza.

7. O licenciador é a persoa ou a entidade que ofrece a obra ou prestación nos termos desta licenza e lle concede os seus dereitos de explotación conforme o disposto nela.

8. Vostede é a persoa ou a entidade que exerce os dereitos concedidos mediante esta licenza e que non violou previamente os seus termos a respecto da obra ou da prestación, ou que recibiu o permiso expreso do licenciador para exercer os dereitos concedidos mediante esta licenza a pesar dunha violación anterior.

9. A transformación dunha obra comprende a súa tradución, adaptación e calquera outra modificación na súa forma da que se derive unha obra diferente. A creación resultante da transformación dunha obra terá a consideración de obra derivada.

10. Enténdese por reprodución a fixación directa ou indirecta, provisional ou permanente, por calquera medio e en calquera forma, de toda a obra ou a prestación ou de parte dela, que permitir a súa comunicación ou a obtención de copias.

11. Enténdese por distribución a posta a disposición do público do orixinal ou das copias da obra ou da prestación, nun soporte tanxible, mediante a súa venda, alugueiro, préstamo ou de calquera outra forma.

12. Enténdese por comunicación pública todo acto polo cal unha pluralidade de persoas, que non pertenceren ao ámbito doméstico de quen a levar a cabo, poida ter acceso á obra ou á prestación sen previa distribución de exemplares a cada unha delas. Considérase comunicación pública a posta a disposición do público de obras ou prestacións por procedementos con fíos ou

sen fíos, de tal forma que calquera persoa poida acceder a elas desde o lugar e no momento que escoller.

13. A explotación da obra ou a prestación comprende a reprodución, a distribución, a comunicación pública e, no seu caso, a transformación.

14. Os elementos da licenza son as características principais da licenza segundo a selección efectuada polo licenciador e indicadas no título desta licenza: Recoñecemento, CompartirIgual.

15. Unha licenza equivalente é:

1. Unha versión posterior desta licenza de Creative Commons cos mesmos elementos de licenza.

2. A mesma versión ou unha versión posterior desta licenza de calquera outra xurisdición recoñecida por Creative Commons cos mesmos elementos da licenza (exemplo: Recoñecemento-CompartirIgual 3.0 Xapón).

3. A mesma versión ou unha versión posterior da licenza de Creative Commons non adaptada a ningunha xurisdición (Unported) cos mesmos elementos da licenza.

4. Unha das licenzas compatibles que aparece en http://creativecommons.org/compatiblelicenses e que foi aprobada por Creative Commons como esencialmente equivalente a esta licenza porque, como mínimo:

1. Contén termos co mesmo propósito, o mesmo significado e o mesmo efecto que os elementos desta licenza.

2. Permite explicitamente que as obras derivadas de obras suxeitas a ela poidan ser distribuídas mediante esta licenza, a licenza de Creative Commons non adaptada a ningunha xurisdición (Unported) ou unha licenza de calquera outra xurisdición recoñecida por Creative Commons, cos seus mesmos elementos de licenza.

2. Límites dos dereitos. Nada nesta licenza pretende reducir ou restrinxir calquera dos límites legais dos dereitos exclusivos do titular dos dereitos de propiedade intelectual de acordo coa Lei de Propiedade Intelectual ou calquera outra lei aplicábel, foren derivados de usos lexítimos, tales como a copia privada ou a cita, ou outras limitacións como o resultante da primeira venda de exemplares (esgotamento).

3. Concesión de licenza. Conforme os termos e ás condicións desta licenza, o licenciador concede, polo prazo de protección dos dereitos de propiedade intelectual e a título gratuíto, unha licenza de ámbito mundial non exclusiva que inclúe os seguintes dereitos:

1. Dereito de reprodución, distribución e comunicación pública da obra ou da prestación.

2. Dereito a incorporar a obra ou a prestación nunha ou máis

coleccións.

3. Dereito de reprodución, distribución e comunicación pública da obra ou da prestación licitamente incorporada nunha colección.

4. Dereito de transformación da obra para crear unha obra derivada sempre e cando se incluír nesta unha indicación da transformación ou modificación efectuada.

