Caracterização de um Processo de Software para Projetos de Software Livre

Christian Robottom Reis
kiko@async.com.br

Orientação:
Profa. Dra. Renata Pontin de Mattos Fortes
renata@icmc.usp.br

Dissertação apresentada ao Instituto de Ciências Matemáticas e de Computação da Universidade de São Paulo para a obtenção do título de Mestre em Ciências da Computação e Matemática Computacional.

São Carlos, São Paulo
Fevereiro de 2003

Resumo

Software Livre é software fornecido com código fonte, e que pode ser livremente usado, modificado e redistribuído. Projetos de Software Livre são organizações virtuais formadas por indivíduos que trabalham juntos no desenvolvimento de um software livre específico. Estes indivíduos trabalham geograficamente dispersos, utilizando ferramentas simples para coordenar e comunicar seu trabalho através da Internet.

Este trabalho analisa esses projetos do ponto de vista de seu processo de software; em outras palavras, analisa as atividades que realizam para produzir, gerenciar e garantir a qualidade do seu software. Na parte inicial do trabalho é feita uma extensa revisão bibliográfica, comentando os principais trabalhos na área, e são detalhadas as características principais dos projetos de software livre. O conteúdo principal deste trabalho resulta de dois anos de participação ativa na comunidade, e de um levantamento realizado através de questionário, detalhando mais de quinhentos projetos diferentes. São apresentadas treze hipóteses experimentais, e os resultados do questionário são discutidos no contexto destas hipóteses.

Dos projetos avaliados nesse levantamento, algumas características comuns foram avaliadas. As equipes da grande maioria dos projetos são pequenas, tendo menos de cinco participantes. Além disso, existe uma distribuição equilibrada entre algumas formas de organização descritas na literatura, incluindo o 'ditador benevolente' de Eric S. Raymond e o 'comitê' exemplificado pelo projeto Apache. Dentre um conjunto de domínios de aplicação determinados, os projetos de software livre se concentram nas áreas de engenharia e desenvolvimento de software, redes e segurança, e aplicações multimídia.

Com relação às atividades do processo de software, pode-se dizer que a maioria dos projetos tem requisitos fundamentalmente definidos pelos seus autores, e que a base de usuários de grande parte dos softwares é composta dos seus desenvolvedores e da comunidade de software livre. Uma parcela significativa dos projetos baseia-se em outros softwares pré-existentes, e em padrões publicados previamente. Pouca ênfase é dada à usabilidade, assim como às atividades de garantia de qualidade convencionais. Surpreendentemente, também recebem pouca atenção as atividades de revisão de código e teste sistemático. Entre as ferramentas que os projetos utilizam, se destacam as listas de discussão e os sistemas de controle de versão.

Foi estabelecida uma correlação entre a dimensão da equipe do projeto e as atividades de engenharia de software que realiza, mas não se confirmou um vínculo entre estas atividades e a idade do projeto. Foram também estabelecidas relações entre o número de linhas de código do software do projeto e a sua idade, tamanho e domínio de aplicação. Estes resultados são exibidos neste trabalho, e estarão publicamente disponíveis no site Web do projeto. O trabalho conclui descrevendo partes essenciais do processo de software em projetos de software livre, e oferecendo sugestões para trabalhos posteriores.

Palavras-Chave: Software Livre, Open Source, Processo de Software, Ciclo de Vida, Modelo de Processo de Software, Engenharia de Software, Desenvolvimento de Software Descentralizado.

Abstract

Free Software (or Open Source) is software provided with source code that may be freely used, modified and redistributed. Free Software Projects are virtual communities of developers that work on a specific free software product. These developers work geographically dispersed, using simple tools to communicate and coordinate their actions over the Internet.

This work analyzes these projects with respect to their software process; in other words, the activities they do to produce, manage and ensure the quality of their software. In the initial sections, I perform a large review of related work in this field, and provide an overview of the main characteristics of free software projects. The main contents of this dissertation is based on two years of participation in free software projects, and on a survey based on a questionnaire that describes over five hundred different free software projects. I present thirteen initial hypothesis, using them as a framework to analyze the results of the survey.

Among the projects evaluated in this survey, some common aspects were observed. For instance, most projects have a small team, with less than five developers. I also found a balanced distribution of leadership systems described in other works, including Eric Raymond's 'benevolent dictator' and the committee exemplified by the Apache project. The domains in which these projects tend to concentrate are software engineering and development, networks and security, and multimedia applications.

Concerning the software process activities, most projects have their functional requirements established by their authors: the user base for most of the projects includes the development team and the free software community. A large proportion of the projects surveyed are based on pre-existing software, or on previously published standards. Little emphasis is placed on usability and conventional forms of quality assurance. Remarkably, this lack of emphasis on conventional process includes activities like review and testing. Among the tools used by the projects, mailing lists and versioning systems such as CVS come out ahead by a strong margin.

A correlation was identified between the size of the project's team and the software engineering activities that it realizes, but I could not establish a link between the age of the project and these activities. Other correlations were established between the size of the project's codebase (in terms of lines of code) and its age, size and application domain. The dissertation concludes describing essential parts of the software process in free software projects, and offering suggestions for future work.

Keywords: Free Software, Open Source, Software Process, Software Lifecycle, Software Process Model, Software Engineering, Distributed Software Development.

Conteúdo

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

O desafio fundamental que a Engenharia de Software enfrenta é o mesmo há décadas: **Como construir software melhor?** Entre processos, ferramentas e organizações alternativas, a literatura consolida experiência e inovação resultantes do trabalho de milhares de pesquisadores, motivados em descrever (e prescrever) formas mais eficazes de lidar com os problemas inerentes à construção de software. Resultam destes esforços um conjunto de modelos e normas: CMM, SPICE e ISO/IEC 12.207 [1, 2].

Simultaneamente, uma proporção pequena (porém crescente) de software vem sendo desenvolvida por grupos de indivíduos independentes, que trabalham geograficamente dispersos segundo uma filosofia que só pode ser descrita como original: o software que produzem pode ser livremente utilizado e modificado por qualquer pessoa que se interessar. Originalmente raros e reduzidos, estes grupos vêm estabelecendo-se gradualmente como organizações mais consolidadas, com nome próprio, equipe e missão: os Projetos de Software Livre.

É curioso que possa ter sucesso mundial um modelo de desenvolvimento aparentemente fundamentado no trabalho de amadores — e voluntários — coordenados de maneira pouco formal usando ferramentas extremamente simples. Mais surpreendente é a percepção informal entre a comunidade Internet e os meios de comunicação de que o software produzido por estes projetos tem qualidade. Exemplos de destaque incluem o Linux, um núcleo de sistema operacional; Apache, o servidor web mais difundido na Internet (`www.netcraft.com`); o Sendmail e o QMail, juntos responsáveis por transferir mais de 50% do tráfego de correio eletrônico mundial [3]. O sucesso aparente destes softwares motiva duas perguntas espontâneas:

1. Como tornar software livre economicamente viável?
2. Como projetos de software livre produzem software?

Um número de publicações da literatura econômica e administrativa se propõe a discutir a primeira pergunta [4, 5]. No entanto, apenas recentemente a comunidade da computação, e em particular da engenharia de software, têm manifestado interesse com relação à segunda. Para garantir que esta comu-

nidade se mantenha em sincronia com o estado da arte do desenvolvimento de software, é importante a existência de um roteiro abrangente que descreva o processo executado nestes projetos de software livre.

Este trabalho de mestrado propõe um roteiro potencial. Não tem, obviamente, a intenção de fornecer O Roteiro Final; busca ao invés disso oferecer definições bem-fundamentadas, experiências práticas e um conjunto rico de dados empíricos para guiar novos estudos e potenciais descobertas nesta área.

1.1 Motivação

Existe interesse em compreender melhor o processo de desenvolvimento em projetos de software livre: o número crescente de publicações na comunidade de engenharia de software demonstra esta premissa. Estes projetos podem fornecer à comunidade idéias alternativas a respeito da organização, motivação e coordenação de equipes de desenvolvimento, enriquecendo o conhecimento relativo ao processo de software. Por sua vez, a comunidade de engenharia de software, dado seu extenso conhecimento acumulado, pode contribuir para que os projetos produzam software melhor, de forma mais eficaz.

Nos últimos anos, um número de publicações científicas tem buscado descrever estes projetos de forma mais detalhada. No entanto, grande parte destas trabalhos se concentra em projetos individuais, ou em aspectos particulares do processo de software. Ao invés de prover uma visão ampla do cenário de projetos de software livre, a síntese destes trabalhos resulta em uma descrição esparsa, com lacunas importantes relacionadas ao processo de software aplicado nos projetos. Quão representativas são estas amostras e análises individuais?

Existem ainda estudos empíricos a respeito de populações maiores de projetos de software livre [6, 7, 8], mas estes abordam predominantemente aspectos de demografia e motivação, alguns em conjunto com análise automatizada do código-fonte produzido. Não há um levantamento objetivo realizado entre uma população de projetos de software livre que consolide informações a respeito de seu processo de desenvolvimento.

1.2 Problema de Pesquisa

A existência de um grande número de projetos de software livre em pleno desenvolvimento indica uma alternativa funcional, mas pouco conhecida, às organizações convencionais que constroem software. Do conjunto de trabalhos científicos realizados a respeito destes projetos, é possível identificar diversos processos de software distintos.

Este trabalho se propõe a discutir a questão fundamental relacionada ao processo de software em projetos de software livre: **como é construído software nestes projetos**, considerando as atividades fundamentais do processo de software. Aborda esta questão de uma maneira abrangente, definindo características do processo para uma população representativa de projetos. Com base nestas informações,

o trabalho busca ainda analisar a existência de um processo comum entre eles, e avaliar diferenças e semelhanças com os processos de software convencionais descritos na literatura.

1.3 Organização da Dissertação

Esta dissertação apresenta uma descrição completa do **domínio do problema**, da **metodologia de pesquisa** aplicada, e dos **resultados obtidos**. A estrutura do trabalho segue a seguinte seqüência:

- Os Capítulos 2, 3 e 4 descrevem o domínio do problema: o Processo de Software, Software Livre e Projetos de Software Livre.
- O Capítulo 5 faz uma visão geral do estado da arte presente na literatura científica a respeito do domínio do problema.
- O Capítulo 6 detalha os problemas de pesquisa, e apresenta a metodologia experimental utilizada, que inclui observação participante e a aplicação de um questionário.
- Os Capítulos 7 e 8 descrevem os resultados obtidos destes experimentos.
- O capítulo 9 discute as conclusões obtidas com este trabalho, e indica trabalhos futuros.

1.4 Convenções de Formato

Ao longo desta dissertação, utiliza-se um padrão de formatação baseado nos seguintes critérios:

- Há um número de trechos em inglês nesta dissertação. Parágrafos em inglês citados da literatura são acompanhados de referência bibliográfica e traduzidos no rodapé da página. Citações curtas em inglês aparecem traduzidas entre parênteses no próprio texto.
- Neste trabalho, um questionário público foi realizado. Questões e comentários obtidos do questionário aparecem literalmente e não são traduzidos, evitando modificar seu sentido original.
- Termos específicos em idioma estrangeiro aparecem em *itálico* apenas na sua primeira ocorrência.
- Definições que aparecem correntemente no texto podem ser destacadas com o uso de **negrito**; ênfase no texto é indicada por *itálico*. Definições e descrições importantes são fornecidas em quadros destacados
- Referências são indicadas numericamente e incluídas ao final do texto. Endereços Web curtos são incluídas no próprio texto, entre parênteses; endereços mais longos são indicados no rodapé.

Capítulo 2

Processo de Software

A proposta desta dissertação é analisar o processo de software implementado nos projetos de software livre. Tendo em vista a amplitude desta proposta, o primeiro desafio é definir um conjunto de aspectos concretos em relação aos quais os projetos serão analisados.

O Processo de Software é redefinido freqüentemente na literatura fundamental da Engenharia de Software [2, 9, 10]. Neste capítulo, é oferecida uma visão prática do que é este processo, baseada numa síntese destas definições.

2.1 Definição

Engenharia de Software consiste no estabelecimento de princípios e boas práticas para a implementação eficaz de software. É um tema amplo e que possui vasta quantidade de trabalhos publicados: uma busca por 'software engineering' entre os documentos indexados pelo serviço de referência de publicações científicas CiteSeer (`www.researchindex.org`) lista mais de 10.000 documentos.

O enfoque deste trabalho é sobre uma área específica deste universo de conhecimento, que discute os **processos** pelos quais pode (ou deve) ser implementado software. Por processo, entende-se um conjunto de passos envolvendo atividades, limitações e recursos que produzem um resultado [2]. Com base nesta afirmação, uma definição útil de processo de software para este trabalho pode ser sintetizada:

> Processo de Software é um conjunto de atividades realizadas para construir software, levando em consideração os produtos sendo construídos, as pessoas envolvidas, e as ferramentas com as quais trabalham.

É importante deixar claro que um processo de software particular, da forma como implementado por uma organização, é algo concreto e individualizado: cada equipe de desenvolvimento **instancia** sua versão. Uma descrição de um processo de software específico oferece uma forma de entender e avaliar seus objetivos, forças e fraquezas. Avaliar Software Livre neste contexto tem potencial significativo:

oferece uma forma estruturada de avaliar as características individuais do processo implementado para sua construção.

Nas seções seguintes será feita uma definição das atividades principais que constituem o processo de software, com base num consenso entre a literatura fundamental. Uma visão mais completa do processo é fornecida pela norma ISO/IEC 12.207 [1].

2.2 Atividades Essenciais

As atividades aqui consideradas essenciais são as fundamentais para a construção de software. Estas atividades podem ser resumidas em:

Análise e Definição de Requisitos: é o processo de descobrir e detalhar quais funcionalidades (e quais limitações funcionais) o software requer; um **requisito** é a descrição de algo que o sistema deve ser capaz de realizar. Em projetos convencionais é freqüentemente realizado um processo de engenharia de requisitos com a participação do usuário final do sistema, já que este é normalmente bom conhecedor do domínio do problema a ser solucionado.

Freqüentemente são associados às atividades do processo de software um ou mais **artefatos**: são seus produtos concretos. O artefato convencionalmente associado à atividade de análise de requisitos é um documento de especificação de requisitos.

Projeto Arquitetural e Detalhado: esta atividade é um detalhamento iterativo de uma solução computacional para atender aos requisitos elicitados. O projeto é normalmente feito em níveis; nos níveis mais abstratos, são definidas as entradas e saídas do software, e as entidades com quais ele se relaciona. Progressivamente refina-se este projeto em partes menores: programas, módulos, funções, variáveis. Aos níveis mais abstratos é dado o nome de projeto arquitetural, a aos mais refinados, projeto detalhado.

Existem diversas técnicas para realizar este refinamento e descrevê-lo; em geral, os artefatos desta atividade são descrições textuais e diagramas que descrevem os componentes e seus relacionamentos. Existem diversas notações diferentes para projeto, incluindo a recente e popular UML (*Unified Modelling Language*) [11].

Codificação: consiste na criação do código-fonte, implementando a funcionalidade especificada nos requisitos, de acordo com o modelo definido no projeto. Esta atividade envolve escrever código, depurá-lo e integrá-lo progressivamente. Para a realização desta atividade, desenvolvedores normalmente utilizam os artefatos de requisitos e projeto produzidos, e interagem com as equipes responsáveis por estes.

Teste de Unidade, de Integração e de Sistema: embora seja comum teste *ad hoc* durante o processo de codificação, a *atividade de teste* envolve uma abordagem mais sistemática. Assim como

projeto, teste pode ser descrito como uma atividade progressiva, indo do elemento menor para o maior. Por exemplo, o teste de unidade cobre partes individuais do sistema (funções, módulos e interfaces); já o teste de integração verifica o funcionamento destas partes uma vez combinadas. O teste de sistema valida os aspectos funcionais do sistema, incluindo desempenho, adequação e a consistência da funcionalidade implementada.

Existem divisões mais refinadas dos tipos de testes realizados em projetos de maior escala. É importante notar que a atividade de testes pode ocorrer simultaneamente à codificação; Kent Beck, por exemplo, na descrição da metodologia XP (*eXtreme Programming*) [12], sugere implementar testes de unidade **antes** mesmo de se codificar funcionalidade.

Lançamento: uma vez o sistema (ou uma parte essencial, para processos iterativos ou cíclicos) estando pronto e validado, pode ser realizado seu lançamento. Durante um lançamento, normalmente ocorre a consolidação da documentação e ajuda *online*, o empacotamento do software, e a sua distribuição, entrega ou instalação. Esta fase é considerada crítica, uma vez que é normalmente o primeiro contato do usuário final com o sistema.

É comum em processos iterativos ou cíclicos ocorrerem múltiplos 'mini-lançamentos'; neste caso, a atividade de lançamento tem impacto reduzido pelo fato de se aliviar os riscos associados à entrada do sistema no ambiente do usuário; esta é feita gradualmente, com cada 'mini-lançamento' individual.

Estas atividades são consideradas as mínimas necessárias para a construção efetiva de software [2]. Não são necessariamente seqüenciais [13], freqüentemente ocorrendo superposição entre elas por períodos de tempo. Em outras palavras, não se pode determinar uma relação temporal estrita entre elas; existe, na realidade, uma relação mais sutil de interdependência que tende a ordenar sua realização.

Por exemplo, a atividade de implementação requer que se saiba o que está sendo construído, de forma que depende de resultados obtidos durante a análise de requisitos. No entanto, não é realista dizer que para *todo projeto* a implementação só se inicia quando os requisitos estiverem completamente levantados e documentados.

2.3 Atividades Auxiliares

Além das atividades essenciais da seção anterior, um processo de software eficaz inclui algumas atividades diferenciadas. Estas atividades existem para viabilizar e apoiar a execução das essenciais, sendo voltadas para otimizar recursos e minimizar os riscos inerentes à nossa profissão. De maneira resumida, as atividades auxiliares fundamentais incluem:

Gerência de Alocação: A tarefa de gerência de alocação consiste em atribuir e administrar recursos (espaço, equipamento) e pessoas entre as tarefas do projeto, com o objetivo de maximizar a

produtividade da equipe.

Gerência de Cronograma: normalmente software é desenvolvido com base em um prazo esperado para sua entrega; em projetos iterativos ou de maior escala é frequente um número de marcos (ou *milestones*) intermediários que permitem verificar o progresso do trabalho. Gerência de cronograma envolve administrar um equilíbrio entre a funcionalidade esperada, o desempenho da equipe, e as datas chaves do projeto.

Gerência de Configuração: este termo cobre um conjunto de políticas e atividades que têm como objetivo controlar os artefatos do projeto (incluindo código-fonte), possibilitando revisar, armazenar, auditar e relatar as alterações produzidas ao longo do tempo.

Uma parte importante da gerência de configuração é o controle de versões do software, normalmente assistido por uma ferramenta de software (como o ClearCase, SourceSafe ou CVS). Outras atividades incluem revisão e inspeção do código, auditoria pós-integração, e relatórios de alterações.

Documentação: ao longo do processo de construção do software, artefatos, código e funcionalidade são normalmente descritos textual ou graficamente na forma de documentação. Existem diversos produtos diferentes que podem ser considerados 'documentação'; sua criação, gerenciamento e atualização são considerados essenciais para um processo saudável.

Em organizações mais complexas, e em projetos de grande porte, podem ocorrer subdivisões importantes destas atividades básicas, assim como a inclusão de outras atividades específicas. No entanto, para uma análise inicial de um processo relativamente desconhecido — como é o caso do processo usado em projetos de software livre — constituem um bom conjunto inicial.

2.4 Garantia de Qualidade de Software

Evidentemente qualquer processo de construção de software busca qualidade. Ao conjunto de esforços feitos para se manter qualidade ao longo deste processo é dado o nome de Garantia de Qualidade de Software[1] (ou QA, de *[Software] Quality Assurance*).

QA não é apenas uma atividade, mas uma série de 'micro-processos' e políticas estabelecidas para supervisionar a qualidade dos artefatos e do próprio processo. Entre estas atividades, pode-se destacar como importantes do ponto de vista de QA:

- revisão e auditoria de artefatos, incluindo código-fonte,
- levantamento e análise de métricas,

[1] Alguns textos tratam a Garantia de Qualidade e as Atividades Auxiliares como equivalentes ou sinônimas; neste trabalho, optamos por caracterizar QA em uma perspectiva ortogonal às atividades do processo de software.

- teste, incluindo teste público (*alfa* e *beta*),
- padronização interna (políticas e regras internas à organização),
- padronização e certificação externa (em relação aos padrões estabelecidos).

Esta lista não é exaustiva, mas oferece uma idéia do tipo de garantia que QA busca oferecer. QA é também importante em projetos de software livre, e será discutida nos capítulos posteriores.

2.5 Modelos de Processo de Software

Para auxiliar a compreensão do processo de software, alguns autores oferecem modelos que descrevem de forma mais concisa o ciclo de vida de um projeto de construção de software. Normalmente são representados graficamente, através de um diagrama. Shari Pfleeger em [2] aponta dois motivos distintos para a utilização de um modelo:

> 'Some [models] are *prescriptions* for the way software development should progress, and others are *descriptions* of the way software development is done in actuality. In theory, the two kinds of models should be similar or the same, but in practice, they are not.'

Um objetivo deste trabalho é estabelecer modelos descritivos que possam auxiliar a compreensão do processo implementado em projetos de software livre. A literatura apresenta uma classificação dos modelos pré-existentes de acordo com a sua seqüencialidade:

Modelos Seqüenciais: tem como característica uma seqüência linear de atividades; dos requisitos seguem projeto arquitetural e detalhado, codificação, e assim por diante. Estes modelos são normalmente prescritivos uma vez que raramente são construídos softwares obedecendo uma seqüencia estritamente linear.

Existem diversas variantes dos modelos seqüenciais que buscam reduzir a sua estrita linearidade. Uma técnica utilizada se chama prototipação, que envolve a construção de um software que ofereça uma primeira experiência ao usuário final, mas que não implemente funcionalidade completa. Após construído o protótipo, existem algumas alternativas: pode ser descartado em favor de implementar sistema completo, ou pode ser evoluído gradativamente até atingir a funcionalidade requisitada.

Modelos Evolucionários ou Iterativos: ao invés de uma seqüência linear, o software é gradualmente produzido e lançado; para cada ciclo de lançamento as atividades básicas são realizadas de acordo com o ritmo de execução. Uma variante possível é um ciclo, normalmente aplicado a software que não contempla uma entrega final; em outra palavras, software que permanece em desenvolvimento contínuo, como é o caso de grande parte dos softwares modernos de consumo geral.

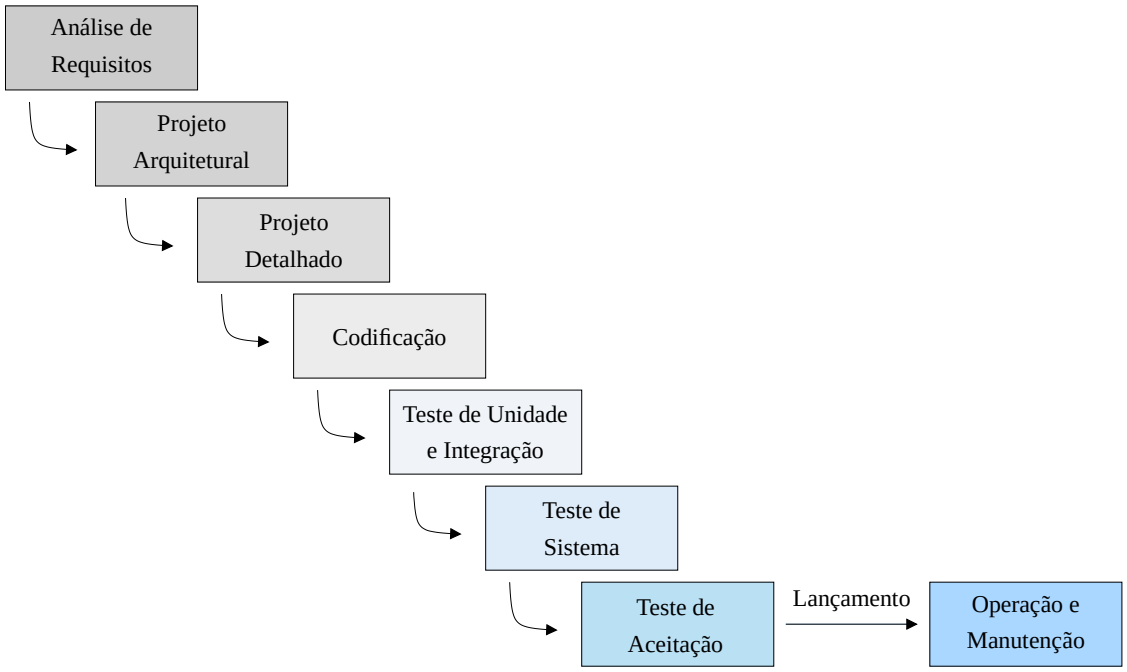


Figura 2.1: Um diagrama do ciclo de vida cascata, descrito por Winston Royce em 1970.

A Figura 2.1 apresenta um diagrama que descreve um dos primeiros modelos a ser propostos, o modelo cascata (‘*waterfall*’) de Winston Royce [14]. Embora seja um modelo prescritivo, e diversas críticas à sua natureza tenham sido publicadas [2], o modelo cascata é pertinente para este trabalho por apresentar claramente uma distinção entre as atividades de desenvolvimento do projeto e a passagem para o estágio de manutenção. Em projetos de software livre, assim como em uma parte dos produtos de software de consumo em massa [15], a fase mais importante da sua existência ocorre justamente *após* seu lançamento. Ao longo do trabalho uma discussão será feita a respeito desta característica fundamental.

Existe um grande número de modelos distintos na literatura, e sua descrição detalhada foge do escopo deste trabalho; para uma análise mais profunda veja [9]. Com base nesta visão geral do processo de software, nos capítulos seguintes se inicia a discussão a respeito do domínio do nosso objeto de pesquisa: os projetos de software livre.

Capítulo 3

Software Livre e *Open Source*

O tema deste trabalho é a classe de softwares denominada software livre ou open source, que são produtos de software comuns, distribuídos com direitos e restrições especiais, e freqüentemente desenvolvidos em projetos voluntários. Para descrever este domínio, e esclarecer a terminologia que será adotada no trabalho, serão apresentadas neste capítulo algumas definições importantes. Como grande parte das definições se apóia em conceitos relacionados à propriedade intelectual, é relevante discutir este assunto em primeiro lugar.

3.1 Propriedade Intelectual

Todo produto de software é derivado de atividade intelectual, e como tal, é protegido por um conjunto de leis que tratam de propriedade intelectual, ou *copyright*. O conceito fundamental do copyright é simples: o autor original do trabalho determina a forma pela qual sua obra será utilizada. Mais especificamente, copyright permite ao autor determinar direitos de uso, cópia, modificação e distribuição (incluindo aluguel, empréstimo e transmissão), entre outros [16].

O copyright tem um prazo de validade[1], após o qual o uso da obra se torna completamente irrestrito (o chamado **domínio público**). As condições e direitos específicos do autor são determinados pela legislação de copyright de cada país, grande parte deles sendo membros da *World Intellectual Property Organization* (WIPO), a organização que consolida as diretrizes internacionais de copyright.

As proteções determinadas pelo copyright não necessitam ser explicitamente descritas: a obra já nasce intrinsicamente protegida, e para fazer qualquer uso desta além do que é considerado ‘uso justo’ (do inglês *fair use*) deve ser feita uma solicitação ao autor [17]. O que constitui exatamente este ‘uso justo’ é assunto de debate e jurisprudência, mas, demonstrando através de um exemplo, a citação é uma forma coberta por este tipo de exceção.

[1] Esta colocação é uma enorme simplificação da realidade. A questão do prazo de validade do copyright é um dos assuntos mais polêmicos na atualidade, sendo tema de diversos processos legais em andamento, mas que foge do escopo deste trabalho. Maiores informações sobre o problema das extensões de copyright podem ser obtidas em `http://eldred.cc/`.

No caso de software, copyright é um assunto bastante importante pela natureza especial que os produtos têm: grande facilidade de replicação e difusão (em termos econômicos, custo marginal de reprodução muito baixo [18]). Além disso, estando disponível o código-fonte, modificar o produto se torna também muito simples. Por estes motivos, softwares normalmente são associados a um documento adicional que explicita os direitos que são oferecidos a seu receptor, a **licença de software**.

3.2 Licenças de Software

É comum o uso de licenças de software para determinar mais especificamente a forma como um software pode ser usado. A licença é um documento (não necessariamente registrado ou validado com nenhum órgão ou organização) veiculado junto ao software, que determina as condições pelas quais pode ser utilizado. Se baseia integralmente nos termos especificados pelas leis de copyright, e normalmente é elaborada por alguém que tenha boa compreensão dos aspectos legais envolvidos [19]. Exemplos comuns de licença são as *click-wrap*, que são exibidas ao usuário quando um software é instalado ou utilizado pela primeira vez, e que requerem uma confirmação da aceitação do usuário para liberar o acesso ao software.

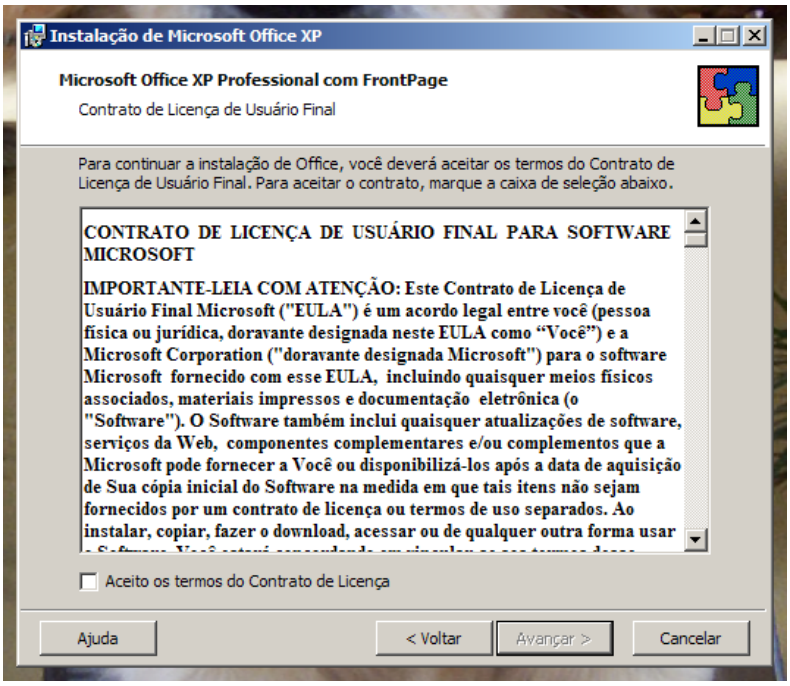


Figura 3.1: Exemplo de uma licença 'click-wrap': o usuário, ao instalar o software, deve ler e decidir se aceita ou não a sua licença antes de usá-lo. É provável que uma parte dos usuários ignore o texto integralmente, apenas confirmando sua aceitação — sem saber exatamente o que aceitam. ('Microsoft®', 'Microsoft Office XP Professional com FrontPage®' e 'Microsoft Office XP®' são marcas registradas da Microsoft.)

3.2.1 Graus de Restrição em Licenças de Software

Embora a maior parte das licenças existentes atualmente detalhe restrições ao uso do software a que se aplicam, existem licenças que têm como finalidade **garantir ao usuário** um conjunto de direitos[2]. É possível categorizar um software de acordo com a forma como sua licença protege ou restringe direitos ao usuário; a lista de categorias de software descrita em 'Categories of Free and Non-Free Software' do Free Software Foundation (FSF) [20] inclui um grande número de tipos diferentes. Estão listadas a seguir as mais relevantes para esta discussão, ordenando-as das mais às menos restritivas:

Software Proprietário: software que proíbe redistribuição e alteração pelo usuário. A maior parte dos softwares comercialmente distribuídos hoje se enquadra nesta categoria.

***Freeware*:** software que permite redistribuição, mas não modificação, e portanto para o qual geralmente não há código-fonte disponível. Os termos Software Livre e Freeware diferem bastante em significado, e seu uso como sinônimos é considerado incorreto.

***Shareware*:** software que permite redistribuição, mas que restringe o uso de acordo com uma condição específica, normalmente associada a um tempo limite de uso, após o qual precisa ser adquirida uma licença comercial. Normalmente não há código-fonte disponível para shareware.

Software Livre: software que oferece ao usuário o direito de **usar, estudar, modificar e redistribuí-lo** (as '4 liberdades básicas' descritas em [21]).

Domínio Público: software sem copyright, software cujo proprietário rescindiu qualquer direito que possuía sobre o software, ou ainda software cujo copyright já expirou. Este tipo de software pode ser utilizado sem qualquer restrição.

Desta lista, pode ser feita uma observação importante: que o termo 'software livre' inclui software no domínio público, mas não shareware e freeware.

3.2.2 Licenças de Software Livre

Licenciamento tem grande importância para software livre; é natural que existam, portanto, diversas licenças de software livre com particularidades individuais. Uma discussão detalhada sobre licenças e suas particularidades vai além do escopo desta seção, podendo ser encontrada na publicação do FSF, 'Various licenses and comments about them' [22]. Aqui são apresentadas apenas as licenças mais importantes, e as restrições associadas a elas:

GNU GPL: (`www.gnu.org/copyleft/gpl.html`)

[2] Esta estratégia de oferecer garantias é conhecida como *copyleft*, descrita em `http://www.gnu.org/copyleft/`

A licença de software livre mais importante atualmente: de acordo com as estatísticas do site `freshmeat.net`, um serviço de registro de softwares livres, em torno de 70% dos softwares livres são licenciados pela GNU GPL. Esta licença é **não-permissiva**: Permite redistribuição apenas se for mantida a garantia de liberdade aos receptores da cópia redistribuída; obriga versões modificadas a serem também livres, acompanhadas de código-fonte.

BSD, X, MIT, Apache: (`www.freebsd.org/copyright/license.html`)

Permitem redistribuição livre do software. A licença BSD original inclui uma cláusula que obriga cópias redistribuídas a manter visível um aviso do copyright; já as licenças X e MIT, não. São **permissivas**: permitem que versões modificadas possam ser redistribuídas de forma não-livre.

MPL, GNU LGPL (`www.mozilla.org/MPL/` e `www.gnu.org/copyleft/lesser.html`)

São não-permissivas, permitindo redistribuição do código apenas quando mantida a garantia de liberdade inalterada. No entanto, permitem que este código seja usado em um 'produto maior' sem que este tenha que ser licenciado livremente. Se modificações forem feitas ao código licenciado pelo MPL ou LGPL, estas devem ser fornecidas acompanhadas de código-fonte. Esta restrição não cobre o código-fonte do 'produto maior'.

3.3 Software Livre e Open Source

Parte da dificuldade em discutir software livre deriva do termo assumir múltiplos significados, seu sentido variando de acordo com o contexto em que ocorre [20, 23, 24]. Existe uma forma bastante objetiva de definir software livre:

> Software Livre é qualquer software cuja licença garanta ao seu usuário liberdades relacionadas ao uso, alteração e redistribuição. Seu aspecto fundamental é o fato do **código-fonte** estar livremente disponível para ser lido, estudado ou modificado por qualquer pessoa interessada.

Este sentido é o adotado pela FSF e pela *Open Source Initiative* (`www.opensource.org`), entidades importantes entre os grupos que lidam com Software Livre[3]. No entanto, esta definição analisa o termo apenas do ponto de vista do produto final. A realidade é que raramente software livre aparece dissociado de um conjunto de outras características importantes:

[3]Um dos aspectos interessantes desta definição é que não restringe a comercialização do software. É perfeitamente possível cobrar um valor pelo software livre. Na prática, já que é possível copiar o código de qualquer pessoa que já o tenha, o preço que o usuário paga por um software livre tende a ser baixo o suficiente para motivar as pessoas a comprarem e não o copiarem.

Distribuição via Internet: o software é disponibilizado através de algum serviço Internet, incluindo sites Web e FTP, repositórios CVS e correio eletrônico[4].

