Um modelo de projeto para o projeto FreeBSD

Niklas Saers

Um modelo de projeto para o projeto FreeBSD

Niklas Saers

Revisão: 52263

Índice

Lista de Figuras

Prefácio

Até agora, o projeto FreeBSD lançou várias técnicas descritas para fazer diferentes partes do trabalho. No entanto, um modelo de projeto resumindo como o projeto é estruturado é necessário devido à quantidade crescente de membros do projeto. [1] Este artigo fornecerá esse modelo de projeto e será doado ao projeto de Documentação do FreeBSD, onde ele poderá evoluir junto com o projeto, de modo que ele possa, a qualquer momento, refletir a maneira como o projeto funciona. É baseado em [Saers, 2003].

Gostaria de agradecer às pessoas a seguir por dedicarem tempo para explicar coisas que não estavam claras para mim e por revisar o documento.

- Andrey A. Chernov <ache@freebsd.org >
- Bruce A. Mah <bmah@freebsd.org >
- Dag-Erling Smørgrav <des@freebsd.org >
- Giorgos Keramidas<keramida@freebsd.org >
- Ingvil Hovig <ingvil.hovig@skatteetaten.no >
- Jesper Holck<jeh.inf@cbs.dk >
- John Baldwin <jhb@freebsd.org >
- John Polstra <jdp@freebsd.org >
- Kirk McKusick <mckusick@freebsd.org >
- Mark Linimon <linimon@freebsd.org >
- Marleen Devos
- Niels Jørgenssen<nielsj@ruc.dk >
- Nik Clayton <nik@freebsd.org >
- Poul-Henning Kamp <phk@freebsd.org >
- Simon L. Nielsen <simon@freebsd.org >

[1] Isso vai de mãos dadas com a lei de Brooks onde “adicionar outra pessoa a um projeto atrasado irá atrasa-lo ainda mais” pois irá aumentar as necessidades de comunicação Brooks, 1995. Um modelo de projeto é uma ferramenta para reduzir as necessidades de comunicação.

Capítulo 1. Visão geral

Um modelo de projeto é um meio de reduzir a sobrecarga de comunicações em um projeto. Conforme mostrado por [Brooks, 1995], aumentar o número de participantes do projeto aumenta exponencialmente a comunicação no projeto. O FreeBSD tem aumentado nos últimos anos tanto sua massa de usuários ativos quanto de committers, e a comunicação no projeto aumentou de acordo com esse crescimento. Esse modelo de projeto servirá para reduzir essa sobrecarga, fornecendo uma descrição atualizada do projeto.

Durante as eleições do Core em 2002, Mark Murray declarou: “Me oponho a um longo livro de regras, pois isso satisfaz tendências de advogados e é contrário ao tecnocentrismo de que o projeto tanto necessita.” [FreeBSD, 2002B]. Este modelo de projeto não pretende ser uma ferramenta para justificar a criação de imposições para desenvolvedores, mas como uma ferramenta para facilitar a coordenação. Isso tem significado como uma descrição do projeto, com uma visão geral de como os diferentes processos são executados. É uma introdução ao funcionamento do projeto FreeBSD.

O modelo do projeto FreeBSD será descrito a partir de 1º de julho de 2004. É baseado no paper de Niels Jørgensen [Jørgensen, 2001], documentos oficiais do FreeBSD, discussões em listas de discussão do FreeBSD e entrevistas com os desenvolvedores.

Depois de fornecer as definições dos termos usados, este documento delineará a estrutura organizacional (incluindo descrições de funções e linhas de comunicação), discutirá o modelo de metodologia e, depois de apresentar as ferramentas usadas para controle de processos, apresentará os processos definidos. Finalmente, ele delineará os principais subprojetos do projeto FreeBSD.

[FreeBSD, 2002A, Seção 1.2 e 1.3] fornece a visão e as diretrizes arquitetônicas do projeto. A visão é “Produzir o melhor pacote de sistema operacional semelhante ao UNIX, respeitando a ideologia das ferramentas de software originais, bem como usabilidade, desempenho e estabilidade.” As diretrizes de arquitetura ajudam a determinar se um problema que alguém quer que seja resolvido está dentro do escopo do projeto

Capítulo 2. Definições

2.1. Atividade

Uma “atividade” é um elemento do trabalho realizado durante o curso de um projeto [PMI, 2000]. Ele tem uma saída e leva a um resultado. Tal saída pode ser uma entrada para outra atividade ou parte da entrega do processo.

2.2. Processo

Um “processo” é uma série de atividades que levam a um resultado específico. Um processo pode consistir em um ou mais subprocessos. Um exemplo de um processo é o design de software.

2.3. Hat (Definição/função especifica para algumas pessoas)

Uma “hat” é um sinônimo de função. Uma hat tem certas responsabilidades em um processo e para o resultado do processo. O hat executa atividades. Está bem definido por quais problemas o hat deve ser contatado pelos membros do projeto e pessoas fora do projeto.

2.4. Resultado

Um “resultado” é a finalização do processo. Isso é sinônimo de entrega, que é definido como “qualquer finalização mensurável, tangível, verificável ou item que deve ser produzido para concluir um projeto ou parte de um projeto. Frequentemente usado de forma mais restrita em referência a uma entrega externa, que é uma entrega sujeita à aprovação do patrocinador ou cliente do projeto”, por [PMI, 2000]. Exemplos de resultados são um software, uma decisão tomada ou um relatório escrito.

2.5. FreeBSD

Ao dizer “FreeBSD” queremos dizer o sistema operacional FreeBSD derivativo do BSD semelhante ao UNIX, enquanto ao dizer “o projeto FreeBSD” queremos dizer a organização do projeto.

Capítulo 3. Estrutura organizacional

Enquanto ninguém assume a propriedade do FreeBSD, a organização do FreeBSD é dividida em core, committers e colaboradores e isso faz parte da comunidade do FreeBSD que vive em torno dele.

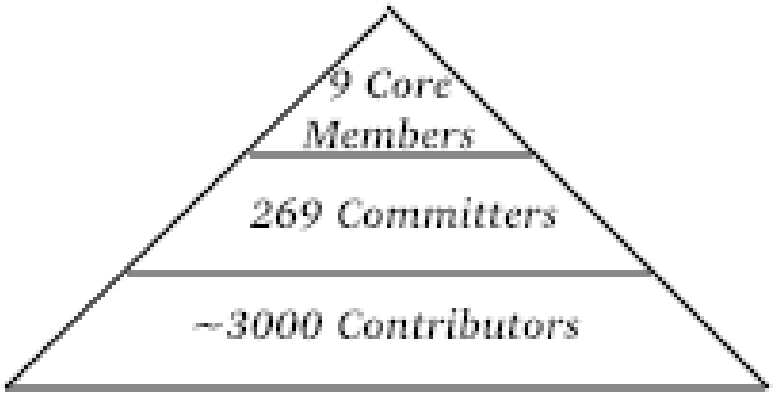

Figura 3.1. A estrutura do projeto FreeBSD

O número de committers foi determinado passando pelos logs do CVS de 1º de janeiro de 2004 a 31 de dezembro de 2004 e pelos colaboradores, passando pela lista de contribuições e relatórios de problemas.

O principal recurso na comunidade do FreeBSD são seus desenvolvedores: os committers e colaboradores. É com suas contribuições que o projeto pode avançar. Desenvolvedores regulares são referidos como colaboradores. Até 1º de janeiro de 2003, havia uma estimativa de 5500 colaboradores no projeto.

Os committers são desenvolvedores com o privilégio de poder aplicar mudanças (fazer commit). Geralmente, são os desenvolvedores mais ativos que estão dispostos a gastar seu tempo não apenas integrando seu próprio código, mas integrando o código enviado pelos desenvolvedores que não têm esse privilégio. Eles também são os desenvolvedores que elegem a equipe principal (core) e têm acesso a discussões fechadas.

O projeto pode ser agrupado em quatro partes separadas distintas, e a maioria dos desenvolvedores focará seu envolvimento em uma parte do FreeBSD. As quatro partes são desenvolvimento de kernel, desenvolvimento de userland, ports e documentação. Ao se referir ao sistema base, nos referimos tanto o kernel quanto o userland.

Esta divisão muda nosso triângulo para ficar assim:

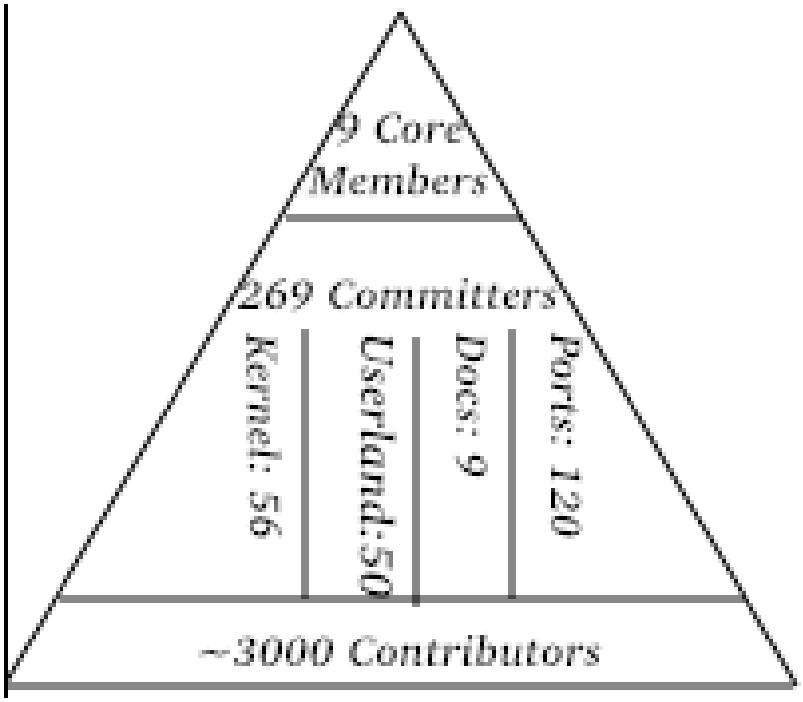

Figura 3.2. A estrutura do Projeto FreeBSD com committers em categorias

O número de committers por área foi determinado passando por logs do CVS de 1 de janeiro de 2004 a 31 de dezembro de 2004. Observe que muitos committers trabalham em várias áreas, fazendo com que o número total seja maior que o número real de committers. O número total de committers naquele momento era 269.