5. Dereito de reprodución, distribución e comunicación pública de obras derivadas creadas a partir da obra licenciada.

6. Dereito a extraer e reutilizar a obra ou a prestación dunha base de datos.

7. Para evitar calquera dúbida, o titular orixinario:

1. Conserva o dereito a percibir as remuneracións ou compensacións previstas por actos de explotación da obra ou prestación, cualificadas pola lei como irrenunciábeis e inalienábeis e suxeitas a xestión colectiva obrigatoria.

2. Renuncia ao dereito exclusivo a percibir, tanto individualmente como mediante unha entidade de xestión colectiva de dereitos, calquera remuneración derivada de actos de explotación da obra ou prestación que vostede realizar.

Estes dereitos pódense exercer en todos os medios e formatos, tanxíbeis ou intanxíbeis, coñecidos no momento da concesión desta licenza. Os dereitos mencionados inclúen o dereito a efectuar as modificacións que foren precisas tecnicamente para o exercicio dos dereitos noutros medios e formatos. Todos os dereitos non concedidos expresamente polo licenciador quedan reservados, incluíndo a título enunciativo mais non limitativo, os dereitos morais irrenunciábeis recoñecidos pola lei aplicábel. Na medida en que o licenciador tiver dereitos exclusivos previstos pola lei nacional vixente que implementa a directiva europea en materia de dereito sui generis sobre bases de datos, renuncia expresamente aos devanditos dereitos exclusivos.

4. Restricións. A concesión de dereitos que supón esta licenza atópase suxeita e limitada ás restricións seguintes:

1. Vostede pode reproducir, distribuír ou comunicar publicamente a obra ou prestación soamente baixo os termos desta licenza e debe incluír unha copia dela, ou o seu Identificador Uniforme de Recurso (URI). Vostede non pode ofrecer ou impoñer ningunha condición sobre a obra ou prestación que alterar ou restrinxir os termos desta licenza ou o exercicio dos seus dereitos por parte dos seus concesionarios. Vostede non pode sublicenciar a obra ou prestación. Vostede debe manter intactos todos os avisos que se refiriren a esta licenza e á ausencia de garantías. Vostede non pode reproducir, distribuír ou comunicar publicamente a obra ou prestación con medidas tecnolóxicas que

controlen o acceso ou o uso dun xeito contrario aos termos desta licenza. Esta sección 4.a tamén afecta á obra ou prestación incorporada nunha colección, mais iso non implica que esta no seu conxunto fique automaticamente ou deba ficar suxeita aos seus termos. No caso que lle for requirido, logo de comunicación do licenciador, se vostede incorporar a obra nunha colección e/ou crear unha obra derivada, deberá quitar calquera crédito requirido no apartado 4.c, na medida do posíbel.

2. Vostede pode distribuír ou comunicar publicamente unha obra derivada no sentido desta licenza soamente baixo os seus termos ou outra licenza equivalente. Se vostede empregar esta mesma licenza debe incluír unha copia ou ben o seu URI, con cada obra derivada que vostede distribuír ou comunicar publicamente. Vostede non pode ofrecer ou impoñer ningún termo respecto á obra derivada que altere ou restrinxa os termos desta licenza ou o exercicio dos seus dereitos por parte dos seus concesionarios. Vostede debe manter intactos todos os avisos que se refiriren a esta licenza e á ausencia de garantías cando distribuír ou comunicar publicamente a obra derivada. Vostede non pode ofrecer ou impoñer ningún termo a respecto das obras derivadas ou das súas transformacións que alteraren ou restrinxiren os termos desta licenza ou o exercicio dos seus dereitos por parte dos seus concesionarios. Vostede non pode reproducir, distribuír ou comunicar publicamente a obra derivada con medidas tecnolóxicas que controlaren o acceso ou o uso da obra dun xeito contrario aos termos desta licenza. Se empregar unha licenza equivalente debe cumprir cos requisitos que esta establecer cando distribuír ou comunicar publicamente a obra derivada. Todas estas condicións aplícanse a unha obra derivada en tanto que incorporada a unha colección, mais non implica que esta teña que estar suxeita aos termos desta licenza.