Desenvolvimento descentralizado via Internet: se mais de um desenvolvedor trabalha no software, este desenvolvimento é feito de maneira colaborativa, usando a Internet como meio de comunicação.

Usuários participantes: é comum se formar um grupo de usuários finais que se comunicam com alguma regularidade com os desenvolvedores e entre si, comunicando problemas e trocando experiências do uso do software.

Interesse pessoal do autor: o autor é um usuário do software, e portanto tem motivação pessoal na sua criação e manutenção. Além disso, o autor normalmente busca apoiar seus usuários e fazer o grupo de pessoas envolvidas com o software crescer.

Ferramentas de comunicação e desenvolvimento distribuídas: são usados um número de ferramentas e serviços para suportar a comunicação entre estes indivíduos.

Forte Individualismo: tanto desenvolvedores quando usuários se tratam pessoalmente, usando nomes próprios, e não em termos das organizações que possam representar; não há um senso de hierarquia neste sentido (embora haja outras hierarquias relevantes). No arquivo contendo o software lançado (também chamado de **pacote**) são incluídos os nomes dos autores e contribuidores.

Essas características oferecem uma visão mais completa do que realmente representa software livre e o grupo de pessoas que o sustenta. Neste trabalho, o termo **Projeto de Software Livre** é usado para descrever um conjunto vinculado a um software livre, composto por pessoas, código-fonte, processo e ferramentas de desenvolvimento. A Figura 3.2 esquematiza os elementos fundamentais de um projeto de software livre, e o Capítulo 4 descreve estas características em maior detalhe.

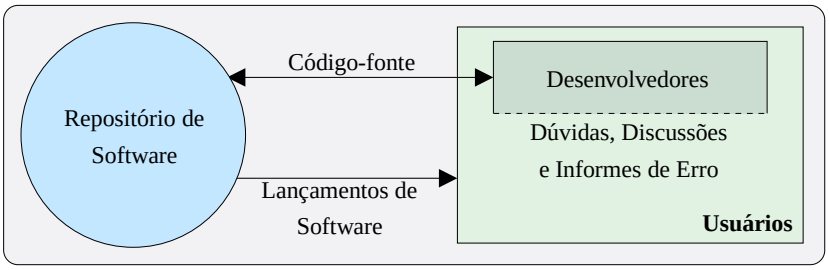


Figura 3.2: Diagrama de um projeto de software livre em alto nível. As entidades principais representadas constituem a base de usuários do software, uma parte da qual são desenvolvedores, e o repositório de software onde é armazenado e oferecido o software do projeto.

[4]Para uma descrição dos serviços Internet citados acima, incluindo Web (ou WWW), FTP, correio eletrônico veja o 'Guia Internet para Iniciantes' em `http://www.icmc.usp.br/manuals/BigDummy/`

3.3.1 Open Source

A expressão *Open Source* é freqüentemente usada para descrever software da mesma categoria que Software Livre, em contextos bastante parecidos. Por que existe esta multiplicidade de nomes?

O termo 'Open Source' foi criado em 1998 com o objetivo de desenfatizar o teor filosófico associado à **liberdade** que o termo 'Software Livre' possui [23]. Existia uma motivação para quebrar barreiras de preconceito vinculadas à defesa deste princípio, o que melhoraria a chance de convencer empresas a adotar este tipo de software e sua forma de desenvolvimento. Na prática, uma licença considerada Open Source (e não de software livre) tende a ser mais permissiva em relação à geração de produtos derivados não-livres.

De certa forma, 'open source' enfatiza os aspectos do processo e da organização social, enquanto 'software livre' enfatiza os aspectos de livre redistribuição e troca de conhecimento. Existe uma certa predisposição de certos grupos, como o FSF, em considerar o termo open source ligeiramente menos correto ou digno que free software. Nesta dissertação, buscando evitar discussões que não sejam estritamente relacionadas ao processo e produtividade dos projetos de software livre, optei por usar exclusivamente o termo software livre.

3.4 Histórico

Software Livre é um conceito antigo, embora não tenha sido usado esse termo específico até a década de 80. Grande parte do software existente durante as décadas de 60 e 70 foi desenvolvido de forma colaborativa e aberta em diversas instituições e empresas [25]; o Unix original é um exemplo desta tendência [26]. Redistribuição e aprimoramento do software eram vistos como positivos, e o software geralmente era fornecido com o seu código-fonte, mesmo que as licenças não resguardassem explicitamente a liberdade [27, 28].

A década de 80 trouxe uma alteração importante: a mudança gradual para licenças restritivas, que não permitiam redistribuição ou modificação. O software fornecido, que até então era visto como uma combinação de código-fonte com código executável, passou a significar apenas o executável [28].

3.4.1 GNU e a Free Software Foundation

Em 1985, Stallman criou o **Free Software Foundation**, cuja finalidade era promover a criação de um sistema operacional completamente livre. O sistema operacional tinha o objetivo de ser compatível com Unix, e foi chamado de GNU (uma sigla recursiva que significa *GNU's not Unix*). Para garantir sua liberdade, Stallman também criou uma licença para este software, a GNU GPL, descrita anteriormente como a licença de software livre mais utilizada. O desafio de construir o sistema operacional envolvia não apenas a criação de um núcleo de sistema operacional, mas também a criação de uma coleção de bibliotecas e aplicativos que permitissem ao sistema compatibilidade com o Unix original. Durante

esta década o FSF apoiou o desenvolvimento do conjunto de compiladores GCC (originalmente *GNU C Compiler*, hoje *GNU Compiler Collection*), o editor de textos Emacs e as bibliotecas padrão libc e glibc, entre outros[5].

O processo de desenvolvimento deste período não é documentado na literatura, mas é certo dizer que já existia uma comunidade de desenvolvedores interessados nos produtos da FSF e que contribuía com código-fonte e pacotes para completar as lacunas no sistema operacional GNU. O meio de comunicação óbvio era a Internet, que começava a se tornar mais facilmente acessível, embora versões dos pacotes GNU fossem também distribuídas em fitas magnéticas. Correio eletrônico e FTP eram os veículos pelos quais os desenvolvedores comunicavam seus lançamentos.

3.4.2 O núcleo Linux

Em 1991, um estudante da Universidade de Helsinki, Linus Torvalds, iniciou o desenvolvimento de um núcleo de sistema operacional, o Linux. O núcleo Linux é considerado o mais importante exemplo moderno de um software livre. Uma diferença fundamental no desenvolvimento do Linux é a 'abertura' do processo de desenvolvimento: Torvalds convidou outros desenvolvedores a participarem ativamente da codificação e manutenção do núcleo, e, de forma surpreendentemente rápida, este evoluiu para se tornar um software com funcionalidade similar aos núcleos Unix comerciais [29].

Em grande parte este processo 'aberto' foi inaugurado com o Linux, mas deve-se notar que seu surgimento é fortemente associado à ampla disponibilidade de acesso à Internet, o que permitiu que de fato se formassem os laços de distribuição de software e comunicação interpessoal usada por Linus e outros interessados. Outro fator que se considera essencial para o sucesso do Linux, do ponto de vista do processo de desenvolvimento, é a escolha pela licença GPL, que garantiu aos colaboradores a preservação do trabalho contribuído. O sucesso aparente do processo de desenvolvimento utilizado no Linux o levou a ser imitado e replicado por outros autores interessados em produzir software livre.

3.4.3 Outros pacotes de software livre

Não eram apenas a comunidade Linux e o projeto GNU que ofereciam pacotes de software. Desenvolvedores interessados em implementar um software para um fim específico, que não existisse como software livre, possuíam duas opções imediatas: usar um sistema operacional proprietário e comprar um software que atendesse às suas necessidades, ou implementar uma versão do software para Linux. Além disso, existiam softwares livremente disponíveis que executavam em versões proprietárias do Unix, e que gradualmente foram sendo convertidos para também executar sobre a plataforma combinada Linux e GNU.

[5]GCC, Emacs e glibc são pacotes oferecidos pelo projeto GNU. Visite suas páginas respectivas a partir do endereço `http://www.gnu.org/`

A este conjunto de autores e usuários interessados, freqüentemente é associado o nome de 'comunidade de software livre' (ou comunidade open source). É importante deixar claro que este nome é uma forma conveniente de se referir ao total de pessoas envolvidas de uma forma ou outra com software livre, mas que não tem semântica própria definida. Em outras palavras, os desenvolvedores originais não consideravam necessariamente que estavam contribuindo para uma comunidade, ou mesmo que esta comunidade existisse. Com o passar do tempo, no entanto, ficou claro que a maior parte destes autores tinha em comum três características importantes:

- Por definição, desenvolviam software para ser redistribuído livremente, através de uma licença de software livre.
- Dividiam um conjunto de valores básicos derivados dos valores originais do Unix [30]: resumidamente, um zelo por excelência técnica, modularidade, reuso, portabilidade e observância a padrões publicados.
- Trabalhavam em isolamento geográfico: não existia mais uma grande sala com computadores onde todos sentavam juntos para trabalhar. Cada autor tinha seu ambiente individual de trabalho, e colaborava-se usando ferramentas de comunicação pela Internet. De fato, a distribuição dos autores era mundial, concentrando-se em países onde existia fácil acesso à Internet, larga disponibilidade de computadores, e conhecimento do idioma inglês.

O fato de muitos softwares livres se apoiarem uns nos outros — afinal, o compilador necessitava de um núcleo para executar, e um núcleo sem aplicativos é inútil — também foi um dos agentes de coesão desta comunidade: o desenvolvedor que dominava um determinado software muitas vezes oferecia auxílio aos desenvolvedores dos softwares 'dependentes'. Com o passar do tempo, foram sendo criados sites Web e listas de correio eletrônico dedicadas a discussão e veiculação de notícias de interesse geral para os envolvidos de alguma forma com software livre. Exemplos destes concentradores são o site de notícias `www.slashdot.org` e um arquivo de lançamentos de software, o `freshmeat.net`. Estes veículos figuram como fatores importantes para estabelecer uma consciência real de comunidade entre os desenvolvedores geograficamente dispersos.

3.4.4 As distribuições Linux

Com o surgimento e aperfeiçoamento do Linux, era finalmente possível construir um sistema operacional usando como base as ferramentas construídas pelo projeto GNU. Esta combinação (núcleo mais software básico) agregado a um conjunto de aplicações que tinha sido criado para outros sistemas operacionais (o próprio Minix, e os variantes Unix já existentes) resultava em um sistema utilizável por desenvolvedores e outros interessados. Nesta época, esse conjunto começou a ser distribuído de forma ad hoc; gradualmente começaram a surgir projetos para produzir um 'meta-pacote' que seria constituído de um grande número destes pacotes individuais. Estes projetos ganharam o nome de **'distribuições'**, e

são os precursores das distribuições modernas como Redhat, Conectiva e Debian. O nome 'distribuição Linux' passou a ser usado para descrever esta combinação; o pronunciamento oficial de Richard Stallman e da FSF é de que este nome não dá o crédito necessário ao projeto GNU e recomenda o uso do nome GNU/Linux para representar esta combinação [6].

A distribuição nada mais é do que um agregado de pacotes individuais, cada um com seu próprio autor e comunidade de desenvolvimento, com um sistema de instalação simplificado e uma marca associada; esta própria dissertação foi elaborada em um sistema com a distribuição Debian. No entanto, seu surgimento representa uma mudança intencional da visão original de um sistema para amadores em direção a um sistema de uso geral, facilmente instalável e já incluindo uma série de aplicações pré-configuradas. Enquanto pacotes individuais não podiam ser comercializados eficientemente, era possível vender um CD-ROM contendo uma distribuição completa e de instalação mais fácil. A partir deste momento, passou a haver atenção substancialmente maior sobre o trabalho realizado por este conjunto de desenvolvedores. Quem eram estas pessoas que doavam seu trabalho para criar softwares que podiam ser redistribuídos livremente? É interessante observar os paralelos entre a era inicial da computação, descrita acima, e esta nova forma de organização.

Distribuições GNU/Linux

As distribuições incluem de centenas a milhares de pacotes distintos (um dos quais será o núcleo Linux). Cada pacote individual é resultado do trabalho de um ou mais autores, representando um micro-processo de desenvolvimento que é ocultado pela presença da distribuição. Existe uma certa natureza simbiótica entre o pacote e a distribuição: a distribuição depende dos autores individuais para criar o software que oferece; por sua vez, os pacotes podem atingir um público muito maior se oferecidos por uma distribuição.

Uma contrapartida desta simbiose é que as distribuições passaram a ter interesse especial nos pacotes mais críticos; as empresas que mantêm e comercializam as distribuições freqüentemente são responsáveis por financiar desenvolvedores para garantir seu desenvolvimento. Além disso, realizam um trabalho importante de garantia de qualidade, já que são responsáveis imediatas pelo software que distribuem (independente de produzí-lo ou não!)

O resultado mais surpreendente desta evolução foi o fato da qualidade aparente das distribuições ser alta. Embora não haja um volume de literatura que compare a qualidade de sistemas proprietários a sistemas desenvolvidos como software livre (em parte porque é tão difícil medir qualidade de software),

[6] Ver `http://www.gnu.org/philosophy/free-sw.html` para uma justificativa.

o que existe realmente confirma esta impressão. Barton P. Miller em seu trabalho 'Fuzz Revisited: A Re-examination of the Reliability of Unix Utilities and Services' [31] compara ferramentas provindas de versões de Unix comercialmente disponíveis com as presentes na distribuição Linux Slackware. Sua conclusão: a confiabilidade das ferramentas no Slackware era notavelmente mais alta do que a dos sistemas comerciais.

É possível questionar a representatividade desta avaliação, tanto do ponto de vista de cobertura, quanto do ponto de vista do que representa confiabilidade dentro de uma visão geral de qualidade. No entanto, a boa reputação das distribuições Linux do ponto de vista da estabilidade ajuda a entender porque se criou uma impressão positiva em torno da qualidade percebida do sistema. Aliada à alta produtividade aparente nos projetos individuais, e à imagem do contribuidor voluntário 'trabalhando pela causa', ficou claro para muitos pesquisadores que estavam presenciando algo extraordinário [32].

3.4.5 Projetos de Software Livre

Enquanto as distribuições implementavam melhorias no sistema de instalação, desenvolviam seus canais de vendas e aprimoravam a escolha e qualidade dos seus pacotes, os pacotes individuais continuavam sendo mantidos por seus respectivos desenvolvedores. A partir da segunda metade da década de 90, é possível perceber o aparecimento de um número de grupos e organizações informais dedicadas aos **pacotes** individuais.

Originalmente produzidos por um único autor que interagia com sua comunidade de usuários e desenvolvedores esporádicos, os pacotes mais interessantes agora começavam a atrair grupos maiores de desenvolvedores ativos. O exemplo claro disto é, novamente, o Linux, que desde o início tinha este caráter aberto e convidativo. Gradualmente estes grupos de desenvolvedores consolidaram-se em organizações informais para o aprimoramento de pacotes de software livre.

Essa organização, o **Projeto de Software Livre**, é uma comunidade virtual, com um nome próprio, centrada em torno de um pacote de código-fonte, com participação espontânea e variável de usuários e desenvolvedores. O contexto contemporâneo é o de milhares de projetos de software livre independentes, cada um responsável por um software específico, implementando um processo de software individualmente determinado, com ferramentas desenvolvidas para apoiar a sua natureza fundamentalmente distribuída. É neste contexto que esta dissertação busca uma melhor compreensão tanto do processo quanto das conseqüências deste em relação ao software produzido.

Capítulo 4

Projetos de Software Livre

No capítulo anterior, software livre foi abordado do ponto de vista de um **produto**: código-fonte licenciado de forma a resguardar liberdades fundamentais ao seu usuário. Este enfoque permite que sejam compreendidos os conceitos fundamentais relacionados ao software em si; no entanto, para estudar o processo pelo qual este tipo de software é desenvolvido e mantido, é essencial estudar melhor os grupos que o produzem. Note que boa parte das definições neste capítulo foram estabelecidas durante este presente trabalho, e não provêm da literatura.

4.1 Definição

Um Projeto de Software Livre é uma organização virtual dedicada à manutenção de um produto de software livre. Embora o termo não possua uma definição claramente estabelecida na literatura, no contexto deste trabalho oferecemos a seguinte:

> Um Projeto de Software Livre é uma organização composta por um **conjunto de pessoas** que usa e desenvolve **um único software livre**, contribuindo para uma base comum de código-fonte e conhecimento. Este grupo terá à sua disposição **ferramentas de comunicação e trabalho colaborativo**, e um conjunto de experiências e opiniões técnicas que usam para discutir o andamento do projeto.

Por definição, este 'conjunto' de pessoas é estabelecido pelo simples fato de discutirem por correio eletrônico (ou outra ferramenta de comunicação) assuntos relacionados ao software do projeto; desenvolvedores importantes podem ter alguma permissão especial para integração de código-fonte, mas de forma geral pode-se dizer que é uma combinação volátil, mantida apenas pelo interesse e disponibilidade de seus membros. Não é possível estabelecer em um determinado momento o número exato de pessoas que fazem parte de um projeto de software livre; desenvolvedores podem perder interesse ou não dispor de tempo para promover seu desenvolvimento; usuários podem optar por usar outro software, ou mesmo reduzir sua interação com o grupo pelo fato do software atender bem sua necessidade.

Sob esse ponto de vista, apenas o código tem um aspecto persistente. Ao longo do tempo, a comunidade de um projeto pode variar arbitrariamente; no entanto a sua base de código, com raras exceções, é um produto da evolução da primeira versão integrada pelo autor original.

Todo projeto tem um nome próprio pelo qual é identificado; normalmente este nome reflete o nome do produto principal que desenvolve. Exemplos são o projeto Linux, que tem como produto principal o núcleo Linux, e o projeto Apache, que produz um servidor HTTP.

4.2 Organizações Alternativas

Embora pela nossa definição um projeto de software livre seja voltado para um software específico, o nome 'Projeto de Software Livre' também é freqüentemente usado para descrever três formas alternativas de organização, aqui definidas: o **meta-projeto**, a **distribuição** e o **grupo de usuários**.

Meta-projeto: é um agregado de **projetos** de software livre. Existem diversos exemplos importantes, os mais famosos sendo os projetos Mozilla, KDE e GNOME. Possui uma estrutura de liderança que é freqüentemente composta de desenvolvedores chave dos projetos individuais mais importantes.

O meta-projeto oferece uma 'bandeira' sob a qual diversos produtos são afiliados, desenvolvidos e lançados. Por exemplo, o meta-projeto Mozilla sustenta um conjunto de aplicações distintas, incluindo os navegadores Web Mozilla e Phoenix, uma infra-estrutura de desenvolvimento baseada em interpretadores para XUL, XBL e Javascript, e ferramentas de apoio ao desenvolvimento como Bugzilla, Tinderbox e Bonsai.

A tarefa do meta-projeto é organizar estes projetos individuais em alguns aspectos diferentes, que podem incluir os seguintes:

- Provendo uma estrutura política com objetivos mais abrangentes que um projeto de software livre individual; por exemplo, o meta-projeto GNOME tem como objetivo promover um sistema *desktop* com base em um conjunto de aplicações e um framework para desenvolvimento.
- Provendo padronização de interfaces de programação (API) e interfaces com o usuário (HCI); novamente, o projeto GNOME oferece documentos que padronizam sua arquitetura e recomendações para a criação de interfaces com melhor usabilidade.

Distribuição: é um agregado de **pacotes** de software livre, com o objetivo de garantir qualidade do produto integrado, simplificar a instalação destes produtos, e explorar um potencial mercado consumidor.

A diferença básica entre uma distribuição e um meta-projeto é seu objetivo; a distribuição tem como meta oferecer ao usuário final uma forma conveniente de acesso ao software livre; por sua

vez o meta-projeto tem o objetivo de fomentar e auxiliar os projetos que engloba. O envolvimento de uma distribuição com um projeto é mais distante: seu papel se assemelha ao de um usuário ativo (ou desenvolvedor esporádico) do projeto. Além disso, não oferecem afiliação: um único projeto pode ser incluído em múltiplas distribuições, e nunca se diria que 'é membro da distribuição X'.

Grupo de usuários: é um agregado de **usuários** de software livre, em geral dedicados a um tema particular. Normalmente possui a característica de centralização geográfica, unindo usuários que vivem em uma mesma região. Por exemplo, existem ao redor do mundo centenas de grupos de usuários Linux (chamados de LUGs, de *Linux User Groups*). O grupo de usuários pode ter uma série de objetivos, incluindo:

- Auxiliar novos usuários a usar um projeto, meta-projeto ou distribuição de software livre.
- Promover o uso de software livre na comunidade local.
- Dar suporte de primeiro nível a usuários do software. Membros dos grupos de usuários são freqüentemente desenvolvedores esporádicos e trabalham com os membros do projeto de software livre propriamente dito para resolver problemas que identificam no seu grupo de usuários local.

4.3 Aspectos de um Projeto de Software Livre

Como definido anteriormente, um projeto de software livre representa um conjunto formado por software, comunidade e ferramentas. As seções seguintes discutem aspectos individuais desta tríade.

4.3.1 Software

Por definição, o código-fonte que o projeto produz é distribuído através de uma licença de software livre. Esta base de código é freqüentemente mantida em um repositório público de onde qualquer pessoa pode copiá-la. Alterações nesta base são normalmente feitas por um grupo pequeno de pessoas — em muitos casos, por uma pessoa só, que é chamada de *maintainer* ou **mantenedor**.

A base de código inicial é normalmente escrita por uma pessoa isoladamente, e lançada através de uma mensagem enviada para uma lista de discussão ou em um arquivo online de projetos (como os sites `freshmeat.net` ou `sourceforge.net`). O código-fonte (possivelmente acompanhado de versão pré-compiladas para arquiteturas e distribuições específicas) é normalmente oferecido de um repositório Internet através de Web ou FTP.

Em alguns casos, a base de código original deriva parcial ou totalmente do código-fonte de um outro projeto; a este tipo de projeto é dado o nome informal de *code fork*, já que representa um veio evolutivo independente do projeto inicial, com um ancestral comum. Os motivos para se fazer um code

fork podem variar: insatisfação dos desenvolvedores em relação aos mantenedores da versão original; intenção de implementar alterações potencialmente arriscadas; desejo de evoluir o código-fonte em uma direção distinta do projeto original.

Normalmente o desenvolvimento é feito de forma isolada; cada desenvolvedor cria alterações ao código-fonte localmente. Uma vez decidido que a alteração é correta e desejável (possivelmente tendo passada por revisão de projeto e código de outros desenvolvedores), o processo de integração é iniciado. Se o desenvolvedor tem permissão para escrever código diretamente no repositório, faz a integração pessoalmente; se não, envia sua alteração para outro que tem autoridade o suficiente para integrá-la. Durante este processo, é comum que diversas pessoas comentem sobre a alteração, e que esta seja revisada e mesmo reescrita caso necessário; isto ocorre tanto na fase pré-integração como pós-integração. Este ciclo se repete para cada alteração a ser introduzida na base de código-fonte.

4.3.2 Comunidade

As pessoas envolvidas em um projeto podem ter sua participação classificada pelo tipo de papel que exercem:

Usuários não-participantes, sem interesse em discutir ou ajudar no desenvolvimento do software. A maioria das pessoas envolvidas com software livre fará parte desta categoria; fazem teste funcional do software mas em geral não informam erros quando os encontram.

Usuários participantes, que ativamente contribuem para o projeto informando problemas e discutindo funcionalidade desejada. Estes usuários são responsáveis por boa parte do teste funcional, provendo *feedback* para os autores em relação a regressões e inconsistências.

Desenvolvedores esporádicos, usuários com conhecimento de programação e disposição para alterar código-fonte e produzir alterações. Normalmente suas alterações terão caráter localizado, reparando um pequeno defeito ou implementando uma pequena extensão de funcionalidade. Alguns membros da comunidade de software livre tem o papel de desenvolvedor esporádico em um conjunto grande de projetos, contribuindo reparos quando encontram um problema simples.

Desenvolvedores ativos, que têm responsabilidade por módulos do código-fonte ou por implementar funcionalidade mais extensa ou mais complexa. Entre estes normalmente figura o autor original como mantenedor central.

Esta taxonomia inicial será reavaliada nos capítulos posteriores à medida que outros trabalhos forem discutidos. Vale notar que esta divisão não é rígida; pessoas podem migrar de um papel para outro conforme escolha e oportunidade pessoal.

Para coordenar o trabalho que realizam, os membros da comunidade de um projeto utilizam a Internet através de ferramentas simples amplamente disponíveis: correio eletrônico, páginas Web, listas

de discussão, em conjunto com ferramentas de desenvolvimento de software, controle de versão, e acompanhamento de defeitos.

4.3.3 Comunicação escrita

Os membros da comunidade se comunicam normalmente através de listas de correio eletrônico. Alguns projetos utilizam adicionalmente repositórios públicos de defeitos (descritos adiante na Seção 4.3.5) e formas de comunicação em tempo real. Em geral, usuários participantes tendem a usar estes veículos para comunicar experiências, problemas e solicitações. Também fazem suporte de primeira ordem, ajudando outros usuários com problemas comuns.

Desenvolvedores também participam ativamente das discussões, fazendo boa parte do suporte ao usuário, e argumentando com outros desenvolvedores questões de projeto e implementação. Em projetos de dimensão um pouco maior existem listas e canais separados para discussão relacionada ao desenvolvimento em si. Desta forma, ficam mais isoladas as discussões entre usuários, que geralmente não estão interessados nas minúcias técnicas associadas ao desenvolvimento.

O fato das ferramentas apoiarem a troca de experiência na comunidade tem como efeito colateral a possibilidade de arquivar-se esta comunicação para análise futura. Normalmente estes arquivos online são indexados por serviços de busca na Web e servem como uma fonte importante de documentação para os usuários, já que freqüentemente apresentam problemas recorrentes, já discutidos no passado por outros.

Normalmente o conteúdo transmitido por estes meios será texto simples, escrito na grande maioria dos casos em inglês. A comunidade de software livre tende a não utilizar e-mail HTML, por exemplo. Muitas pessoas têm habilidade de formatação bastante avançada, usando recursos como hierarquização, listas e até diagramas criados usando símbolos texto. A Figura 4.1, por exemplo, demonstra o uso de uma mensagem de correio eletrônico para descrever uma interface gráfica. O uso do denominador comum texto garante acesso ao conteúdo, não importando qual ferramenta de visualização é utilizada.

É importante notar que os desenvolvedores trocam código em formato texto, mas não enviam os arquivos inteiros. Ao invés disso, criam *diffs* (de *differences*, ou diferenças). Um diff é um arquivo texto contendo um conjunto de fragmentos de código, representando adições e subtrações ao código anterior. Estes diffs são gerados por uma ferramenta apropriadamente chamada `diff`. A maior parte dos desenvolvedores possui boa habilidade na leitura de diffs, e existem ferramentas que permitem que sejam facilmente visualizados. A Figura 4.4 exemplifica uma destas ferramentas.

```
Date: Wed, 29 May 2002 04:15:16 +1200
From: Matthew Thomas <mpt@myrealbox.com>
To: Christian Reis <kiko@async.com.br>
Subject: Re: [priority++] Re: UI review for Payment dialogs

Christian Reis wrote:
> There is something I forgot to ask if we can make fit: the
> default interval is 30 days, but sometimes somebody will ask
> for 15 or 45 days between installments. In this case, it
> usually applies to all installments; how can we integrate it?

Ok, maybe you really do want more than one dialog. :-)

How about this?
```

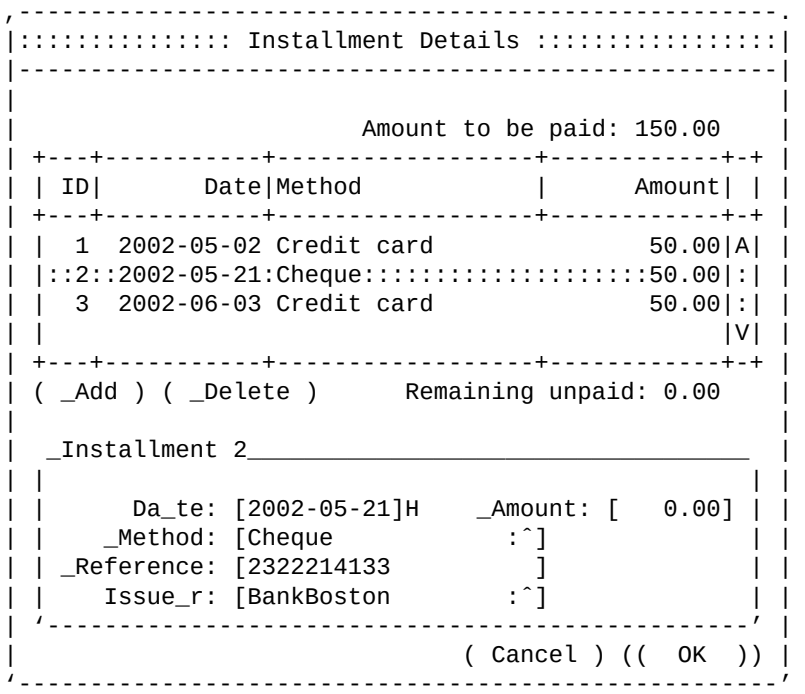


Figura 4.1: Exemplo de uma comunicação entre o autor e Matthew Thomas discutindo uma interface gráfica. Note que Matthew é neo-zelandês e não usa a mesma ferramenta de criação de interfaces que eu uso; não poderia facilmente criar uma simulação da interface, e nem desenhar em papel uma figura para comunicar sua idéia. Neste sentido, o uso de texto simples é um recurso bastante valioso.

4.3.4 Gerência do Código-fonte

Para armazenar a base de código do projeto, normalmente é usada uma ferramenta de controle de versões. É frequente o uso da ferramenta CVS [33]: apesar de ter limitações reconhecidas [34], a ferramenta aparentemente tem uso maciço por ser bastante simples e ter ampla disponibilidade. Compreender seu esquema de funcionamento ajuda a entender a forma de trabalho distribuído da comunidade; a Figura 4.2 representa este esquema graficamente.

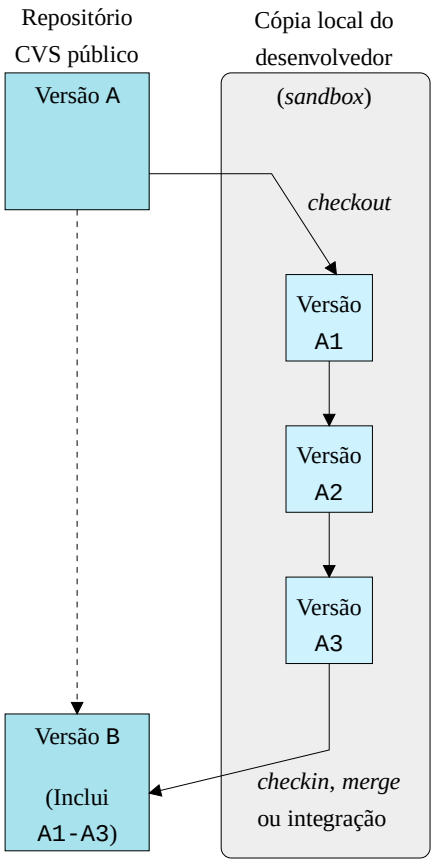


Figura 4.2: Diagrama descrevendo o fluxo de trabalho normal utilizado em ferramentas de controle de versão que utilizam um *sandbox* local, como o CVS. Note que as versões `A1-A3` existem apenas no espaço local do desenvolvedor, e nenhum outro desenvolvedor tem acesso a elas.

O modelo básico de funcionamento do CVS permite trabalho *offline*, tornado possível que o desenvolvedor faça alterações em uma cópia local do código-fonte, sem necessitar estar conectado ao servidor que o hospeda. Além disso, não utiliza um mecanismo de exclusão mútua, o que permite que múltiplos desenvolvedores modifiquem o código nas suas cópias locais independentemente. Vale notar que o CVS serve para armazenar e controlar versão de qualquer documento baseado em texto simples, e não apenas código-fonte. Seus aspectos principais são discutidos a seguir.

Repositório: Um repositório CVS consiste de um conjunto de arquivos mantido em um local central; normalmente é armazenado em um servidor conectado à Internet que permite acesso anônimo, evitando que usuários que queiram acessar o código-fonte tenham que registrar-se. O repositório CVS opera (principalmente) segundo um modelo de incrementos negativos: armazena as cópias mais atuais dos arquivos do projeto, e as diferenças entre esta e as versões anteriores.

Cópia Local: O conceito fundamental do CVS é que não é necessário travar ou trabalhar diretamente o repositório (na realidade, o repositório nunca é manipulado diretamente; todo o acesso é feito através de uma ferramenta de *front-end* chamada `cvs`). Ao invés disso, qualquer desenvolvedor interessado solicita ao servidor (usando esta ferramenta) uma cópia do código de uma certa versão - na maior parte das vezes a versão mais atual, chamada CVS HEAD, é solicitada. Uma vez criada esta cópia local, também chamada de *workplace* ou *sandbox*, o desenvolvedor tem acesso aos arquivos que deseja construir. Toda a estrutura de compilação e teste deve ser previamente preparada para poder ser utilizada pelo desenvolvedor usando as ferramentas que possui no seu ambiente local.

É alterando esta cópia local que o desenvolvedor implementa suas modificações. Usando a ferramenta `cvs`, ele pode solicitar atualização de sua base de código local, 'puxando' alterações que foram integradas ao repositório principal depois da criação da sua cópia local. A ferramenta `cvs`

também gera diffs automaticamente a partir das diferenças entre a cópia local e o repositório. Desta forma é mais fácil auditar exatamente o que está sendo alterado.

Integração: Ao alterar o código, o desenvolvedor está livre de qualquer interação com o repositório. Quando julga que a versão alterada está pronta para ser integrada à base principal, usa a ferramenta `cvs` para submeter o código ao repositório central. A esta ação de integração é dado o nome de *commit* ou *checkin*.

> Se múltiplos desenvolvedores alteram o mesmo trecho de código, conceitualmente ocorre um **conflito**. O processo de resolução de conflitos é razoavelmente complexo: o **primeiro** desenvolvedor a integrar o código ignora, a princípio, o fato de outros desenvolvedores estarem trabalhando no mesmo trecho. Cada desenvolvedor subseqüente que iniciar uma ação de checkin receberá do repositório a mensagem de que precisa atualizar a sua versão local (já que uma integração no mesmo arquivo foi feita pelo primeiro desenvolvedor).
>
> Neste momento, ao atualizar a cópia local (um *update* nos termos do CVS) a ferramenta irá informar que houveram alterações provindas do repositório no mesmo trecho de código alterado localmente. Estes trechos do código são marcados com indicadores que diferenciam a versão do repositório da versão local.
>
> O desenvolvedor precisa analisar com cuidado as alterações e decidir de que forma o código deve ficar; ele pode resolver o conflito removendo uma das versões; mais freqüentemente a opção correta é integrar mudanças de ambas as versões num trecho único. Em parte, o mecanismo de identificação de conflitos do CVS também incentiva o uso de formato texto simples, já que fica bastante difícil reparar conflitos em formatos complexos.

Versões: Cada arquivo modificado no repositório recebe uma versão numérica. É possível criar *aliases* (apelidos) que abstraem as versões de um conjunto de arquivos em um determinado instante do tempo. Com isso, é possível reverter alterações de arquivos individuais para versões passadas, e também regenerar uma versão completa do repositório em um determinado momento.

Do ponto de vista do desenvolvedor, o CVS permite que se compare facilmente as alterações feitas na sua cópia local com qualquer versão integrada no repositório, e torna possível reverter alterações integradas no repositório central para versões anteriores. Este último recurso é normalmente utilizado para remover alterações problemáticas; por exemplo, código que cause um defeito ou regressão funcional no software.