Os committers se enquadram em três grupos: committers que estão preocupados apenas com uma área do projeto (por exemplo, sistemas de arquivos), committers que estão envolvidos apenas com um subprojeto e committers que se comprometem com partes diferentes do código, incluindo subprojetos. Como alguns committers trabalham em partes diferentes, o número total na seção commiters do triângulo é maior que no triângulo acima.

O kernel é o principal bloco de construção do FreeBSD. Enquanto os aplicativos em userland são protegidos contra falhas em outros aplicativos do userland, todo o sistema é vulnerável a erros no kernel. Isso, combinado com a grande quantidade de dependências no kernel e que não é fácil ver todas as consequências de uma mudança no kernel, exige que os desenvolvedores tenham uma compreensão relativamente completa do kernel. Múltiplos esforços de desenvolvimento no kernel também requerem uma coordenação mais próxima do que os aplicativos em userland requerem.

Os principais utilitários, conhecidos como userland, fornecem a interface que identifica o FreeBSD, interface do usuário, bibliotecas compartilhadas e interfaces externas para conectar clientes. Atualmente, 162 pessoas estão envolvidas no desenvolvimento e manutenção da userland, muitas delas sendo mantenedoras de sua própria parte do código. A manutenção será discutida na seção Maintainership.

A documentação é tratada por The FreeBSD Documentation Project e inclui todos os documentos em torno do projeto FreeBSD, incluindo as páginas web. Durante o ano de 2004, 101 pessoas fizeram commits para o Projeto de Documentação do FreeBSD.

Ports é a coleção de meta-dados necessários para fazer com que os pacotes de software sejam compilados corretamente no FreeBSD. Um exemplo de port é o port do navegador Mozilla. Ele contém informações sobre onde buscar o código fonte, quais correções aplicar e como o pacote deve ser instalado no sistema. Isso permite que ferramentas automatizadas busquem, criem e instalem o pacote. Até esta data, existem mais de 12600 ports disponíveis. [1], variando de servidores web a jogos, linguagens de programação e a maioria dos tipos de aplicativos que estão em uso em computadores modernos. Os ports serão discutidos em mais detalhes na seção The Ports Subproject.

[1]As estatísticas são geradas contando o número de entradas no arquivo buscado pelo portsdb até 1 de abril de 2005. portsdb é uma parte do port sysutils/portupgrade.

Capítulo 4. Modelo de metodologia

4.1. Modelo de desenvolvimento

Não existe um modelo definido para como as pessoas escrevem código no FreeBSD. No entanto, Niels Jørgenssen sugeriu um modelo de como o código escrito é integrado ao projeto.

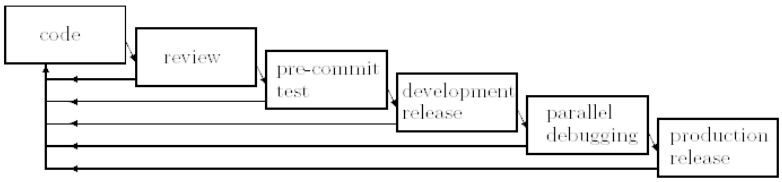

Figura 4.1. O modelo de Jørgenssen para integração de mudanças

A “versão de desenvolvimento” é a branch (ramificação) FreeBSD-CURRENT ("-CURRENT") e a “versão de produção ” é a branch (ramificação) FreeBSD-STABLE ("-STABLE") [Jørgensen, 2001].

Este é um modelo para uma mudança e mostra que, após a codificação, os desenvolvedores buscam a revisão da comunidade e tentam integrá-la com seus próprios sistemas. Depois de integrar a mudança na versão de desenvolvimento, chamada FreeBSD-CURRENT, ela é testada por muitos usuários e desenvolvedores na comunidade FreeBSD. Depois de passar por testes suficientes, é feito o merge (aplicado) na versão de produção, chamada FreeBSD-STABLE. A menos que cada estágio seja concluído com êxito, o desenvolvedor precisa voltar e fazer modificações no código e reiniciar o processo. Integrar uma mudança com -CURRENT ou -STABLE é chamado de fazer um commit.

Jørgensen descobriu que a maioria dos desenvolvedores do FreeBSD trabalha individualmente, o que significa que este modelo é usado em paralelo por muitos desenvolvedores nos diferentes esforços de desenvolvimento em andamento. Um desenvolvedor também pode estar trabalhando em várias alterações, de modo que, enquanto ele aguarda revisão ou que pessoas testem uma ou mais de suas alterações, ele pode estar escrevendo outra alteração.

Como cada commit representa um incremento, este é um modelo massivamente incremental. Os commits são tão freqüentes que durante um ano [1], 85427 commits foram feitos, fazendo uma média diária de 233 commits.

Dentro do processo “code” na figura de Jørgensen, cada programador tem seu próprio estilo de trabalho e segue seus próprios modelos de desenvolvimento. O suporte poderia muito bem ter sido chamado de “desenvolvimento”, pois inclui coleta e análise de requisitos, sistema e projeto detalhados, implementação e verificação. No entanto, a única saída desses estágios é o código-fonte ou a documentação do sistema.

Da perspectiva de um modelo em etapas (como o modelo em cascata), os outros suportes podem ser vistos como verificação adicional e integração do sistema. Essa integração do sistema também é importante para ver se uma alteração é aceita pela comunidade. Até que o código seja "committed", o desenvolvedor é livre para escolher quanto se deve comunicar sobre o restante do projeto. Para que o -CURRENT funcione como um buffer (para que as ideias brilhantes que tinham algumas desvantagens não descobertas possam ser recuperadas), o tempo mínimo que um "commit" deve estar no -CURRENT antes de fazer o merge (aplicar o código) para -STABLE é de 3 dias. Esse merge (aplicação) é referido como um MFC (Merge From Current).

É importante notar a palavra “change (mudança)”. A maioria dos commits não contém novos recursos radicais, mas são atualizações de manutenção.

As únicas exceções desse modelo são correções de segurança e alterações nos recursos que estão obsoletos na branch (ramificação) -CURRENT. Nesses casos, as alterações podem ser "committed" diretamente na branch -STABLE.

[1] O período de 1º de janeiro de 2004 a 31 de dezembro de 2004 foi examinado para encontrar esse número.

Além de muitas pessoas trabalhando no projeto, há muitos projetos relacionados ao Projeto FreeBSD. Estes são projetos desenvolvendo novos recursos, subprojetos ou projetos cujo resultado é incorporado ao FreeBSD [2]. Esses projetos se encaixam no Projeto FreeBSD, assim como os esforços regulares de desenvolvimento: eles produzem código que são integrados ao Projeto FreeBSD. No entanto, alguns deles (como Ports e Documentação) têm o privilégio de serem aplicáveis às duas branchs (ramificações) ou de commit diretamente na -CURRENT e na -STABLE.

Não há padrões para como o design deve ser feito, nem o design é coletado em um repositório centralizado. O design principal é o do 4.4BSD. [3] Como o design é parte do processo "Code (Código)" no modelo de Jørgenssen, cabe a cada desenvolvedor ou sub-projeto como isso deve ser feito. Mesmo que o projeto deva ser armazenado em um repositório central, a saída dos estágios do projeto seria de uso limitado, pois as diferenças de metodologias as desvalorizariam se de alguma forma interoperáveis. Para o design geral do projeto, o projeto conta com os subprojetos para negociar interfaces adequadas entre si, em vez de ditar interfaces.

4.2. Lançamento de versões (Release branches)

Os lançamentos do FreeBSD são melhor ilustrados por uma árvore com muitas branches (ramificações), onde cada branch principal representa uma versão principal. Versões secundárias são representadas por branches das branches maiores.

Na árvore de versões a seguir, as setas que seguem uma a outra em uma determinada direção representam uma branch. Caixas com linhas completas e encorporadas representam lançamentos oficiais. Caixas com linhas pontilhadas representam a branch de desenvolvimento nesse momento. Branchs de segurança são representadas por ovais. As encorporadas diferem das caixas em que representam um fork, definindo um lugar onde uma branch se divide em duas branchs, onde uma das branches se tornam uma sub-branch (sub ramificação). Por exemplo, em 4.0-RELEASE, a branch 4.0-CURRENT é dividida em 4-STABLE e 5.0-CURRENT. No 4.5-RELEASE, a branch retirou uma branch de segurança chamada RELENG_4_5.

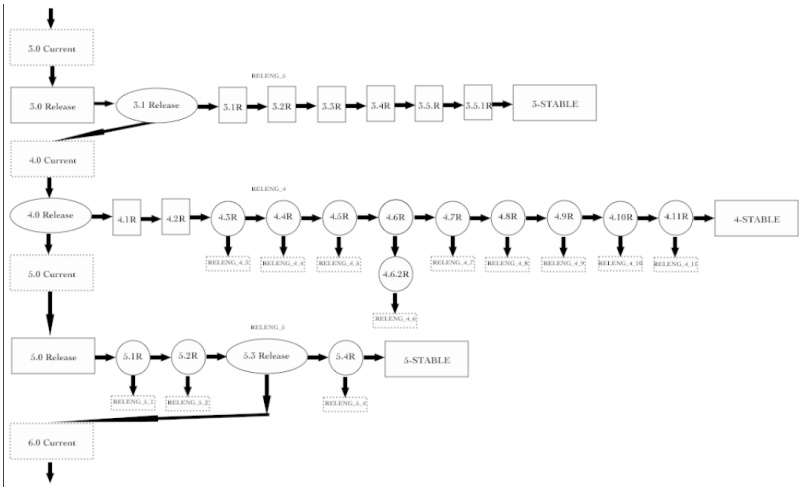

Figura 4.2. A árvore de versões (release) do FreeBSD

A última versão -CURRENT é sempre referida como -CURRENT, enquanto a versão mais recente -STABLE é sempre referida como -STABLE. Nessa figura, -STABLE se refere a 4-STABLE, enquanto -CURRENT refere-se a 5.0-CURRENT seguindo para 5.0-RELEASE. [FreeBSD, 2002E]

[2] Por exemplo, o desenvolvimento da pilha Bluetooth começou como um subprojeto até ser considerada estável o suficiente para ser feito o merge (aplicado) na branch -CURRENT. Agora é parte do sistema principal do FreeBSD.