3. Se vostede reproducir, distribuír ou comunicar publicamente a obra ou a prestación, unha colección que a incorporar ou calquera obra derivada, debe manter intactos todos os avisos sobre a propiedade intelectual e indicar, de xeito razoábel conforme ao medio ou aos medios que vostede estiver a empregar:

1. O nome do autor orixinal, ou o pseudónimo se for o caso, así como o do titular orixinario, se lle for facilitado.

2. O nome daquelas partes (por exemplo: institución, publicación, revista) que o titular orixinario e/ou o licenciador designaren para ser recoñecido no aviso legal, as condicións de uso, ou de calquera outro xeito razoábel.

3. O título da obra ou da prestación se lle for facilitado.

4. O URI, se existir, que o licenciador especificar para ser vinculado á obra ou á prestación, a menos que tal URI non se referir ao aviso legal ou á información sobre a licenza da obra ou a prestación.

5. No caso dunha obra derivada, un aviso que identifique a transformación da obra na obra derivada (p. ex., "tradución galega da obra de Autor Orixinal, ou "guión baseado en obra orixinal de Autor Orixinal").

Este recoñecemento debe facerse de xeito razoábel. No caso dunha obra derivada ou incorporación nunha colección estes créditos deberán aparecer como mínimo no mesmo lugar onde se acharen os correspondentes a outros autores ou titulares e de forma comparábel a estes. Para evitar a dúbida, os créditos requiridos nesta sección só serán empregados para os efectos de atribución da obra ou a prestación no xeito especificado anteriormente. Sen un permiso previo por escrito, vostede non pode afirmar nin dar a entender implicitamente nin explicitamente ningunha conexión, patrocinio ou aprobación por parte do titular orixinario, o licenciador e/ou as partes recoñecidas cara a vostede ou cara ao uso que fai da obra ou a prestación.

4. Para evitar calquera dúbida, debe facerse notar que as restricións anteriores (parágrafos 4.a, 4.b e 4.c) non son de aplicación a aquelas partes da obra ou á prestación obxecto desta licenza que unicamente poidan ser protexidas mediante o dereito sui generis sobre bases de datos recollido pola lei nacional vixente que traspoña a directiva europea de bases de datos.

5. Exención de responsabilidade

SALVO QUE FOR ACORDADO MUTUAMENTE DOUTRO XEITO POLAS PARTES, O LICENCIADOR OFRECE A OBRA OU A PRESTACIÓN SEN MODIFICACIÓNS (ON AN "AS-IS" BASIS) E NON CONFIRE NINGUNHA GARANTÍA DE NINGÚN TIPO A RESPECTO DA OBRA OU DA PRESTACIÓN OU DA PRESENZA OU AUSENCIA DE ERROS QUE PUIDEREN OU NON SER DESCUBERTOS. ALGUNHAS XURISDICIÓNS NON PERMITEN A EXCLUSIÓN DAS DEVANDITAS GARANTÍAS, POLO QUE TAL EXCLUSIÓN PODE NON SERLLE DE APLICACIÓN A VOSTEDE.

6. Limitación de responsabilidade. SALVO QUE O DISPUXER EXPRESA E IMPERATIVAMENTE A LEI APLICÁBEL, EN NINGÚN CASO O LICENCIADOR SERÁ RESPONSÁBEL ANTE VOSTEDE POR CALQUERA DANO RESULTANTE, XERAL OU ESPECIAL (INCLUÍDO O DANO EMERXENTE E O LUCRO CESANTE), FORTUÍTO OU CAUSAL, DIRECTO OU INDIRECTOS PRODUCIDO EN CONEXIÓN CON ESTA LICENZA OU O USO DA OBRA OU A PRESTACIÓN, MESMO SE O LICENCIADOR FOR INFORMADO DA POSIBILIDADE DE TALES DANOS.