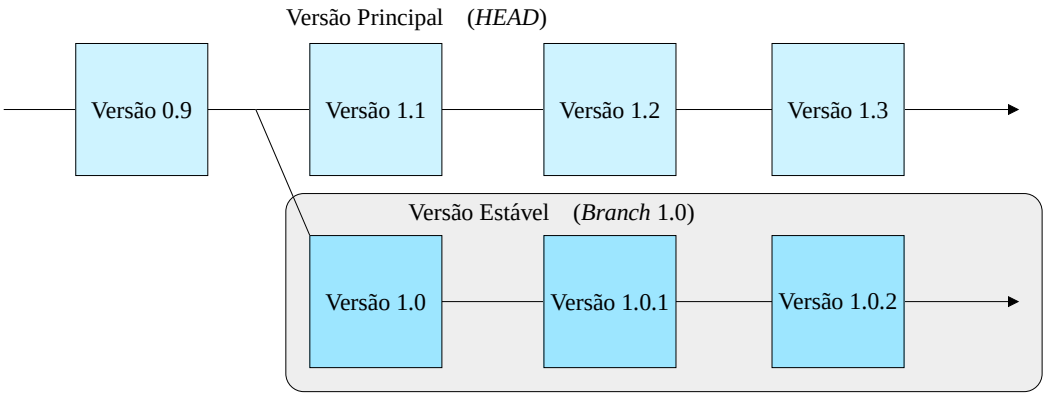


Figura 4.3: Diagrama descrevendo múltiplas versões e múltiplos *branches*. Cada branch pode possuir um conjunto arbitrário de versões. Aqui, o branch estável foi criado logo após o lançamento da versão 0.9, e é mantido separadamente da versão principal, recebendo apenas reparos para bugs — 3 versões estáveis já foram lançadas (1.0-1.0.2). Na versão principal desenvolvimento e adição de funcionalidade continuam normalmente.

***Branches*:** Além das versões alternativas da base de código principal, o CVS permite que se crie uma linha de versões paralela à principal. Resumidamente, é estabelecido um *branch* (ou bifurcação) a partir do código principal (este chamado *trunk* ou tronco). Alterações integradas sobre este branch são isoladas, não refletindo no tronco. É possível criar um número arbitrário de branches; cada um deles terá um nome simbólico, que o desenvolvedor especifica ao usar a ferramenta `cvs`. Eventualmente alterações de um branch podem ser 'aplicadas' no tronco principal; esta operação tem o nome de *merge*.

É comum a prática entre os projetos de software livre de manter branches estáveis (Figura 4.3), conforme descrito a seguir:

Branch Estável: neste branch tendem a ser integradas apenas correções de erros. **Tendem** é a palavra chave neste contexto, dado que esta regra não é estrita. Esta versão é tratada de forma especial, e usuários têm expectativas de que entre uma versão e outra do branch estável não ocorram regressões.

Branch Instável: nesta versão são aceitos tanto reparos de erros quanto adições de funcionalidade. Não há uma garantia de que as versões funcionem de forma confiável, especialmente nas primeiras versões lançadas deste ciclo. Com o passar do tempo, é declarada uma moratória à adição de funcionalidade (o chamado *feature freeze*), e a versão instável passa por um período de estabilização. Ao final deste período, uma versão considerada 'estável o suficiente' é batizada de 'nova versão estável', e um novo ciclo se inicia.

Além de manter versões estáveis, branches CVS são utilizados para implementar mudanças que tenham impacto extenso sobre a base de código. Como visto anteriormente, cada desenvolvedor trabalha em relativo isolamento, com sua cópia local. Se não existissem branches, a integração de

mudanças extensas seria muito mais difícil, já que por grandes períodos de tempo corre-se o risco de outro desenvolvedor ter alterado o mesmo trecho do código. Utilizando um branch, é possível desenvolver a alteração paralelamente, e integrar mudanças para o tronco de forma incremental.

Alguns projetos existem com o objetivo de prover um software de controle de versões que contorne as limitações do CVS; um dos principais é o projeto Subversion (`www.subversion.org`). Vale notar que no início de 2002, Linus Torvalds anunciou que iria fazer uso do sistema Bitkeeper para controlar o código-fonte do núcleo Linux. Esta decisão é polêmica, levando em consideração o fato de que Bitkeeper não é software livre, estando livremente disponível apenas para **uso** para a maior parte das pessoas[1].

Esta dissertação não discute a fundo a ferramenta Bitkeeper; no entanto, a equipe do núcleo Linux afirma que a ferramenta tem **alto valor** e que proporcionou um ganho significativo de produtividade. Bitkeeper foi especialmente projetada para lidar com grandes números de micro-alterações concorrentes que são características de projetos grandes como o núcleo, e permanece como um objeto de estudos pouco conhecido mas muito interessante.

4.3.5 Apoio ao Processo de Desenvolvimento

Esta seção descreve outras ferramentas que são usadas pelos desenvolvedores regularmente. Como as descritas nas seções anteriores, precisam levar em consideração alguns valores fundamentais: suporte a texto simples (o que garante interoperabilidade), respeito às características individuais dos desenvolvedores, e suporte ao modelo offline de desenvolvimento.

Essas ferramentas podem ser divididas em alguns grupos básicos:

Ferramentas de Desenvolvimento: embora exista uma grande variedade de ferramentas disponível para desenvolvimento, boa parte dos desenvolvedores aparenta utilizar alguma variante dos dois editores de texto fundamentais disponíveis para Unix: o Emacs e o `vi`. Existem ambientes integrados para desenvolvimento, e algumas ferramentas para desenvolvimento de interfaces gráficas, mas à primeira vista parece existir um consenso em usar ferramentas familiares que suportam formatos padronizados, como texto e a família de linguagens *markup* do W3C (*World Wide Web Consortium*; ver `www.w3.org`).

Ferramentas de Configuração e Compilação: Uma das filosofias básicas do software livre é a portabilidade, e para tanto existem alguns sistemas para apoiar autoconfiguração e compilação condicional. O padrão de fato entre projetos de software livre parece ser baseado em um conjunto de ferramentas: `autoheader`, `autoconf`, `automake`, e `make` [35]. Estas ferramentas tornam

[1] Os termos exatos da licença do Bitkeeper têm variado, e são motivo de polêmica adicional. Para um exemplo comentado, visite `http://kerneltrap.org/node.php?id=444`.

relativamente simples a tarefa de compilar um novo software, e permitem que seja verificada a presença de pré-requisitos para a compilação.

Visualização de Código-Fonte: Existe uma variedade de formatadores e visualizadores de código-fonte. Os mais populares oferecem interfaces Web para visualização dos módulos de código-fonte. Exemplos deste tipo de ferramentas são o ViewCVS e CVSWeb, que oferecem visualização (somente de leitura) para o repositório CVS; o LXR, uma ferramenta que gera hiperlinks a partir de símbolos do código-fonte; e o Bonsai, uma ferramenta que oferece uma visão temporal das alterações integradas no repositório CVS.

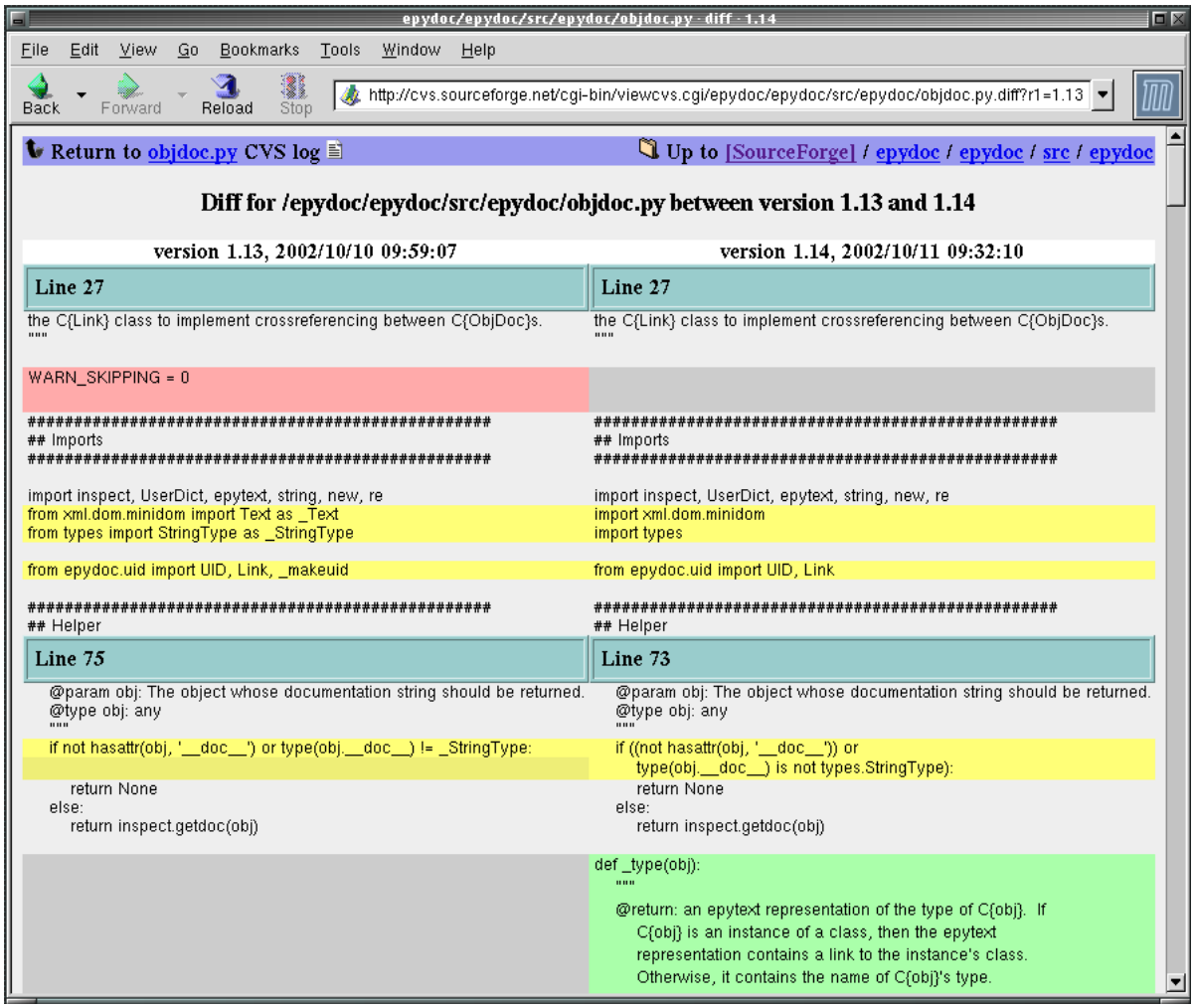


Figura 4.4: Tela exemplificando a função de *diff* da ferramenta ViewCVS, que permite visualizar as alterações entre duas versões de um arquivo controlado por CVS. Seções em verde, vermelho e amarelo correspondem às partes adicionadas, removidas e alteradas, respectivamente. Esta ferramenta, como a grande maioria de ferramentas de visualização em software livre, é baseada na Web, e pode ser acessada por qualquer pessoa. O código em questão é do software `epydoc`, e é mantido no serviço `sourceforge.net`.

Acompanhamento de Defeitos (ou Alterações): nos últimos anos tem havido um progresso consi-

derável em relação à criação de repositórios de informes de erros entre os projetos de software livre. Diversos projetos individuais existem para promover este tipo de ferramentas, incluindo o Bugzilla e o GNATS, e são atualmente usados por um grande número de projetos de software livre.

Estas ferramentas normalmente têm interface Web ou via correio eletrônico, e têm como função principal registrar mensagens de texto apontando a presença de erros, enviadas por usuários do software. Cada erro é registrado em um **informe** individual; cada informe criado possui um ciclo de vida que vai da sua criação (no momento em que é informado) até seu fechamento, que normalmente ocorre quando o erro é reparado na base de código. A ferramenta permite que sejam facilmente localizados e confirmados defeitos, incluindo informes que apontam erros duplicados, e informes que não se aplicam à versão atual do software.

O Bugzilla integra, além deste modelo fundamental, funcionalidade de revisão de código. A cada informe pode ser vinculado uma alteração de código provisória, que deve ser revisada e aprovada para integração na base de código principal.

4.3.6 Serviços de Hospedagem de Projetos

Nos últimos anos, com o advento do pioneiro site Web Sourceforge.net, algumas opções gratuitas para hospedar um projeto de software livre têm surgido. Um serviço de hospedagem oferece um conjunto de facilidades para um projeto:

- Espaço para hospedagem de páginas Web para o projeto.
- Espaço para oferecer versões lançadas do código-fonte para *download*.
- A possibilidade de criar listas de discussão próprias.
- Um repositório CVS, incluindo um sistema de visualização de código.
- Uma ferramenta de acompanhamento de defeitos.
- Controle de acesso para funções restritas (acesso de escrita ao CVS, e administração das listas de discussão, por exemplo).
- Formas de publicidade para o projeto através de concentradores de lançamentos, diretórios e serviços de busca.

Os serviços de hospedagem públicos oferecem interface via Web para manipulação dos serviços individuais dos projetos. A conveniência de ter todos estes recursos já configurados e disponíveis para um projeto é considerável, e muitos projetos de pequeno e médio porte têm migrado para este tipo de solução.

Um ponto a ressaltar é que os meta-projetos de maior porte, em sua maioria, já possuem esta infra-estrutura montada e custeada com recursos próprios, e portanto seus projetos não fazem uso destes serviços de hospedagem (alguns, como o Apache e o Mozilla, por exemplo, oferecem hospedagem a projetos de voluntários em serviços próprios como o `apache.org` e o `mozdev.org`).

4.4 Exemplos de Projetos de Software Livre

Entre os milhares de projetos de software livre existentes, há um número de projetos que se destacam pela sua grande base de usuários, qualidade ou originalidade. Esta seção descreve alguns projetos importantes, oferecendo uma visão geral de instâncias particulares do processo de software em projetos de software livre. Ao longo desta dissertação, outros projetos serão apresentados e descritos, adicionalmente.

4.4.1 Núcleo de Sistema Operacional: Linux

> '...com duas semanas do anúncio de Torvalds [sobre o lançamento do núcleo] em outubro de 1991, 30 pessoas tinham contribuído cerca de 200 informes de erros, contribuições de utilitários e drivers e melhorias para ser adicionados ao núcleo [...] em Julho de 1995, mais de 15.000 pessoas de 90 países e 5 continentes tinham contribuído comentários, informes de erro, correções, alterações, reparos e melhorias.' [36]

O Linux[2] (`www.kernel.org`) é um núcleo compatível com Unix. É um sistema operacional multi-tarefa preemptivo baseado na API definida pelo grupo POSIX do IEEE [37]. Seu autor original e mantenedor até o presente momento é Linus Torvalds [38]. Linux é inteiramente escrito em C e Assembler, somando mais de 2 milhões de linhas de código; o núcleo é também especialmente portátil, suportando atualmente mais de dez arquiteturas diferentes.

A equipe de desenvolvimento do núcleo aparenta ser muito grande; a média do número de indivíduos que participa semanalmente da lista de discussão principal do desenvolvimento do núcleo gira em torno de 500, e são resumidas as principais discussões no concentrador Kernel Traffic (`kt.zork.net/kernel-traffic/`). O desenvolvimento ocorre em ritmo acelerado, versões freqüentemente sendo lançadas com poucas semanas de intervalo.

O Linux, no entanto, é um dos projetos mais minimalistas em relação à infra-estrutura de desenvolvimento: até o início de 2002 apenas listas de discussão e e-mail eram utilizadas para discutir o desenvolvimento do núcleo. Como dito anteriormente, Linus e outros desenvolvedores vêm usando o Bitkeeper para controle de versões, e recentemente foi inaugurada uma instalação do Bugzilla para apoiar o Linux.

[2] Como dito anteriormente, existe alguma ambiguidade quando é usada o nome Linux: pode tanto se referir ao projeto de software livre dedicado ao núcleo quanto às distribuições baseadas nele. Neste texto, o nome Linux se refere exclusivamente ao projeto de software livre.

Numeração

Uma política de numeração freqüentemente observada entre projetos de software livre é utilizada no Linux. Cada núcleo tem um número de versão composto de 3 partes: nos referimos neste texto às partes individuais como primária, secundária e terciária (para a versão `2.4.20`, temos primário `2`, secundário `4` e terciário `20`). O conceito central da política de numeração: são mantidos 'troncos' ou 'ciclos' alternativos da base de código principal; os com numeração secundária **ímpar** são considerados instáveis; os com numeração secundária **par** são estáveis. Diversas versões de cada tronco podem existir; no nosso exemplo, `2.4.20` indica que 20 versões do tronco estável `2.4` foram lançadas.

Podem também existir múltiplos troncos estáveis. Atualmente, existem três em manutenção: `2.0`, `2.2` e `2.4`, este último mantido por um brasileiro chamado Marcelo Tosatti. Linux é um projeto peculiar por ter apenas um mantenedor principal por tronco para integrar alterações provindas desta enorme equipe; as pessoas em quem Linus confia agem, desta forma, como filtros para estas alterações.

Em resumo, uma característica deste projeto é a existência de diversas versões paralelas mantidas por estes 'filtros humanos', cada uma servindo como um veículo público de teste de integração, onde a alteração é testada antes de ser enviada ao Linus. Por exemplo, um dos desenvolvedores mais importantes do núcleo se chama Alan Cox, que mantém uma versão paralela do núcleo cujo número vem com um sufixo `ac` (ou seja, poderíamos ter uma versão paralela chamada `2.4.20-ac1`).

4.4.2 Servidor Web: Apache

O servidor Apache HTTP Server tem como função principal servir páginas Web. Faz parte do meta-projeto Apache Software Foundation (`www.apache.org`), uma fundação registrada e sem fins lucrativos. O projeto é baseado no código do servidor NCSA `httpd`; se iniciou como um conjunto informal de alterações mantidas por Brian Behlendorf, evoluindo para um primeiro lançamento público em abril de 1995. O software é hoje o servidor mais utilizado na Internet, correspondendo a mais de 60% do total de servidores Web (`www.netcraft.com`).

O projeto se organiza em torno de um núcleo de desenvolvedores, que são responsáveis por boa parte do código implementado; não há um líder único como no caso do Linux. O núcleo delibera internamente sobre a evolução do projeto e a entrada de novos membros. A interação com usuários e outros desenvolvedores é, como de costume, realizada através de listas de discussão e informes de erro. O projeto recentemente adotou o Bugzilla, substituindo o sistema de registro de erros BUGDB original. O projeto possui um repositório CVS público.

4.4.3 Navegador Web: Mozilla

Mozilla (`www.mozilla.org`) é um projeto criado pela Netscape/America Online para desenvolver um navegador Web. O projeto é um dos maiores entre os projetos de software livre existentes, com

dimensão comparável ao núcleo Linux, e se destaca por produzir software para usuários finais.

O projeto Mozilla é especial por demonstrar uma preocupação nítida com engenharia de software, e em particular com o processo de garantia de qualidade. Uma série de ferramentas de engenharia de software foi desenvolvida internamente para apoiar o projeto, e o conjunto de documentos detalhando processo, autoridade e status (`www.mozilla.org/roadmap.html`) demonstra que o processo de software tem importância dentro do projeto.

Além das ferramentas citadas acima, o Mozilla conta com CVS público, um site Web bastante completo, sua própria rede de IRC (um serviço de comunicação Internet em tempo real), e uma série de listas de discussão integradas com correio Usenet. A ferramenta Bugzilla, utilizada amplamente, foi escrita especificamente para atender ao projeto.

A autoridade pelo código no projeto Mozilla é dividida por módulos. Existe um ‘proprietário’ para cada módulo, responsável pelas decisões locais. É aplicada uma política firme de revisão de código, e cada alteração precisa ser formalmente revisada e aprovada antes de ser integrada.

Capítulo 5

Trabalhos Relacionados

Software Livre tem sido crescentemente abordado na literatura científica, e parte do objetivo desta dissertação é resumir o conteúdo disperso entre os principais trabalhos. Este capítulo cobre três tipos básicos de publicação: as consideradas fundamentais por prover as definições e o conhecimento essencial para o tópico deste trabalho; as que descrevem aspectos de projetos específicos de software livre; e as que tratam do assunto software livre como parte dos temas engenharia e processo de software.

5.1 Literatura Essencial

5.1.1 A Catedral e o Bazar de Eric Raymond

O trabalho de Eric Raymond 'The Cathedral and The Bazaar' [32] é considerado por muitos a definição fundamental de um processo de software para projetos de software livre (no livro, o termo 'Open Source' é usado); certamente é a publicação mais popular, citada recorrentemente pelos outros artigos apresentados nesta seção. É possível identificar entre grande parte dos trabalhos sobre software livre uma polarização em relação ao artigo de Raymond: ou apóiam seus preceitos sem questionar, ou fazem observações céticas e críticas ao seu conteúdo.

A idéia principal estabelecida no artigo é a existência de uma divisão clara entre o que o autor considera duas formas distintas de desenvolvimento de software livre: a 'Catedral', que descreve projetos de software livre desenvolvidos por grupos fechados, com pequena abertura para participação externa; e o 'Bazar', que descreve projetos desenvolvidos de forma mais transparente, abertos à participação de qualquer desenvolvedor que tenha interesse.

- **A Catedral** é tratada por Raymond como uma forma conservadora e tradicionalista de desenvolvimento; ao descrever este modelo, alude à criação de uma grande obra de engenharia, onde um grupo pequeno de sábios projeta e refina uma construção perfeita. Característico deste tipo de projeto são os longos ciclos de desenvolvimento, e a grande demora entre lançamentos de versões públicas. Raymond cita projetos como o GNU Emacs como exemplos desta forma de

desenvolvimento.

- **O Bazar** por sua vez é descrito como sendo um projeto colaborativo aberto ao ponto da 'promiscuidade' - usuários estão livres e bem-vindos a participar e opinar sobre o desenvolvimento em qualquer fase. Lançamentos ocorrem freqüentemente, e um grande número de 'olhos' tem acesso ao código-fonte, permitindo que erros e regressões sejam rapidamente avaliados e informados. Raymond cita o Linux e seu próprio projeto, o Fetchmail, como exemplos deste modelo de desenvolvimento.

O artigo coloca alguns pontos fundamentais a respeito do desenvolvimento de software livre, dos quais incluo os mais recorrentemente citados:

1. "Given enough eyeballs, all bugs are shallow" (Se exposto a um número suficiente de olhos, todos os bugs são superficiais), que Raymond chama 'A lei de Linus'. É uma metáfora que defende a importância de ter uma grande base de usuários com acesso ao código-fonte do ponto de vista da garantia da qualidade. O corolário desta 'lei' é a frase 'Debugging is paralelizable' (Depuração é paralelizável).

   Raymond considera este aspecto do processo de software livre fundamental para contrapor o princípio básico enunciado por Fred Brooks no seu livro 'The Mythical Man-Month', onde afirma que à medida que a equipe de um projeto cresce, menor é sua eficiência dado o custo crescente de comunicação e coordenação [39].

2. "Release early. Release often. And listen to your customers" (Lance cedo, lance freqüentemente, e ouça o que seus clientes têm a dizer), que defende a prática de oferecer aos usuários a oportunidade de testar freqüentemente a base de código mais recente.

Existe uma série de discussões e críticas [40] publicadas a respeito deste trabalho. A maior parte destas coloca restrições com relação à cisma radical do modelo — as diferenças entre o Bazar e a Catedral não são tão evidentes quanto Raymond apresenta inicialmente. Tanto no Linux quanto em projetos mais conservadores é exercida uma boa medida de controle, e a 'promiscuidade' dos projetos estilo Bazar que o artigo anuncia é em grande parte um reflexo sutil da maneira como são tratados novos participantes. Certos projetos têm uma equipe estimulante e amigável; outros uma equipe mais fechada e reservada.

Também fica pouco claro no contexto do artigo o que exatamente significa fazer 'lançamentos freqüentes' quando o contexto atual de software livre é um de acesso quase instantâneo a código-fonte através de CVS e outras ferramentas de desenvolvimento colaborativo. Mesmo o Emacs, considerado por Raymond o exemplo supremo do estilo de desenvolvimento Catedral, possui um repositório de código e listas de discussão publicamente disponíveis (`savannah.gnu.org/projects/emacs`).

O artigo de Raymond é importante sobretudo por ser um pioneiro na descrição do que consistem projetos de software livre, e por incluir conhecimento provindo da experiência prática de Raymond na operação de um de seus projetos de software livre, o Fetchmail.

5.1.2 Desenvolvimento Distribuído: Herbsleb e Grinter

> 'Despite the necessity, and perhaps even desireability of geographically distributed development, it is extremely difficult to do so successfully[1].'

Um trabalho importante que descreve experiência prática com o desenvolvimento de software feito por equipes distribuídas é o artigo 'Splitting the Organization and Integrating the Code: Conway's Law Revisited', de James Herbsleb e Rebecca Grinter [41]. O artigo relata um projeto de desenvolvimento de software, distribuído entre equipes da Lucent na Alemanha e no Reino Unido.

Neste artigo, os autores apontam o problema fundamental do processo distribuído: **dificuldade de comunicação** entre os membros das equipes. Esta limitação essencial acaba produzindo problemas face às inevitáveis alterações nos requisitos, projeto e interface dos componentes desenvolvidos nas respectivas localidades: não há um veículo de comunicação implícito ou informal para transmití-las.

O extraordinário neste trabalho é que fica clara a divergência natural que ocorre entre o processo idealizado e o que é implementado de fato, e o quanto é essencial a comunicação – muitas vezes informal – para permitir o constante ajuste e reavaliação deste processo. A ausência de mecanismo e ferramentas para facilitar a comunicação resulta em defeitos, problemas de integração, falta de coordenação entre as equipes, e, em uma perspectiva mais ampla, custos elevados e eficiência mais baixa.

Esse artigo é especialmente importante por descrever uma situação geograficamente distribuída muito semelhante à encontrada nos projetos de software livre, mas com um processo de software distinto. O artigo traz à tona a questão: **se desenvolvimento distribuído é tão difícil, como é que os projetos de software livre têm êxito, dada a relativa simplicidade do seu processo e de suas ferramentas?**

5.1.3 A avaliação empírica de Sandeep Krishnamurthy

Sandeep Krishnamurty, no seu artigo 'Cave or Community? An Empirical Examination of 100 Mature Open Source Projects' [42] realiza um levantamento que pode ser considerado um precursor deste presente trabalho de mestrado. É uma pesquisa empírica baseada em dados de 100 projetos de software livre hospedados no site `sourceforge.net`, e nele o autor verifica algumas propriedades dos projetos observados:

- A grande maioria dos projetos já maduros é desenvolvida por um grupo pequeno de indivíduos:

[1] 'Apesar da necessidade e até desejo em implementar desenvolvimento distribuído, é extremamente difícil fazê-lo com sucesso.'

o número mediano de desenvolvedores envolvidos por projeto não passa de 4, e somente 29% dos projetos avaliados possui mais de 5 desenvolvedores. 22% dos projetos possui apenas um desenvolvedor.

- Com relação às listas de discussão por correio eletrônico, a maior parte dos projetos produz muito pouca discussão; poucos projetos geram discussão intensa.
- Existe uma correlação entre a idade do projeto e o número de desenvolvedores, o que é naturalmente esperado uma vez que projetos com maior longevidade têm por definição um período mais longo para atrair uma comunidade.

A conclusão do artigo é que o modelo mais reconhecido para desenvolvimento de software livre, que é o de uma comunidade grande de desenvolvedores interagindo intensamente, não se aplica a grande parte dos projetos existentes atualmente: a maior parte dos projetos tem equipe pequena e reduzida participação do público. Os fatores que levam a esta situação ainda precisam ser melhor estudados, mas o autor coloca algumas hipóteses: a dificuldade que projetos pequenos têm para atrair desenvolvedores, já que o benefício global da sua contribuição parece pequeno; e a tendência dos projetos grandes de concentrar a atenção da comunidade por sua alta visibilidade e conseqüente alta recompensa para participantes.

5.1.4 Detratores Iniciais

Os trabalhos discutidos nesta seção apresentam dúvidas ou restrições ao processo de desenvolvimento de software livre. É interessante observar que, embora não existisse uma definição concreta de como exatamente este processo seria, são pertinentes as perguntas colocadas, e as limitações apontadas. Por este motivo, pode-se dizer que esses artigos constituem a inspiração deste trabalho de mestrado, motivando a busca por explicações, respostas e esclarecimentos para estes desafios.

Open Source Methodology: Ready For Prime Time? Este artigo de Steve McConnell [43] é uma crítica ao alarde inicial feito a respeito das vantagens do desenvolvimento de software livre. Concede que o processo de alguma forma ‘funciona’, mas discute a real confiabilidade do software produzido, e a ausência de requisitos e projeto de alto nível. Aponta uma propriedade fundamental de software livre: a potencial ineficiência gerada pelo trabalho simultâneo de diversos desenvolvedores sobre uma mesma base de código.

The Sociology of Open Source: Of Cults and Communities: O artigo de Robert Glass [44] coloca alguns paradoxos aparentes relacionados às motivações de participar em um projeto de software livre: como é possível que os desenvolvedores apreciem dar manutenção em software, ler e estudar código alheio, e trabalhar sem remuneração, já que historicamente se mostram avessos a estas tarefas e situações?

Is the Open-Source Community Setting a Bad Example? Greg Wilson [45] critica neste trabalho a despreocupação prevalecente na comunidade de software livre com relação ao processo de software, e em particular a inexistência ou desatualização das ferramentas de desenvolvimento.

5.2 Estudos sobre Projetos Específicos

5.2.1 O Servidor Apache

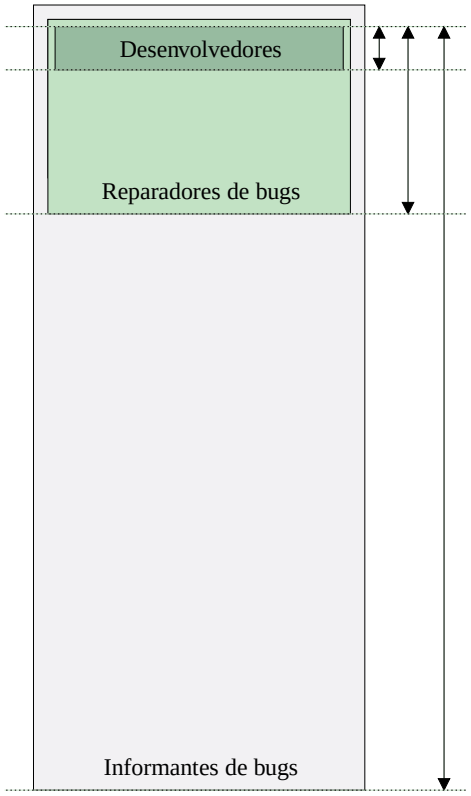


Figura 5.1: Uma representação gráfica da comunidade de desenvolvedores como enunciada por Mockus et al. em sua terceira hipótese: há um aumento de uma ordem de grandeza entre a dimensão de cada população indicada. (O diagrama não representa esta escala exata.)

O artigo 'A Case Study of Open Source Software Development: The Apache Server', de Audris Mockus, Roy Fielding e James Herbsleb [46] é um dos primeiros e mais completos tratamentos científicos feitos sobre um projeto de software livre. O artigo faz uma avaliação das vantagens e eficiências hipotéticas do processo de software livre, relacionando-as com a situação prática de um projeto importante.

No artigo é feita uma análise do conjunto de mensagens de correio eletrônico, do repositório CVS do projeto, e da base de informes de erro. A partir destas fontes de dados os autores levantam informações sobre densidade de defeitos, a distribuição do número de alterações por desenvolvedor, e o nível de 'propriedade de código' por módulo.

As conclusões do artigo lançam um conjunto de sete hipóteses experimentais. As mais relevantes para a presente análise são suas hipóteses 3, 6 e 7:

- Hipótese 3: Em projetos de software livre bem-sucedidos, um grupo maior do que o grupo de desenvolvedores por uma ordem de magnitude irá submeter reparos para defeitos. Outro grupo *ainda* maior, por *outra* ordem de magnitude, irá informar defeitos.

  É interessante notar como esta hipótese parece ser confirmada no trabalho de Krishnamurthy descrito anteriormente: o grupo de desenvolvedores da grande maioria dos projetos é pequeno, e a maior parte do código-fonte será escrita por poucos indivíduos.

- Hipótese 6: Em projetos de software livre bem-sucedidos, os desenvolvedores serão também usuários do software.

- Hipótese 7: Projetos de software livre oferecem resposta muito rápida a problemas reportados por usuários finais.

Há um outro artigo, escrito por Roy Fielding, ‘Shared Leadership in the Apache Project’ [47]. Este trabalho apresenta um histórico organizacional do projeto Apache, e descreve subjetivamente as tarefas de coordenação e o processo de tomada de decisão no projeto. O Apache é um exemplo interessante de um projeto liderado por um comitê, e não um líder único.

5.2.2 O Núcleo Linux

O núcleo de sistema operacional Linux é o maior e mais famoso projeto de software livre, e por este motivo há um grande conjunto de trabalhos que o estudam sob diferentes perspectivas [38, 48]. Além dos artigos discutidos nesta seção, o núcleo tem servido de base para estudos que vão desde ferramentas de análise de código-fonte a propostas de otimização para subsistemas do núcleo.

Dinâmica Social Distribuída

Sendo o primeiro projeto de software livre a receber atenção particular, é evidente que estudos sobre os mecanismos sociais que sustentam o desenvolvimento tenham se direcionado ao Linux. Dois importantes artigos existem e discutem o núcleo com relação aos preceitos colocados por Raymond.

O primeiro destes é o artigo de Jae Yun Moon e Lee Sproull, ‘Essence of Distributed Work: The Case of the Linux Kernel’ [36]. Descreve os antecedentes do projeto, e oferece uma visão geral da evolução de sua comunidade de desenvolvedores do núcleo desde o lançamento inicial de Linus em 1991. É uma boa fonte de informações sobre o projeto como um todo, descrevendo ferramentas de comunicação, estrutura de liderança e a política de numeração de versões. O trabalho conclui com uma série de explicações sobre o sucesso do modelo de desenvolvimento: acesso à Internet onipresente, uma licença de software adequada, a necessidade de liderança, a paralelização de trabalho, o envolvimento pessoal e a resultante motivação dos desenvolvedores.

O segundo é um longo texto de Ko Kuwabara, ‘Linux: A Bazaar at the Edge of Chaos’ [49]. Este texto é especial por reunir fragmentos de discussões em mensagens de correio eletrônico para caracterizar os aspectos de liderança e alocação de trabalho, e a forma como são discutidos e arbitrados estilo, projeto e implementação de funcionalidade. Uma das contribuições mais importantes deste trabalho é a descrição do Linux como um sistema complexo, onde a evolução parece ser não-direcionada e onde não existe planejamento ‘top-down’ além de um consenso interno de que as decisões devem ser tecnicamente excelentes.

Evolução

Entre os diversos trabalhos sobre o Linux, o artigo de Michael W. Godfrey e Quiang Tu 'Evolution in Open Source Software: A Case Study' [50] se destaca por fazer uma análise empírica do crescimento do núcleo[2], com base em 96 versões lançadas entre março de 1994 e dezembro de 1999. Sua conclusão — que o crescimento do Linux é super-linear e que não mostra sinais de redução — é interessante tendo em vista estudos prévios sobre a evolução de sistemas de software, que sugerem que o crescimento de um software encontra barreiras ao longo da sua vida, e que seu ritmo de desenvolvimento diminui à medida que se torna mais complexo [51].

O artigo discute algumas explicações para o progresso rápido do núcleo: alta modularidade, propensão natural de crescimento dos dispositivos de hardware, e a entrada de grandes contribuições para suporte a novas arquiteturas; no entanto, fica claro que é inesperado o resultado obtido, especialmente quando contemplado o peculiar processo de desenvolvimento.

Outro artigo que discute a manutenibilidade do núcleo é o de Stephen R. Schach et al., 'Maintainability of the Linux Kernel' [52]. O trabalho analisa o acoplamento e a interdependência entre módulos do núcleo utilizando uma ferramenta que processa código-fonte. A conclusão do trabalho é que o núcleo possui alto acoplamento, e que este acoplamento cresce exponencialmente à medida que o núcleo em si cresce. Esta propriedade, em teoria, reflete numa crescente dificuldade na manutenção do código do núcleo. Resta observar a validade prática desta conclusão, já que o desenvolvimento do núcleo (incluindo reestruturação e evolução arquitetural) continua normalmente até o presente momento.