[3] De acordo com Kirk McKusick, depois de 20 anos desenvolvendo sistemas operacionais UNIX, as interfaces são, na maioria das vezes, resolvidas. Portanto, não há necessidade de muito design. No entanto, novas aplicações do sistema e novo hardware levam a algumas implementações sendo mais benéficas do que aquelas que costumavam ser ter preferencia. Um exemplo é a introdução da navegação Web que tornou a conexão TCP/IP normal uma sequência curta de dados, em vez de um fluxo contínuo durante um período de tempo mais longo.

Um “lançamento principal” é sempre feito a partir da branch -CURRENT. No entanto, a branch -CURRENT não precisa ser feito fork nesse momento, mas pode concentrar-se na estabilização. Um exemplo disso é que, após 3.0-RELEASE, 3.1-RELEASE também era uma continuação da branch -CURRENT, e -CURRENT não se tornou uma branch de desenvolvimento verdadeira até que esta versão fosse lançada e fosse feito o fork da branch 3-STABLE. Quando -CURRENT retorna para se tornar uma branch de desenvolvimento, ele só pode ser seguido por um lançamento principal. Prevê-se que seja feito o fork do 5-STABLE a partir da branch 5.0-CURRENT em torno da 5.3-RELEASE. Não é feito até que seja feito o fork da 5-STABLE, então a branch de desenvolvimento será marcada como 6.0-CURRENT.

Uma “versão secundária” é feita a partir da branch -CURRENT após uma versão principal ou da branch -STABLE.

Seguindo e incluindo, 4.3-RELEASE [4], quando uma liberação secundária foi feita, ela se torna uma “branch de segurança”. Isso se destina a organizações que não desejam seguir a branch -STABLE e os possíveis recursos novos/alterados que oferece, mas que exigem um ambiente absolutamente estável, atualizando apenas para implementar atualizações de segurança. [5]

Cada atualização para uma branch de segurança é chamada de “patchlevel (à nível de correção)”. Para cada aprimoramento de segurança que é feito, o número de patchlevel é aumentado, tornando mais fácil para as pessoas rastrearem a branch para ver quais melhorias de segurança implementaram. Nos casos em que houve falhas graves de segurança, uma nova versão inteira pode ser feita a partir de uma branch de segurança. Um exemplo disso é a 4.6.2-RELEASE.

4.3. Resumo do modelo

Para resumir, o modelo de desenvolvimento do FreeBSD pode ser visto como a seguinte árvore:

[4] A primeira versão onde isso aconteceu foi na 4.5-RELEASE, mas as branches de segurança foram criadas ao mesmo tempo para 4.3-RELEASE e 4.4-RELEASE.

[5] Existe uma terminologia que se sobrepõe à palavra "stable (estável)", o que leva a alguma confusão. A branch -STABLE ainda é uma branch de desenvolvimento, cujo objetivo é ser útil para a maioria das pessoas. Se nunca for aceitável que um sistema obtenha alterações que não são anunciadas no momento em que é implantado, esse sistema deve executar uma branch de segurança.

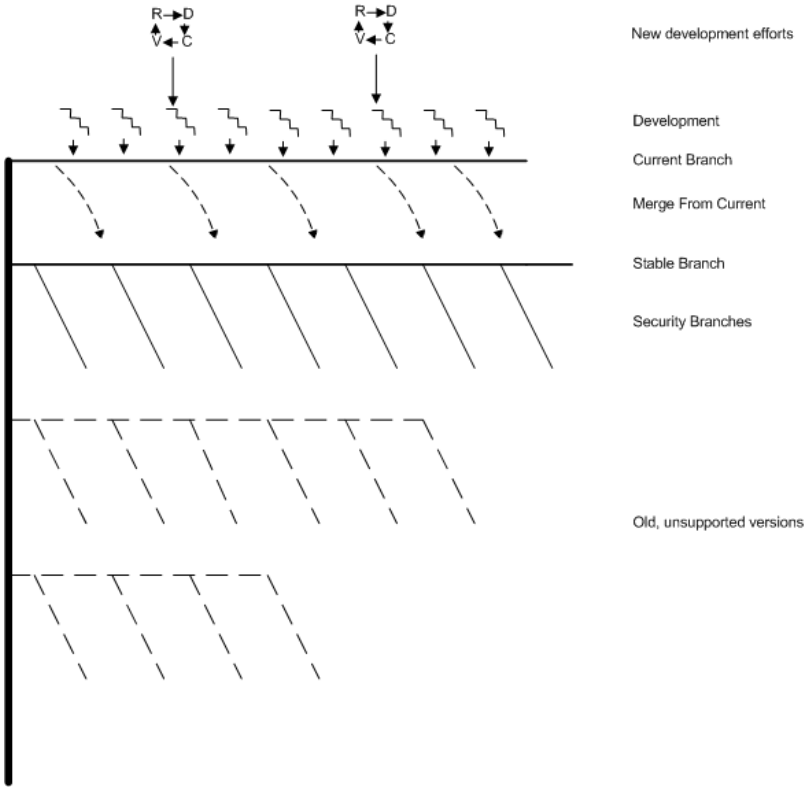

Figura 4.3. O modelo geral de desenvolvimento

A árvore do desenvolvimento do FreeBSD com esforços contínuos de desenvolvimento e integração contínua.

A árvore simboliza as versões de lançamento com versões principais gerando novas branchs principais e versões secundárias sendo versões da branch principal. A branch superior é a branch -CURRENT, onde todo o desenvolvimento novo é integrado, e a branch -STABLE é a branch diretamente abaixo dela.

Nuvens de esforços de desenvolvimento pairam sobre o projeto, onde os desenvolvedores usam os modelos de desenvolvimento que eles acham adequados. O produto de seu trabalho é então integrado em -CURRENT, onde ele é depurado paralelamente e finalmente é feito o merge (aplicado o código) do -CURRENT na -STABLE. As correções de segurança são feitos os merges (aplicado os códigos) da -STABLE para as branches de segurança.

Capítulo 5. Hats (Funções)

Muitos committers têm uma área especial de responsabilidade. Esses papéis são chamados de hats. Esses títulos podem ser funções do projeto, como diretor de relações públicas ou mantenedor de uma determinada área do código. Como esse é um projeto em que as pessoas doam voluntariamente seu tempo livre, as pessoas com hats atribuídos nem sempre estão disponíveis. Eles devem, portanto, nomear um substituto que possa desempenhar o cargo de hat em sua ausência. A outra opção é ter o cargo ocupado por um grupo.

Muitos desses hats não são formalizados. Hats formalizados têm uma carta indicando o propósito exato do hat, juntamente com seus privilégios e responsabilidades. A redação de tais cartas é uma nova parte do projeto e, portanto, ainda não foi concluída para todos os hats. Essas descrições de hats não são uma formalização, e sim um resumo da função com links para a carta quando disponíveis e endereços de contato.

5.1. Hats Universais

5.1.1. Colaborador

Um Colaborador contribui para o projeto do FreeBSD como desenvolvedor, como autor, enviando relatórios de problemas ou contribuindo de outras formas para o progresso do projeto. Um colaborador não tem privilégios especiais no projeto do FreeBSD. [FreeBSD, 2002F]

5.1.2. Committer

Uma pessoa que possui os privilégios necessários para adicionar seu código ou documentação ao repositório. Um committer fez um commit nos últimos 12 meses. [FreeBSD, 2000A] Um committer ativo é um committer que fez uma média de um commit por mês durante esse tempo.

Vale a pena notar que não existem barreiras técnicas para impedir que alguém, uma vez tendo ganho privilégios de commit para o main- ou um sub-projeto, de fazer commits em partes do código desse projeto que o committer não obteve permissão especificamente para modificar. No entanto, quando quiser fazer modificações em partes que um committer não tenha participado antes, ele deve ler os logs para ver o que aconteceu nesta área antes, e também ler o arquivo MAINTAINER para ver se o mantenedor desta parte tem quaisquer pedidos especiais sobre como as alterações no código devem ser feitas

5.1.3. Equipe principal (Core Team)

A equipe principal é eleita pelos committers da lista de committers e serve como a diretoria do projeto FreeBSD. Promove colaboradores ativos para committers, atribui pessoas a hats bem definidos e é o mediador final de decisões envolvendo o caminho que o projeto deve seguir. Em 1º de julho de 2004, o core consistia de 9 membros. As eleições são realizadas a cada dois anos.

5.1.4. Maintainership

Maintainership significa que essa pessoa é responsável pelo que é permitido entrar nessa área do código e tem a palavra final caso ocorram discordâncias sobre o código. Isso envolve trabalho proativo destinado a estimular contribuições e trabalho reativo na revisão de commits.

Com o código fonte do FreeBSD vem o arquivo MAINTAINERS que contém um resumo de uma linha de como cada mantenedor gostaria que as contribuições fossem feitas. Ter este aviso e informações de contato permite que os desenvolvedores se concentrem no esforço de desenvolvimento, em vez de ficarem presos em uma correspondência lenta, caso o mantenedor não esteja disponível por algum tempo.

Se o mantenedor não estiver disponível por um período de tempo excessivamente longo, e outras pessoas fizerem uma quantidade significativa de trabalho, a manutenção pode ser trocada sem a aprovação do mantenedor. Isto é baseado na postura de que a manutenção deve ser demonstrada, não declarada.

A manutenção de uma parte específica do código é um hat que não é mantido como um grupo.

5.2. Hats oficiais (Pessoas com funções definidas)

Os hats oficiais no Projeto FreeBSD são hats que são mais ou menos formalizados e, principalmente, com funções administrativas. Eles têm autoridade e responsabilidade por sua área. A ilustração a seguir mostra as linhas de responsabilidade. Depois disso segue uma descrição de cada hat, incluindo quem os mantém.

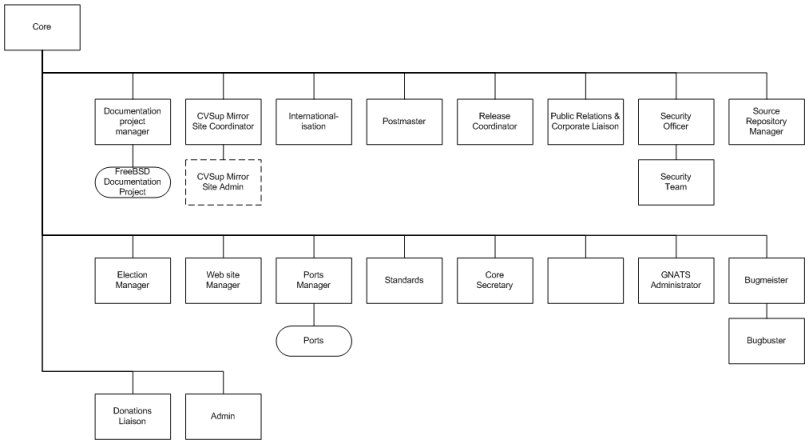

Figura 5.1. Visão geral dos hats (funções) oficiais

Todas as caixas consistem em grupos de committers, exceto as caixas pontilhadas onde os detentores não são necessariamente committers. Os círculos planificados são subprojetos e consistem em committers e pessoas que não são committers do projeto principal.