7. Finalización da licenza

1. Esta licenza e a concesión dos dereitos que contén rematarán automaticamente en caso de calquera incumprimento dos seus termos. As persoas ou entidades que recibiran de vostede obras derivadas ou coleccións baixo esta licenza, non obstante,

non verán as súas licenzas finalizadas, sempre que tales persoas ou entidades se mantiveren no cumprimento íntegro desta licenza. As seccións 1, 2, 5, 6, 7 e 8 permanecerán vixentes malia calquera finalización desta licenza.

2. De conformidade coas condicións e termos anteriores, a concesión de dereitos desta licenza é vixente por todo o prazo de protección dos dereitos de propiedade intelectual segundo a lei aplicábel. Sen prexuízo do anterior, o licenciador resérvase o dereito a divulgar ou publicar a obra ou a prestación en condicións distintas ás presentes, ou de retirar a obra ou a prestación en calquera momento. Non obstante, iso non suporá dar por concluída esta licenza (ou calquera outra licenza que fose concedida, ou sexa necesario ser concedida, baixo os termos desta licenza), que continuará vixente e con efectos completos a non ser que finalizara consonte o establecido anteriormente, sen prexuízo do dereito moral de arrepentimento nos termos recoñecidos pola Lei de Propiedade Intelectual aplicábel.

8. Miscelánea

1. Cada vez que vostede realice calquera tipo de explotación da obra ou a prestación, ou dunha colección que a incorpore, o licenciador ofrece aos terceiros e sucesivos licenciatarios a concesión de dereitos sobre a obra ou a prestación nas mesmas condicións e termos que a licenza concedida a vostede.

2. Cada vez que vostede realizar calquera tipo de explotación dunha obra derivada, o licenciador ofreceralle aos terceiros e sucesivos licenciatarios a concesión de dereitos sobre a obra obxecto desta licenza nas mesmas condicións e termos que a licenza concedida a vostede.

3. Se algunha disposición desta licenza resultar inválida ou inaplicábel segundo a Lei vixente, iso non afectará a validez ou aplicabilidade do resto das cláusulas desta licenza e, sen ningunha acción adicional por calquera das partes deste acordo, tal disposición entenderase reformada no estritamente necesario para facer que tal disposición sexa válida e executiva.

4. Non se entenderá que existe renuncia a respecto de ningún termo ou disposición desta licenza, nin que se consente ningunha violación desta, a menos que tal renuncia ou consentimento figurar por escrito e levar a sinatura da parte que renunciar ou consentir.

5. Esta licenza constitúe o acordo pleno entre as partes a respecto da obra ou da prestación obxecto da licenza. Non caben interpretacións, acordos ou condicións con respecto á obra ou a prestación que non se atopen expresamente especificados na presente licenza. O licenciador non estará obrigado por ningunha disposición complementaria que poida aparecer en calquera comunicación que faga chegar vostede. Esta licenza non se pode modificar sen o mutuo acordo por escrito entre o licenciador e

vostede.

Aviso de Creative Commons

Creative Commons non é parte desta licenza, e non ofrece ningunha garantía en relación coa obra ou a prestación. Creative Commons non será responsábel fronte a vostede ou calquera parte, por calquera danos resultantes, incluíndo, mais non limitando danos xerais ou especiais (incluído o dano emerxente e o lucro cesante), fortuítos ou causais, en conexión con esta licenza. Sen prexuízo das dúas (2) oracións anteriores, se Creative Commons se identificou expresamente como o licenciador, terá todos os dereitos e obrigas do licenciador.

Excepto para o propósito limitado de indicar ao público que a obra ou a prestación está licenciada baixo a CCPL, ningunha parte empregará a marca rexistrada "Creative Commons" ou calquera marca rexistrada ou insignia relacionada con "Creative Commons" sen o seu consentimento por escrito. Calquera uso permitido farase de conformidade coas pautas vixentes en cada momento sobre o uso da marca rexistrada por "Creative Commons", en tanto que sexan publicadas no seu sitio web (website) ou sexan proporcionadas por pedimento previo. Para evitar calquera dúbida, estas restricións no uso da marca non forman parte desta licenza.

Pode contactar con Creative Commons en:
http://creativecommons.org/