5.2.3 O Projeto Mozilla

O artigo 'An Overview of the Software Engineering Process in the Mozilla Project' [53], elaborado como parte deste trabalho de mestrado, oferece uma visão geral do projeto Mozilla do ponto de vista do processo de software. Descreve como são realizadas as atividades básicas de requisitos, projeto, implementação e garantia de qualidade, e detalha as ferramentas utilizadas para apoiar a comunidade de desenvolvedores.

São descritos os mecanismos de revisão e aprovação formal de alterações, usados e apoiados pela ferramenta de gerência de defeitos do projeto; esta atividade é especial por ser raramente encontrada de maneira tão formal em outros projetos de software livre. Também é especial o produto final do projeto: um navegador Web para o usuário final, o que não é comum para um projeto de software livre; o artigo aponta como é problemática a determinação dos requisitos de interface e usabilidade, e de como é necessário ainda trabalho nesta área para estabelecer um processo eficiente de arbitragem.

O Projeto Mozilla é especialmente interessante por usar e oferecer uma série de ferramentas para apoiar o processo de desenvolvimento, e o artigo descreve resumidamente as ferramentas utilizadas: Bonsai, que permite fazer consultas no histórico de alterações no repositório de código; Bugzilla, uma

[2] O artigo também avalia o projeto VIM, um editor de texto compatível com o editor `vi` original.

ferramenta que integra um repositório de informes de erro e um sistema de apoio para revisão; e LXR, uma ferramenta que gera páginas Web a partir de código-fonte, com hiperlinks entre seus símbolos.

5.2.4 O Projeto FreeBSD

O trabalho de Niels Jørgensen, 'Putting it All in the Trunk: Incremental Software Development in the FreeBSD Open Source Project' [54] descreve um projeto muito importante e relativamente pouco conhecido. O texto detalha o processo de desenvolvimento aplicado no projeto, que é caracteristicamente iterativo e incremental. O surpreendente do processo é sua grande escala: o FreeBSD possui mais de 200 indivíduos com permissão para integrar código-fonte no repositório central, o que é surpreendente e sugere complexidade de gerenciamento muito alta[3].

O artigo descreve detalhadamente um ciclo de vida para alterações de código, e em particular cobre a forma como estas são controladas e auditadas. Ressalta que boa parte do problema de gerenciamento do código é resolvido pelo fato do nome do autor da alteração estar publicamente vinculado ao código: 'Embarassment is a powerful thing', e que as questões restantes são raras e resolvidas caso a caso pelo grupo de desenvolvedores principal.

O artigo apresenta outro ponto interessante: que derivados do Unix original podem evitar boa parte do esforço de projeto arquitetural porque possuem um roteiro pronto, elaborado ao longe de décadas de experiência. Ressalta, no entanto, que inovação de fato ocorre em áreas não contempladas previamente, como suporte a multiprocessamento, e que o processo de software utilizado varia de acordo com o tipo e a extensão da alteração a ser implementada.

5.3 Processo de Software e Projetos de Software Livre

Nesta seção são analisados trabalhos da literatura que tratam das atividades e aspectos gerais do processo de software em projetos de software livre.

5.3.1 Obtenção de Requisitos

Bart Massey discute no seu artigo 'Where do Open Source Requirements come from (And what should we do about it?)' [55] as diferenças entre o processo de engenharia de requisitos em projetos convencionas e o processo em projetos de software livre. Ressalta a ausência de um processo formal de requisitos, dando atenção particular à colocação de Raymond que descreve a origem de boa parte dos projetos de software livre: *'Every good work of software starts by scratching a developer's personal itch'* (traduzido livremente, todo bom software começa por atender a uma necessidade pessoal do seu desenvolvedor). Como colocado por Mockus et al. anteriormente, esta é uma propriedade importante

[3]Na realidade, não é incomum um projeto de maior porte ter um grande número de desenvolvedores com permissão para integrar código; no caso do Mozilla, por exemplo, o número é similar.

de grande parte dos projetos: o desenvolvedor também é usuário do software.

O artigo levanta uma série de hipóteses para explicar a ausência deste processo de requisitos, citando fontes possíveis para os requisitos em projetos de software livre:

1. Podem originar-se diretamente dos desenvolvedores.
2. Podem originar-se de usuários de software livre, através de feedback pós-lançamento.
3. Podem resultar da implementação de padrões estabelecidos explicitamente.
4. Podem resultar da implementação de padrões de fato ou de mercado.
5. Podem resultar da necessidade de implementar protótipos com fins didáticos ou acadêmicos.

Estas hipóteses são repetidas informalmente em diversos outros trabalhos, e são um ponto inicial para um trabalho de elucidação do processo de requisitos em software livre. Massey termina por identificar as fraquezas deste processo: baixa escalabilidade, *inbreeding*, e uma tendência à concentração em nichos específicos, *versus* a possibilidade de expandir para áreas onde desenvolvedores não são normalmente usuários.

5.3.2 Qualidade

Dois artigos se destacam entre os trabalhos que analisam o processo de qualidade em software livre. O primeiro, 'A Survey on Quality Related Activities in Open Source' de Luyin Zhao e Sebastian Elbaum [56] descreve um levantamento feito por meio de um questionário veiculado a um grupo de desenvolvedores de software livre. O artigo cita alguns dados interessantes: 80% dos entrevistados afirmou não usar um plano de testes para seus produtos, e a interface com o usuário foi a área crítica mais citada, representando 28% das áreas apontadas como mais problemáticas. Infelizmente o questionário utilizado apresenta problemas de usabilidade — utiliza muitos termos do processo da engenharia de software convencional, o que prejudica sua compreensão e aplicabilidade a software livre — e tendo isto em vista, as medidas de tempo de testes, em particular, parecem ter pouca significância.

O segundo artigo, 'High Quality and Open Source Software Practices', de T.J. Halloran e William L. Scherlis [57], analisa o processo de qualidade através de uma observação das práticas de qualidade em um conjunto de projetos importantes de software livre. Sugere a existência de diversos mecanismos para manutenção de qualidade nestes projetos:

- O código do projeto é isolado atrás de uma barreira; existe uma forte distinção entre a grande abertura em relação à disponibilidade de código, e a política bastante restritiva de integração de contribuições. Esta configuração permite grande liberdade individual (já que o desenvolvedor é livre para alterar o código que ele tem nas mãos) sem prejuízo para a estabilidade geral do produto de software.

- Os processos são fortemente mediados pelas ferramentas de comunicação utilizadas, removendo do líder parte da obrigação de garantir o respeito às políticas do projeto.

  Exemplos claros são o processo de revisão no projeto Mozilla (apoiado pela ferramenta Bugzilla) e a política de integração embutida na ferramenta de controle de versões CVS, usada em um grande número de projetos.

- Para apoiar a entrada de novos participantes, os projetos tendem a usar um conjunto de ferramentas de infraestrutura padronizado para minimizar a barreira de adaptação. Este conjunto inclui o CVS, documentos em texto simples, listas de discussão e correio eletrônico, e repositórios como o Bugzilla e o `sourceforge.net`.

O texto conclui que, embora sejam interessantes melhorias nos processos de desenvolvimento em projetos de software livre, devem ser implementadas e integradas às ferramentas de forma incremental. Esta conclusão parece bastante razoável tendo em vista a padronização prática destas ferramentas e o alto custo de mudanças no fluxo de trabalho de uma comunidade grande de desenvolvedores.

5.3.3 Comunicação

O artigo de Yutaka Yamauchi et al, 'Collaboration with Lean Media' [58] oferece uma visão geral dos mecanismos de comunicação usados em projetos de software livre, concentrando-se em uma análise das comunicações feitas entre desenvolvedores dos projetos GCC e FreeBSD-Newconfig, e generalizando as conclusões no contexto maior que é CSCW (*Computer Supported Collaborative Work*).

O que o artigo aponta é inesperado: apesar da tendência acadêmica de produzir ferramentas complexas para suportar trabalho colaborativo desta natureza, projetos de software livre sobrevivem e comprovam a eficácia do uso de ferramentas e meios de comunicação extremamente simples ('lean', nos termos do artigo): correio eletrônico, arquivos texto, CVS. Particularmente notável é que esta conclusão reflete precisamente o padrão de ferramentas do trabalho de Halloran e Scherlis descrito na seção anterior.

Além de observações sobre a tecnologia usada nas ferramentas, o artigo mostra duas tendências que ajudam a definir a comunidade de software livre: a parcialidade em relação à ação prática, que é caracterizada pela famosa frase de Linus Torvalds, 'Show me the code!' (mostre-me o código!); e a cultura altamente racional dos membros, que buscam explicar suas ações através de raciocínio lógico e justificativas tecnológicas. Certamente ambos os aspectos são passíveis de observação diária nas listas de discussão de qualquer projeto de software livre de maior porte.

5.3.4 Gerenciamento de Configuração

O trabalho de André van der Hoek, 'Configuration Management and Open Source Projects' [34], traz uma justificativa do uso generalizado de CVS, e uma análise de suas fraquezas e possíveis melhorias. Um ponto muito importante que o artigo apresenta, e que não é reproduzido em nenhum outro

artigo nesta seção, é que o **CVS é amplamente usado por modelar muito bem o processo de desenvolvimento de software descentralizado, assíncrono e não-intrusivo que caracteriza projetos de software livre**.

U. Asklund e L. Bendix, no seu trabalho 'A study of configuration management in open source software projects' [59], fazem um levantamento geral das práticas de gerenciamento de configuração (GC) em software livre. Descrevem as principais atividades de GC realizadas em projetos de software livre:

- Controle de versões, seleção de configuração e gerência da área de trabalho, que são feitos usando ferramentas como o CVS, e que portanto seguem seu fluxo de funcionamento básico.

- Gerência de compilação, que é feita usando ferramentas como o `make` e `autoconf`, como descrito na Seção 4.3.5. Interessante é a observação de que é normalmente pouco custoso regenerar ou recompilar o software a partir da versão obtida do repositório.

- Controle de alterações, que é particular a cada projeto, mas que em termos gerais difere substancialmente do processo convencional por não utilizar pré-alocação de trabalho. Descreve adicionalmente os dois processos distintos de integração: quando há permissão de alteração direta no repositório, e quando é necessário o envio da alteração a um moderador.

Os autores apontam fatores importantes que as ferramentas de controle de versão devem oferecer para atender às necessidades deste processo: entre outras, suporte a múltiplas plataformas, suporte ao modelo de trabalho offline e a longo prazo semelhante ao que CVS oferece, facilidade e conveniência de obtenção e visualização de versões arbitrárias do código-fonte, e o suporte a múltiplos *baselines* (ou branches). Detalham também um aspecto essencial do processo de software, que é social: desenvolvedores são controlados em grande parte pelo desejo de manter uma boa reputação perante a comunidade, e por isso têm cuidado especial com o trabalho que buscam integrar.

O artigo, no entanto, apresenta algumas imprecisões e algumas informações desatualizadas. Entre elas, afirma que não há processo de lançamento de software (com garantia de qualidade, documentação, e ferramentas para instalação), o que contradiz os grandes esforços para lançamento que projetos como o Mozilla, o GNOME e o FreeBSD executam. Sua avaliação do projeto Mozilla é pouco precisa, ignorando a forte política de revisão e integração de código. Afirma que Linux não possui um mecanismo para recuperar versões anteriores do código, o que é incorreto em dois níveis: primeiro, porque diversos repositórios armazenam **todas** as versões publicamente lançadas, e segundo, porque a partir do início de 2002 Linus Torvalds adotou publicamente a ferramenta Bitkeeper (`www.bitkeeper.com`) para gerenciamento de versões do núcleo. O artigo data de 2001, e por este motivo pode ter ignorado o fato do Bitkeeper estar sendo avaliado por Linus há algum tempo. Estas falhas não comprometem significativamente as recomendações principais do artigo.

5.3.5 Progresso, Eficiência e Evolução

Estes assuntos parecem atrair a maior parte das pesquisas relacionadas a software livre, e existem diversos bons trabalhos e levantamentos. O primeiro destes é o trabalho de Bart J Dempsey et al. 'A Quantitative Profile of a Community of Open Source Linux Developers' [60], que faz uma avaliação dos projetos hospedados no repositório `www.ibilio.org` (na época da publicação chamado `metalab.unc.edu`, e originalmente `sunsite.unc.edu`). O repositório mantém um conjunto muito grande de pacotes de software livre, não necessariamente relacionados ao núcleo ou distribuições Linux (e por este motivo é um pouco infeliz a escolha do título do trabalho). Utilizando uma ferramenta para avaliar automaticamente os pacotes disponíveis e suas mudanças ao longo do tempo, fazem uma série de classificações com base na velocidade de atualização, no ritmo de criação de novos pacotes, e no perfil de participação de desenvolvedores individuais.

Entre os dados levantados mais notáveis estão:

- A grande quantidade de autores que contribuíram apenas com um pacote. Em outras palavras, há uma forte tendência à contribuição e manutenção de um software específico por desenvolvedor; são raros desenvolvedores que mantêm um conjunto numeroso de softwares. Isto confirma a tendência apontada no trabalho de Krishnamurthy descrito na Seção 5.1.3.

- A regularidade e longevidade das atualizações aos pacotes. Durante o período de um mês, foram observadas todas as alterações feitas aos pacotes em um subconjunto do repositório; ao longo deste mês, 1.5% dos pacotes deste subconjunto foram atualizados, e foram acrescentados novos pacotes perfazendo 4% do total. Estas cifras sugerem uma constante e saudável atualização dos pacotes hospedados no repositório.

- A população de desenvolvedores responsáveis por pacotes provém de um grande número de países, com base nos endereços de correio eletrônico individuais. A maior parte dos endereços são provenientes de domínios tipicamente norte-americanos (`.com, .edu, .net` e `.org`). Os países melhor representados, baseando-se estritamente nos domínios Internet, são Alemanha, Reino Unido, Holanda, Austrália e França[4].

Ecologia da População de Projetos

> "Open Source Software is a hybrid - part economic project, part network organization, part social movement[5]."

[4] Esta avaliação não é particularmente precisa por ser comum o uso de um endereço de correio eletrônico não-específico ao país onde mora o desenvolvedor, mas é uma aproximação aceitável, levando-se em consideração a predominância dos TLDs (*top level domains*) principais.

[5] 'Software open source é um híbrido - em parte projeto econômico, em parte organização de rede, em parte movimento social.'

Outro importante artigo é 'The Ecology of Open-Source Software Development', de Kieran Healy [61]. Este artigo faz um levantamento com base nos mesmos dados de Krishnamurthy, mas com um enfoque distinto, buscando entender os padrões de liderança e a estrutura organizacional do projeto de software livre típico. O que se constatou é que a intensidade da atividade nos projetos de software livre tem uma distribuição bastante tendenciosa; apenas um conjunto *muito* pequeno de projetos tem mais do que 5 participantes. Além disso, o número de commits feitos ao repositório CVS, o número de mensagens, o número de downloads dos pacotes e o número de visitas ao site apresentam padrão bastante similar. Resumidamente, os resultados sugerem existir grandes disparidades entre os projetos individuais, e que um número pequeno de projetos concentra grande parte da atenção e atividade da comunidade.

Healy também critica as colocações de Raymond, comparando o Bazar teórico ao projeto típico, que não possui mais do que um punhado de desenvolvedores — como é possível ocorrer revisão de código intensa para garantir qualidade se não há volume de pessoas suficiente para sequer ler este código?

Algumas hipóteses são estabelecidas para explicar estas discrepâncias. Por exemplo, boa parte dos projetos hospedados no `sourceforge.net` nunca produzirá código utilizável, muitos sendo abandonados após o estabelecimento de um documento de intenções[6]. Healy conclui sugerindo três potenciais justificativas:

- A concentração de desenvolvedores experientes pode ser um determinante do sucesso dos projetos; em outras palavras, quanto maior o sucesso de um projeto de software livre, mais experientes serão seus desenvolvedores.

- A tarefa de liderança e organização do grupo de desenvolvedores é provavelmente crucial; projetos de sucesso têm líderes respeitados e tecnicamente competentes. Healy sugere que, neste aspecto, projetos de software livre tenham forte semelhança com movimentos sociais, e que seu sucesso seja ligado diretamente às habilidades de seu líder.

- Por último, contrapondo a visão do Bazar não-hierárquico proposto por Raymond, Healy indica que hierarquia cumpre um papel importante na determinação do sucesso de projetos de software livre. Deve-se notar que isto não implica uma hierarquia rígida, e sim um grau de hierarquia que seja benéfico ao projeto, e que permita um convívio e coordenação sustentável.

Padrões Evolutivos

O artigo 'Evolution Patterns of Open-Source Software Systems and Communities', de Kumiyo Nakakoji et al. [62], é um estudo de caso baseado em quatro projetos *envolvidos* com software livre – dois são grupos de usuários locais, um é parte de um projeto de software livre (cuidando de tradução e

[6] Minha participação pessoal em um projeto abandonado, hospedado no site `sourceforge.net`, confirma esta explicação.

adaptação da base de dados PostgreSQL para novas arquiteturas), e um é de fato um projeto de software livre (Jun, uma biblioteca multimídia).

Por si só esta é uma importante contribuição do artigo: discute a existência destas organizações secundárias (os grupos de usuários) que têm como meta apoiar o uso de software livre em uma comunidade local. No artigo, os motivos colocados para sua existência são tanto técnicos quanto culturais — a barreira do idioma, a falta de familiaridade com o processo de desenvolvimento de software livre e a barreira de costumes oriental-ocidental. Servem como uma ponte essencial entre os desenvolvedores dos projetos e os usuários com menor conhecimento técnico, mas boa experiência prática no uso da ferramenta.

O trabalho ainda contribui com dois modelos que buscam generalizar a participação individual e a evolução da base de código dos projetos. O modelo de participação oferece um detalhamento em relação às categorias que defini na Seção 4.3.2; classifica os membros de uma comunidade de software livre em 8 níveis, aqui listados em ordem crescente de envolvimento com o desenvolvimento do projeto:

- passivo (que é usuário mas não interage com a comunidade).
- leitor (que assina e lê as listas de discussão do projeto),
- *bug reporter* (que informa defeitos),
- *bug fixer* (que faz reparo ocasional de defeitos),
- desenvolvedor ocasional,
- desenvolvedor ativo,
- membro do núcleo,
- líder,

Embora não seja aplicável a qualquer projeto, este modelo é uma boa referência para descrever a participação individual em projetos maiores. É bastante similar ao modelo proposto por Cristina Gacek et al. no artigo ‘The Many Meanings of Open Source’ [23], sendo que este último trabalho oferece ainda um mapeamento das categorias para as atividades que realizam dentro da comunidade.

5.3.6 Análises de Código-fonte

Um número de pesquisas se concentra nos produtos de software dos projetos de software livre. É notável como este tipo de análise em larga escala só pode ser realizada na prática com software livre, já que seria impraticável realizar uma pesquisa desta abrangência entre um conjunto extenso de software proprietário.

A Distribuição Redhat

Dois trabalhos de David Wheeler descrevem estudos da dimensão da distribuição Redhat Linux: 'Estimating Linux's Size' e 'More Than a Gigabuck: Estimating GNU/Linux's Size' [63, 64]. A ferramenta escrita e utilizada por Wheeler nestes trabalhos, `sloccount`, também é disponível como software livre e foi utilizada no presente trabalho de mestrado para fazer parte do tratamento dos dados.

Wheeler mede a distribuição de licenças entre os pacotes, o esforço de desenvolvimento em termos de homens-ano, e o custo em termos do modelo COCOMO [65] e do salário médio de um programador americano. A conclusão mais notável desta medida é que a distribuição Redhat teria custado mais de 1 bilhão de dólares para ser criada segundo um modelo de desenvolvimento convencional. Os corolários desta conclusão: que é possível ser elaborado, por um grupo de desenvolvedores independentes – em grande parte voluntários – um sistema operacional completo de dimensão significativa; que compartilhar código livremente é de fato uma forma eficaz de se construir sistemas complexos.

A Distribuição Debian

Jesús M. González-Barahona et al. descrevem no artigo 'Counting Potatoes: The size of Debian 2.2' [66] uma investigação realizada sobre a distribuição Debian. Como nos artigos de Wheeler, resulta desse trabalho uma análise detalhada do código que compõe a distribuição, classificado por linguagem de programação e por pacote individual. Faz também uma comparação dos totais obtidos com a dimensão publicada de outros softwares modernos, tanto livres quanto não-livres.

Nesta pesquisa também foi utilizada a ferramenta de David Wheeler, o `sloccount`. Esse trabalho é particularmente interessante por descrever a metodologia utilizada, incluindo os critérios para exclusão dos pacotes e as limitações e ressalvas que afetam uma análise em grande escala de código-fonte.

Os estudos Orbiten e FLOSS

Dois estudos semelhantes foram realizados sobre grandes conjuntos de software livre com o objetivo de determinar autoria. O primeiro estudo, realizado entre 1998 e 2000, é o 'Orbiten Free Software Survey' de Rishab Aiyer Ghosh e Vipul Ved Prakash [7]. Este estudo se fundamentou no código da distribuição Redhat 6.1, sobre o qual foi realizado um processamento automatizado com base em uma ferramenta chamada CODD. Esta ferramenta analisa o código-fonte com base em um conjunto de heurísticas (descritas em `orbiten.org/codd/`) e relata a distribuição de propriedade dos arquivos de código-fonte. Com base nestes relatórios, é possível construir listas de autores com as proporções que contribuíram.

O segundo estudo, FLOSS[7], é composto de um conjunto grande de análises realizadas para pro-

[7] Disponível em: `http://www.infonomics.nl/FLOSS/report/`

mover uma visão melhor do cenário de projetos de software livre. Entre estas análises se destaca uma reprodução do estudo Orbiten com uma escala ainda maior de projetos de software livre [67], que é pertinente para a discussão atual. O relatório do projeto inclui detalhes sobre a propriedade do código, fazendo relação com a dimensão total de autores.

Os resultados de ambos os trabalhos se complementam: descrevem uma concentração de responsabilidade pelo código semelhante à descrita na hipótese 3 de Mockus et al., mas ainda mais extrema. Apontam que a maior parte do código de projetos de porte médio e pequeno é escrita por um número pequeno de pessoas, na maior parte dos projetos este número não passando de dois. O relatório FLOSS ainda vincula a dimensão do código ao número de desenvolvedores, um resultado esperado tendo em vista os ganhos de escala que projetos grandes parecem apresentar.

Fazendo uma retrospectiva, é interessante perceber que os estudos empíricos que avaliam a autoria e composição dos projetos — Krishnamurthy, Orbiten, FLOSS — tendem a confirmar uma propriedade peculiar: que o número de autores de cada pacote é bastante reduzido, mas que o número de pacotes em si é muito grande.

5.3.7 Demografia

Há alguns estudos importantes sobre a demografia da comunidade de software livre. Pelo fato de despertar curiosidade sobre suas motivações, procedência e experiência, desenvolvedores de projetos de software livre já foram analisados quantitativamente em algumas pesquisas empíricas. Esta seção apresenta dois destes estudos.

BCG Hacker Survey

Um dos principais estudos demográficos realizados sobre a população de participantes em projetos de software livre é o de Karim Lakhani e Bob Wolf, do Boston Consulting Group (BCG), intitulado ‘The Boston Consulting Group Hacker Survey’ [8]. O estudo se baseou em uma população de líderes de projeto e desenvolvedores principais, registrados no site de hospedagem de projetos `sourceforge.net`.

O levantamento foi feito em duas fases, com 536 e 169 respostas individuais, respectivamente. A primeira fase avaliou a população de desenvolvedores de uma seleção aleatória dos projetos, e a segunda apenas os desenvolvedores provindos de projetos classificados como já maduros. O estudo produziu diversas conclusões importantes com relação às motivações e propriedades do grupo como um todo:

- Os desenvolvedores apreciam de fato o trabalho que desenvolvem. Por exemplo, 72% dos entrevistados disseram que quando estão programando, perdem a noção do tempo por estarem muito entretidos, e tempo de sono foi considerada a contrapartida mais significativa resultante da

participação em um projeto de software livre.

- A maior parte dos participantes (70% do total) eram voluntários e não seriam financeiramente recompensados pelo seu trabalho.
- A idade dos participantes se concentra na faixa de 22 a 38 anos, a média ficando em torno de 30 anos.
- Confirmando o trabalho de Dempsey et al. descrito na seção anterior, a população de desenvolvedores da pesquisa se mostrou ser realmente global, provindo de dezenas de países, e sendo representada principalmente por desenvolvedores dos EUA, Alemanha, Reino Unido, Austrália e Canadá.
- Outro resultado importante é que 45% dos participantes eram programadores profissionais, e que a média de experiência com programação do grupo analisado era de 11 anos.
- A visão de gerência transmitida pelos participantes também é original. Afirmaram esperar do líder do projeto a realização de tarefas práticas: criação da base de código original, contribuições de código constantes, uma boa visão a longo prazo, iniciativa para diálogo, e disposição para integrar contribuições. Em contrapartida, enfatizaram em menor grau tarefas como alocação e gerência de cronograma, motivação dos desenvolvedores, e apoio à entrada de novos membros.

FLOSS Survey of Developers

Outro relatório produzido sobre o estudo FLOSS, descrito na Seção 5.3.6, detalha os resultados de um extenso levantamento quantitativo realizado em uma população de 2784 desenvolvedores [6]. Com base em um questionário divulgado de forma indireta (através de listas de discussão e sites Web), este trabalho produziu um conjunto de medidas a respeito das características pessoais e profissionais desta população. Entre os resultados interessantes, está um perfil do desenvolvedor:

- Homens constituem 98.9% dos entrevistados.
- 65% da população possui menos de 30 anos, a distribuição concentrando-se em torno de 27 anos. Esta medida se assemelha aos resultados da pesquisa BCG, na qual a média era de 30 anos. É possível que a média um pouco superior reflita o fato da pesquisa BCG ter sido veiculada a uma população de líderes e desenvolvedores principais.
- 70% são formados, com curso superior completo; deste total, 28% possui mestrado e 9% doutorado.
- 65% são empregados e 14% autônomos, a maioria dos desenvolvedores trabalhando em áreas relacionadas à computação, e outra parcela significante sendo estudantes.

Destas medidas pode-se obter um cenário pessoal interessante da população de desenvolvedores. Os dados relativos à sua participação em projetos de software livre também merecem destaque: a maior parte dos desenvolvedores passa pouco tempo (até 10 horas por semana) trabalhando em projetos de software livre, e tendem a contribuir para projetos dos domínios Redes, serviços Web e produtividade pessoal. Grande parte dos entrevistados afirmou trabalhar em até 5 projetos simultaneamente, a maioria limitando-se apenas a 1.

Parte dos resultados do levantamento são inesperados e merecem uma análise particular. Por exemplo, a distribuição das nacionalidades dos participantes do levantamento é significativamente européia. A maior parte provindo da França (15%), EUA (12%), Alemanha (12%), Itália (8%), Reino Unido (6%) e Holanda (6.5%). Embora não difira totalmente de outros levantamentos desta natureza, a baixa participação dos EUA, Suécia e Austrália parece indicar uma tendência do questionário a concentrar-se em certos países europeus, talvez pela natureza da sua forma de divulgação.

Os resultados do estudo FLOSS apontam para uma população jovem e altamente educada, ativa economicamente e, como esperado, concentrando-se nas áreas profissionais compostas na sua maioria por desenvolvedores. A reduzida participação em projetos de software livre pode representar uma confirmação dos resultados da pesquisa de Mockus et al. (Seção 5.2.1) e dos trabalhos Orbiten e FLOSS (Seção 5.3.6): que grande parte dos projetos terá seu código escrito por um número pequeno de pessoas, e que uma parcela grande da população terá envolvimento esporádico e reduzido.

Capítulo 6

Caracterização do Processo de Software

6.1 Desafios

Este trabalho, como descrito na introdução, se propõe a levantar e quantificar aspectos essenciais do processo de software utilizado nos projetos de software livre. Apresenta uma série de desafios do ponto de vista da pesquisa:

- **O pouco conhecimento *prático* existente sobre o processo de desenvolvimento realizado em projetos de software livre.** Embora muitos softwares livres sejam amplamente conhecidos e utilizados, seus processos de desenvolvimento são ainda obscuros para boa parte da comunidade científica. Por exemplo, sobrevive o estigma de que o Linux é um sistema operacional de "desenvolvimento aberto e caótico", onde "qualquer um implementa o que quer"; no entanto, na literatura [38, 36] e no espaço de discussão público do projeto estão documentados alguns aspectos do processo pelo qual é desenvolvido. Sugerem algo distinto de um projeto descontrolado.

  Um dos objetivos deste trabalho foi acumular experiência prática e descrever melhor o que de fato ocorre durante o desenvolvimento desses projetos. Com base nesta experiência prática tornou-se possível guiar um levantamento de dados quantitativo que pudesse efetivamente estabelecer a presença e intensidade do processo em questão.

- **A incerteza em relação à população total de projetos de software livre.** Não há como aferir o tamanho total da população de projetos de software livre, uma vez que podem ser iniciados e divulgados independentemente — não há necessidade de registrar o projeto se o autor não o desejar. No entanto, existem alguns sites Web que mantêm índices públicos de projetos, e que permitem que seja avaliado o número de projetos *mainstream* (em outras palavras, mais conhecidos). Destes sites, é possível garantir que a população de projetos esteja na ordem dos milhares (uma parcela destes sendo projetos inativos ou abandonados; veja `unmaintained.sourceforge.net`).

- **A inexistência de uma instituição formal que represente o projeto.** O projeto é um conjunto

independente de desenvolvedores e contribuidores; muitas vezes não há nenhuma formalização além de um acordo verbal de interesse entre os membros do projeto. Para levantar dados diretamente do projeto (sem utilizar outras fontes, como a bibliografia), é necessário identificar uma pessoa de contato.

Além disso, não há forma de estabelecer este contato que não seja por meio de solicitação direta e pessoal. Esta limitação contrasta diretamente com levantamentos realizados em empresas, que oferecem pontos de contato bem-documentados, e cuja hierarquia e organização permitem que sejam feitas solicitações formais para participação em uma pesquisa. Em um projeto de software livre, o pesquisador conta primordialmente com a boa vontade da pessoa de contato.

- **A natureza independente e não-uniforme dos projetos e de seus produtos.** Cada projeto de software livre possui sua forma de organização, suas regras internas, seus meios de contato, seus diferentes artefatos, e diferentes formas de organizar e oferecer estes artefatos. Este fato dificulta qualquer processo de padronização dos projetos, seja por questionário, seja por um levantamento sistemático dos artefatos, ambos executados neste trabalho.

- **A incerteza com relação ao conhecimento geral dos conceitos de engenharia de software.** Como cada projeto é freqüentemente formado por voluntários interessados no seu desenvolvimento, é impossível garantir que estes voluntários tenham conhecimento na área de engenharia de software – nem mesmo é possível garantir seu grau de instrução em computação ou informática. Isto torna essencial que qualquer questão relacionada aos termos técnicos da engenharia de software seja abordada de forma clara e detalhada.

- **O uso exclusivo do idioma inglês na literatura e comunicação entre pessoas da comunidade de software livre.** Embora seja freqüentemente ignorado, a não-utilização do idioma português para o desenvolvimento de software livre representa uma barreira no nosso país tanto para participantes em potencial quanto para pesquisadores interessados em entender o seu processo de software. De certa forma, boa parte do conhecimento em toda a área da Computação é consolidado em inglês; no entanto, como a comunidade de software livre é realmente internacional, não possuindo identidade geográfica ou cultural uniforme, torna-se imperativo o uso deste idioma.

Durante a execução deste projeto de pesquisa, a metodologia a ser utilizada foi refinada continuamente. Cada desafio verificado resultou em uma reavaliação da proposta inicial e em uma análise das possíveis estratégias para sua superação.

6.2 Refinando o Problema de Pesquisa

Como visto no capítulo anterior, conhecemos elementos importantes do processo de software realizado em alguns projetos de software livre específicos através de trabalhos descritivos. Dos levantamentos

quantitativos realizados também é possível obter características da população de desenvolvedores e do código-fonte dos softwares. No entanto, esta perspectiva 'caixa-preta' não permite que se verifique a presença ou a intensidade das atividades do processo de software utilizado.

Na introdução, estabelecemos um problema de pesquisa central. Este problema de pesquisa pode ser refinado nas seguintes questões:

I. **De que forma os projetos de software livre trabalham para produzir software?** Em outras palavras, qual é o processo de software realizado nos projetos, e quais atividades o compõem?

II. **Existe um processo comum entre os projetos de software livre?** Levando-se em consideração a diversidade apontada entre os projetos, é pouco provável que o processo aplicado seja uniforme, mas é possível que exista um conjunto de atividades mínimo que é comum à maioria dos projetos.

III. **De que forma estes processos diferem do processo de software convencional documentado na literatura de engenharia de software?**

Estas questões sintetizam os aspectos essenciais deste projeto de mestrado. Delas podem ser derivadas um número de perguntas menores, de natureza mais específica, relacionadas aos aspectos individuais da engenharia de software. Estes problemas de pesquisa derivados incluem os seguintes:

P1. Que tipos de software são gerados pelos projetos?

P2. Que tipo de trabalho fazem os membros das equipes?

P3. Como se dá o processo de engenharia de requisitos?

P4. Como são realizadas as atividades de projeto?

P5. Qual é o fluxo de trabalho do processo de alteração e integração de código?

P6. Quais são as atividades de documentação importantes?

P7. Que atividades de garantia de qualidade são realizadas?

P8. De que tamanho são as equipes que trabalham nos projetos?

P9. Que idades têm os diferentes projetos?

P10. Que formas de liderança são aplicadas?

P11. Que tipos de ferramentas são usadas?

P12. Existe relação entre o tamanho das equipes e o processo de software?

P13. Existe relação entre a idade do projeto e o processo de software?

6.3 Visão Geral da Metodologia

Para realizar o levantamento e abordar os problemas de pesquisa descritos, foi estabelecida uma linha de ação, levando em consideração os desafios apontados no início do capítulo. As seguintes etapas foram selecionadas:

1. **Revisão bibliográfica**; levantamento da literatura pré-existente a respeito de projetos de software livre e processo de software.

2. **Participação ativa**; participação prática em um conjunto de projetos de software livre com o objetivo de acumular experiência e formar relacionamentos com membros importantes da comunidade que pudessem auxiliar posteriormente no processo de pesquisa.

3. **Projeto de questionário**; detalhamento e implementação da ferramenta a ser usada na avaliação quantitativa dos projetos de software livre.

4. **Veiculação do questionário**; seleção da população de projetos, solicitação e comunicação com os respondentes (ou **participantes**), e tabulação dos resultados obtidos.

5. **Análise** do questionário, e síntese dos resultados.

Esta escolha de etapas foi feita com o objetivo de gradativamente acumular uma base de conhecimento a respeito do processo de software em questão. Nas seções seguintes são discutidas estas etapas em maior detalhe.

6.4 Revisão Bibliográfica

O conhecimento a respeito do domínio do problema, descrito nos capítulos anteriores, foi levantado em grande parte através da revisão bibliográfica realizada ao longo do trabalho, em conjunto com experiência prévia na área, expandida através da participação ativa em projetos de software livre.

É interessante observar que houve publicação de um grande número de trabalhos a respeito de software livre nos últimos três anos. No início deste projeto de mestrado, havia um pequeno corpo de literatura essencial, e poucos projetos de pesquisa anunciados ou em execução; ao longo deste período, um grande número de levantamentos empíricos e trabalhos teóricos foi publicado. Este fato comprova a importância crescente do assunto na comunidade científica.

Esta dissertação buscou levar em consideração a grande maioria dos trabalhos disponíveis na literatura da área que tratassem deste assunto, e mesmo publicações bastante recentes foram analisadas e resumidas. O Capítulo 5 apresentou um *survey* dos trabalhos mais importantes na área.