5.2.1. Gerente de Projetos de Documentação

The FreeBSD Documentation Project é o arquiteto responsável por definir e acompanhar as metas de documentação para os committers no projeto Documentação.

Hat mantido pela: Equipe do DocEng <doceng@FreeBSD.org >. O Capitulo DocEng.

5.2.2. Postmaster

O Postmaster é responsável pelo envio correto do email para o endereço de email dos committers. Ele também é responsável por garantir que as listas de discussão funcionem e deve tomar medidas contra possíveis interrupções de correspondência, como filtros de vírus, spam e trolls.

Hat atualmente mantido pelo: Time Postmaster <postmaster@FreeBSD.org >.

5.2.3. Coordenação de Release (Versões/Lançamentos)

As responsabilidades da Equipe de Engenharia de Release são

- Definir, publicar e seguir um cronograma de lançamento para releases (versões) oficiais
- Documentando e formalizando procedimentos de engenharia da release
- Criação e manutenção de branches de código
- Coordenando com as equipes de Ports e Documentação para ter um conjunto atualizado de pacotes e documentação lançados com as novas releases
- Coordenar com a equipe de segurança para que as versões pendentes não sejam afetadas por vulnerabilidades divulgadas recentemente.

Mais informações sobre o processo de desenvolvimento estão disponíveis na seção release engineering.

Hat mantido pela: Equipe de Engenharia de Release <re@FreeBSD.org >. O Capitulo de Engenharia de Release.

5.2.4. Relações Públicas & Contato Corporativo

Relações Publicas & As responsabilidades do contato corporativo são:

- Fazer declarações de imprensa quando acontecem coisas que são importantes para o projeto FreeBSD.
- Ser a pessoa de contato oficial para corporações que estão trabalhando em estreita colaboração com o Projeto FreeBSD.
- Tomar medidas para promover o FreeBSD tanto na comunidade Open Source quanto no mundo corporativo.
- Lidar com a lista de discussão “freebsd-advocacy”.

Este hat não está atualmente ocupado.

5.2.5. Oficial de segurança

A principal responsabilidade do Security Officer (Oficial de Segurança) é coordenar a troca de informações com outras pessoas na comunidade de segurança e no projeto FreeBSD. O agente de segurança também é responsável por tomar medidas quando os problemas de segurança são relatados e promover um comportamento proativo de desenvolvimento quando se trata de segurança.

Devido ao receio de que informações sobre vulnerabilidades possam vazar para pessoas com intenções maliciosas antes que um patch esteja disponível, apenas o Oficial de Segurança, composto por um oficial, um adjunto e dois membros do Core team, recebe informações confidenciais sobre problemas de segurança. No entanto, para criar ou implementar um patch, o Oficial de Segurança tem a equipe de segurança oficial <security-team@FreeBSD.org > para ajudar no trabalho.

5.2.6. Gerenciador do Repositório de Código Fonte

O Source Repository Manager (Gerenciador do Repositório de Código Fonte) é o único que tem permissão para modificar diretamente o repositório sem usar a ferramenta SVN. É sua responsabilidade garantir que os problemas técnicos que surgem no repositório sejam resolvidos rapidamente. O gerenciador do repositório de código fonte possui a autoridade para voltar commits, se isso for necessário para resolver um problema técnico do SVN.

Hat mantido pelo: Source Repository Manager <clusteradm@FreeBSD.org >.

5.2.7. Gerente de Eleições

O Gerente de Eleições é responsável pelo processo Core election. O gerente é responsável por executar e manter o sistema eleitoral, e é a autoridade final caso eventos imprevistos menores ocorram no processo eleitoral. Grandes imprevistos devem ser discutidos com o Core team

Hat realizado apenas durante as eleições.

5.2.8. Gerenciamento de sites

O hat Web site Management é responsável por coordenar o lançamento de páginas Web atualizadas em espelhos ao redor do mundo, pela estrutura geral do site principal e pelo sistema em que está sendo executado. O gerenciamento precisa coordenar o conteúdo com The FreeBSD Documentation Project e atua como mantenedor da árvore “www”.

Hat mantido pelos: Webmasters do FreeBSD <www@FreeBSD.org >.

5.2.9. Gerente de Ports

O Gerente de Ports atua como uma conexão entre The Ports Subproject e o projeto principal, e todas as solicitações do projeto devem ir para o gerente de ports.

Hat mantido pela: Equipe de Gerenciamento de Ports <portmgr@FreeBSD.org >. O Capitulo Portmgr.

5.2.10. Padrões

O Standards (Padrões) é responsável por garantir que o FreeBSD cumpra com os padrões com os quais está comprometido, mantendo-se atualizado sobre o desenvolvimento desses padrões e notificando os desenvolvedores do FreeBSD sobre mudanças importantes que lhes permitam assumir um papel proativo e diminuir o tempo entre atualização de padrões e complacência do FreeBSD.

Hat atualmente mantido por: Garrett Wollman <wollman@FreeBSD.org >.

5.2.11. Secretário do Core (do time do Core)

A principal responsabilidade do Secretário do Core é redigir os drafts e publicar os Relatórios finais do Core. O secretário também mantém a agenda central, assegurando assim que nenhuma bola seja descartada sem solução.

Hat atualmente mantido por: Joseph Mingrone <jrm@FreeBSD.org >.

5.2.12. Bugmeister

O Bugmeister é responsável por garantir que o banco de dados de manutenção esteja em ordem, que as entradas sejam categorizadas corretamente e que não existam entradas inválidas.

Hat atualmente mantido pelo: Time Bugmeister <bugmeister@FreeBSD.org >.

5.2.13. Oficial de Contratos de doações

A tarefa do oficial de contratos de doações é combinar os desenvolvedores com necessidades com pessoas ou organizações dispostas a fazer uma doação. A carta de ligação de doações está disponível aqui

Hat mantida pelo: Escritório de Contratos de Doações <donations@FreeBSD.org >.

5.2.14. Admin

(Também chamado de “Administrador de Cluster do FreeBSD”)

A equipe administrativa consiste nas pessoas responsáveis por administrar os computadores dos quais o projeto depende, para que seu trabalho distribuído e comunicação sejam sincronizados. Consiste principalmente naquelas pessoas que têm acesso físico aos servidores.

Hat mantido pela: Equipe de Admin <admin@FreeBSD.org >.

5.3. Hats dependentes de processo

5.3.1. Criador do relatório

A pessoa originalmente responsável pelo preenchimento de um Relatório de Problemas.

5.3.2. Bugbuster

Uma pessoa que irá encontrar a pessoa certa para resolver o problema, ou fechar o PR se for uma duplicata ou não uma interessante.

5.3.3. Mentor

Um mentor é um committer que assume a responsabilidade de introduzir um novo committer no projeto, tanto em termos de garantir que a nova configuração de committers seja válida, que o novo committer conheça as

ferramentas disponíveis necessárias em seu trabalho quanto que o novo committer saiba o que se espera dele em termos de comportamento.

5.3.4. Fornecedor

A(s) pessoa(s) ou organização de quem vem o código externo e para quem os patches são enviados.

5.3.5. Revisores

Pessoas na lista de discussão em que a solicitação de revisão é postada.

Capítulo 6. Processos

A seção a seguir descreverá os processos definidos do projeto. Questões que não são tratadas por esses processos acontecem em uma base ad-hoc com base no que era costume fazer em casos semelhantes.

6.1. Adicionando novos committers e removendo committers antigos

O Core team tem a responsabilidade de fornecer e remover os privilégios de commit aos colaboradores. Isso só pode ser feito por meio de votação na lista de discussão do core. Os subprojetos de ports e documentação podem conceder privilégios de commit a pessoas que trabalham nesses projetos, mas até o momento não removeram esses privilégios.

Normalmente, um colaborador é recomendado para o core por um committer. Para colaboradores ou pessoas de fora entrar em contato com o core pedindo para ser um committer não é algo bem pensado e geralmente é rejeitado.

Se a área de interesse particular para o desenvolvedor potencialmente se sobrepuser à área de manutenção de outros committers, a opinião desses commiters mantenedores é solicitada. No entanto, é frequentemente esse committer que recomenda o desenvolvedor.

Quando um colaborador recebe status de committer, ele recebe um mentor. O committer que recomendou o novo committer, no caso geral, assumirá a responsabilidade de ser o novo mentor dos committers.

Quando um colaborador recebe seu commit bit, um e-mail assinado PGP é enviado de Core Secretary, Ports Manager ou nik@freebsd.org para ambos admins@freebsd.org, o mentor designado, o novo committer e o core confirmando a aprovação de uma nova conta. O mentor então reúne uma senha, a chave pública SSH 2 e a chave PGP do novo committer e as envia para Admin. Quando a nova conta é criada, o mentor ativa o commit bit e orienta o novo committer pelo resto do processo inicial.

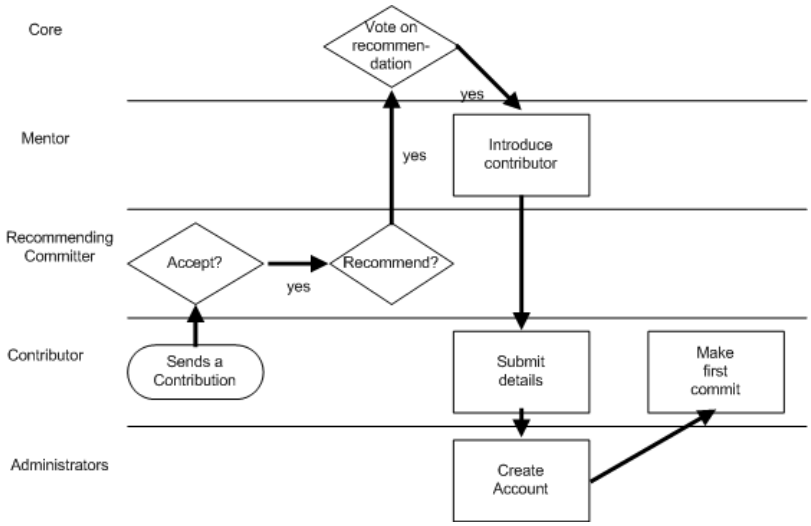

Figura 6.1. Resumo do processo: adicionando um novo committer

Quando um colaborador envia uma parte do código, o committer que recebe pode optar por recomendar que o colaborador receba privilégios de commit. Se ele recomendar isso para o core, eles irão votar essa recomendação. Se eles votarem a favor, um mentor é designado ao novo committer e o novo committer tem que enviar seus dados para os administradores para que uma conta seja criada. Depois disso, o novo committer está pronto para fazer seu primeiro commit. Por tradição, isso é feito adicionando seu nome à lista de committers.