6.5 Participação Ativa

Um dos aspectos inovadores deste trabalho foi a decisão de utilizar a técnica de observação participante para levantar experiência prática a respeito do funcionamento de alguns projetos de software livre. Nos últimos 24 meses, estive envolvido com diversos projetos de software livre, participando ativamente através de discussão em listas de correio eletrônico, submissão de alterações de código-fonte, e outras atividades comuns entre os projetos.

A minha experiência prévia com projetos de software livre, e a afinidade com a minha atividade profissional, tornaram a técnica de observação participante uma opção relevante. Esta técnica permite ao pesquisador vivenciar o processo de construção de software e obter um entendimento mais íntimo das relações interpessoais e do fluxo de trabalho dentro do grupo de estudo [68]. Para projetos de software livre, é particularmente interessante esta abordagem por dois motivos:

1. **A barreira de ingresso em um projeto de software livre é facilmente transposta.** Contribuir quantidades significativas de código (ou tornar-se um participante conhecido e acreditado) é uma conquista alcançada ao longo de um período de participação, e não deve ser uma pretensão de curto prazo. No entanto, assumir a posição de um usuário ativo, e gradativamente escolher tarefas que estejam ao seu alcance é algo perfeitamente factível. A maior parte das pessoas envolvidas nos projetos tem grande interesse em receber qualquer auxílio, e é aceitável contribuir quanto tempo ou esforço lhe for conveniente.

2. **As contribuições realizadas durante o processo de observação participante consistem em benefícios concretos para o projeto e seu software.** Como conseqüência direta do meu trabalho nos projetos dos quais participei, é possível apreciar diversos efeitos positivos: melhorias no código, implementação de nova funcionalidade, auxílio a outros usuários, escrita de documentação e diversas outras atividades esporádicas.

O trabalho de participação ativa neste projeto de mestrado envolveu uma série de projetos de software livre, concentrando-se em quatro projetos particulares, com contribuições de caráter distinto em cada um deles. Nas seções seguintes é descrito em ordem cronológica o trabalho realizado em cada um deles. No próximo capítulo são discutidos os resultados desta participação.

6.5.1 Mozilla

O projeto Mozilla (`www.mozilla.org`) foi descrito na Seção 4.4.3; é um projeto de software livre que se dedica à criação de um navegador Web e de uma plataforma de desenvolvimento de aplicações. A tela principal do navegador Web Mozilla é apresentada na Figura 6.1. Algumas observações de destaque sobre o Mozilla seguem:

- É um dos projetos de maior porte; centenas de desenvolvedores e milhares de contribuidores ocasionais gravitam ao redor do projeto. Sua base de código soma mais de dois milhões de linhas de código, o que o coloca entre os grandes projetos de software livre, ao lado do núcleo Linux e do ambiente gráfico XFree86 [64].

- Produz um dos softwares mais importantes para a comunidade de software livre: um navegador Web com suporte a um conjunto muito grande de plataformas de hardware e sistema operacionais, visando apoio completo aos padrões definidos pelo W3C.

- Tem apoio de uma série de empresas renomadas, incluindo a Netscape/AOL (que fundou o projeto), a Sun Microsystems, e a IBM. Este aspecto é pouco usual entre projetos de software livre.

- É um projeto que investe no seu processo de software, documentando-o, e utilizando uma série de ferramentas desenvolvidas internamente para apoiá-lo.

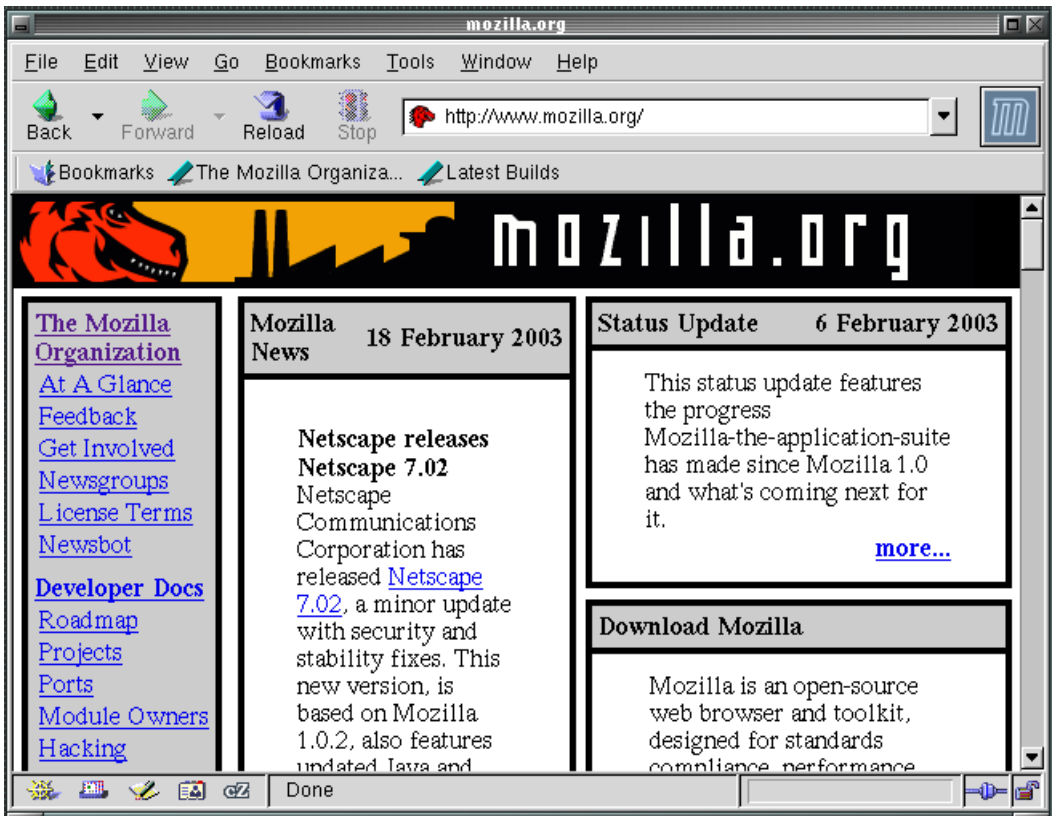


Figura 6.1: Exemplo de tela do navegador Web Mozilla, um dos projetos escolhidos neste trabalho para observação participante. O Mozilla é um dos mais importantes projetos de software livre, tendo grande porte, amplo apoio institucional e um processo de software bem definido.

Por este conjunto de propriedades, o Mozilla foi escolhido como o primeiro projeto a ser estudado. O trabalho executado no projeto Mozilla teve como objetivo entender melhor as relações entre os mem-

bros da equipe do projeto, suas subdivisões, e de que forma se dá a gerência de um projeto de software livre fortemente apoiado por uma empresa.

6.5.2 Bugzilla

O projeto Mozilla investiu grande esforço na criação de ferramentas para apoiar seu processo de desenvolvimento. Em parte isso deriva da herança proprietária do projeto (que já utilizava um conjunto de ferramentas de apoio ao processo de desenvolvimento [69]), e do interesse da Netscape/AOL em obter uma certa medida de garantia do seu projeto de software livre piloto. As ferramentas foram especificamente elaboradas para apoiar desenvolvimento distribuído de uma grande base de código, algumas sendo adaptadas de outras ferramentas já existentes (no caso do LXR e do sistema `autoconf/automake` utilizado na compilação), e outras sendo utilizadas sem alteração alguma (o CVS, por exemplo).

A ferramenta que tem maior importância para o projeto é o sistema de controle de alterações Bugzilla (`www.bugzilla.org`). É utilizado diariamente por centenas de pessoas em todas as etapas do processo de software do projeto Mozilla, desde definição de funcionalidade até projeto, implementação e revisão de alterações. Um registro (ou 'bug') do Bugzilla está exposto na Figura 6.2.

Este projeto foi escolhido pelo fato de ser uma ferramenta de engenharia de software importante para diversos projetos de software livre. Compreender seu funcionamento e ter contato direto com seus usuários permitiria uma boa compreensão dos mecanismos de participação da comunidade. Pela experiência que obtive como usuário do Bugzilla (enquanto participando do projeto Mozilla), e por ser escrita principalmente em Perl, HTML e Javascript (é uma aplicação Web), seria viável contribuir efetivamente com código para a ferramenta.

6.5.3 PyGTK

Parte do trabalho desenvolvido na Async Open Source – empresa onde trabalho – envolve o uso da linguagem Python em conjunto com uma biblioteca de componentes gráficos para a criação de aplicações comerciais. Esta biblioteca se chama PyGTK , e adapta a biblioteca gráfica GTK+ à linguagem Python[1].

Este projeto possui características bastante diferentes dos projetos anteriormente estudados. Primeiro, a comunidade de usuários é bem mais restrita. Segundo, a biblioteca tem funcionalidade bastante delimitada, tanto pela implementação da biblioteca GTK+, quanto pela semântica e padrões da linguagem Python. Embora muitas contribuições tivessem sido feitas por outros, a maior parte do código tinha sido escrita por uma única pessoa, e menos de quatro pessoas contribuem com alguma regularidade para o projeto.

Uma deficiência séria do projeto PyGTK é a virtual ausência de documentação; pelo fato de grande

[1]Python: `www.python.org` PyGTK: `www.daa.com.au/~james/pygtk/` GTK+: `www.gtk.org`

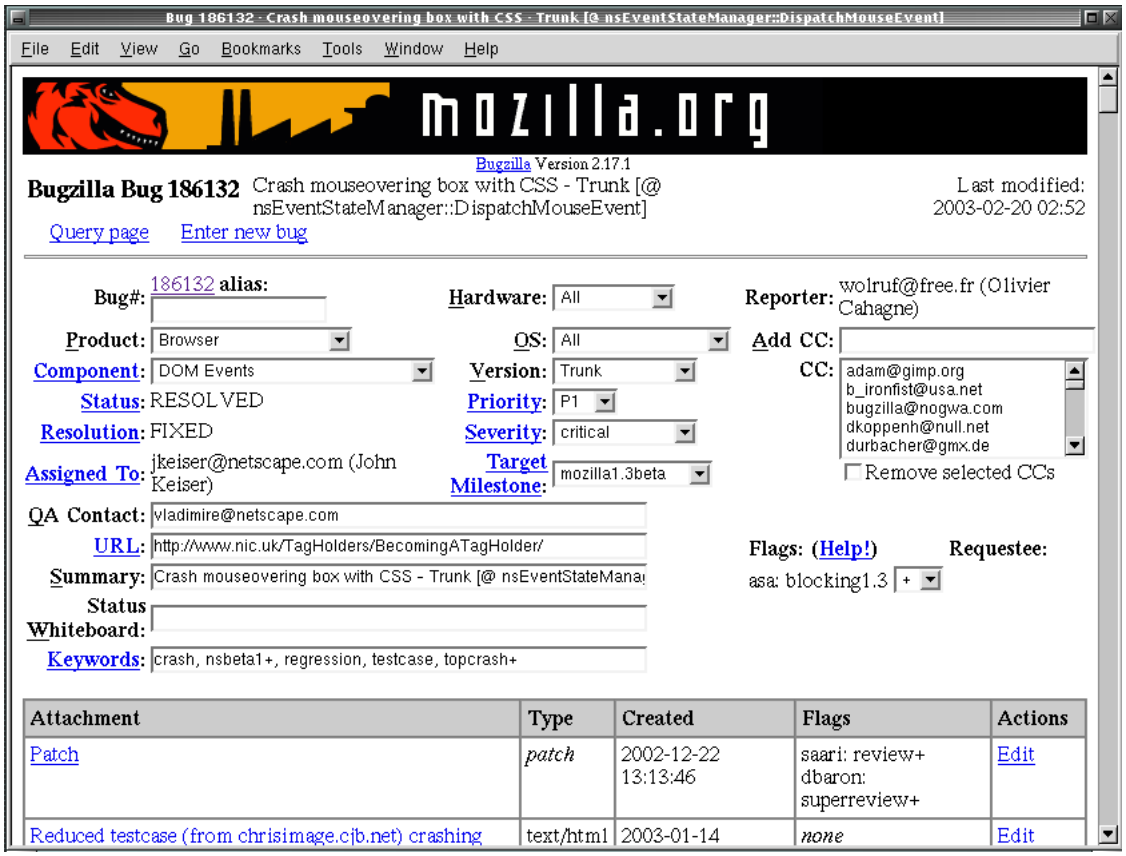


Figura 6.2: Exemplo de tela do Bugzilla, a base de informes de erro utilizada em diversos projetos de software livre, sendo ele mesmo um projeto de software livre. O Bugzilla é utilizado diariamente por milhares de desenvolvedores em empresas e outras instituições no mundo todo, grande parte envolvida no desenvolvimento de software livre.

parte do projeto ser um simples mapeamento entre linguagens, pouca ênfase se deu a esta atividade. Além disso, a API[2] extensa dificulta consideravelmente seu uso, tendo muitas vezes semântica pouco clara ou bastante diferente da implementação do GTK+ em C.

Para remediar esta situação, além de prover auxílio a outros usuários nas listas de discussão do projeto, trabalhei elaborando um documento FAQ (*Frequently Asked Questions*) utilizando uma ferramenta também escrita em Python, chamada `faqwiz`. Fiz também contribuições de código para reparar pequenos problemas e completar funções da API GTK+ ainda não mapeadas.

6.5.4 Kiwi

O último projeto estudado foi iniciado em 2001 para apoiar um projeto interno da Async Open Source, que necessitava de um *framework* para apoiar o desenvolvimento de aplicações gráficas usando o PyGTK. Este projeto é intitulado Kiwi [70]. A experiência com o Kiwi é diferente dos projetos anteri-

[2] *Application Programming Interface*, o conjunto de funções e classes exportadas por uma biblioteca.

ores por ser de minha autoria. Foi escrita integralmente neste período, e é completamente original; não inclui código-fonte proveniente de nenhuma outra aplicação. O objetivo da biblioteca é permitir que seja possível escrever aplicações com interface gráfica em Python com mínimo esforço e alto reuso. A Figura 6.3 apresenta um exemplo desenvolvido usando PyGTK e Kiwi.

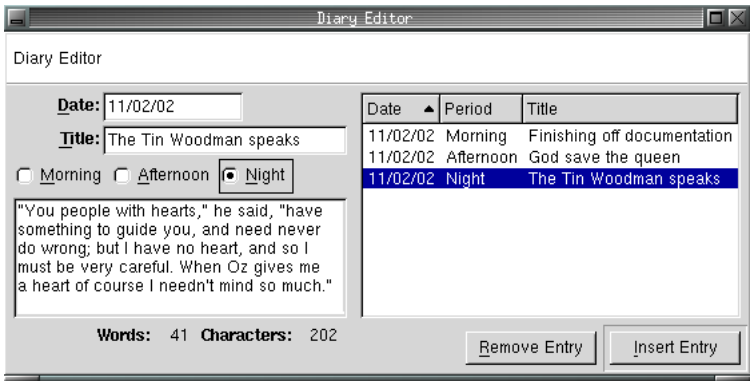


Figura 6.3: Exemplo de aplicação escrita utilizando PyGTK e Kiwi, respectivamente o terceiro e quarto projetos estudados como observador participante. O PyGTK difere dos projetos anteriores por ser menor, e por ter requisitos bastante delimitados pelo fato de ser uma biblioteca que serve de elo entre uma implementação em C e a linguagem Python. Kiwi é um framework completo para desenvolvimento de aplicações gráficas.

A biblioteca é descrita em [70], contando com documentação completa e um grande número de exemplos. Um histórico do projeto é fornecido na Seção 7.4.1.

6.6 Projeto do Questionário

No projeto de pesquisa avaliado para o exame de qualificação deste trabalho de mestrado, foi estabelecido que seria realizado um levantamento empírico, sendo utilizado um questionário para avaliar os projetos de software livre. Na fase inicial do projeto ficou evidente, no entanto, que existiam ainda grandes incertezas relacionadas ao processo de software em questão; estas incertezas acarretavam risco em produzir um levantamento inconsistente ou de pouco valor. Seria também bastante difícil decidir quais questões e alternativas seriam apropriadas.

Dado que levantamentos anteriores aplicados à mesma população já haviam evidenciado a necessidade de utilizar linguagem e formato aceitáveis para esta população qualificada e crítica, escolheu-se adiar este processo de levantamento e enriquecer o trabalho com a participação ativa descrita na seção anterior. Esta experiência se mostrou fundamental para uma melhor compreensão do problema de pesquisa, e para a elaboração das hipóteses experimentais a ser avaliadas.

6.6.1 Hipóteses Experimentais

Com base na literatura, identificou-se um número de hipóteses enunciadas a respeito do processo de desenvolvimento de software livre; alguns exemplos já foram citados na seção anterior[3]. Para expandir estas hipóteses em um conjunto adequado ao objetivo deste trabalho, foi adotada a seguinte técnica: **sintetizar os problemas de pesquisa da Seção 6.2, buscando paralelos na literatura e codificando-os em termos compreensíveis para a população em questão.**

A primeira hipótese provém de uma 'verdade assumida' em todas as publicações, sejam científicas ou informais, e confirmada em parte pelos levantamentos descritos em [8, 60]:

H1. O trabalho é realizado por equipes geograficamente dispersas.

O segundo grupo de hipóteses a (H2 a H7) foi coletado de trabalhos descritivos anteriormente realizados [32, 46, 54]. Este grupo descreve aspectos considerados fundamentais para caracterizar um projeto de software livre. Em particular, a hipótese 5 incorpora a questão essencial do fluxo de trabalho modelado através do CVS, levantada por André van der Hoek e descrita na Seção 5.3.4.

H2. O desenvolvedor do software é também seu usuário e contribui efetivamente para determinar grande parte de sua funcionalidade.

H3. Todo projeto utiliza um sistema de liderança particular para coordenar e arbitrar disputas, existindo duas variantes principais: o 'ditador benevolente' de Raymond, e o grupo central (ou *core*) de desenvolvedores (exemplificados pelos projetos Apache e FreeBSD, discutidos na Seção 5.2).

H4. Há uso amplo de ferramentas para viabilizar a comunicação entre a equipe geograficamente dispersa.

H5. Há uso amplo de ferramentas de controle de versão, em particular do CVS.

H6. A maior parte dos projetos possui equipe pequena, e a média do número de indivíduos por equipe não passa de 5.

H7. A maior parte das equipes possui membros com mais de 5 anos de experiência em desenvolvimento de software.

[3] Uma observação merece ser feita a respeito das hipóteses. Na época em que foram elaboradas, eu ainda não havia tomado contato com o trabalho de Krishnamurthy, que descrevo na Seção 5.1.3. É interessante observar como ele discute algumas hipóteses muito similares às descritas nesta seção. Esta coincidência permite explorar uma certa consistência experimental entre os dois trabalhos.

Da experiência que obtive com a participação ativa descrita anteriormente, elaborei um terceiro conjunto de explicações possíveis (H8 a H13) para descrever aspectos adicionais dos projetos. Estas hipóteses foram adaptadas de acordo com o objetivo inicial de detalhar as atividades e recursos do processo de software, incluindo engenharia de requisitos, dimensão e habilidade da equipe, documentação, e garantia de qualidade.

H8. O trabalho de engenharia de requisitos é freqüentemente facilitado por levantamentos anteriores feitos por outras equipes, seja através de padrões estabelecidos e/ou documentados, seja através de código-fonte herdado de outro projeto.

H9. Existe um compromisso pessoal por parte dos membros da equipe do projeto em garantir a qualidade do software lançado.

H10. Existe uma forte política de controle e revisão relacionada à aceitação, integração e auditoria de contribuições de código.

H11. O tamanho da base de código do projeto está diretamente relacionado à dimensão da equipe e ao tempo de existência do projeto.

H12. A equipe do projeto tende a crescer com o tempo.

H13. Pouca atenção é dada à usabilidade do software.

Decidiu-se incluir uma hipótese (H13) sobre usabilidade: no projeto Mozilla, e posteriormente no PyGTK, me chamou a atenção o fato da atividade de engenharia de usabilidade ainda ser muito incipiente entre projetos de software livre. Este aspecto também é descrito por Matthew Thomas em seu célebre trabalho ‘Why Free Software usability tends to suck’, e por outros pesquisadores da área [71, 72, 73].

Este conjunto de hipóteses cobre boa parte do escopo intencionado para este trabalho. Relacionadoas aos problemas de pesquisa expostos na Seção 6.2 é possível observar algumas omissões importantes, que são justificadas a seguir.

- Não há hipótese associada ao tipo de software produzido (problema P1). Existe um consenso implícito de que software livre se concentra em domínios onde o desenvolvedor de software é um usuário; esta afirmação já está vinculada à hipótese H2. Além disso, através do questionário seria possível obter uma distribuição completa dos domínios de aplicação.

- Não há hipótese associada à atividade de projeto arquitetural ou detalhado (problema P4). Uma particularidade que observei nos projetos dos quais participei é que a atividade de projetar software é muito difícil de ser elicitada. É possível que em parte isso se deva ao fato de software

livre, por estar permanentemente em manutenção, possuir projeto já estabelecido e implementado. Pode também derivar do fato do trabalho de desenvolvimento ser feito isoladamente, apenas sendo comunicado publicamente a alteração ao código-fonte resultante.

Existe alguma documentação de projeto dos softwares livres de maior porte, mas raramente é detalhada, e por isso tampouco seria possível avaliar este processo através de seus artefatos. Dada a dificuldade de obter medidas confiáveis a respeito desta atividade, optei por excluí-la da lista de hipóteses. Paul Vixie em [74] faz uma colocação informal mas pertinente sobre projeto:

> 'There usually just is no system-level design for an unfunded Open Source effort. Either the system design is implicit [...] or it evolves over time (like the software itself). Usually by Version 2 or 3 of an open-source system, there actually is a system design even if it doesn't get written down anywhere[4].'

- Não há hipótese associada à atividade de documentação (problema P6). Dada a variedade de documentos possíveis, e conhecendo os esforços dos diversos projetos de documentação independentes, seria mais indicado utilizar o questionário para determinar que tipos de documentação são considerados importantes pela comunidade de software livre.

Descritas as hipóteses que busco discutir, e comparadas estas com os problemas de pesquisa, na próxima seção é apresentada a forma de avaliação principal utilizada neste experimento – o questionário.

6.6.2 Visão Geral do Questionário

Para obter dados para discutir as hipóteses apresentadas, existiam algumas opções: levantamento de dados através de informações existentes em sites Web e outros documentos dos projetos; análise dos pacotes de código-fonte produzidos; estudos de caso isolados; entrevistas com desenvolvedores importantes. No entanto, o objetivo de sintetizar informações de uma grande população de projetos motivou a aplicação de um processo multi-modos, sendo realizada a coleta principal de dados por meio de um questionário.

O questionário seria veiculado a pessoas que tivessem bom conhecimento do fluxo de trabalho e da equipe de um projeto de software livre específico; preferencialmente, ao *mantenedor responsável pelo projeto*. Além do questionário, seria feita uma análise do pacote de código-fonte do projeto, obtendo um segundo conjunto de dados quantitativos que poderia ser comparado aos resultados do questionário.

[4] 'Geralmente não há projeto arquitetural para um projeto de software livre que não seja patrocinado por uma empresa. Ou o projeto é implícito, ou evolui com o tempo (como o próprio software). Geralmente por volta da versão 2 ou 3 de um sistema de software livre já há um projeto arquitetural, mesmo que não tenha sido documentado em nenhum lugar.'

6.6.3 Requisitos

A natureza geograficamente distribuída dos projetos de software livre torna atrativo o uso dos formatos Web e correio eletrônico para oferecer o questionário. Existe literatura ampla descrevendo as particularidades que devem ser observadas quando se trabalhando com questionários online [75, 76, 77], e uma síntese das recomendações principais segue:

- O projeto do questionário deve levar em consideração a impossibilidade de determinar qual tipo de software será usado para acessá-lo; deve, portanto, ser projetado para ser usado com uma grande variedade de agentes de usuário (navegadores Web, ferramentas de correio eletrônico). Este aspecto é especialmente importante levando em consideração o fato da população em questão exibir forte opinião em favor da padronização e acessibilidade[5].

  O questionário foi construído fazendo uso exclusivo de tecnologias padronizadas (basicamente texto e HTML), utilizadas de forma a permitir visualização com agentes mais antigos. Quando utilizados recursos mais modernos (o uso do elemento `<LABEL>` para facilitar o acesso às opções do questionário, ou de CSS e DOM para a ajuda online, por exemplo) testes extensivos garantiram que não haveriam problemas em navegadores que não os suportassem.

- Deve-se assumir que parte dos usuários possui conexão com a Internet a baixa velocidade. Levando este aspecto em consideração, o questionário foi elaborado sem o uso de imagens, e a maior página, que carrega o questionário propriamente dito, tem 38 KBytes no total.

- O questionário deve ter formato compacto, reduzindo o tempo necessário para seu preenchimento. Este fator se mostra crítico para obter boa relação de resposta para o questionário Web.

- Devem ser oferecidos múltiplos formatos, permitindo ao participante optar pelo que lhe for mais conveniente. Os formatos Web e correio eletrônico são alternativas aceitáveis dada a característica de dispersão geográfica da população em questão.

- Deve-se evitar utilizar perguntas abertas (como ‘Qual é sua ocupação?’), pela característica que têm de proporcionar respostas vagas ou incompletas, e por demandar maior tempo de resposta. No entanto, para cada questão deve ser oferecida uma opção personalizada (‘`Outro:`’) e um espaço para comentários.

A estas recomendações obtidas da literatura foi acrescentado um requisito adicional que julguei importante pela avaliação dos questionários anteriores descritos na literatura:

- A linguagem deve ser acessível a um conjunto grande de desenvolvedores, evitando o uso de termos técnicos — por exemplo, termos particulares do processo de software — que pudessem ser desconhecidos ou mal interpretados.

[5] Para exemplos, ver `www.anybrowser.org` e `www.webstandards.org`

6.6.4 Implementação do Questionário

A necessidade de obter alta flexibilidade e usabilidade, e a experiência prévia com a elaboração de aplicações Web usando PHP me motivou a não reutilizar uma solução pronta para questionários, das quais algumas existem (`mod_survey` e `phpESP` sendo dois exemplos de ferramentas de questionário disponíveis como software livre). Esta escolha permitiu alta flexibilidade e total liberdade para manipular os dados e as interfaces, mas consumiu esforço considerável para sua construção.

A elaboração do questionário se mostrou uma tarefa desafiadora. Além da dificuldade inerente em preparar uma ferramenta para avaliação online, o tema do questionário — o processo de software — é amplo o suficiente para garantir uma oferta inesgotável de pontos para avaliação. Levando este aspecto em consideração, a ferramenta foi projetada de forma a permitir alterar tanto as perguntas quanto as alternativas sem impacto significativo no código-fonte ou na base de dados. A versão texto, para veiculação por correio eletrônico, foi gerada a partir do questionário Web e mantida manualmente.

Um ponto importante a destacar é o fato de toda a comunicação escrita com os indivíduos participantes do questionário ter sido feita *em inglês*. Como citado no início do capítulo, o uso do inglês é imperativo dada a natureza geograficamente distribuída dos projetos em questão; essencial, portanto, foi conhecimento do idioma inglês e a revisão oferecida por terceiros durante a pesquisa.

Software Utilizado

Partiu-se da suposição que mais desenvolvedores mostrariam preferência pelo uso do questionário Web. Além disso, alguma infra-estrutura para armazenar as respostas haveria de existir. Por estes motivos, optou-se por utilizar uma aplicação Web para coletar e armazenar as informações obtidas do questionário.

Quando realizando uma pesquisa sobre projetos de software livre, é razoável obedecer à restrição de utilizar apenas software livre para sua realização; se por nenhum outro motivo, pelo fato de permitir uma avaliação subjetiva da qualidade das ferramentas utilizadas. Para a elaboração do questionário foram utilizados os seguintes softwares:

Servidor Web: utilizou-se o servidor Apache (`www.apache.org/httpd`), apresentado anteriormente na Seção 4.4.2.

Servidor de Base de Dados: o sistema gerenciador de banco de dados (SGBD) PostgreSQL (`www.postgresql.org`) foi utilizado para persistir as informações coletadas e sintetizar os dados durante a análise. Este SGBD suporta tipos complexos, incluindo listas (ou arranjos) de valores, o que foi essencial para implementar um sistema flexível de questões e respostas: não é necessário alterar o esquema da base de dados para modificá-las.

Linguagem de Programação: foi utilizado o PHP (`www.php.net`), uma linguagem interpretada desenvolvida especificamente para aplicações Web. PHP simplifica consideravelmente a tarefa de

desenvolver um questionário pelo fato de possuir tipos de alto nível (como listas e *hashes*), tipagem dinâmica, e facilidade de manipulação de formulários.

Geração de Gráficos: na análise dos dados do questionário, um conjunto grande de gráficos foi elaborado. Para sua criação, optei por utilizar uma biblioteca que pudesse manipular diretamente as informações disponíveis na base de dados: o JPGraph (`www.aditus.nu/jpgraph`), também escrito em PHP.

Outras ferramentas: Todo o código-fonte e material enviado aos participantes foi criado com o editor de texto VIM (`www.vim.org`). As páginas Web foram verificadas inúmeras vezes pela ferramenta de validação online do W3C (`validator.w3.org`).

No total, a ferramenta implementada para o questionário possui cerca de 4.000 linhas de código, com aproximadamente 40% deste total correspondendo a código PHP, e o restante sendo HTML, SQL, texto, comentários e documentação.

6.6.5 Fluxo de Trabalho

O questionário Web desenvolvido oferece um conjunto restrito de funcionalidades publicamente expostas. As páginas principais incluem:

- Uma página de apresentação, descrevendo a pesquisa, sua afiliação, e fornecendo um roteiro de participação.
- Uma página de informações adicionais sobre a pesquisa, incluindo motivações, um histórico e informações para contato.
- Um FAQ, descrevendo aspectos pertinentes do trabalho.
- Uma página de Ajuda, que descreve em detalhe as perguntas do questionário.
- A página de registro do projeto, que leva a uma série de páginas específicas ao preenchimento do questionário, descritas a seguir.

Este conjunto de páginas oferece um suporte mínimo ao visitante, e foi elaborado para esclarecer qualquer dúvida diretamente, sem a necessidade de intervenção do administrador. Tendo início na página de registro do projeto, a navegação pelo questionário segue os seguintes passos:

1. Registro do projeto: é solicitado ao participante o nome do projeto e o endereço de sua página Web. Inicialmente, se planejou o levantamento de forma a permitir que múltiplos questionários pudessem ser preenchidos para um mesmo projeto. Por dificultar o processo de análise de dados, abandonou-se esta idéia; no entanto, este passo adicional de registro permaneceu. Seria razoável

integrar este passo com o passo posterior, o que não foi feito neste trabalho de mestrado para evitar desestabilizar a ferramenta em uso.

2. **Registro do questionário:** o participante fornece seu endereço de correio eletrônico, que é usado posteriormente para enviar mensagens de confirmação, comprovando o correto processamento do questionário respondido. Note que não há autenticação alguma nesta fase; qualquer pessoa poderia se registrar e responder por qualquer projeto. Na prática, todos os questionários respondidos foram precedidos de contato por e-mail, e os endereços e respostas foram verificados; nenhum caso extraordinário foi identificado.

3. **Questionário:** nesta página, o participante tem acesso ao formulário contendo as perguntas e alternativas de resposta.

4. **Verificação:** ao submeter o questionário, é apresentada uma página de confirmação que inclui todas as respostas escolhidas. Esta página permite adicionalmente que o participante assinale o questionário preenchido como **privado**, indicando que respostas e comentários não deverão ser expostos publicamente.

5. **Agradecimento:** oferece ao usuário uma confirmação visual do recebimento do questionário, e inclui hiperlinks para as páginas de resultados.

Existe ainda a opção de retornar a um questionário já preenchido para alterar suas respostas. Esta funcionalidade foi acrescentada para permitir aos projetos com liderança múltipla delegar parte do preenchimento do questionário a pessoas diferentes, e está descrita na Figura 6.4 como o caminho B.

Não ficaram publicamente expostos os resultados durante este período; todos os hiperlinks para as páginas de resultados eram veiculados apenas ao participante, após este ter preenchido e verificado o questionário. Esta foi uma forma simples de evitar induzir as respostas dos novos questionários, também preservando a banda de conexão Internet do servidor.

Algumas ferramentas para visualização e análise dos resultados foram elaboradas durante o preenchimento do questionário e são descritas na Seção 6.8.1.

6.6.6 Conteúdo

Foram necessárias duas iterações para produzir o questionário aplicado, sendo feito um piloto simples ao final de cada uma. Esta seção discute o processo pelo qual foram escolhidos o formato e o conteúdo das questões e de suas alternativas.

Questionário 1: Longo demais, Vago demais

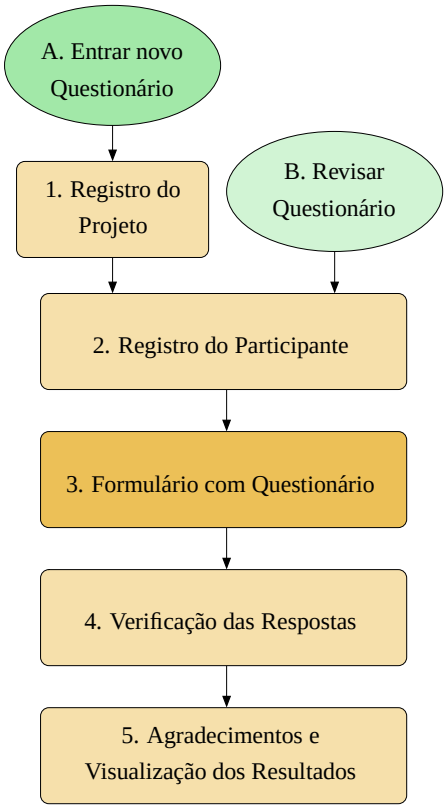


Figura 6.4: Diagrama das etapas a serem seguidas para preencher o questionário.

Com base nos requisitos descritos anteriormente, foi projetado o site Web que hospedaria o questionário, e estudou-se o formato a ser utilizado. Pelas recomendações anteriores e pelo tipo de hipótese a ser avaliada, ficou evidente que deveriam ser aplicadas perguntas objetivas, com opções de resposta pré-definidas. Além disso, para cada questão, seria oferecida uma alternativa personalizada (com o nome *'Other:'*) e um espaço livre para comentários. Alguns protótipos foram rapidamente esquematizados e avaliados, e optou-se por usar uma estrutura de seções, cada seção contendo um conjunto de perguntas seqüenciais.

Para projetar as perguntas individuais, foram analisadas as hipóteses experimentais e os problemas de pesquisa. Ao invés de codificar objetivamente as hipóteses em perguntas e alternativas, optou-se por utilizar uma seção de perguntas gerais, e um conjunto de seções cobrindo o processo de software, cada seção detalhando uma parte do ciclo de vida do projeto. Esta divisão em múltiplas seções para o processo de software merece uma discussão adicional:

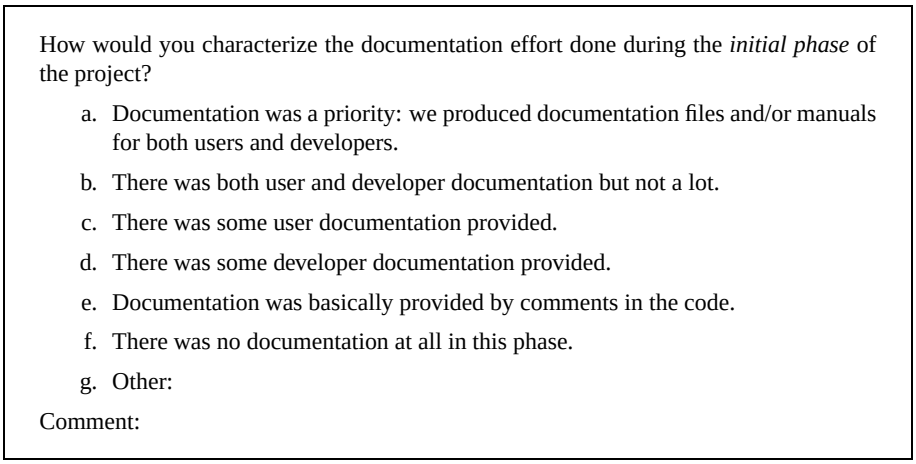
- Da literatura, e da experiência prévia com projetos de software livre, espera-se que estes tenham um ciclo de vida particular, caracterizado por um desenvolvimento inicial isolado (como Raymond mesmo afirma [32]) seguido de um lançamento público, quando o software pode então ser obtido e utilizado pela comunidade. Após o lançamento, o software passa a ser desenvolvido de maneira semelhante a um projeto convencional em fase de manutenção.