Lembre-se de que um committer é considerado alguém que tenha feito algum commit de código nos últimos 12 meses. No entanto, somente após 18 meses de inatividade terem se passado, os privilégios de commit são

qualificados para serem revogados. [FreeBSD, 2002H] Não há, no entanto, procedimentos automáticos para fazer isso. Para reações a consentimentos de privilégios de commit não acionados pelo tempo, veja a seção 1.5.8.

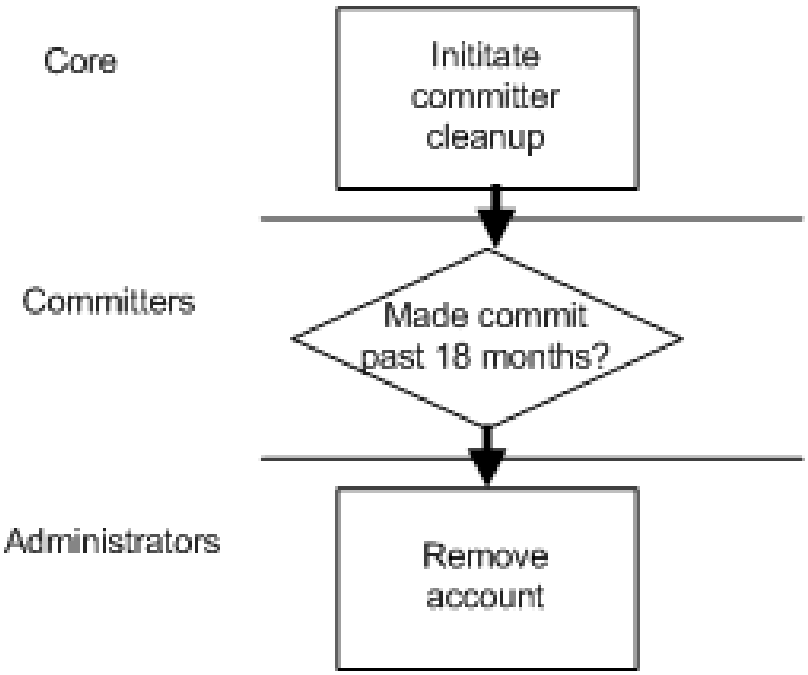

Figura 6.2. Resumo do processo: removendo um committer

Quando o Core decide limpar a lista de committers, eles checam quem não fez um commit nos últimos 18 meses. Os committers que não fizeram isso têm seus commit bit revogados.

Também é possível que os committers solicitem que seu commit bit seja retirado se, por alguma razão, eles não estiverem mais se comprometendo ativamente com o projeto. Nesse caso, ele também pode ser restaurado posteriormente pelo core, caso o committer peça.

Funções neste processo:

1. Core team
2. Contributor
3. Committer
4. Maintainership
5. Mentor

[FreeBSD, 2000A] [FreeBSD, 2002H] [FreeBSD, 2002I]

6.2. Enviando código para o projeto (Committing code)

O processo de commit de um código novo ou modificado é um dos processos mais frequentes no projeto FreeBSD e geralmente acontece muitas vezes ao dia. O commit do código só pode ser feito por um "committer". Committers aplicam código escrito por eles mesmos, código enviado a eles ou código enviado através de um relatório de problemas.

Quando o código é escrito pelo desenvolvedor que é não trivial, ele deve procurar uma revisão de código da comunidade. Isso é feito enviando e-mails para a lista relevante solicitando a revisão. Antes de enviar o código para

revisão, ele deve garantir que ele seja compilado corretamente com a árvore inteira e que todos os testes relevantes sejam executados. Isso é chamado “teste de pré-commit”. Quando o código contribuído é recebido, ele deve ser revisado pelo committer e testado da mesma maneira.

Quando uma alteração é "committed" em uma parte do código fonte que foi contribuída por um Vendor externo, o mantenedor deve garantir que o patch seja repassado ao fornecedor. Isso está de acordo com a filosofia de código aberto e facilita a sincronização com os projetos externos, pois os patches não precisam ser reaplicados sempre que uma nova versão é feita.

Depois que o código estiver disponível para revisão e nenhuma alteração adicional for necessária, o código será "committed" na branch de desenvolvimento, -CURRENT. Se a alteração se aplicar também à branch -STABLE ou às outras branches, uma contagem regressiva de um “Merge From Current” ("MFC") será definida pelo committer. Após o número de dias que o committer escolheu ao configurar o MFC, um email será enviado automaticamente ao committer, lembrando-o de enviá-lo para a branch -STABLE (e possivelmente também para branches de segurança). Apenas alterações críticas de segurança devem ser aplicadas a branch de segurança.

Atrasar o commit para -STABLE e outras branches permite “depuração paralela” onde o código "committed" é testado em uma ampla gama de configurações. Isso faz alterações no -STABLE para conter menos falhas e, assim, dar seu nome à branch.

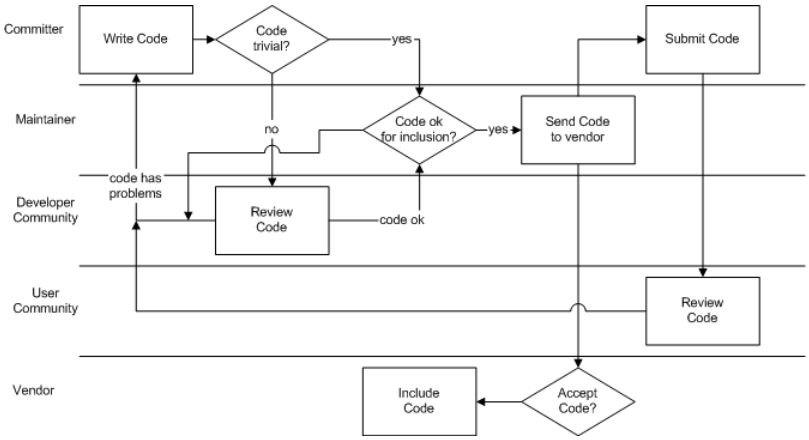

Figura 6.3. Resumo do processo: um committer aplica o código

Quando um committer escreveu um pedaço de código e quer fazer o seu commit, ele primeiro precisa determinar se é trivial o suficiente entrar sem uma análise prévia ou se deve ser revisado pela comunidade de desenvolvedores. Se o código é trivial ou foi revisado e o committer não é o mantenedor, ele deve consultar o mantenedor antes de continuar. Se o código for contribuído por um fornecedor externo, o mantenedor deve criar um patch que seja enviado de volta ao fornecedor. O código é então confirmado e implantado pelos usuários. Caso encontrem problemas com o código, isso será relatado e o committer poderá voltar a escrever um patch. Se um fornecedor for afetado, ele pode optar por implementar ou ignorar o patch.

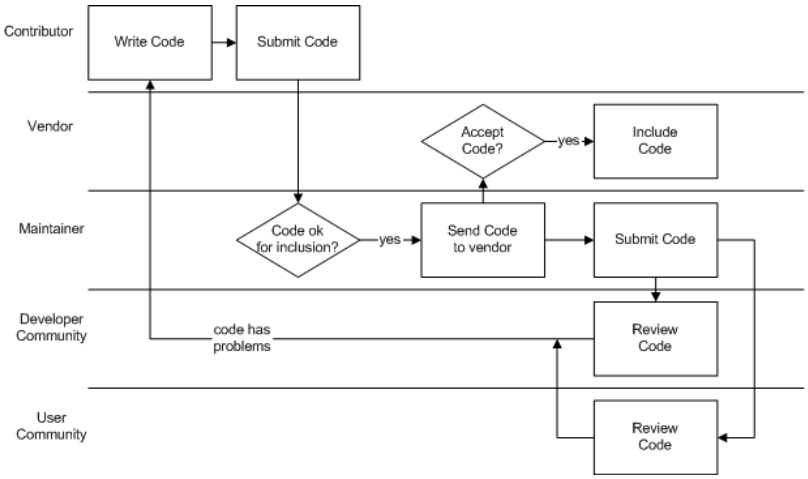

Figura 6.4. Resumo do processo: um colaborador envia o código

A diferença quando um colaborador faz uma contribuição de código é que ele envia o código através da interface do Bugzilla. Este relatório é escolhido pelo mantenedor que revisa o código e faz o seu commit.

Hats incluídos neste processo são:

1. Committer
2. Contributor
3. Vendor
4. Reviewer

[FreeBSD, 2001] [Jørgensen, 2001]

6.3. Eleição do Core

As eleições do core são realizadas pelo menos a cada dois anos. [1] Nove membros do core são eleitos. Novas eleições serão realizadas se o número de membros do core ficar abaixo de sete. Novas eleições também podem ser realizadas se pelo menos 1/3 dos committers ativos exigirem isso.

Quando uma eleição é realizada, o core anuncia isso com pelo menos 6 semanas de antecedência e nomeia um gerente eleitoral para dirigir as eleições.

Somente committers podem ser eleitos para o core. Os candidatos precisam apresentar sua candidatura pelo menos uma semana antes do início das eleições, mas podem refinar suas declarações até o início da votação. Eles são apresentados na lista de candidatos. Ao escrever suas declarações eleitorais, os candidatos devem responder a algumas perguntas padrões submetidas pelo gerente eleitoral.

Durante as eleições, a regra que um committer deve ter feito um commit durante os 12 meses passados é seguida estritamente. Apenas esses committers estão qualificados para votar.

Ao votar, o committer pode votar uma vez em apoio de até nove candidatos. A votação é feita durante um período de quatro semanas com lembretes sendo postados na lista de discussão "developers" que está disponível para todos os committers.

Os resultados das eleições são divulgados uma semana após o término da eleição, e a nova equipe do core toma posse uma semana após o lançamento dos resultados.

[1] A primeira eleição do core foi realizada em setembro de 2000

Se houver um empate de votação, isso será resolvido pelos novos membros do core, eleitos sem ambiguidade.

Votos e declarações de candidatos são arquivados, mas os arquivos não estão publicamente disponíveis.

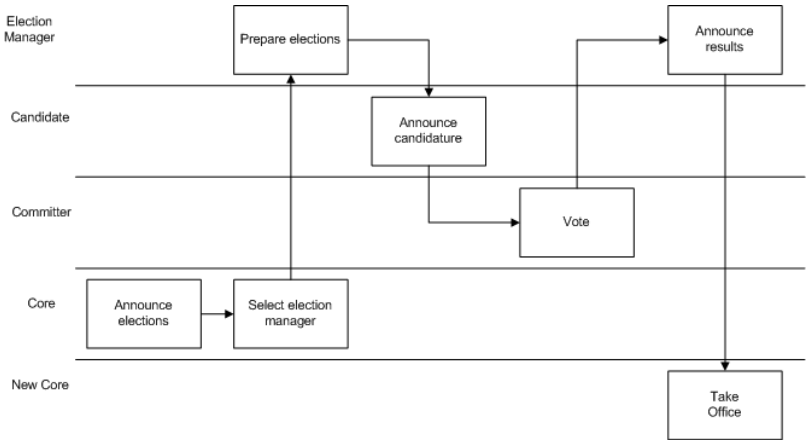

Figura 6.5. Resumo do processo: Eleições do Core

O Core anuncia a eleição e seleciona um gerente eleitoral. Ele prepara as eleições e, quando estiver pronto, os candidatos podem anunciar suas candidaturas através da apresentação de suas declarações. Os committers então votam. Após o término da votação, os resultados das eleições são anunciados e a nova equipe do core toma posse.

Hats nas eleições do Core são:

- Core team
- Committer
- Election Manager

[FreeBSD, 2000A] [FreeBSD, 2002B] [FreeBSD, 2002G]

6.4. Desenvolvimento de novos recursos

Dentro do projeto existem subprojetos que estão trabalhando em novos recursos. Esses projetos geralmente são feitos por uma pessoa [Jørgensen, 2001]. Todo projeto é livre para organizar o desenvolvimento como achar melhor. No entanto, quando é feito o merge do projeto (aplicado) à branch -CURRENT, ele deve seguir as diretrizes do projeto. Quando o código foi bem testado na branch -CURRENT e considerado estável o suficiente e relevante para a branch -STABLE, ele é mergeado à branch -STABLE.

Os requisitos do projeto são fornecidos como o desenvolvedor desejar, solicitações da comunidade em termos de solicitações diretas por correio, relatórios de problemas, financiamento comercial para o desenvolvimento de recursos ou contribuições da comunidade científica. Os desejos que estão sob a responsabilidade de um desenvolvedor são dados àquele desenvolvedor que prioriza seu tempo entre o pedido e seus desejos. Uma maneira comum de fazer isso é manter uma lista de tarefas mantidas pelo projeto. Itens que não são de responsabilidade de alguém são coletados em TODO-lists, a menos que alguém seja voluntário para assumir a responsabilidade. Todos os pedidos, sua distribuição e acompanhamento são tratados pela ferramenta Bugzilla.

A análise de requisitos acontece de duas maneiras. As solicitações que entram são discutidas em listas de discussão, tanto dentro do projeto principal quanto no subprojeto ao qual a solicitação pertence ou é gerada pela solicitação. Além disso, os desenvolvedores individuais no subprojeto avaliarão a viabilidade das solicitações e determinarão a priorização entre elas. Além dos arquivos das discussões que ocorreram, nenhum resultado é criado por essa fase que é incorporada ao projeto principal.

Como os pedidos são priorizados pelos desenvolvedores individuais com base em fazer o que eles acham interessante, necessário ou são financiados para fazer, não há estratégia geral ou priorização de solicitações

que considerem como requisitos e acompanhamento de sua implementação correta. No entanto, a maioria dos desenvolvedores tem uma visão compartilhada de quais problemas são mais importantes e podem solicitar diretrizes da equipe de engenharia de release.

A fase de verificação do projeto é dupla. Antes de fazer o commit do código para a branch atual, os desenvolvedores solicitam que seu código seja revisado por seus pares. Esta revisão é, na maior parte, feita por testes funcionais, mas a revisão de código também é importante. Quando o código é "committed" para a branch, um teste funcional mais amplo ocorrerá, o que pode acionar mais revisões de código e depuração, caso o código não se comporte conforme o esperado. Esta segunda forma de verificação pode ser considerada como verificação estrutural. Embora os próprios subprojetos possam escrever testes formais, como testes de unidade, eles geralmente não são coletados pelo projeto principal e geralmente são removidos antes que o código seja "committed" na branch atual. [2]

6.5. Manutenção

É uma vantagem para o projeto que para cada área do código fonte tenha pelo menos uma pessoa que conheça bem essa área. Algumas partes do código designaram mantenedores. Outros têm mantenedores de fato, e algumas partes do sistema não possuem mantenedores. O mantenedor é geralmente uma pessoa do subprojeto que escreveu e integrou o código, ou alguém que o portou da plataforma para a qual foi escrito. [3] O trabalho do mantenedor é garantir que o código esteja em sincronia com o projeto de onde vem o código, se for um código contribuído, e aplicar correções enviadas pela comunidade ou escrever correções em problemas descobertos.

A maior parte do trabalho que é colocado no projeto FreeBSD é a manutenção. [Jørgensen, 2001] fez uma figura mostrando o ciclo de vida das mudanças.

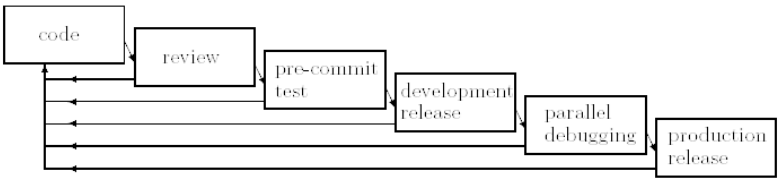

Figura 6.6. O modelo de Jørgenssen para integração de mudanças

Aqui, “desenvolvimento de release (versão)” refere-se a branch -CURRENT, enquanto o “release em produção” refere-se a branch -STABLE. O “teste de pré-commit” é o teste funcional feito por desenvolvedores de peers quando solicitado a fazê-lo ou a testar o código para determinar o status do subprojeto. “Debug paralelo” é o teste funcional que pode acionar mais revisões e debugs quando o código é incluído na branch -CURRENT.

A partir desta escrita, havia 269 committers no projeto. Quando eles fazem o commit de uma mudança em uma branch, isso constitui uma nova release (versão). É muito comum os usuários da comunidade rastrearem uma determinada branch. A existência imediata de uma nova release torna as mudanças amplamente disponíveis imediatamente e permite um feedback rápido da comunidade. Isso também dá à comunidade o tempo de resposta que eles esperam em questões que são importantes para eles. Isso torna a comunidade mais engajada e, assim, permite mais e melhores feedbacks que estimulam mais a manutenção e, por fim, deverá criar um produto melhor.

Antes de fazer alterações no código em partes da árvore que tem um histórico desconhecido para o committer, o committer é obrigado a ler os logs de commit para ver porque certos recursos são implementados da maneira que eles estão, para não cometer erros que foram anteriormente pensado ou resolvido.

[2] No entanto, mais e mais testes são executados ao construir o sistema (“make world”). Esses testes são, no entanto, uma adição muito nova e ainda não foi criada nenhuma estrutura sistemática para esses testes.

[3] sendmail e named são exemplos de código que foi feito merge (aplicado o código) de outras plataformas.

6.6. Relatório de Problemas

Antes do FreeBSD 10, o FreeBSD incluía uma ferramenta de relatório de problemas chamada `send-pr`. Os problemas incluem relatórios de bugs, solicitações de recursos, aprimoramentos de recursos e avisos de novas versões de software externo incluídas no projeto. Embora o `send-pr` esteja disponível, os usuários e desenvolvedores são incentivados a enviar problemas usando nosso formulário de relatório de problemas.

Os relatórios de problemas são enviados para um endereço de e-mail, onde são inseridos no banco de dados de manutenção de Relatórios de Problemas. Um Bugbuster classifica o problema e o envia ao grupo ou mantenedor correto dentro do projeto. Depois que alguém assume a responsabilidade pelo relatório, o relatório estará sendo analisado. Esta análise inclui verificar o problema e pensar em uma solução para o problema. Muitas vezes é necessário feedback do criador do relatório ou até mesmo da comunidade do FreeBSD. Uma vez que um patch para o problema é feito, o criador pode ser solicitado a testá-lo. Finalmente, o patch de trabalhado é integrado ao projeto e documentado, se aplicável. Ele passa pelo ciclo de manutenção regular, conforme descrito na seção maintenance. Estes são os estados em que um relatório de problemas pode estar: aberto, analisado, feedback, corrigido, suspenso e fechado. O estado suspenso é para quando um progresso adicional não é possível devido à falta de informação ou quando a tarefa exigiria tanto trabalho que ninguém está trabalhando nela no momento.

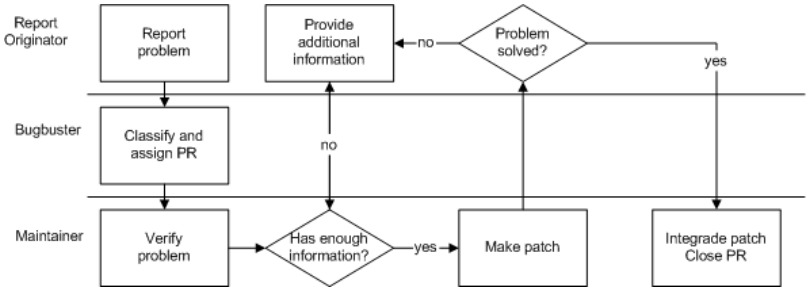

Figura 6.7. Resumo do processo: Relatório de Pproblemas

Um problema é relatado pelo criador do relatório. É então classificado por um bugbuster e entregue ao mantenedor correto. Ele verifica o problema e discute o problema com o criador até que ele tenha informações suficientes para criar um patch de trabalhado. Este patch é então "committed" e o relatório de problemas é fechado.