- É natural esperar que este evento, o lançamento, caracterize uma grande alteração no processo de software utilizado. Esta mudança certamente ocorreu no projeto Kiwi (o framework descrito na Seção 6.5.4), tanto no uso de ferramentas de desenvolvimento quanto na forma em que alterações no código eram avaliadas e implementadas[6].

- Concluiu-se desta premissa que seria importante avaliar as atividades e recursos do processo de forma separada, dedicando uma seção do questionário a cada fase do ciclo de vida do projeto. Em cada seção, se repetiria uma parte das questões relacionadas às atividades do processo, se alterando apenas a fase do ciclo de vida a ser avaliada (pré-lançamento, lançamento e pós-lançamento).

[6]Em retrospectiva, acredito que este cuidado adicional provém em grande parte do desejo de evitar efeitos desagradáveis à minha comunidade de usuários incipiente. Um histórico do projeto Kiwi é apresentado na Seção 7.4.1.

No total, o questionário produzido na primeira iteração tinha trinta questões, três delas questões de múltipla escolha com aproximadamente nove opções cada, e as restantes questões de escolha simples, com cerca de seis opções cada. As questões de escolha simples ofereciam uma série de afirmações, entre as quais o participante selecionaria a que melhor se ajustasse. Uma destas questões, referente à atividade de documentação, pode ser vista na Figura 6.5.


How would you characterize the documentation effort done during the *initial phase* of the project?

a. Documentation was a priority: we produced documentation files and/or manuals for both users and developers.
b. There was both user and developer documentation but not a lot.
c. There was some user documentation provided.
d. There was some developer documentation provided.
e. Documentation was basically provided by comments in the code.
f. There was no documentation at all in this phase.
g. Other:

Comment:

Figura 6.5: Uma das perguntas do questionário inicial, relativa à atividade de documentação. Note que esta pergunta é específica à *fase inicial* do projeto; havia outra pergunta bastante similar relativa à fase pós-lançamento, e ainda outra relativa à situação atual.

Haviam três questões de múltipla escolha. Duas buscavam levantar o perfil das ferramentas usadas no projeto, e a terceira, os veículos de comunicação utilizados para publicar o lançamento do software. Esta última, embora não associada diretamente aos problemas de pesquisa, buscava avaliar quais mecanismos de publicidade eram mais conhecidos e utilizados pelos projetos; foi omitida da versão final do questionário, mas segue sendo uma indagação pertinente.

Uma vez elaborado e implementado o questionário, foi realizado um teste piloto para verificar sua adequação e aplicabilidade. Foram convidados dois revisores da comunidade – David Miller e Matthew Thomas, com quem eu havia trabalhado no projeto Bugzilla – que aceitaram revisar o trabalho. O questionário também foi avaliado por Elisa Yumi, do ICMC USP/São Carlos, que enviou sugestões e discutiu o conteúdo das questões. Os revisores fizeram uma série de comentários que nos indicaram problemas estruturais:

- O questionário era longo demais, violando a diretriz que orienta a elaboração de um questionário compacto: todos os revisores levaram mais de quinze minutos para preenchê-lo. O questionário completo ocupava mais de seis páginas impresso, e mais de doze telas quando visualizado em um navegador Web à resolução de 800x600 pixels, na fonte padrão de doze pontos.
- A repetição das perguntas entre as seções diferentes era cansativa e confusa; os revisores apontaram o problema da desorientação causada por haver questões muito parecidas.

- O questionário teste preenchido por David Miller apontou um problema referente às informações da fase pré-lançamento do software: ele não era o autor original do Bugzilla, e portanto não sabia responder essas questões com confiança.

- David e Matthew afirmaram ser cansativo seu preenchimento; além de longo e aparentemente repetitivo, o assunto não parecia ser interessante o suficiente para justificar o esforço.

Com base neste resultado, decidiu-se retornar à fase de projeto e alterar o questionário de forma a ser mais sucinto e menos enfadonho. **A observação cabível é que um questionário único (e exaustivo) parece ser uma opção ruim para medir as mudanças do processo ao longo do ciclo de vida de um projeto.** O questionário produzido nesta primeira iteração está disponível como o Apêndice A.

Questionário 2 (Final)

Com a experiência obtida na elaboração e revisão do primeiro questionário, partiu-se para a elaboração de uma nova versão. As lições aprendidas com o resultado anterior motivaram alterações na estrutura e no conteúdo do questionário:

- Buscou-se reduzir pela metade o tempo de preenchimento do questionário. Seriam incluídas no máximo quinze questões, este número correspondendo à metade das questões do questionário original.

- Dado o espaço limitado, e o conjunto de problemas apontado com relação às questões separadas por fase do ciclo de vida, optou-se por incluir apenas perguntas referentes à situação atual do projeto. Com isto, foi reduzida a chance do usuário dispersar-se.

- Para reduzir ao máximo o esforço exigido para responder o questionário, decidiu-se utilizar, quando possível, opções de múltipla escolha (‘Marque todas as que se aplicam’). Uma discussão sobre as vantagens e os riscos desta escolha é feita na Seção 8.4.1.

O questionário resultante desta iteração do projeto era sensivelmente mais curto e mais acessível, com doze questões no total. Incluía quatro questões gerais sobre o projeto, três questões sobre a equipe e o modelo de liderança, quatro questões detalhando atividades do processo de software, e uma questão levantando as ferramentas utilizadas no projeto. Está disponível nesta dissertação como o Apêndice B.

Revisão e Avaliação

Esta nova versão foi veiculada a um segundo conjunto de avaliadores que forneceu comentários construtivos e sugestões de melhoria. Em particular, Matthew Thomas (Bugzilla, LinuxStep) ofereceu sugestões para melhorar a legibilidade de certas questões; Bart Massey (Portland State University; veja

Seção 5.3.1) contribuiu com a questão referente aos tipos de usuários esperados do projeto, e com sugestões para as perguntas sobre requisitos e padrão anterior.

Além disso, as seguintes pessoas testaram e ofereceram comentários valiosos sobre o questionário: Ricardo Hasegawa (NILC USP/São Carlos), Johan Dahlin (PyGTK, GNOME), Michele Campeotto (Moleskine), Xavier Orduquoy (Gael).

Ao fim deste ciclo de avaliação, concluiu-se que esta versão do questionário estava adequada para veiculação pública. Nas seções seguintes é discutido o processo de veiculação e avaliação do questionário.

6.7 Veiculação do Questionário

Como apresentado na seção inicial deste capítulo, um dos desafios deste processo de levantamento é o desconhecimento da dimensão total da população a ser avaliada. Seria bastante difícil realizar uma seleção aleatória da população total dos projetos, já que não existe uma lista central e nem uma forma de contactar projetos aleatoriamente.

Utilizando uma amostra não-aleatória, no entanto, há o risco natural de obter uma medida tendenciosa, já que a população é definida com base em um critério estabelecido conscientemente para os fins da pesquisa. Por este motivo, não é estritamente possível generalizar os resultados deste trabalho para a população de projetos de software livre. O critério escolhido para determinar os projetos, no entanto, é representativo de uma parcela importante deles:

> A população que buscamos estudar é constituída de **projetos de software livre cujo processo de software se mostra saudável e sustentável**, e por este motivo, serão projetos que tenham características de *longevidade*, *intensa atividade*, ou *grande e permanente destaque público*.

A partir desta definição, foi estabelecida uma amostra por quotas e casos críticos [78], que são descritas na Seção 6.7.1. Cabe uma justificativa para a escolha desta amostra:

- A longevidade de um projeto sugere sua utilidade continuada, e portanto a sustentabilidade do seu processo de software. Projetos com *poucos usuários* e um *autor particularmente ativo* podem fugir a esta generalização, mas para fins desta pesquisa os julgamos cabíveis de inclusão.
- Projetos que possuem intensa atividade normalmente refletem uma comunidade vibrante e participativa. Este fator sugere a existência de um processo de software eficaz, que estimula a participação dos desenvolvedores e usuários.
- Projetos com grande destaque público podem ter conquistado fama de diversas formas; no entanto, destaque *continuado* sugere que do projeto resulta bom software. Uma das metas da en-

genharia de software consiste justamente em determinar o processo pelo qual 'bom software' é produzido, de forma que justifica-se a inclusão destes projetos.

O critério para este questionário define um foco bastante semelhante ao de Krishnamurthy no estudo descrito na Seção 5.1.3. Uma validação prévia deste critério pode ser obtida analisando os projetos selecionados, e avaliando sua distribuição: com base na lista final de 570 projetos que submeteram questionários válidos, é razoável afirmar que representa uma parcela significante e representativa dos projetos de software livre ativos e saudáveis da atualidade.

6.7.1 Seleção dos Projetos-alvo

Apesar de não existir registro central dos projetos de software livre, dois sites Web (citados anteriormente na Seção 4.3.1) referenciam um conjunto extenso destes projetos: o `freshmeat.net` e o `sourceforge.net`[7]. Ambos listam milhares de projetos, opcionalmente categorizados e ordenados de acordo com um número de índices diferentes (`freshmeat.net/stats`). Entre estes índices, três em particular são relevantes para este estudo:

***Development Level*:** projetos em ambos os sites são categorizados de acordo com seu grau de maturidade, sendo um índice crescente que varia de 1 a 6. O valor 5 corresponde a projetos estáveis, e 6 a projetos maduros. É importante destacar que este critério de maturidade é subjetivo e não tem relação com o critério de maturidade estabelecido pelo CMM [79].

Vitalidade: projetos que fazem grande número de lançamentos são considerados mais ativos, sendo que esta medida leva em consideração o tempo de existência do projeto, evitando que um projeto antigo receba sistematicamente notas baixas de vitalidade.

Popularidade: é um índice calculado a partir da intensidade de procura e visualização das informações do projeto; em outras palavras, indica quantas pessoas se interessaram pelo software.

Do `freshmeat.net`, foram escolhidos aproximadamente 800 projetos entre as categorias delimitadas por estes índices. O site `sourceforge.net` utiliza o mesmo sistema de métricas, mas oferece apenas as categorias 'projetos mais ativos' e 'projetos mais populares'. Estas categorias estão disponíveis em duas opções de período: uma contemplando apenas a última semana ('mais ativos na última semana', exibido na Figura 6.6), e outra geral. Optou-se por utilizar 100 projetos da versão geral de cada uma destas categorias, capturados no dia 12 de novembro de 2002.

Para complementar o conjunto de projetos obtido a partir destes sites, foi elaborada uma lista de 50 projetos críticos para o levantamento. Esta lista foi composta pelos projetos com os quais eu possuía

[7] Ambos os sites registram projetos de software livre; no entanto, enquanto o `freshmeat.net` se restringe a publicar notícias sobre os lançamentos diários, o `sourceforge.net` oferece uma infraestrutura completa para desenvolvimento, conforme visto na Seção 4.3.6.

experiência pessoal (e para os quais eu poderia, portanto, validar as respostas), e pelos projetos mais recorrentemente citados na literatura. Como parte desta lista foram incluídos projetos provindos de dois grupos de extrema importância para a produção de software para usuários finais: os meta-projetos KDE Office e o GNOME Office, que se dedicam à criação de aplicações para produtividade (como planilhas e processadores de texto).

- `freshmeat.net`
  - Projetos maduros (desenvolvimento 6)
    - 100 projetos mais populares
    - 100 projetos mais maduros
  - Projetos estáveis (desenvolvimento 5)
    - 200 projetos mais populares
    - 200 projetos mais maduros
  - 50 projetos mais populares do meta-projeto GNOME
  - 50 projetos mais populares do meta-projeto KDE
  - 50 projetos mais populares do meta-projeto GTK+
- `sourceforge.net`
  - 100 projetos mais ativos, geral
  - 100 projetos mais requisitados, geral
- 50 Projetos Adicionais
  - Projetos essenciais de software livre
  - Projetos do GNOME Office
  - Projetos do KDE Office

Tabela 6.1: Um resumo dos projetos escolhidos para a amostra.

É importante deixar claro que as categorias não são mutuamente exclusivas; em outras palavras, projetos constantes no `sourceforge.net` podem ser também listados no `freshmeat.net`. Por este motivo, o total de projetos contactados não equivale à soma da seleção indicada na Tabela 6.1. Também vale notar que a maior parte dos projetos do conjunto adicional está listada em posição de destaque no site `freshmeat.net`, o que confirma sua relevância.

Desta lista de 1.102 projetos, foi feito um levantamento dos endereços de correio eletrônico dos seus mantenedores. Este levantamento consiste de duas atividades: buscar pelos sites de registro os endereços de correio eletrônico de contato, e falhando esta busca, utilizando o site Web do projeto ou uma ferramenta de busca para localizar seu mantenedor.

6.7.2 Envio da Solicitação

Com a lista de projetos e endereços de correio eletrônico dos mantenedores, iniciou-se o processo de contato para solicitar participação na pesquisa. Utilizou-se uma mensagem em texto padrão detalhando o projeto, a relevância da participação daquele projeto em particular, e uma visão geral da pesquisa

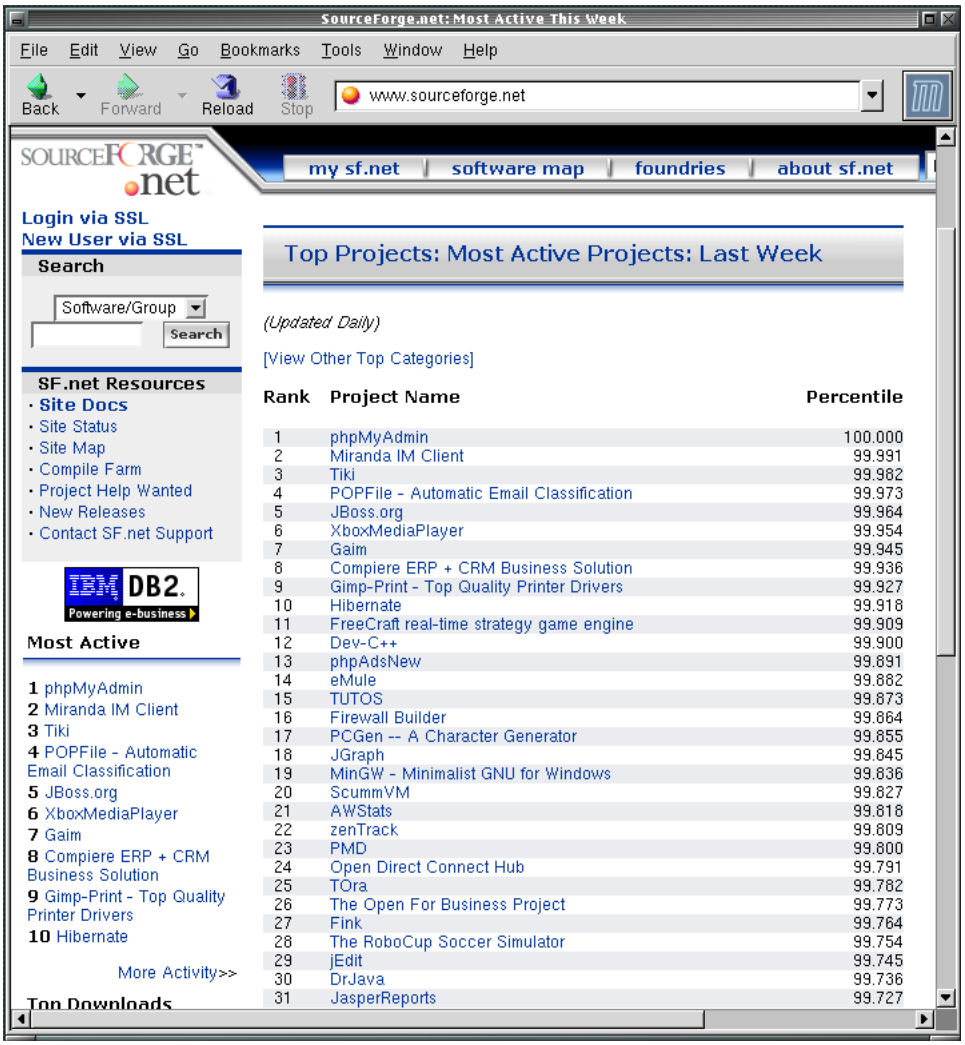


Figura 6.6: Lista dos projetos mais ativos para a semana de 16 de fevereiro do site `sourceforge.net`

executada neste trabalho de mestrado. A mensagem incluía um hiperlink para a página do formulário, e oferecia a opção de solicitar por e-mail o questionário. A mensagem final está disponível como o Apêndice C.

Sendo uma mensagem não-solicitada, houve cuidado particular para evitar desagradar a população em questão. O conteúdo da mensagem foi elaborado com um enfoque pessoal, e oferecia no início do texto uma lista dos projetos mais famosos que já haviam respondidos. Com isso, buscou-se credibilizar o trabalho, associando-o aos nomes de projetos importantes. Além disso, esta mensagem foi ligeiramente alterada entre cada lote de mensagens enviado para atualizá-la ao estado atual do levantamento e melhorar sua legibilidade. Mensagens personalizadas foram enviadas para as pessoas da comunidade de software livre com quem eu já possuía algum contato anterior (ironicamente, escrever estas mensagens demandou mais esforço que todo o processo de elaboração e envio da mensagem padrão).

Pela grande quantidade de endereços para contactar, decidiu-se dividir o envio em quatro lotes. O envio foi realizado entre os meses de setembro e novembro de 2002: um quarto dos endereços listados foi contactado por semana, com uma pausa de duas semanas entre cada lote.

Esta separação em lotes foi particularmente oportuna por permitir comunicação mais efetiva com o grupo de projetos daquela semana em particular, além de garantir tempo livre para atividades de manutenção da ferramenta de questionário e processamento dos endereços de e-mail inválidos — além de interpretar a mensagem de *bounce*, era necessário buscar um endereço atualizado para o projeto, e reenviar a solicitação.

Resposta

A reação ao questionário enviado foi surpreendente. Uma análise mais profunda dos tempos de resposta ainda merece ser feita, mas é possível dizer que aproximadamente 30% dos questionários enviados foi respondido dentro de um período de 6 horas do seu envio. Um total de 6 indivíduos responderam solicitando uma versão por e-mail do questionário, dos quais 5 retornaram suas respostas.

Resta analisar com maior precisão o perfil das mensagens retornadas, das respondidas pessoalmente, e das que resultaram em questionários efetivamente preenchidos. Acredito que estes resultados possam fornecer informações valiosas a respeito da veiculação de questionários para este tipo de população. Uma parcela pequena (aproximadamente 5%) dos indivíduos contactados respondeu diretamente após preencher o questionário com sugestões, comentários e perguntas. A maior parte destes indivíduos manifestou interesse em obter uma versão do relatório final. A recompensa imediata resultante do envio foi um número surpreendente de mensagens recebidas elogiando o estudo, das quais três exemplos seguem:

> ‘I completed the survey. I must say that it is best survey of its kind I’ve participated in. Every question is completely relevant, the proposed answers are reasonable and exhaustive, and the possibility of giving open answers leaves very little chance for error.’

> ‘Thanks for your personal request, as I wasn’t aware of your survey. It sure seems like a good subject and I’ve already seen some interesting stats at your site. [...] Please send me a copy (html format is fine) of the stats when you have finished the project. I’m very interested to see the final outcome.’

> ‘Cool survey! I just completed it. Better than most of this kind that I’ve seen. (Some I’ve given up on because the terminology was too alien.)’

> ‘I feel your e-mails and online form work well; the amount of attention paid to them increased my incentive to fill out the study.’

Embora os totais sejam apresentados e discutidos no Capítulo 8, cabe mencionar aqui a dimensão deste levantamento de dados: um total de 1.102 solicitações foram enviadas, e um total de 570 questionários foram subseqüentemente preenchidos.

6.8 Análise dos Resultados

Uma vez que um número significativo de respostas havia sido acumulado, iniciou-se o processo de análise dos resultados, e coleta secundária de dados. Esta seção descreve estas atividades em maior detalhe, incluindo a motivação para isolar uma parcela dos projetos, e a forma como parte dos dados foi tratada.

> Para qualquer pesquisa veiculada a um público grande e diversificado, existe uma necessidade natural de filtrar as respostas de acordo com seus critérios experimentais. A análise dos resultados deste trabalho teve como política implícita evitar, quando possível, o descarte de respostas obtidas, considerando valiosa cada participação individual, e buscando ao máximo aplicar os critérios de maneira a viabilizar a inclusão do maior número de respostas possível.

É importante deixar claro que a base de respostas acumulada consiste em um recurso muito rico para avaliação posterior. A análise feita neste trabalho é ainda preliminar, dado o pouco tempo que houve para ponderar e concluir sobre os resultados.

O tratamento posterior dos questionários pode ser subdividido nas seguintes tarefas, executadas em grande parte paralelamente:

1. Classificar e isolar projetos de escalas distintas, e os questionários incompletos.

2. Avaliar comentários e respostas personalizadas, identificando problemas excepcionais no questionário e assinalando comentários relevantes.

3. Obter os dados adicionais para cada projeto (caracterizar o domínio do software e processar o seu código-fonte).

4. Gerar agrupamentos com questões correlacionadas.

Todas estas atividades são suportadas pela ferramenta de questionário; é possível indicar respostas incompletas, projetos de escalas indesejadas, comentários interessantes, e iniciar o processamento do pacote de código-fonte.

6.8.1 Ferramentas de Análise

À medida que os questionários foram sendo preenchidos, um conjunto de ferramentas para visualização foi construído. Este conjunto inclui:

- uma página com os totais gerais do questionário, incluindo dados adicionais obtidos do seu processamento, como os países de procedência dos participantes e as médias dos resultados da análise dos pacotes (disponível como o Apêndice E);
- uma página com uma lista dos projetos registrados, contendo hiperlinks que permitem visualizar as respostas individuais de projetos não-privados;
- uma tabela para visualização comparativa dos resultados individuais dos projetos;
- um formulário que permite selecionar respostas individuais e obter destas os projetos que as assinalaram (disposta na Figura 6.7);
- um conjunto grande de gráficos gerados automaticamente a partir da base de dados do projeto.

As ferramentas de visualização são integradas através da interface Web do projeto e estarão disponíveis para acesso público após a defesa desta dissertação.

6.8.2 Classificação e Validação dos Questionários

Dado o grande número de respostas obtidas, foi necessário fazer uma triagem dos projetos participantes do questionário. Nosso objetivo, conforme anteriormente descrito, é caracterizar o processo de software em projetos de software livre; aplicando este último termo no sentido da definição oferecida em 4.1, é necessário fazer distinção entre projetos de escalas diferentes[8].

Durante o processo de seleção dos projetos para o questionário, tomou-se o cuidado de evitar convidar meta-projetos e distribuições; no entanto, por desconhecimento, incorreção ou pelo simples acaso, alguns acabaram sendo registrados. O tratamento dado a estes casos excepcionais está descrito a seguir:

Meta-projetos: uma parcela pequena dos projetos inscritos no questionário foram identificados como meta-projetos (12 projetos, ou 2%). Estes foram assinalados como tal e removidos das contagens globais para evitar imprecisão nos resultados. Meta-projetos que responderam o questionário incluem os projetos Ogg, GNU Enterprise, ALSA, gEDA e ROX Desktop.

Distribuições: assim como os meta-projetos, algumas distribuições foram inscritas (um total de 19, ou 3%); tratamento idêntico foi dado a estas. Exemplos de distribuições que responderam o questionário incluem o Openwall GNU/Linux, uClinux, teTeX, Linux from Scratch, Ark Linux e o GnuWin32.

[8] Uma definição dos termos ‘meta-projeto’ e ‘distribuição’ é feita na Seção 4.2.

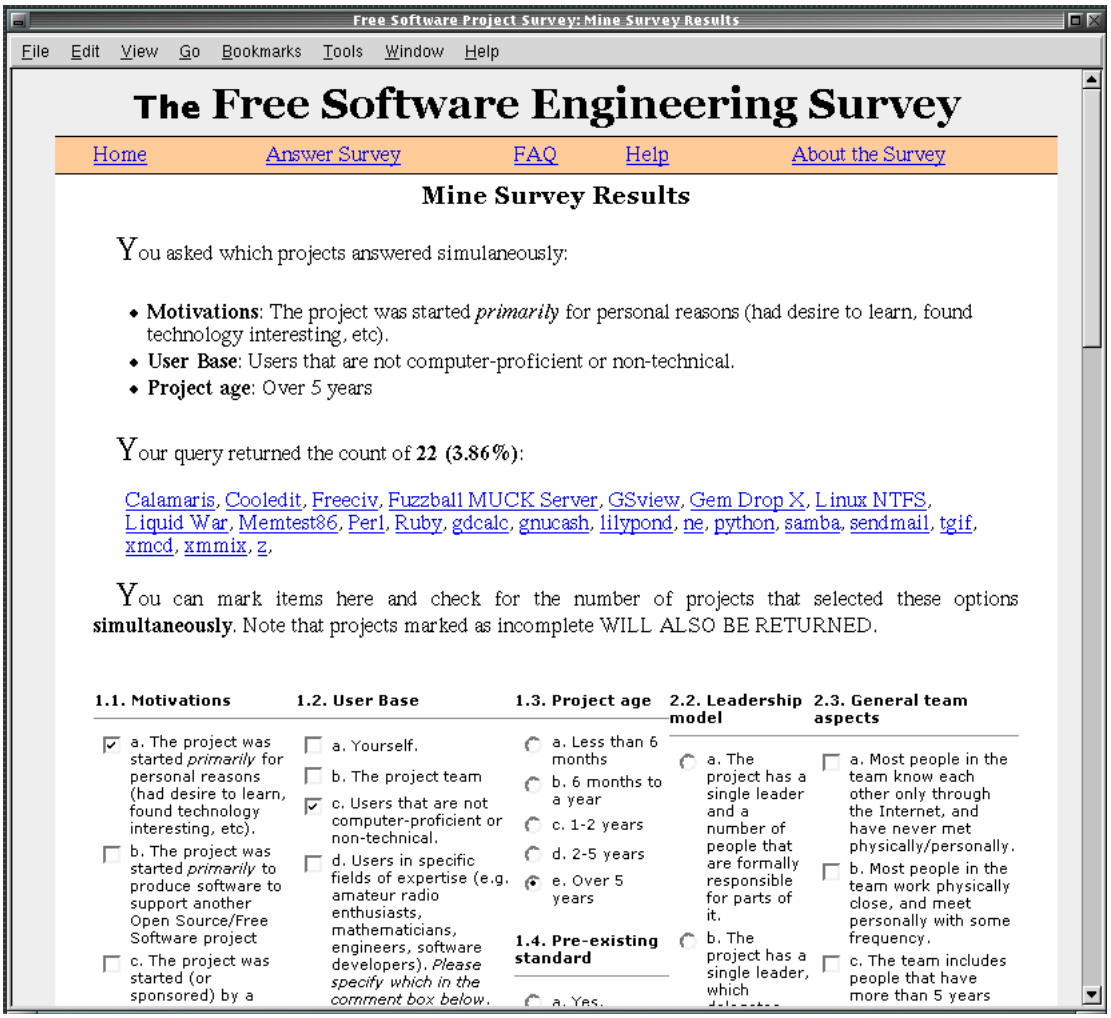


Figura 6.7: Imagem de uma das ferramentas de análise do questionário. Com esta ferramenta, é possível selecionar um conjunto de alternativas e obter os projetos que as selecionaram. Os nomes dos projetos são hiperlinks para suas páginas de resposta individuais.

Questionários Incompletos: Alguns questionários não foram preenchidos completamente (um total de 21, ou 3%). Como regra, qualquer questionário com mais de 5 questões sem itens selecionados foi descartado. A justificativa para este descarte é a grande possibilidade do formulário ter sido incompreendido (ou ignorado) pelo participante.

Além dos questionários incompletos, alguns projetos ofereceram dificuldade na medição do código-fonte do seu software; em particular, projetos que ofereciam mais de um pacote de software (onde o software principal não estivesse claramente marcado), ou projetos que produzem *patches* sobre o código de outros projetos. Estes projetos foram omitidos apenas das medidas e comparações de código-fonte.

Uma exceção interessante ocorreu com o projeto Jabytad. Este projeto foi incorretamente incluído

na lista de envio: era um projeto de *freeware* e não software livre (refira à Seção 3.2.1 para uma discussão dos termos). Ao reavaliar os projetos que haviam respondido o questionário, percebi a inconsistência e entrei em contato com o participante para justificar o motivo pelo qual não poderia usar seu formulário. A resposta do autor Tom Drexl, da Alemanha, segue:

> 'As Jabytad was my first project I wasn't sure whether to make it freely available as Open Source... and as there is no sense in hiding the sources, the source is (intended to be) freely available, currently there is just no link and thus no possibility to find the sources. I will correct this tomorrow evening and put it officially under the GPL, so you can keep the answers.'

O pacote foi obtido da página do projeto, medido e incluído nos totais da pesquisa.

Outras Exceções e Correções

Foram verificados os comentários e as respostas personalizadas obtidas do questionário. Embora a grande maioria dos participantes tenha usado a caixa de comentário para detalhar ou especificar melhor a resposta oferecida, alguns apontaram imprecisões ou dúvidas em relação ao questionário.

- A questão 1.3 determinava há quanto tempo havia sido lançada a primeira versão pública do software. Quatro projetos indicaram ainda não ter lançado versões finais do seu produto. Esta ausência entre as opções é uma falha reconhecida do questionário.
- Na questão 2.1, que buscava determinar o tamanho da equipe, dois projetos utilizaram o campo personalizado para indicar um número específico maior que 50; foram ajustados manualmente para a alternativa correta, evitando invalidar suas outras respostas.

Questão 1.4 (Padrão Pré-existente) Foi verificada uma inconsistência mais significativa na questão 1.4. A intenção da pergunta era determinar a *pré-existência* de um padrão publicado relevante para o projeto. No entanto, a pergunta original omitia a informação de que o padrão solicitado deveria ser de fato pré-existente, e dizia apenas:

> Is there a standard or public specification available that defines a significant part of the project?

No entanto, a página de validação usa o título **Padrão Pré-existente** para identificar esta questão. Por este motivo, alguma confusão resultante pôde ser observada entre os comentários. A verificação desta falha provocou o dilema: descartar a resposta ou indicar claramente uma ressalva nos resultados associados?

Com uma leitura cuidadosa dos comentários, e uma verificação das respostas por meio de material disponível na Web (usando como recurso final contato por correio eletrônico com o responsável pelo preenchimento), foi possível levantar com precisão quais projetos haviam respondido por engano; entre os 184 projetos que responderam positivamente, apenas 9 projetos não haviam incluído um comentário descrevendo exatamente qual padrão utilizavam. No total, foram identificados 18 projetos que incorreram em erro, se referindo a padrões criados após o lançamento ou *durante* o desenvolvimento do projeto.

Dada a segurança com que as respostas podiam ser corrigidas e confirmadas pelos seus autores, optou-se por modificá-las e não descartar esta questão. Para evitar alterar os dados originais durante o processo de correção, uma cópia das respostas foi criada, e corrigiu-se a resposta destas 18 questões. O resultado final não se alterou significativamente, mas é importante registrar esta ressalva, já que indica que uma parte dos dados, ainda que pequena, foi corrigida durante a análise.

O texto da questão foi corrigido[9] ao final deste processo (e, portanto, não foi utilizado por nenhum projeto considerado nesta análise de dados).

Não foi realizada correção nos dados além dos casos supracitados; no entanto, validação posterior do questionário é necessária para garantir que de fato não há inconsistência adicional (infelizmente, não houve tempo para realizar esta validação durante o período do projeto de mestrado).

6.8.3 Classificação por Domínio

Além das respostas dos questionários, havia sido decidido, para cada projeto, levantar um conjunto adicional de dados. Esta coleta consistiu de duas atividades principais: determinar um domínio para o projeto, e medir o pacote de código-fonte distribuído por este.

É desafiador elaborar um sistema de categorias para domínios de aplicação. Além de não conhecer nenhuma taxonomia de domínios publicada na literatura que pudesse ser aplicada aos projetos, tive dificuldade adicional por ser um conjunto de softwares muito grande, uma parte possuindo amplo escopo.

Aceitas estas limitações, identificou-se um conjunto de 16 categorias para classificar o domínio da aplicação, com base na população de projetos capturada na pesquisa. Esta divisão foi realizada utilizando apenas experiência prévia e decisão consensual:

[9] A decisão de corrigir o texto reflete a pretensão de reutilizar o questionário no futuro para avaliar mudanças ao longo do tempo, e de aplicá-lo a uma população de controle para validação das respostas.

- Administração, Financas e E-commerce
- Áudio, Vídeo e Multimídia
- Aplicações Científicas
- Ambientes Gráficos e similares
- Bases de Dados e similares
- Comunicação Pessoal, P2P
- Correio Eletrônico
- Desenvolvimento e Engenharia de Software
- Emulação
- Engenharia
- Gráficos e Animação
- Jogos
- Navegadores e Aplicações de Escritório
- Outros
- Redes e Segurança
- Software Básico

A divisão por domínio teve como objetivo oferecer uma classificação adicional dos resultados com base na área de aplicação do software. Note a existência de uma categoria residual (*Miscellaneous* ou Outros) que serviu para agrupar projetos cujo domínio não se adequaram a nenhuma outra categoria.

É importante observar que o domínio não foi fornecido pelo participante, e sim selecionado manualmente para cada projeto durante esta fase de coleta secundária. Optou-se por indicar manualmente este dado com o objetivo de reduzir o número de questões ao mínimo possível, e garantir precisão com relação ao critério de divisão das categorias.

6.8.4 Análise do Código-Fonte

Cada projeto, como descrito no capítulo anterior, implementa um processo que tem como resultado fundamental a produção e o lançamento de software livre em um pacote contendo seu código-fonte. Este código-fonte representa uma evolução do software do projeto deste seu lançamento inicial, integrando todas as contribuições feitas ao longo deste período.

Uma medida objetiva que pode ser feita para um projeto, portanto, é a avaliação do conteúdo de seu pacote: um estudo desta natureza permite uma avaliação do perfil de código-fonte produzido, e potencialmente oferece um indicativo do desempenho e qualidade do projeto.

No entanto, este enfoque experimental ainda precisa ser melhor pesquisado para confirmar sua validade. Esta etapa do processo tem como objetivo nominal apenas oferecer um conjunto de dados que possa suportar este processo de validação, e os resultados associados ao estudo do código-fonte devem ser analisados levando em consideração esta ressalva.

Um total de três medidas diferentes foi obtida de cada pacote:

1. Tamanho (bruto) do pacote, normalizado para um formato comum.
2. Data de lançamento público do pacote.
3. Um perfil do número de linhas do código-fonte, separada por linguagem e totalizada.