As funções incluídas neste processo são:

1. Report originator
2. Maintainership
3. Bugbuster

[FreeBSD, 2002C]. [FreeBSD, 2002D]

6.7. Reagindo ao mau comportamento

[FreeBSD, 2001] tem um número de regras que os committers devem seguir. No entanto, acontece que essas regras são quebradas. As seguintes regras existem para poder reagir ao mau comportamento. Eles especificam quais ações resultarão e em quanto tempo será uma suspensão dos privilégios de commit do committer.

- Fazer um commit durante o code freeze (tempo de congelamento de código) sem a aprovação da equipe dos Engenheiros de Release - 2 dias
- Fazer um commit em uma branch de segurança sem aprovação - 2 dias
- Guerras por causa de commit - 5 dias para todas as partes participantes
- Comportamento deselegante ou inapropriado - 5 dias

[Lehey, 2002]

Para que as suspensões sejam eficientes, qualquer membro do core pode implementar uma suspensão antes de discuti-la na lista de discussão “core”. Os infratores reincidentes podem, com um voto de 2/3 do core, receber penalidades mais duras, incluindo a remoção permanente dos privilégios de commit. (No entanto, este último é sempre visto como um último recurso, devido à sua tendência inerente de criar controvérsia). Todas as suspensões são postadas na lista de discussão “developers”, uma lista disponível apenas para committers.

É importante que você não possa ser suspenso por cometer erros técnicos. Todas as penalidades vêm de quebrar a etiqueta social.

Hats envolvidos neste processo:

- Core team
- Committer

6.8. Engenharia de Release (Versão)

O projeto FreeBSD possui uma equipe de Engenharia de Release com um engenheiro de release principal responsável por criar versões do FreeBSD que podem ser trazidas para a comunidade de usuários através da rede ou vendidas em lojas de varejo. Como o FreeBSD está disponível em várias plataformas e as releases para as diferentes arquiteturas são disponibilizadas ao mesmo tempo, a equipe tem uma pessoa responsável por cada arquitetura. Além disso, há cargos na equipe responsável por coordenar os esforços de garantia de qualidade, construir um conjunto de pacotes e ter um conjunto atualizado de documentos. Ao se referir ao engenheiro de release, um representante da equipe de engenharia de release é indicado.

Quando uma versão está chegando, o projeto do FreeBSD muda de forma um pouco. Um cronograma de lançamento é feito contendo congelamentos de recursos e códigos, liberação de versões intermediárias e a versão final. Um congelamento de recursos significa que nenhum novo recurso pode ser aplicado na branch sem o consentimento explícito dos engenheiros de release. O congelamento de código significa que nenhuma alteração no código (como bugs-fixes) pode ser aplicada sem o consentimento explícito dos engenheiros de release. Esse congelamento de recurso e código é conhecido como estabilização. Durante o processo de lançamento, o engenheiro de release tem a autoridade total para reverter para versões mais antigas de código e, assim, "desfazer" as alterações, caso ache que as alterações não são adequadas para serem incluídas na versão.

Existem três tipos diferentes de releases:

1. Releases .0 são o primeiro lançamento de uma versão principal. Eles são branches da branch -CURRENT e têm um ciclo de engenharia de release significativamente maior devido à natureza instável da branch -CURRENT
2. As versões .X são releases da branch -STABLE. Elas estão programadas para sair a cada 4 meses.
3. As versões .X.Y são versões de segurança que seguem a branch .X. Eles saem somente quando correções de segurança suficientes foram feito aplicadas desde o último release nessa branch. Novos recursos raramente são incluídos e a equipe de segurança está muito mais envolvida nesses recursos do que em releases regulares.

Para releases da branch -STABLE, o processo de release inicia 45 dias antes da data de lançamento prevista. Durante a primeira fase, os primeiros 15 dias, os desenvolvedores aplicam as alterações que tiveram no -CURRENT e que desejam ter na versão para a branch do release. Quando esse período terminar, o código entra em um congelamento de código de 15 dias, no qual apenas correções de bugs, atualizações de documentação, correções relacionadas à segurança e pequenas alterações de driver de dispositivo são permitidas. Essas alterações devem ser aprovadas pelo engenheiro de release com antecedência. No início do último período de 15 dias, um candidato a reçease é criado para testes generalizados. É menos provável que as atualizações sejam permitidas durante esse período, exceto por importantes correções de bugs e atualizações de segurança. Neste período final, todos os releases são considerados candidatos a release. No final do processo de release, uma release é criada com o novo número da versão, incluindo distribuições binárias em sites e a criação de imagens em CD-ROM. No entanto, a release não é

considerado "realmente liberado" até que uma mensagem PGP-assinada afirmando exatamente isso, seja enviada para a lista de discussão freebsd-announce; Qualquer coisa rotulada como "release" antes disso pode estar em processo e sujeita a alterações antes do envio da mensagem assinada pelo PGP. [4].

Os lançamentos da branch -CURRENT (ou seja, todas as releases que terminam com ".0") são muito semelhantes, mas com o dobro do prazo. Começa 8 semanas antes do lançamento com o anúncio da linha do tempo da release. Duas semanas após o processo de release, o congelamento de recursos é iniciado e os ajustes de desempenho devem ser mantidos ao mínimo. Quatro semanas antes do lançamento, uma versão beta oficial é disponibilizada. Duas semanas antes do lançamento, o código é oficialmente transformado em uma nova versão. Esta versão recebe status de release candidate e, como na engenharia de release do -STABLE, o congelamento de código do release candidate é endurecido. No entanto, o desenvolvimento na branch principal de desenvolvimento pode continuar. Além dessas diferenças, os processos de engenharia de release são semelhantes.

Releases .0 vão para o seu própria branch e são destinadas principalmente a adoções primárias. A branch passa por um período de estabilização, e não é até que o Release Engineering Team [13] decida que as demandas de estabilidade foram satisfeitas e de que o branch se torna -STABLE e -CURRENT segmenta a próxima versão principal. Enquanto isso para a maioria tem sido com versões .1, isso não é uma demanda.

A maioria das versões são feitas quando uma determinada data é considerada longa o suficiente desde o lançamento anterior. Uma data destino é definida para ter grandes releases a cada 18 meses e versões menores a cada 4 meses. A comunidade de usuários deixou bem claro que a segurança e a estabilidade não podem ser sacrificadas por prazos auto impostos e datas de lançamento desejadas. Por um lapso de tempo para não se tornar muito longo no que diz respeito a questões de segurança e estabilidade, é necessária uma disciplina extra ao aplicar alterações na -STABLE.

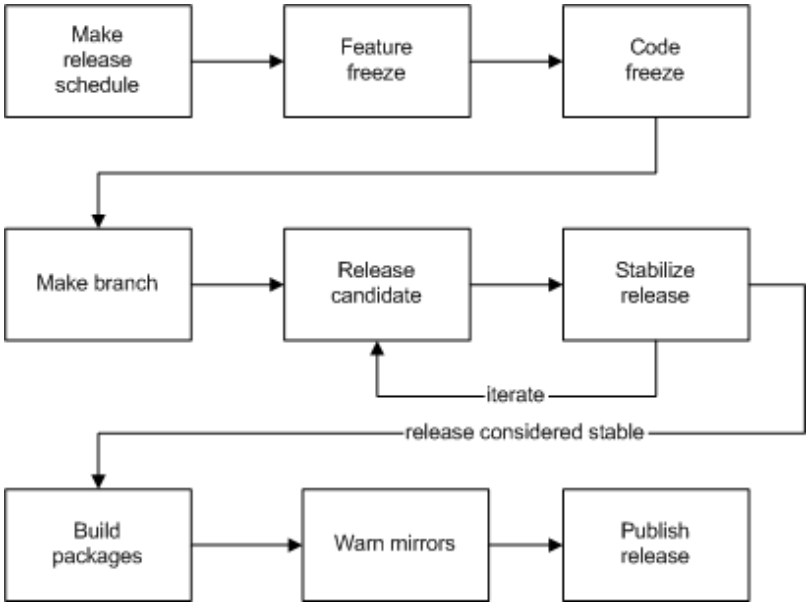

Figura 6.8. Resumo do processo: engenharia de release

Estas são as etapas do processo de engenharia de release. Vários candidatos a release podem ser criados até que a versão seja considerada estável o suficiente para ser liberada.

[FreeBSD, 2002E]

[4]Muitos fornecedores comerciais usam essas imagens para criar CD-ROMs que são vendidos em lojas de varejo.

Capítulo 7. Ferramentas

As principais ferramentas de suporte para suportar o processo de desenvolvimento são Bugzilla, Mailman e OpenSSH. Estas são ferramentas desenvolvidas externamente e são comumente usadas no mundo do código aberto.

7.1. Subversion (SVN)

Subversion ("SVN") é um sistema para lidar com múltiplas versões de arquivos de texto e rastrear quem aplicou mudanças e por quê. Um projeto vive dentro de um "repositório" e diferentes versões são consideradas "branches" diferentes.

7.2. Bugzilla

O Bugzilla é um banco de dados de manutenção que consiste em um conjunto de ferramentas para rastrear bugs em um site central. Ele suporta o processo de rastreamento de bugs para envio e tratamento de bugs, além de consultar e atualizar o banco de dados e editar relatórios de bugs. O projeto usa sua interface web para enviar "Relatórios de Problemas" para o servidor central do Bugzilla. Os committers também possuem clientes web e de linha de comando.

7.3. Mailman

Mailman é um programa que automatiza o gerenciamento de listas de discussão. O Projeto FreeBSD o utiliza para executar 16 listas gerais, 60 listas técnicas, 4 listas limitadas e 5 listas com logs de commit do CVS. Também é usado para muitas listas de discussão configuradas e usadas por outras pessoas e projetos na comunidade FreeBSD. Listas gerais são listas para o público em geral, listas técnicas são principalmente para o desenvolvimento de áreas específicas de interesse, e listas fechadas são para comunicação interna não destinadas ao público em geral. A maior parte de toda a comunicação no projeto passa por essas 85 listas, [FreeBSD, 2003A, Apêndice C].