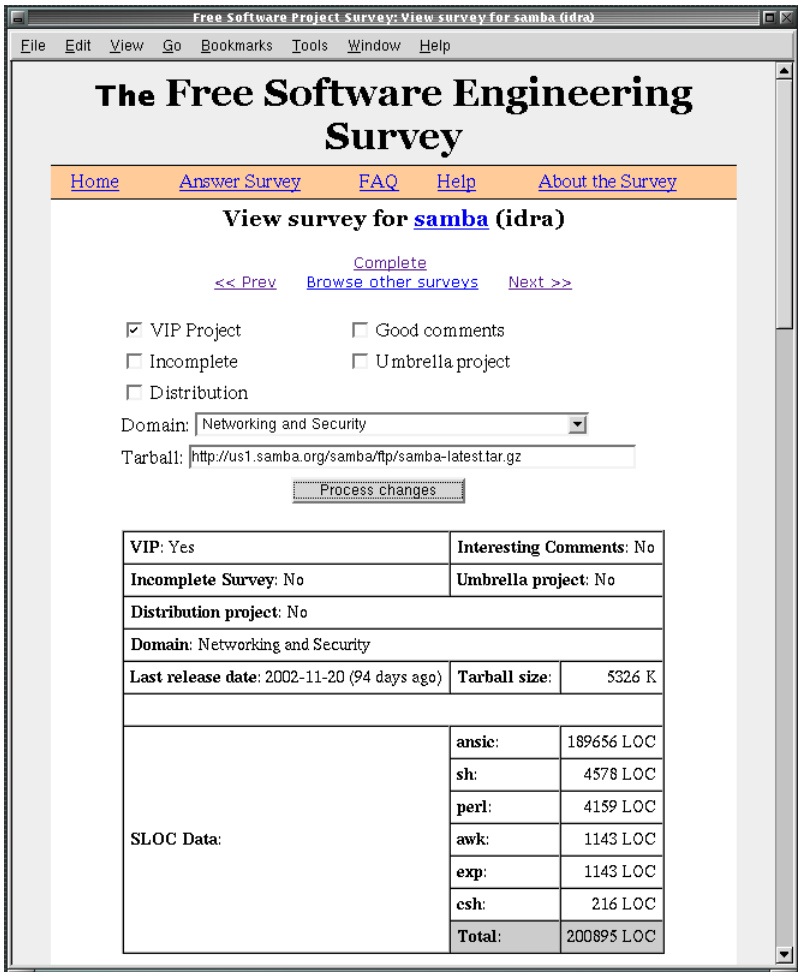


Figura 6.8: Preenchimento dos dados secundários para o projeto samba. Na parte superior da imagem estão os controles para completar os dados. Note o campo 'Tarball' onde é entrado o nome do pacote a ser medido. A ferramenta `process_tarball` é invocada ao submeter este formulário, e realiza a transferência e processamento.

Dada a grande quantidade de projetos a serem medidos — ao final, 521 dos 570 projetos respondidos tiveram seus pacotes analisados — decidiu-se criar uma ferramenta para auxiliar o processo de coleta de pacotes. Esta ferramenta (que tem o nome pouco original de `process_tarball`), obtém um pacote com base em um endereço Internet (um URI ou *Uniform Resource Identifier* nos termos do W3C), realiza as medidas relevantes, e atualiza a base de dados do projeto. Foi escrita na linguagem Python, e é relativamente simples, invocando diversas ferramentas independentes para executar sua função, incluindo o `sloccount` descrito na Seção 5.3.6.

Esta ferramenta foi integrada à interface Web de análise do projeto, sua utilização sendo apresentada na Figura 6.8. Nas seções seguintes é descrito o processo de obtenção e análise dos pacotes.

Obtenção do Pacote

De cada projeto, foi obtido um pacote de código-fonte. Os seguintes critérios foram utilizados para guiar a escolha destes pacotes:

- Para projetos que utilizam versões estáveis e instáveis, foi escolhido **o último pacote publicamente lançado da *versão estável***; em outras palavras, não foram medidos pacotes em teste (alfas, betas ou *release candidates*). A motivação para este critério foi concentrar a pesquisa em versões que o projeto considera *válidas para uso final*, evitando medir pacotes que ainda não tivessem sido lançados como finais.

  De certa forma, este enfoque incorpora a suposição de que apenas pacotes lançados são bons indicativos do processo de software completo, já que terão sido inteiramente realizadas as atividades da garantia de qualidade e o lançamento do software.

- Para projetos que possuem apenas uma versão principal, foi escolhido o último pacote publicamente lançado.

- Para meta-projetos, não foi feita medição, já que seria inconsistente compará-los aos projetos individuais. Da mesma forma, distribuições não participaram da medição[10].

- Para projetos sem versão definitiva lançada publicamente, foi utilizada a última versão lançada, mesmo que não definitiva — ou seja, uma versão alfa, beta ou um *release candidate*. Embora este aspecto possa trazer alguma inconsistência se comparado ao primeiro critério apresentado, apenas 4 projetos não possuíam versão lançada, correspondendo a uma minoria da população.

- Um número pequeno de projetos (4 projetos) oferece apenas lançamentos binários, não havendo um pacote contendo o respectivo código-fonte. Destes projetos, foi obtida uma cópia do código-fonte a partir do CVS, sendo requisitada e medida a versão integrada no mesmo dia do lançamento binário. (Note que foram eliminados os diretórios CVS do pacote processado na medição.)

- Outro conjunto de projetos é distribuído como parte de um 'meta-pacote' maior; por exemplo, as aplicações do KDE Office têm esta propriedade. Estes pacotes foram processados abrindo-se o meta-pacote e medindo apenas o diretório da aplicação. Este processo incorre em algum risco de erro, já que há uma chance de sub-avaliar o total de código do projeto. No entanto, menos de 10 projetos apresentaram esta propriedade.

O tamanho total e a data do arquivo mais recente contido neste pacote[11] foram registrados. Grande parte dos pacotes é distribuído no formato `tar` (que é comum em sistemas Unix) e comprimido utili-

[10] Medir as distribuições representaria, adicionalmente, um problema operacional pela dimensão **muito** maior que têm; a maior parte ocupa múltiplas mídias de CD-ROM.

[11] A data do arquivo mais recente é uma boa forma de medir o momento real de lançamento do software; como visto anteriormente, o veículo de lançamento não é o mesmo para todos os projetos, e a data do pacote em si é pouco confiável por ser freqüentemente alterada durante o processo de transferência do pacote através da Internet.

zando a ferramenta `gzip`. Para manter uma consistência na medição, pacotes distribuídos em outros formatos (PKZip e `bzip`, exclusivamente) foram convertidos para o formato `tar` com `gzip`.

Medição de Linhas de Código

Com base na mesma ferramenta utilizada por David Wheeler na sua análise da distribuição Redhat, o `sloccount`, foi realizada a contagem das linhas de código presentes em cada pacote, classificada por linguagem de programação. Esta medida é interessante por uma série de motivos: permite discutir quais são as linguagens prevalecentes entre os projetos (categorizados de diversas formas), oferece uma medida de dimensão do produto do projeto, e pode ser usada como um indicador, ainda que discutível [80], de desempenho relativo.

A ferramenta processa um conjunto de arquivos de código-fonte e produz um relatório com as linguagens e seus totais em linhas de código 'lógicas' - são eliminadas linhas em branco ou linhas contendo apenas comentários. O resultado produzido pela ferramenta após o processamento do Apache, versão `1.3.27`, segue como exemplo (ligeiramente formatado):

```
SLOC    Directory       SLOC-by-Language (Sorted)
95453   src             ansic=88846, sh=4642, perl=1623,
                                     lex=190, yacc=97, cpp=55
1345    top_dir         sh=1345
131     htdocs          perl=104,lisp=27
31      cgi-bin         sh=24,perl=7
0       logs            (none)
0       conf            (none)
0       icons           (none)

Totals grouped by language (dominant language first):
ansic:        88846 (91.63%)
sh:            6011 (6.20%)
perl:          1734 (1.79%)
lex:            190 (0.20%)
yacc:            97 (0.10%)
cpp:             55 (0.06%)
lisp:            27 (0.03%)
```

Embora bastante conveniente, este método de contagem possui algumas deficiências. A maior delas, e que prejudica a medida de um número de projetos nesta pesquisa, é que a ferramenta não contabiliza todas as linguagens. O mesmo problema foi levantado na medição da distribuição Debian descrita por Jesús M. González-Barahona et al., resumida nesta dissertação na Seção 5.3.6.

Linguagens não contabilizadas pela ferramenta incluem XML, HTML, Javascript/ECMAScript, e Pike. Projetos como o Mozilla, cuja interface com o usuário é integralmente codificada em XML e Javascript, tiveram contagens significativamente alteradas como resultado desta deficiência.

Uma possibilidade para trabalho futuro é implementar uma extensão à ferramenta que processe

estas linguagens adicionais. Houve algum progresso nesse sentido durante este trabalho de mestrado, mas não restaria tempo suficiente para uma implementação completa, e como a ferramenta é vital para o processamento dos pacotes, optou-se por utilizar a versão padrão.

Eliminação de Arquivos Gerados Um problema adicional com o método de contagem merece discussão: entre os pacotes de software livre, é comum a existência de arquivos gerados por outras ferramentas. Em particular, a suíte *autoconf/automake/autoheader*, que é um padrão de fato utilizado em grande parte dos projetos de software livre analisados, gera um número considerável de *scripts* responsáveis por compilação e instalação do software.

Avaliou-se que o volume de código gerado poderia ser grande o suficiente para promover uma alteração significativa nas medidas de código-fonte dos pacotes, e por este motivo optou-se por remover dos pacotes — antes de processá-los com a ferramenta `sloccount` — este conjunto de arquivos, com base no seu nome (exemplos incluem `configure`, `install-sh` e `libtool`). A ferramenta `process_tarball`, que é responsável por executar o `sloccount`, foi alterada para fazer esta remoção automaticamente.

É importante ressaltar que existem outros problemas de medição que não avaliamos nesta pesquisa, e que tem potencial para alterar significativamente os resultados para alguns pacotes. Em particular, alguns pacotes incluem parte ou todo o código-fonte de pacotes produzidos por outros projetos. Esta prática é freqüentemente chamada de *bundling*. O problema, neste caso, é que a contagem do projeto que inclui o pacote externo será super-avaliada. No entanto, se o pacote 'hospedeiro' altera o pacote incluído, torna-se bastante difícil decidir qual o procedimento correto para contagem.

Estes e outros problemas inerentes à medição de código-fonte indicam que é um método que deve ser aplicado considerando suas limitações, e que seus resultados devem ser analisados com cautela.

6.8.5 Agrupamento dos Ítens

Dado o grande número de itens relativos à engenharia de software, uma análise sistemática do questionário pode beneficiar-se de uma quantificação mais geral. A principal análise do trabalho foi realizada com base nas contagens das alternativas individuais; no entanto, qualquer descrição individualizada destas alternativas requer análises (e portanto explicações) muito detalhadas.

Para obter um índice que resuma quanto esforço é dispendido nas atividades de engenharia de software do questionário, criou-se uma lista com as alternativas individuais relevantes, acompanhadas de um peso. Este peso tem valor variando de 1 (esforço desprezível) a 5 (grande esforço). Para calcular o peso de cada alternativa foi realizado um levantamento simples com dois especialistas que responderam, para cada item, à pergunta 'Quanto esforço de engenharia de software é indicado ao assinalar este item'. Os pesos médios estão dispostos na Tabela 6.2.

	Alternativa	**Peso**
	3.1. Requirements	
b.	A significant amount of effort is spent defining what the software functionality and behavior (the requirements) should be.	3.5
d.	There have been meetings or discussions with end-users to define significant parts of the software functionality or behavior.	2
e.	These meetings/discussions occur frequently, and can be considered an important part of defining the project's software.	4
	3.2. Usability	
a.	The user interfaces for the project are designed (or prototyped) and refined before actually being implemented.	3
b.	We have conducted serious usability tests and studies on the project's user interfaces.	4
c.	Developers are not allowed to implement or change the user interfaces before the implementation/change has been reviewed and approved.	4
d.	A part of the team is specifically in charge of UI design.	4
	3.3. Documentation	
a.	We have produced documents that describe at least some of the expected functionality and behavior (requirements) of our software.	2
b.	The project produces and maintains documents for developers that describe how the code is organized (architecture), and/or how parts of it work.	3
c.	There is a reasonably complete coding standards guide that is actively followed by the team.	1
d.	There is documentation for the end-user available for the project's software (consider also third-party documents available).	1
e.	A significant part of the available documentation is frequently updated and revised to be up to date.	5
	3.4. Quality	
a.	There is an [automated] test suite for the project's software, that is used to validate it.	3.5
b.	There is a test plan (a written document describing tests) for the project's software, that is used by the project team.	4.5
c.	Periodic (i.e. nightly, weekly) snapshots of the project's code (or binaries) are distributed and used as an significant means of testing the software.	2
d.	There is an active code review process, where code is read by other members of the team.	4
e.	The project team members have formal rules for integrating code changes into the main codebase, and review is a strict requirement.	4.5
g.	There is a tendency (or policy) to release a public version only when it has been extensively tested by the team.	3
	Máximo Teórico:	57

Tabela 6.2: Tabela com questões e pesos individuais utilizados para calcular o índice de engenharia de sofware.

O peso individual de cada projeto foi dividido pelo peso máximo teórico, e multiplicado por 100 para obter um índice percentual. Exemplificando, um projeto que assinalou apenas os itens `3.1b`, `3.3b` e `3.4c` obteria um índice de engenharia de software resultante da equação:

$$I_{SE} = (P_{3.1b} + P_{3.3b} + P_{3.4c})/P_{max} I_{SE} = 100 * ((3.5 + 3 + 2)/57) = 14.9$$

De forma semelhante, um índice foi criado para descrever o uso de ferramentas de engenharia de software. A pergunta utilizada para determinar este peso é 'O quanto utilizar esta ferramenta auxilia o processo de software?'. Os pesos para este índice se encontram descritos na Tabela 6.3.

	Alternativa	Peso
	3.5. Software Tools	
a.	A project hosting site such as Sourceforge.net, Savannah or Collab.net	2
b.	One or more Web sites	1
c.	A WikiWikiWeb site (SWiki, TWiki, PHPWiki, ZWiki, etc.)	2
d.	A Frequently Asked Questions document	1
e.	A version control tool such as CVS, RCS, Subversion or Bitkeeper	4
f.	A bug database such as GNATS or Bugzilla	5
g.	One or more mailing lists	2
h.	Network news (NNTP)	1
i.	User forums/BBS	1
j.	IRC	1
k.	An instant messaging system like ICQ, Jabber or AIM	1
	Máximo Teórico:	20

Tabela 6.3: Tabela com pesos utilizados para calcular o índice de ferramentas.

Com base nestas tabelas de pesos, índices para cada projeto foram calculados com a ferramenta de visualização de dados. Estes índices são discutidos no Capítulo 8, correlacionando-os com tamanho da equipe, tempo de vida do projeto e tamanho da base de código.

6.9 Conclusões

Nesta seção foi apresentada uma visão geral da metodologia utilizada neste projeto de mestrado. Foram vistas as etapas individuais desta metodologia, incluíndo a participação ativa, o projeto do questionário, e a sua veiculação e análise.

Os resultados mais expressivos deste trabalho são descritos nos dois capítulos posteriores, que tratam da participação ativa e do levantamento, respectivamente. Os resultados da participação ativa têm um caráter predominantemente subjetivo, representando reflexões e conclusões obtidas da experiência como observador participante; já os resultados do questionário oferecem uma base quantitativa para avaliar e descrever o processo de software em questão. Juntos, fornecem uma visão tanto ampla quanto detalhada de um conjunto de projetos de software livre.

Capítulo 7

Resultados da Participação Ativa

Como descrito no capítulo anterior, durante o período do mestrado foi realizada uma experiência como participante em um conjunto de projetos de software livre. Parte desta experiência é descrita em detalhe nos artigos procedentes deste trabalho, e um resumo breve dos resultados obtidos é apresentado nas seções a seguir.

7.1 Mozilla

Minha participação neste projeto cobriu as atividades fundamentais de um usuário avançado, o que inclui:

- Teste funcional, e participação nos testes de aceitação pública (chamados *smoketests*) diários.
- A abertura, discussão e triagem de informes de erro, fazendo parte de um esforço colaborativo que representa parte significativa da Garantia de Qualidade no projeto Mozilla.

Paralelamente a este trabalho, foi realizado um levantamento detalhado do processo de software aplicado no projeto, que resultou na publicação do artigo 'An Overview of the Software Engineering Process in the Mozilla Project' [53]. Para a elaboração deste texto foram envolvidos diversos engenheiros e gerentes do projeto, que contribuíram com métricas, comentários e revisão; além disso, foi realizado um estudo detalhado das ferramentas utilizadas no projeto.

Este artigo foi apresentado no congresso FOSDEM (*Free and Open Source Developers' Meeting*), em Bruxelas, Bélgica, e no Open Source Software Development Workshop, realizado em Newcastle Upon Tyne, Inglaterra, em Fevereiro de 2002. Uma versão atualizada foi submetida e arquivada no repositório de artigos online do Free/Open Source Research Community (`opensource.mit.edu`).

Minha participação no projeto Mozilla se iniciou no final de 2001 e continua até o presente momento, embora em intensidade menor do que nos seis meses iniciais.

7.2 Bugzilla

Da experiência no projeto Bugzilla, um conjunto significativo de resultados foi obtido. Além de participar ativamente das listas de discussão e informes de defeito do projeto, me tornei um de seus desenvolvedores plenos[1], sendo responsável pela implementação de algumas funcionalidades importantes, e sendo freqüentemente atribuído à atividade de revisão de código. Tive oportunidade de desenvolver meu conhecimento das tecnologias envolvidas — incluindo Perl e a ferramenta de geração dinâmica de HTML Template Toolkit, que são utilizados intensamente no projeto — e participar pela primeira vez do desenvolvimento de um projeto de software livre amplo e intensamente utilizado.

O trabalho no projeto Bugzilla ocorreu com maior intensidade nos meses de outubro de 2001 a fevereiro de 2002 e continua até o presente momento. Boa parte dos frutos deste trabalho estão descritos no artigo 'Uma Visão Geral do Bugzilla, uma Ferramenta de Acompanhamento de Alterações' [81], apresentado no Workshop de Ferramentas do XVI Simpósio Brasileiro de Engenharia de Software.

7.3 PyGTK

Dos projetos com os quais tive experiência, o PyGTK e o Kiwi receberam as maiores contribuições práticas, representadas na forma de submissões de código-fonte e documentação. É irônico que estes projetos sejam tão minimalistas em termos do processo de software; são exemplos de esforços existentes na comunidade que produzem software com muito pouca infra-estrutura e processo formal.

A participação como usuário no PyGTK iniciou-se nos primeiros meses de 2001, e a contribuição efetiva de código iniciou-se no final de 2002, continuando até o presente momento. Em novembro de 2002, passei a contribuir ativamente para a versão estável atual (o branch `0.6`), logo me tornando um de seus responsáveis principais.

O PyGTK em particular representa uma amostra pequena porém significativa do meta-projeto GNOME, que tem como objetivo produzir um conjunto de bibliotecas para aplicação e ferramentas de produtividade para usuários finais. Este meta-projeto reúne centenas de projetos distintos; projetos importantes incluem o Gnumeric, uma planilha eletrônica, o Abiword, um processador de textos e o Gnucash, um aplicativo para controle financeiro pessoal.

Além do trabalho de desenvolvimento, que envolve principalmente correção de defeitos e extensão da cobertura da API GTK+, me envolvi em um projeto para melhorar a documentação disponível para programadores usuários da biblioteca. Como descrito anteriormente, não havia documentação para a biblioteca além de um conjunto pequeno de exemplos, usuários sendo remetidos constantemente à documentação da API GTK+ ou à lista de discussão. Esta falta prejudicava em particular os usuários do software com quem tinha contato no meu ambiente de trabalho, e por isso me dediquei à elaboração de um documento contendo um guia técnico para tarefas específicas associadas à biblioteca, na forma

[1] Disponível em `http://www.bugzilla.org/who_we_are.html`

de um FAQ[2].

Para a elaboração do FAQ, me baseei fundamentalmente em conhecimento e estudo pessoal, explorando adicionalmente arquivos da lista de discussão e a documentação GTK+. O FAQ atualmente registra grande parte do conhecimento a respeito da biblioteca, sendo muito utilizado pela comunidade, e incluindo mais de 150 perguntas e respostas. É notável observar como um documento desta natureza – informalmente escrito, mas rico em conteúdo – possa se tornar um recurso tão valioso em tão pouco tempo, recebendo em média 50 visitas diárias no site onde é hospedado.

Como resultado da minha participação no PyGTK, além disso, será lançada nos próximos meses a versão `0.6.12`, que consolidará a correção de diversos erros fatais da biblioteca, e oferecerá publicamente as extensões adicionais à API atualmente integradas no repositório CVS.

7.4 Kiwi

No contexto da minha experiência com projetos de software livre, Kiwi é um projeto especial por ser o primeiro projeto de software livre que iniciei a atingir um porte maior (possui mais de 8000 linhas de código Python).

Este projeto foi iniciado oficialmente em 2001, mas seu desenvolvimento mais intenso ocorreu no segundo semestre de 2002. No momento há um número considerável de alterações a ser implementadas que dependem de tempo disponível para integrar e testá-las. O projeto conta com documentação de referência completa, e um tutorial para implementação incluindo mais de 10 exemplos. Tanto código quanto documentação estão disponíveis a partir do endereço `www.async.com.br/projects/kiwi`.

Minha experiência com o Kiwi permite fazer uma observação subjetiva a respeito das vantagens deste processo: embora o investimento inicial em desenvolvimento e documentação tenha sido grande, a partir do lançamento público foi possível contar com a comunidade de usuários (mesmo que ainda restrita) para contribuir opiniões importantes sobre a evolução da arquitetura, teste e informes de erro detalhados, e inspiração e motivação para desenvolvimento continuado. A próxima seção faz uma breve descrição da origem do projeto Kiwi.

7.4.1 Um Comentário Histórico do Projeto Kiwi

Python, a ferramenta de desenvolvimento predominantemente usada na Async Open Source, é uma linguagem bastante expressiva; um artigo do seu autor Guido van Rossum[3] cita uma relação média de 3 a 5 linhas de Java (e 5 a 10 de C++) para uma linha de Python. Tendo em vista esta alta expressividade, seria bastante interessante a possibilidade de utilizar uma biblioteca gráfica que fizesse uso dos recursos

[2]Este documento está disponível online em `http://www.async.com.br/faq/pygtk`
[3]Disponível em `http://www.python.org/doc/essays/comparisons.html`

da linguagem para oferecer uma interface de alto poder de abstração e reuso.

Para desenvolvimento de aplicações gráficas em Python, existem algumas bibliotecas alternativas, incluindo Tkinter, wxWindows e PyGTK; esta última é utilizada em uma série de projetos da Async Open Source, e por este motivo foi um dos projetos com os quais pude tomar contato durante este período de mestrado. PyGTK tem um problema inerente que deriva da sua proximidade à biblioteca GTK+: apesar de ser visualmente agradável, e de oferecer um conjunto de componentes altamente flexível, a API da biblioteca é pouco expressiva e requer grande esforço de codificação. A extensão e complexidade do código de interface resultante prejudica sua manutenção, e o desenvolvimento de novas interfaces é dificultado pela quantidade de trabalho que requer.

Por este motivo, decidi implementar uma nova biblioteca que utilizasse o PyGTK internamente, mas que oferecesse ao programador uma interface completamente orientada a objetos. Através desta, seria possível criar janelas, diálogos e *slaves*, que nos termos usados no Kiwi são porções de uma interface gráfica *windowless* (sem janela própria), que podem ser reusadas em diversas interfaces distintas. Kiwi também oferece o conceito de um *visual proxy*, que é uma interface ao qual um objeto do domínio pode ser vinculada; alterações na interface são propagadas automaticamente para este objeto.

Kiwi foi implementado em duas fases principais antes do primeiro lançamento pleno. A primeira fase se caracterizou pelo desenvolvimento de componentes gráficos adicionais ao PyGTK, pouca ênfase sendo dada ao *framework* orientado a objetos em si. Todo o desenvolvimento foi feito individualmente, sendo testado por mim em aplicações piloto simples. Parte da equipe da Async Open Source ajudou a testar durante esta fase, sendo a primeira comunidade 'beta' a usar o software. Ao final deste período, importei o código-fonte para um repositório CVS, e uma versão preliminar foi lançada publicamente. Elaborei uma página Web simples para descrever o projeto, contendo alguns exemplos de código e hiperlinks para os pacotes de código-fonte. Esta versão foi utilizada internamente pela equipe da Async Open Source, mas recebeu pouco interesse externo; menos de 10 usuários escreveram para obter maiores informações, e apenas um usuário confirmou de fato estar usando o software.

Após um período, pude me dedicar ao desenvolvimento do projeto mais uma vez, e desenvolvi as classes do framework; durante este processo, também trabalhei escrevendo documentação para o usuário. Parte da motivação em fornecer documentação foi tentar incentivar o uso externo da biblioteca; o tutorial que a acompanha [70] oferece um exemplo da classe de documentação que pode ser oferecida por um projeto de software livre. Este processo culminou no lançamento da versão mais recente, 0.5.0, que é a primeira versão publica que considero funcionalmente completa.

Após este lançamento público alterei a página para incluir a documentação escrita para o projeto, e criei uma lista de discussão para o projeto. De fato, com a versão 0.5.0, interesse considerável pela biblioteca se despertou. Infelizmente, por se basear na versão estável do PyGTK (a versão 0.6), que está em processo de substituição por uma nova versão (PyGTK2), a maior parte dos usuários não pôde de fato testar e usá-la. Quando a biblioteca for atualizada para o PyGTK 2.0 é esperado que a comunidade de desenvolvedores cresça mais acentuadamente. Os usuários que de fato decidiram usar a biblioteca

mantém contato regular comigo, e algumas discussões sobre adição de funcionalidade e projeto se iniciaram recentemente.

Esta descrição oferece uma visão do processo inicial da criação de um projeto de software livre. Um resumo dos pontos principais envolvidos com o processo de software segue:

- O projeto se iniciou de maneira bastante informal, como um experimento para criar classes que estendessem o PyGTK. Foi motivado integralmente pelas necessidades identificadas pela minha equipe de desenvolvimento.
- Passei a usar o CVS logo antes do lançamento da primeira versão pública; anteriormente, não houve necessidade de controlar versões, em parte por estar ainda explorando a arquitetura e organização do código. A primeira versão integrada ao CVS possuía menos de 1.500 linhas de código.
- Todos os lançamentos foram enviados para a lista de discussão do PyGTK, o projeto do qual o Kiwi depende, e registrados no site `freshmeat.net`. Como o projeto possui sua própria lista de discussão, os próximos lançamentos serão também lá veiculados.
- O lançamento inicial foi de uma versão bastante reduzida do código, mas mesmo assim houve interesse público; um usuário externo passou de fato a se comunicar ativamente, demonstrando interesse pelo trabalho, e usando a ferramenta em um projeto próprio de sua empresa.
- Menos de 200 linhas de código foram contribuídas por usuários externos à Async Open Source, e todas estas foram alteradas radicalmente antes da sua integração.
- Existe uma pequena suíte de testes, mas não é automatizada pelo fato de envolver a manipulação da interface gráfica para verificar regressões. Antes dos lançamentos, todos os testes e exemplos são verificados, e teste funcional ad hoc é realizado. Alguma revisão de código é feita por outros desenvolvedores da comunidade e da equipe local.
- O interesse pelo projeto tem crescido gradualmente; nos últimos meses um número grande de usuários já solicitaram versões para o PyGTK2, e usuários ativos têm contribuído com diversos informes de erro e algumas alterações de código-fonte.
- É impossível determinar quantos usuários o projeto possui; embora existam 7 usuários membros da lista de discussão, aproximadamente 900 downloads foram realizados das 5 versões públicas lançadas até o momento. Por ser uma biblioteca usado em outros projetos, é possível que o número de usuários seja ainda maior.

Embora um projeto pequeno, na minha experiência, o Kiwi tem características iniciais bastante similares à maior parte dos projetos incipientes: comunidade reduzida, um autor responsável por escrever a maior parte do código, e um conjunto de ferramentas simples para suportar comunicação.

Capítulo 8

Resultados do Levantamento

A veiculação do questionário foi iniciada em setembro de 2002, e se estendeu até o mês de novembro deste mesmo ano. O objetivo do questionário foi levantar atividades executadas durante a produção de software livre, com base nas respostas preenchidas por um conjunto representativo de líderes e desenvolvedores ativos de projetos específicos. Este capítulo apresenta os resultados quantitativos obtidos através do questionário, e faz comentários discutindo a validade e as particularidades destes resultados.

É importante ressaltar que a base de dados que contém estes resultados será disponibilizada publicamente (excluindo os questionários marcados como privados) e poderá ser analisada e manipulada via a ferramenta Web do projeto[1].

8.1 Resultados Gerais

Foram enviadas 1.102 mensagens convocando para o preenchimento do questionário. As mensagens foram remetidas aos **responsáveis por projetos de software livre classificados como estáveis ou maduros**, conforme o critério discutido na Seção 6.7.1. Na Tabela 8.1 é detalhada a proporção de respostas.

Total de mensagens enviadas	1102	100%
Projetos que preencheram questionários	570	51%
Respostas válidas	548	49%
Respostas válidas excluindo meta-projetos e distribuições	**519**	**47%**

Tabela 8.1: Proporções de resposta obtidas no levantamento quantitativo. Esta proporção é alta considerados outros estudos realizados com esta mesma população.

Todas as respostas capturadas foram arquivadas no site Web do levantamento. Cada projeto possui um registro individual que permite que sejam constatadas suas respostas individuais e comentários. Um

[1] Estes resultados podem ser provisoriamente acessados a partir do endereço `http://www.async.com.br/~kiko/fsp/results.php`

exemplo de uma resposta está disponível como o Apêndice F.

Uma conquista especial deste levantamento foi a inclusão de um grande número de projetos de grande importância. Linux, Mozilla, Perl, Tcl, MySQL, diversos projetos GNOME e diversos projetos KDE estão representados e têm questionários públicos que podem ser revistos e avaliados independentemente. Esta contribuição é significativa porque resultou de grande esforço pessoal em contactar os líderes e incentivar que participassem[2].

8.1.1 Comparações a Outros Levantamentos

A proporção de respostas obtidas é comparável a outros levantamentos realizados com a mesma comunidade. Vale notar, no entanto, que os questionários mencionados nesta seção tinham como objeto de estudo *indivíduos* e não projetos.

Nas duas fases do levantamento BCG, de Lakhani e Wolf, descrito na Seção 5.3.7, um total de 2.221 mensagens foram enviadas, e 695 respostas foram obtidas, o que constitui um retorno de 31%.

No estudo FLOSS, outro estudo demográfico extenso descrito na Seção 5.3.6, não se utilizaram chamadas pessoais por correio eletrônico; ao invés disso, foram utilizados mecanismos de dispersão como listas de discussão e chamadas em sites Web importantes. O total é expressivamente maior: 2.783 respostas foram obtidas no total, mas é impossível determinar quantas pessoas entraram em contato com a chamada, e portanto a proporção de resposta é desconhecida.

Existe uma diferença notável em utilizar chamadas individuais: a solicitação passa a ser direcionada, e a seleção da amostra, portanto, clara e intencional. Nesta pesquisa de mestrado foram selecionados projetos específicos de acordo com os critérios estabelecidos na Seção 6.7.1.

Em contrapartida, pesquisas não-direcionadas (como no estudo FLOSS) têm chance de obter uma distribuição mais variada (ou inesperada) de participantes. Resta ser verificada, no entanto, a inexistência de tendências nas amostras obtidas desta maneira. Por exemplo, é possível que apenas indivíduos com tempo livre substancial tenham disposição para participar de questionários online que não lhes fossem direcionados especificamente.

8.1.2 Significância da Amostra

Não foi estabelecida empiricamente a representatividade da amostra deste trabalho. No entanto, é possível fazer uma estimativa do total de projetos de software livre nas categorias que buscamos observar, **assumindo que os sites de registro de projetos sejam representativos** da população de projetos de software livre.

- Existem aproximadamente 27.000 projetos registrados no `freshmeat.net`, dos quais 6124

[2] Um número de chamadas foi reforçada através de solicitações através do IRC, uma rede de comunicação pessoal. Na minha opinião este recurso foi importante para convencer alguns dos projetos principais a participar.

são considerados estáveis e 895 maduros (na classificação discutida na Seção 6.7.1). Em outras palavras, a população-alvo constitui pouco mais do que 25% do total de projetos registrados.

- No `sourceforge.net` a proporção de projetos é semelhante: 6557 projetos são categorizados como estáveis, e 674 como maduros. O total de projetos registrados é aproximadamente 57.000 projetos, de forma que a população-alvo representa uma proporção ainda menor do total de projetos: menos de 13%[3].

- Com base nestes dados, e assumindo, de maneira conservadora, que mais de 40% dos projetos esteja registrada em ambos os sites simultaneamente, é provável que a população total de projetos estáveis e maduros não passe de 11.000.

Com base nesta dedução, o número de respostas obtidas pode representar até 5% do total de projetos maduros e estáveis registrados, o que indica boa cobertura para um estudo empírico desta natureza.

8.1.3 Localização Geográfica dos Participantes

Um perfil de domínios Internet pode ser obtido a partir dos endereços de correio eletrônico dos participantes. Os 10 domínios mais recorrentes estão exibidos na Tabela 8.2:

Domínio	.com	.org	.net	.de	.edu	.uk	.fr	.au	.se	.nl
Participantes	128	119	83	42	23	14	14	11	10	9

Tabela 8.2: Proporção de participantes por domínio Internet a partir de seu endereço de correio eletrônico. É importante notar que os domínios `.com`, `.org` e `.net` são internacionais, e que o domínio não indica necessariamente o país de residência do desenvolvedor.

Uma observação pertinente é que os domínios genéricos `.com`, `.org` e `.net` não são mais vinculados a um país específico, embora originalmente correspondessem a sites nos EUA. Usuários com e-mails nestes domínios têm localização indeterminada. O domínio `.edu` corresponde a universidades nos EUA e Canadá. É provável que uma proporção dos participantes tenha endereço de e-mail distinto do seu país de localização; por exemplo, é comum que desenvolvedores dos países escandinavos tenham endereços no domínio `.nu`, por significar 'novo' em alguns idiomas da região.

Consideradas estas ressalvas, é possível observar o quanto a população avaliada é de fato internacional. Há uma concentração nos domínios genéricos, o que sugere – mas não garante – presença importante dos EUA. Também é notável que os países destacados são praticamente os mesmos que os descritos no trabalho de Dempsey, descrito na Seção 5.3.5, realizado em 1999.

É importante observar que o estudo FLOSS descreve uma população majoritariamente européia. É impossível determinar o quanto isso representa de fato a proporção mundial, mas devem ser destacados

[3] Estes números sugerem que o `sourceforge.net` hospeda grande número de projetos incipientes ou inativos: 11.490 projetos são descritos como 'em planejamento'

dois fatores que podem influenciar esta proporção. Primeiro, o levantamento FLOSS possuía perguntas específicas relacionadas ao local de residência do desenvolvedor em questão, determinando com muito melhor precisão a localização do desenvolvedor. Segundo, sendo realizado por uma instituição européia, o estudo pode ter um apelo maior à população da região por razões subjetivas.

Finalmente, vale ressaltar que apenas sete projetos na avaliação possuem alguma relação com o Brasil, dois destes sendo projetos da Async Open Source (Kiwi e IndexedCatalog), respondidos na fase de teste do questionário e mantidos nos resultados finais. O questionário para o núcleo Linux for respondido por Rik van Riel, um desenvolvedor holandês que trabalha na Conectiva, S.A., uma empresa brasileira dedicada ao desenvolvimento de uma distribuição. Os outros projetos mantidos por brasileiros são: `apt` for RPM e Windowmaker (Alfredo Kojima), UebiMiau (Aldoir Ventura) e The Lua Language (Roberto Ierusalimschy).

8.1.4 Domínio de Aplicação

O gráfico da Figura 8.1 descreve a distribuição dos projetos nas 16 categorias estabelecidas para os domínios de aplicação. É importante lembrar que a classificação não foi determinada pelo entrevistado, mas durante a coleta de dados secundária, conforme descrito na Seção 6.8.3.

A concentração em domínios relacionados à computação — desenvolvimento de software, redes e segurança — é um indicativo do tipo de usuário que os projetos têm. Esta concentração também faz referência à premissa de que o software tem no seu desenvolvedor seu primeiro – e muitas vezes mais importante – usuário. Pode-se fazer uma comparação destes resultados com o levantamento FLOSS, que reportou redes, serviços Web e aplicações de escritório como sendo as categorias de preferência dos desenvolvedores estudados.