7.4. Pretty Good Privacy

Pretty Good Privacy, mais conhecida como PGP, é um sistema criptográfico que usa uma arquitetura de chave pública para permitir que as pessoas assinem e/ou criptografem digitalmente as informações, a fim de garantir a comunicação segura entre as duas partes. Uma assinatura é usada ao enviar informações a muitos destinatários, permitindo que eles verifiquem se as informações não foram adulteradas antes de recebê-las. No Projeto FreeBSD este é o principal meio de assegurar que a informação tenha sido escrita pela pessoa que afirma ter escrito, e não alterada em trânsito.

7.5. Secure Shell

O Secure Shell é um padrão para efetuar login com segurança em um sistema remoto e para executar comandos no sistema remoto. Ele permite que outras conexões, chamadas túneis, sejam estabelecidas e protegidas entre os dois sistemas envolvidos. Este padrão existe em duas versões primárias, e somente a versão dois é usada para o projeto FreeBSD. A implementação mais comum do padrão é o OpenSSH, que faz parte da distribuição principal do projeto. Uma vez que sua fonte é atualizada com mais frequência do que as liberações do FreeBSD, a versão mais recente também está disponível na árvore de ports.

Capítulo 8. Sub-projetos

Subprojetos são formados para reduzir a quantidade de comunicação necessária para coordenar o grupo de desenvolvedores. Quando uma área problemática é suficientemente isolada, a maior parte da comunicação estaria dentro do grupo, focando no problema, exigindo menos comunicação com os grupos com os quais eles se comunicam do que se o grupo não estivesse isolado.

8.1. Subprojeto Ports

Um "port" é um conjunto de meta-dados e patches que são necessários para buscar, compilar e instalar corretamente um software externo em um sistema FreeBSD. A quantidade de ports cresceu a um ritmo tremendo, como mostra a figura a seguir.

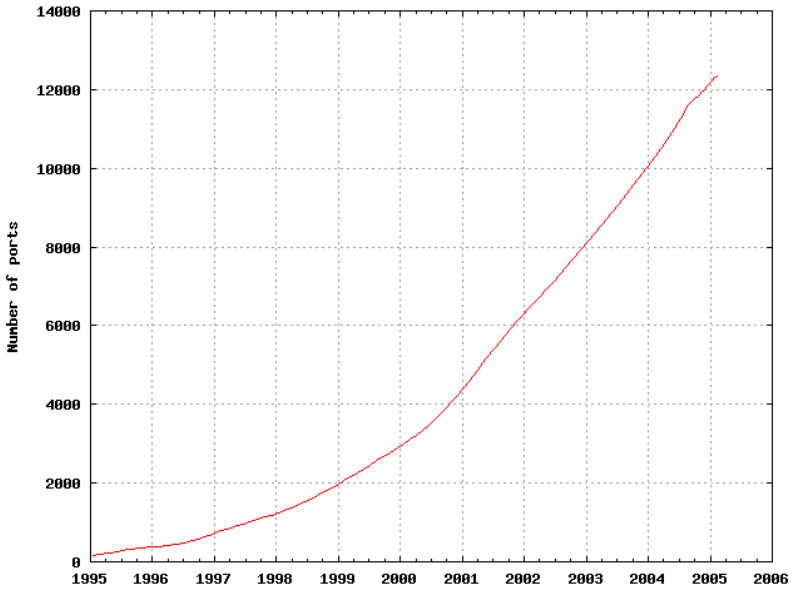

Figura 8.1. Número de ports adicionados entre 1996 e 2005

Figura 8.1, "Número de ports adicionados entre 1996 e 2005" é obtido do site do FreeBSD. Ele mostra o número de ports disponíveis para o FreeBSD no período de 1995 a 2005. Parece que a curva cresceu primeiro exponencialmente e, desde meados de 2001, cresceu linearmente.

Como o software externo descrito pelo port geralmente está em desenvolvimento contínuo, a quantidade de trabalho necessária para manter os ports já é grande e está aumentando. Isso fez com que a parte dos ports do projeto FreeBSD ganhasse uma estrutura mais poderosa, e está se tornando cada vez mais um subprojeto do projeto FreeBSD.

Ports tem seu próprio core team (time principal) com o Ports Manager como seu líder, e esta equipe pode indicar committers sem a aprovação do FreeBSD Core. Ao contrário do Projeto FreeBSD, onde muitas tarefas de manutenção são frequentemente recompensadas com um commit bit, o subprojeto ports contém muitos mantenedores ativos que não são committers.

Ao contrário do projeto principal, a árvore de ports não é ramificada (tem uma branch própria). Cada versão do FreeBSD segue a coleção atual de ports e, portanto, disponibiliza informações atualizadas sobre onde encontrar programas e como construí-los. Isso, no entanto, significa que um port que faz dependências no sistema pode precisar ter variações dependendo de qual versão do FreeBSD é executada.

Com um repositório de ports não ramificado (sem branches), não é possível garantir que qualquer port seja executado em algo diferente de -CURRENT e -STABLE, em particular liberações secundárias e antigas. Não há infraestrutura nem tempo de voluntariado necessário para garantir isso.

Para eficiência de comunicação, as equipes que dependem de ports, como a equipe de engenharia de release, têm suas próprias conexões com ports.

8.2. O projeto de documentação do FreeBSD

O projeto de Documentação do FreeBSD foi iniciado em janeiro de 1995. Desde o grupo inicial de um líder de projeto, quatro líderes de equipe e 16 membros, eles são agora um total de 44 committers. A lista de discussão de documentação tem pouco menos de 300 membros, indicando que há uma grande comunidade em torno dela.

O objetivo do projeto de Documentação é fornecer uma documentação boa e útil do projeto FreeBSD, facilitando assim que novos usuários se familiarizem com o sistema e detalhando recursos avançados para os usuários.

As principais tarefas no projeto de Documentação são trabalhar em projetos atuais no “Conjunto de Documentação do FreeBSD”, e traduzir a documentação para outros idiomas.

Como o Projeto FreeBSD, a documentação é dividida nas mesmas branches. Isso é feito para que sempre haja uma versão atualizada da documentação de cada versão. Apenas erros de documentação são corrigidos nas branches de segurança.

Como o sub-projeto ports, o projeto de Documentação pode indicar committers de documentação sem a aprovação do FreeBSD Core. [FreeBSD, 2003B].

O projeto de Documentação possui uma cartilha. Isso é usado para apresentar novos membros de projeto às ferramentas e sintaxes padrão e atua como referência ao trabalhar no projeto.

Referências

[1] Frederick P. Brooks. Copyright © 1975, 1995 Pearson Education Limited. 0201835959. Addison-Wesley Pub Co. *The Mythical Man-Month*. Essays on Software Engineering, Anniversary Edition (2nd Edition).

[2] Niklas Saers. Copyright © 2003. *Um modelo de projeto para o projeto FreeBSD*. Tese de Candidatus Scientiarum. http://niklas.saers.com/thesis.

[3] Niels Jørgensen. Copyright © 2001. *Colocando tudo no porta-malas*. Desenvolvimento Incremental de Software no Projeto Open Source do FreeBSD. http://www.dat.ruc.dk/~nielsj/research/papers/freebsd.pdf.

[4] Instituto de Gerenciamento de Projeto. Copyright © 1996, 2000 Instituto de Gerenciamento de Projetos. 1-880410-23-0. Instituto de Gerenciamento de Projetos. Newtown Square Pennsylvania USA . *Guia PMBOK*. A Guide to the Project Management Body of Knowledge, 2000 Edition.

[5] Copyright © 2002 O projeto FreeBSD. *Estatuto do Core*. https://www.freebsd.org/internal/bylaws.html.

[6] Copyright © 2002 O Projeto de Documentação do FreeBSD. *Manual do desenvolvedor do FreeBSD*. https://www.freebsd.org/doc/en_US.ISO8859-1/books/developers-handbook/.

[7] Copyright © 2002 O projeto FreeBSD. *Eleição da equipe do Core em 2002*. http://election.uk.freebsd.org/candidates.html.

[8] Dag-Erling Smørgrav e Hiten Pandya. Copyright © 2002 O Projeto de Documentação do FreeBSD. O projeto de documentação do FreeBSD. *Diretrizes para manuseio de Relatórios de Problemas*. https://www.freebsd.org/doc/en/articles/pr-guidelines/article.html.

[9] Dag-Erling Smørgrav. Copyright © 2002 O Projeto de Documentação do FreeBSD. O projeto de documentação do FreeBSD. *Escrevendo Relatórios de Problemas do FreeBSD*. https://www.freebsd.org/doc/en/articles/problem-reports/article.html.

[10] Copyright © 2001 O Projeto de Documentação do FreeBSD. O projeto de documentação do FreeBSD. *Guia para Commiters*. https://www.freebsd.org/doc/en/articles/committers-guide/article.html.

[11] Murray Stokely. Copyright © 2002 O Projeto de Documentação do FreeBSD. O projeto de documentação do FreeBSD. *Engenharia de Release do FreeBSD*. https://www.freebsd.org/doc/en_US.ISO8859-1/articles/releng/article.html.

[12] Projeto de Documentação do FreeBSD. *Manual do FreeBSD*. https://www.freebsd.org/doc/en_US.ISO8859-1/books/handbook.

[13] Copyright © 2002 O Projeto de Documentação do FreeBSD. O projeto de documentação do FreeBSD. *Colaboradores para o FreeBSD*. https://www.freebsd.org/doc/en_US.ISO8859-1/articles/contributors/article.html.

[14] Copyright © 2002 O projeto FreeBSD. O projeto FreeBSD. *Eleições da equipe do Core em 2002*. http://election.uk.freebsd.org.

[15] Copyright © 2002 O projeto FreeBSD. O projeto FreeBSD. *Política de expiração de commit bit*. 2002/04/06 15:35:30. https://www.freebsd.org/internal/expire-bits.html.

[16] Copyright © 2002 O projeto FreeBSD. O projeto FreeBSD. *Novo procedimento de criação de conta*. 2002/08/19 17:11:27. https://www.freebsd.org/internal/new-account.html.

[17] Copyright © 2002 O Projeto de Documentação do FreeBSD. O projeto de documentação do FreeBSD. *Capitulo da Equipe do DocEng do FreeBSD*. 2003/03/16 12:17. https://www.freebsd.org/internal/doceng.html.

[18] Greg Lehey. Copyright © 2002 Greg Lehey. Greg Lehey. *Dois anos nas trincheiras*. A evolução de um projeto de software. http://www.lemis.com/grog/In-the-trenches.pdf.