Pode-se notar a presença forte de projetos dedicados a áudio e vídeo, o que pode ser um resultado da crescente digitalização e distribuição de mídias como música, na forma de arquivos MP3 e Ogg, e vídeo, na forma de arquivos DivX e DVD. Também é notável a presença de grande quantidade de aplicações de produtividade, possivelmente relacionada à inclusão dos projetos GNOME e KDE na amostra.

Outro ponto de destaque é o menor número de aplicações para engenharia e áreas associadas a negócios. A reduzida quantidade de projetos nesta área pode sugerir algumas barreiras: reduzida motivação para iniciar projetos desta natureza, ou a dificuldade de obter requisitos funcionais.

Não foi realizada uma validação desta classificação, de forma que não é possível afirmar que ela é generalizável a outras populações de softwares. No entanto, o fato de apenas 27 projetos (5%) terem sido classificados como Outros (*Miscellaneous*) sugere que a divisão ao menos tem boa cobertura das áreas predominantes. Um potencial trabalho futuro consistiria em validar esta classificação utilizando as categorias de domínio dos sites de registro de projetos utilizados para determinar a amostra, `freshmeat.net` e `sourceforge.net`.

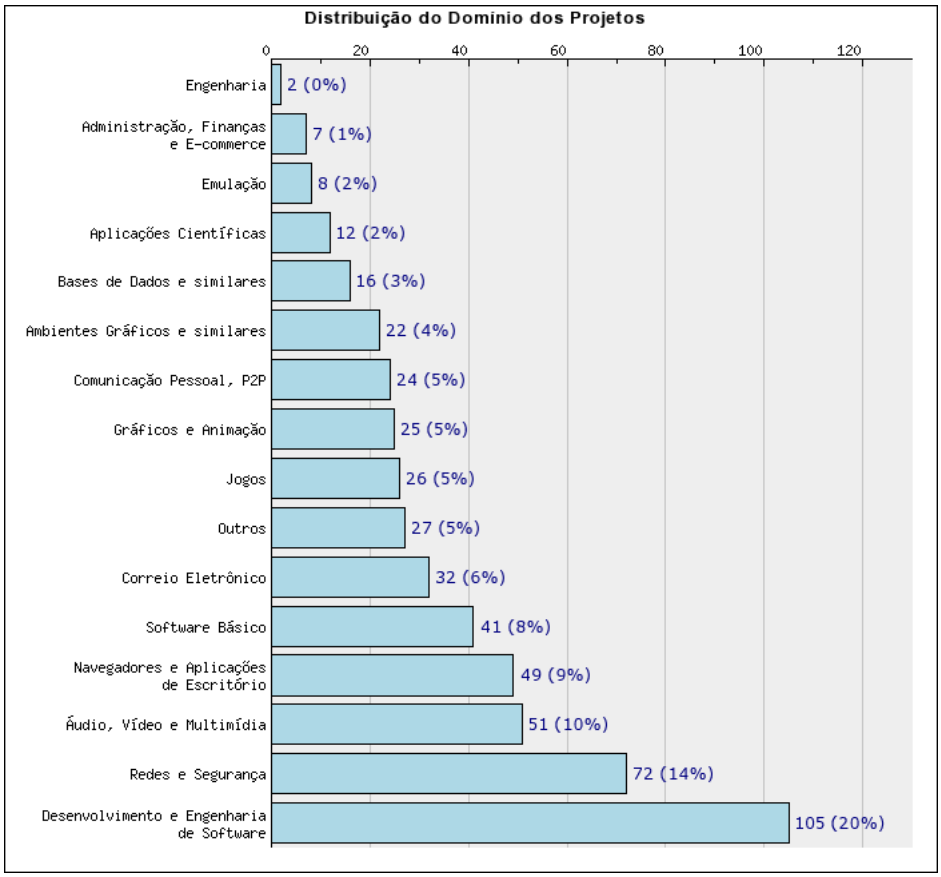


Figura 8.1: Um gráfico da distribuição dos domínios de aplicação entre os projetos participantes. Note a concentração nas áreas onde o desenvolvedor desempenha simultaneamente o papel de usuário.

8.1.5 Código Fonte

Diretamente da análise de código-fonte realizada para os projetos participantes, algumas médias podem ser obtidas. Note que 4 projetos com questionários completos não tiveram seu código-fonte avaliado por motivo técnico (dificuldade de obter o pacote, ou de determinar quanto código era de autoria do projeto), de forma que o total de projetos avaliado nesta contagem é 515.

Linguagens por projeto: O número médio de linguagens por projeto é de 3.2, o que não é surpreendente levando-se em consideração a tradição de utilizar ferramentas diferentes para cada tipo de tarefa. Por exemplo, um sistema de instalação baseado no `autoconf`, a ferramenta mais utilizada entre projetos de software livre, é constituído predominantemente de scripts `m4` e `sh`.

Alguns projetos se destacam pela grande quantidade de linguagens diferentes utilizadas: GRASS GIS (16 linguagens); Mozilla, Beonex e GCC (15 linguagens) e o XFree86 (14 linguagens). A maior parte dos pacotes (409 pacotes, ou 79%) inclui código com mais de duas linguagens.

A linguagem predominante entre os projetos é ANSI C, utilizado em 367 projetos do total. A

Tabela 8.3 detalha as 10 linguagens mais importantes entre os projetos desta pesquisa:

Linguagem	ansic	sh	perl	cpp	yacc	python	asm	awk	sed	java
Total de Projetos	367	342	204	170	87	74	50	47	45	43

Tabela 8.3: Distribuição de linguagens entre os projetos participantes. Note que a maioria dos pacotes possui código escrito em mais de uma linguagem.

Tamanho do Pacote: O tamanho médio do pacote avaliado é de 1.717 KBytes (KB), variando entre 58.496 KB para o maior projeto (o XFree86, versão 4.2.0) e 6 KB para o menor (qsubst, o único projeto do levantamento considerado 'morto' pelo seu autor).

A distribuição é bastante tendenciosa: apenas 14 pacotes (2%) possuem mais que 10.000 KB, e apenas 58 (11%) pacotes possuem mais de 3000 KBytes.

Linhas de código: A média de linhas de código (ou LOC, de *Lines of Code*) por projeto foi de 60.206 (ou ainda 60 KLOC, usando o multiplicador K). É importante ressaltar as limitações desta medição, conforme discutido na Seção 6.8.4.

Observou-se que o número de linhas de código está fortemente associado ao tamanho do pacote. O gráfico *scatter* da Figura 8.2 relaciona estas duas grandezas. A conclusão preliminar desta avaliação é que o tamanho do pacote reflete com boa precisão o número de linhas de código do projeto.

Do gráfico é possível prever que a distribuição de LOC medido será também tendenciosa, o que se confirma nos dados individuais. O LOC por projeto varia entre 2.945.597 LOC para o maior projeto (Linux, versão 2.4.19) e 103 LOC para o projeto SIPAG. Apenas 56 (10%) projetos possuem mais de 100.000 LOC, e apenas 95 projetos (18%) possuem LOC maior do que a média geral de 60.206.

Linhas de código ajustadas: além da contagem absoluta de linhas de código, realizou-se uma medida preliminar com base em uma tabela de conversão entre linguagens elaborada por Casper Jones (`www.theadvisors.com/langcomparison.htm`). Nesta tabela são listados índices que resumem a expressividade das linguagens; por exemplo, o índice para a linguagem C é 2.50, o índice para C++ e Java é 6. O índice base é para o Assembly, e vale 1.

Este índice foi utilizado apenas para verificar qual efeito teria sobre os resultados. Como nenhuma das correlações descritas na Seção 8.3 se modificou significativamente com seu uso, não estão detalhados aqui estes resultados.

A Tabela 8.4 lista os 20 maiores projetos do questionário, comparando LOC a LOC ajustado. Pode-se perceber desta tabela que quase não há variação na composição dos 10 maiores projetos quando comparado LOC a LOC ajustado — apenas suas ordens se alteram. Este fato resulta do

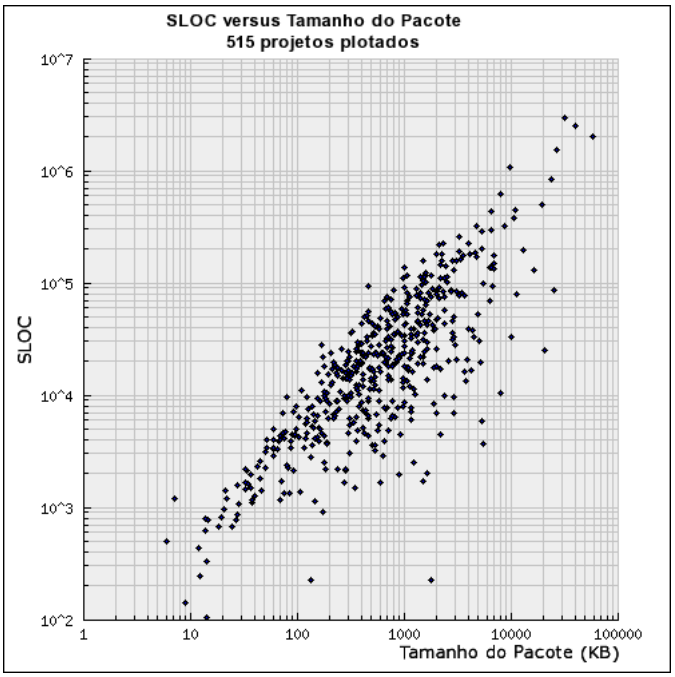


Figura 8.2: Um gráfico comparando o tamanho do pacote comprimido com o número de linhas de código (LOC) que este pacote contém. Note que as escalas são hiperbólicas. A concentração formada na diagonal do gráfico indica o limite da compressão do texto: pacotes que contém apenas código-fonte têm máxima relação entre LOC e o total comprimido em bytes. O corolário desta observação: projetos cujo pacote contém documentação, exemplos e mídia não-textual (como imagens) têm relação mais baixa entre LOC e o total comprimido, e portanto, estarão mais distantes desta diagonal principal.

uso predominante da linguagem C, e da dimensão e disparidade muito grande entre os maiores projetos. A partir da décima posição, a dimensão dos projetos se torna mais equilibrada e o ajuste faz um efeito maior.

Nesta lista, note que quatro dos 20 maiores projetos são compiladores ou interpretadores de linguagens de programação: GCC, Python, Perl, Tcl; dois dos 20 são emuladores: Wine e `xmame`.

Os resultados da análise de linhas de código indicam que o forte ritmo de crescimento identificado por Godfrey e Tu, descrito na Seção 5.2.2, continua presente no Linux: o núcleo atualmente soma quase 3 milhões de linhas de código, um aumento de mais de 30% em relação aos 2.200.000 (aproximadamente) existentes em Dezembro de 1999.

8.2 Resultados do Questionário

Ao total, o questionário veiculado contém mais de 70 itens individuais; considerando os 519 projetos avaliados, existe uma rica base de informação para se retirar dados quantitativos. Os totais estão apresentadas seqüencialmente como o Apêndice E, e nesta seção são descritas e comentadas as questões individuais.

	Ordem por LOC		Ordem por LOC Ajustado	
1	Linux Kernel	2945597	Mozilla	12043149
2	Mozilla	2509445	Linux Kernel	7224051
3	XFree86	2035366	GCC	6289217
4	GCC	1544643	XFree86	5866282
5	xmame	1085144	Perl	4716120
6	GRASS GIS	845149	python	3801566
7	Wine	624332	GRASS GIS	2844143
8	AbiWord	493657	xmame	2757436
9	Perl	444383	AbiWord	2202685
10	python	438435	TYPO3	2069610
11	MySQL	376239	MySQL	1985360
12	wxWindows	324497	Wine	1952081
13	GTK+	321189	wxWindows	1712446
14	Tcl	295550	doxygen	1279234
15	Blender	286359	GNUstep	1203096
16	MPlayer	259827	Blender	1191070
17	doxygen	225616	Tcl	1120949
18	CUPS	225163	Berkeley DB	1033353
19	TightVNC	215852	phpMyAdmin	836940
20	samba	200895	LyX	832236

Tabela 8.4: Uma comparação entre os 20 maiores projetos do ponto de vista de linhas de código (LOC) versus LOC ajustado pela tabela de equivalência de linguagens de Casper Jones. Apesar das ordens se alterarem, apenas 5 projetos estão presentes em apenas uma lista; se deve ao fato da maior parte dos softwares ser escrito predominantemente em C, e da grande dimensão dos projetos maiores. Note também como 4 dos projetos são linguagens de programação, e como há dois emuladores importantes na lista (Wine e `xmame`).

8.2.1 Motivações (Questão 1.1)

Esta questão discute os aspectos relacionados à criação do projeto. As opções oferecidas permitem que sejam detalhados os motivos que levaram à criação do projeto, e o apoio institucional fornecido. A Tabela 8.5 descreve os totais para as alternativas desta questão.

Motivação Pessoal: É inusitado o alto índice de projetos iniciados por motivos pessoais (71%). A maior parte dos projetos de software livre é iniciado porque o autor deseja ou necessita do software, não tendo contribuição significativa de uma organização formal. Mesmo uma parte dos projetos de maior sucesso, como o Linux, Perl, samba e Python, afirmaram ter início como projetos pessoais.

Em contrapartida, outra boa parte dos projetos famosos – Mozilla, MySQL, Blender, Berkeley DB, por exemplo – foram criados com significativo apoio institucional. O Blender, em particular, é um caso importante de uma conversão para software livre a partir de proprietário (Do comentário de Ton Roosendaal, seu líder: *'Rescue the software from a bankrupt company for the user community'*).

Projetos de Apoio: Entre os poucos projetos (15%) que indicaram terem sido criados para apoiar outro projeto de software livre, é interessante observar a presença do Bugzilla, Advanced Linux Sound

1.1. With respect to the initial motivations for starting this project, please mark ALL the sentences below that are true:

a.	The project was started primarily for personal reasons (had desire to learn, found technology interesting, etc).	369	(71%)
b.	The project was started primarily to produce software to support another Open Source/Free Software project	80	(15%)
c.	The project was started (or sponsored) by a company or organization.	69	(13%)
d.	The project was started as part of academic (university) work or research.	62	(12%)
e.	The project's intention from the beginning was to produce at least part of its software as Open Source/Free Software.	304	(59%)
f.	The project began based on pre-existing Open Source/Free Software (code fork, etc.).	113	(22%)
g.	The project began based on pre-existing proprietary/closed source software.	31	(6%)
h.	Other	59	(11%)
	(Sem resposta)	5	(1%)

Tabela 8.5: Motivações e aspectos da criação inicial do projeto.

Architecture (ALSA), e GTK+, que possuem motivos bastante diversos para sua existência. O Bugzilla, como descrito na Seção 6.5.2, foi criado para apoiar o Mozilla; o ALSA para oferecer uma plataforma de áudio para o núcleo Linux; o GTK+, para facilitar a criação do editor gráfico Gimp. Este resultado sugere que projetos de apoio representem uma parcela restrita, porém importante, dos projetos avaliados. Esta alternativa agrega diversos projetos notáveis, incluindo o Wine e praticamente toda a família de ferramentas GNU participantes do levantamento.

Intenção prévia em criar software livre: Há prevalecente intenção de criar o projeto já vislumbrando lançar software livre, indicada por 59% dos projetos avaliados. Este resultado sugere grande consciência entre a comunidade a respeito das vantagens e compromissos em assumir um projeto de software livre.

Projetos baseados em outros softwares: Um número considerável de projetos afirmou ter sido iniciado com base em outro projeto de software, a maior parte em software livre. Uma fraqueza das opções `(f)` e `(g)` é sua falta de especificidade; seria interessante obter dados mais precisos em relação ao reuso ou não do código-fonte. No entanto, a proporção é grande o suficiente para indicar um aspecto da natureza evolutiva dos projetos: novos projetos freqüentemente 'pegam carona' em projetos mais antigos e disparam um novo processo de criação, muitas vezes reutilizando o código-fonte original. Os grandes *code forks* (como são chamadas estas derivações voluntárias)

dos projetos EGCS[4], samba-TNG[5] e BSD[6] são exemplos mais visíveis desta ocorrência, mas o alto resultado nesta questão indica que ocorre em uma escala mais geral.

Outros: A maioria dos projetos que assinalou respostas personalizadas descreve uma motivação importante que as opções oferecidas não contemplavam: afirmaram ter sido iniciados porque inexistia um projeto de software livre com a funcionalidade necessária. Esta motivação está associada à opção `(a)`, mas é melhor descrita pela máxima de Eric Raymond descrita na Seção 5.3.1: *'Every good work of software starts by scratching a developer's personal itch'*.

Diversos participantes citam como motivação, nos comentários, desejar contribuir algo para o movimento de software livre (também foram citados 'open source' e 'Linux'). Seriam interessantes dados adicionais que especificassem esta motivação explicitamente; Richard Stallman, líder do projeto GNU, ao rever o questionário (após sua veiculação pública), sugeriu a inclusão de uma opção que indicasse desejo de contribuir para o movimento de software livre.

8.2.2 Perfil de Usuários (Questão 1.2)

1.2. Who are the main users for your project's software? Please mark ALL that apply.:

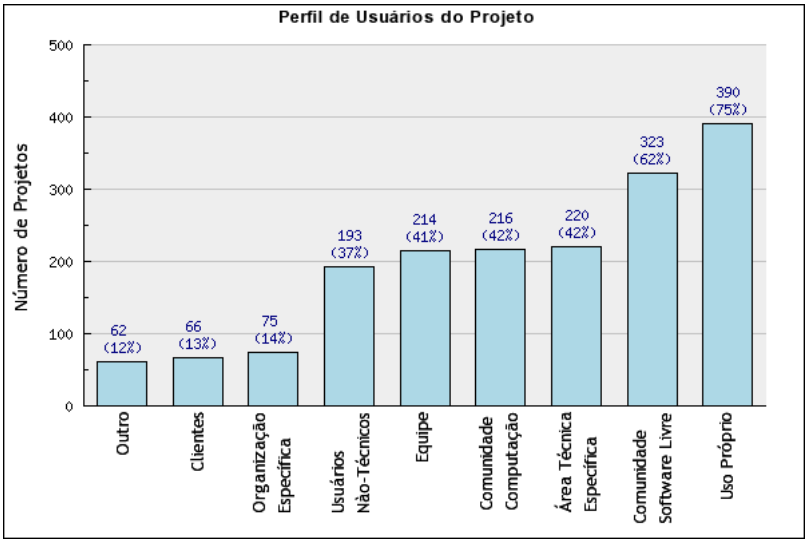


Figura 8.3: Um gráfico exibindo a proporção de projetos que identificaram um tipo de usuário particular. Note o destaque dado a uso próprio e comunidades técnicas.

Esta questão objetiva detalhar a base de usuários considerados importantes para o projeto. A Figura 8.3 oferece uma representação gráfica destes resultados.

[4]Ver `http://news.com.com/2100-1001-268658.html`

[5]Ver `http://lists.samba.org/pipermail/samba-ntdom/2000-October/029294.html`

[6]Um conjunto de projetos acabou se derivando do Unix BSD, inclusive o conjunto de Unix proprietários. Os derivados livres principais são o OpenBSD, NetBSD e FreeBSD.

De acordo com os resultados do domínio de aplicação apresentados anteriormente, a grande maioria dos projetos (75%) indicou ter o próprio autor ou líder atual como um usuário importante. Este dado confirma em particular a hipótese de Bart Massey descrita na Seção 5.3.1: requisitos podem originar-se diretamente dos desenvolvedores.

Também era esperada a predominância de usuários técnicos; esta concentração reflete a fama de 'sistema para hackers' que software livre possui. No entanto, um total de 110 projetos (21%) marcou pelo menos uma entre as opções `(e)` (Cliente) e `(f)` (Organização Específica), o que indica boa aceitação de software livre em situações mais convencionais.

Uma série de análises futuras pode ser realizada com estes dados. Por exemplo, seria interessante levantar quais projetos marcaram um número grande de alternativas, o que indica uma base variada de usuários. Realizar uma análise histórica deste tipo de dado seria também oportuno para prover parâmetros sobre a expansão de software livre para áreas de uso ainda pouco exploradas, e para verificar a evolução do uso por organizações tradicionais.

8.2.3 Idade do Projeto (Questão 1.3)

1.3. How long ago was the first public version of the project's software released?

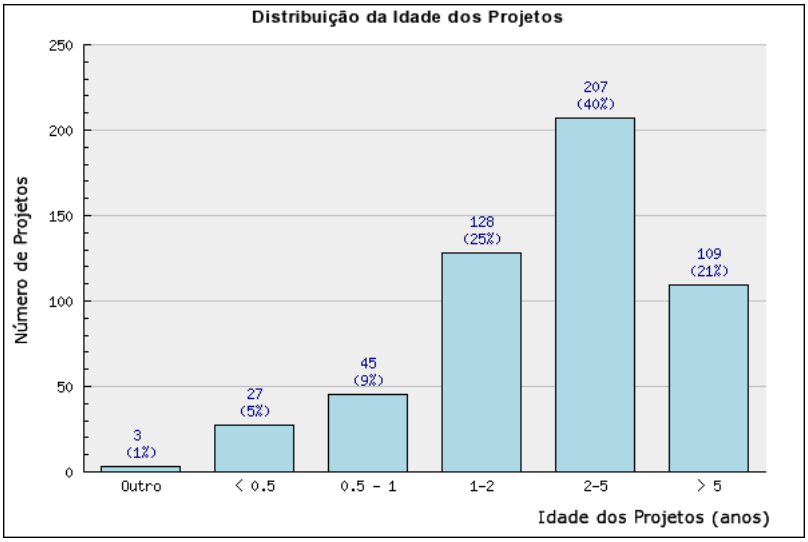


Figura 8.4: Gráfico dos tempos de lançamento da versão pública inicial do projeto – considerada nesta pesquisa como sendo a 'idade do projeto'. Note a relação esperada entre a população definida para o projeto e sua concentração resultante.

Esta questão busca levantar a idade do projeto, definida como o número de anos desde o lançamento da primeira versão pública do software. É importante ressaltar que esta medida é influenciada diretamente pelo critério de seleção da amostra. Não representa a proporção real dos projetos de software livre existentes — dados do `sourceforge.net` e `freshmeat.net` indicam ser *muito* mais fre-

quente a existência de projetos no estado inicial, alfa ou beta.

O gráfico da Figura 8.4 exibe o perfil de idades do projeto. Como esperado, a grande maioria dos projetos tem idades concentrando-se entre 2 e 5 anos. Os projetos de idade reduzida (menor que um ano) constituem uma minoria, indicando a presença de projetos de idade menor entre os projetos estáveis ou maduros do `freshmeat.net`, e entre os projetos ativos e populares provindos do `sourceforge.net`.

Alguns projetos descreveram idades particularmente longas nos comentários. Keith Bostic, do projeto Berkeley DB: '*First public release was probably around 1990.*'; Alexandre Julliard, do Wine: '*Wine will be 10 years old in a few months...*' (relativo a novembro de 2002); e Roberto Ierusalimschy, do projeto Lua: *'The first version Lua 1.1 was released in 1994. The first* ***public*** *version Lua 2.1 was released in 1995'*.

Em resumo, é importante notar a tendência clara que este gráfico indica quando interpretando os resultados deste levantamento: na amostra experimental predominam projetos já estabelecidos ou maduros, e de certa forma, com algum sucesso.

8.2.4 Tamanho da Equipe (Questão 2.1)

2.1. How many people would you estimate make up the project team? If in doubt, consider the project team is made up of the frequent and active contributors (including non-code contributors).

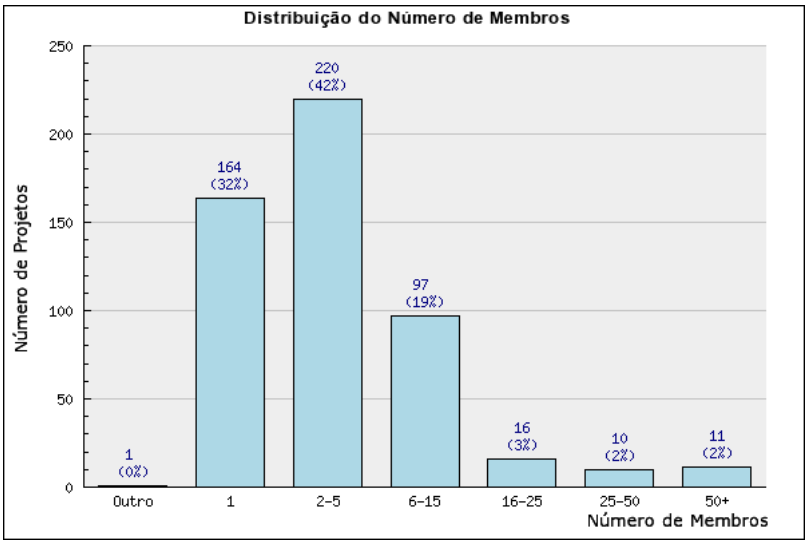


Figura 8.5: Gráfico detalhando a distribuição do tamanho da equipe entre os projetos de software livre analisados. Note o alto número de projetos com equipes com menos de 5 integrantes.

A dimensão da equipe é uma grandeza difícil de avaliar, mesmo para o entrevistado. Este problema deriva de uma incerteza semântica: o que exatamente significa 'ser membro da equipe do projeto'? O

questionário explicita que representam a equipe do projeto os indivíduos que contribuem regularmente, mesmo que não seja com código-fonte (ou seja, artistas e documentadores seriam também contados); no entanto, resta alguma ambiguidade relativa à freqüência e autonomia das contribuições.

O gráfico da Figura 8.5 indica a proporção dos tamanhos das equipes. Este resultado materializa grande parte das conclusões feitas por Krishnamurthy e os estudos Orbiten e FLOSS: os projetos possuem predominantemente uma equipe pequena, menor que 5 pessoas. Uma parcela considerável dos projetos (164, ou 32%) é mantida por uma única pessoa[7].

Dos projetos de uma única pessoa, existem alguns que possuem destaque público; LILO (Linux Loader), GnuWin32 e o GNU make são exemplos de projetos importantes que são mantidos apenas por uma pessoa. No entanto, é possível afirmar que a grande maioria dos projetos com base extensa de código possui equipes maiores. Dos 10 maiores projetos por linhas de código, apenas 3 possuem equipes menores de 25 pessoas – a maioria possui equipes de mais de 50 pessoas. Há uma discussão desta correlação na Seção 8.3.1.

Mais uma vez, é importante ressaltar que a dificuldade de determinar a dimensão exata da equipe tende a produzir alguma imprecisão na sua avaliação. Este fator influencia principalmente as categorias de dimensão maior, já que o líder tem mais dificuldade em recordar quantos desenvolvedores exatamente participam do grupo. Já antecipando esta situação, no projeto das opções de resposta do questionário foram usadas categorias maiores, com faixas de valores mais amplas. Para projetos de menor dimensão, é esperado um erro pequeno dada a facilidade de enumerar os envolvidos em um grupo reduzido. Um comentário de Geoff Kuenning, do projeto ispell, descreve esta dificuldade: *'It depends on how you count. Ispell itself has only one team member. However there are many independent dictionaries each with one or more maintainers. I would guess 16-25 active dictionary maintainers'*.

É interessante observar que o *único* projeto confirmado a indicar 'Outro' é o `qsubst`, um dos menores projetos em termos de linhas de código. Nas palavras de seu autor autor, Der Mouse, *'Zero is my best guess [...] As far as I know nobody has touched qsubst in ages.*'.

8.2.5 Modelo de Liderança (Questão 2.2)

Esta questão aborda o tema organização: como se estrutura o sistema de liderança do projeto. As categorias foram escolhidas com base em modelos distintos apresentados na literatura, incluindo os modelos 'ditador benevolente' de Raymond e o 'comitê de desenvolvedores' do projeto Apache.

A Figura 8.6 permite visualizar a distribuição dos diferentes modelos de liderança. Mais uma vez, é notável a predominância de projetos de pessoa única — neste caso, é um indicativo ainda mais forte da existência de um grande número de projetos de dimensão reduzida, já que esta resposta representa a estrutura de decisão do projeto. Em projetos de maior porte, no entanto, é natural que surja a necessida-

[7] É interessante ver este dado confirmado, visto as críticas feitas ao trabalho de Krishnamurthy por utilizar projetos provindos exclusivamente do `sourceforge.net`. Para uma discussão deste assunto entre a comunidade, ver `http://developers.slashdot.org/article.pl?sid=02/06/05/1530208&mode=thread&tid=156`

2.2. If you were to choose, what sort of leadership model best describes the project today?

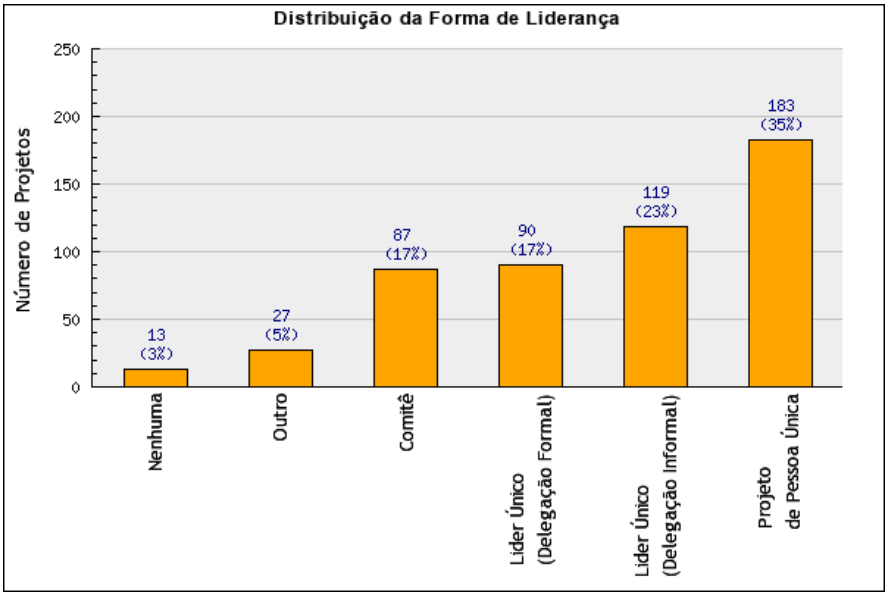


Figura 8.6: Este gráfico descreve as formas de liderança entre os projetos avaliados. Notar a predominância dos projetos de pessoa única, e o resultado equilibrado para as formas de liderança especificadas.

de de uma organização hierárquica, ainda que informal. Os aspectos mais importantes deste resultado são discutidos a seguir.

Hierarquia Predominante: Existe um relativo equilíbrio entre as diferentes formas de liderança, embora a hierarquia informal baseada no 'ditador benevolente' seja a mais utilizada. Uma explicação plausível para esta preferência é que requer o menor esforço de organização - é como um estado para o qual o projeto 'evolui naturalmente'.

As formas de organização mais complexas — líder com delegação formal de atribuições, e o comitê — são presentes em número praticamente idêntico. É interessante notar este empate: ambas as formas de organização requerem algum tipo de esforço organizacional do ponto de vista do líder, mas o comitê a princípio requer maior esforço de coordenação, como discutido a seguir:

- No caso do líder único com delegação formal, é necessário apontar e apoiar as pessoas que serão responsáveis pelos sub-sistemas ou tarefas particulares.
- Já no caso do comitê, é necessário determinar quais desenvolvedores participarão do núcleo de decisão interno, e de que forma serão resolvidas decisões e conflitos dentro do grupo. Roy Fielding, um dos membros do núcleo do projeto Apache, descreve as dificuldades em elaborar um sistema de liderança [47]: *'[...] we needed to determine group consensus,*

without using synchronous communication, and in a way that would interfere as little as possible with the project's progress[8]*'*.

Poucos projetos indicaram a opção 'nenhuma organização', o que sugere um forte consenso entre os participantes de que alguma estrutura de liderança precisa de fato existir, embora exatamente qual possa variar. É notável que os poucos projetos que selecionaram esta opção são, na sua maioria, projetos de porte menor: apenas 3 dos 19 projetos 'sem organização' indicaram possuir uma equipe com mais de 5 indivíduos.

Outros: A presença de um número razoável de respostas marcadas 'Outros' indica uma necessidade de analisar melhor as respostas individuais. Uma avaliação preliminar sugere que, para alguns projetos, as formas de organização dos projetos não sejam tão facilmente evidenciadas.

Um comentário que se destaca entre os personalizados é o de Rik van Riel para o projeto Linux. Não descreve a organização segundo o modelo 'ditador benevolente', como é tipicamente atribuído ao Linux, mas como *'Other: Single gate-keeper trying to fight off a storm of patches.'*. Seu comentário explica a colocação: *'There is a huge community of developers and an excess of new code and new ideas. Linux maintainers are mostly in the business of refusing new code in order to make sure the few things that do get in are of high enough quality to maintain for the following years to come'*.

Uma das respostas obtidas oferece um indício do tipo de população sendo avaliada: *"I do not like the word 'leader' "*. Como visto na Seção 3.3, individualismo é um das características usadas para descrever a comunidade de software livre.

Os resultados desta questão refletem uma das justificativas apontadas por Healy, descritas na Seção 5.3.5: a tarefa de organização é crucial, e ao contrário de um sistema caótico, há forte presença de liderança e hierarquia.

8.2.6 Aspectos da Equipe (Questão 2.3)

Esta questão é a primeira do conjunto que busca os aspectos chave do processo de software nos projetos de software livre. Os itens desta questão oferecem colocações a respeito da equipe do projeto, e o entrevistado deveria selecionar apenas os que se aplicassem ao seu projeto específico. Estes resultados estão dispostos na Tabela 8.6:

Distribuição Geográfica: O resultado imediato das opções `(a)` e `(b)` confirma a situação de distribuição geográfica: 62% dos participantes conhece os membros da equipe apenas através de Internet. Este resultado confirma o que se espera da população de desenvolvedores de software livre. No entanto, as seguintes observações precisam ser feitas a respeito do resultado.

[8] 'Era necessário determinar o consenso do grupo sem se apoiar em comunicação síncrona, e de uma forma que fosse interferir o mínimo possível com o progresso do projeto.'

2.3. With relation to this project's team and community, please mark ALL phrases below that apply.

a.	Most people in the team know each other only through the Internet, and have never met physically/personally.	323	(62%)
b.	Most people in the team work physically close, and meet personally with some frequency.	57	(11%)
c.	The team includes people that have more than 5 years experience in serious software development.	290	(55%)
d.	The project provides a high barrier of entry to new participants, even to those that are skilled in software development.	104	(20%)
e.	A contributor to the project will often participate in more than one project activity: functionality definition, architectural design, designing user interfaces , coding, testing, project management, etc.	208	(40%)
f.	Code contributors tend to work only on one specific language or technology used in the project (the one which they are most familiar with).	117	(22%)
g.	Other	46	(8%)
	(Nenhum selecionado)	35	(6%)

Tabela 8.6: Totais para as respostas da questão 2.3, relacionada aos aspectos da equipe do projeto.

- Primeiro, as opções indicavam duas situações opostas, o que não é uma boa prática considerando o formato escolhido para esta questão. De qualquer forma, poucos projetos indicaram ambas as opções, e Ian Hixie justificou esta escolha para o Mozilla da seguinte forma: *'a and b contradict each other but basically there's a large group that works together in one building and another group that has never met anyone'*.
- A natureza das opções força uma polarização grande de respostas. É possível identificar, através dos comentários, projetos que não selecionaram nenhuma das opções por habitar um espaço intermediário. A polarização da primeira opção em particular é problemática levando-se em consideração um fato apontado por muitos: os eventos internacionais realizados nos últimos anos tem contribuído para aproximar os desenvolvedores, e muitos agora se conhecem pessoalmente, embora não trabalhem juntos com freqüência. Este fator indica que a proporção de pessoas que trabalham de forma distribuída é ainda maior que os 62% indicados.
- A questão de distribuição requer interpretação adicional se avaliada a situação dos projetos com tamanho de equipe 1. De certa forma, o correto seria não assinalar a opção `(a)` para nenhum dos projetos; no entanto, avaliou-se que 9% do total de projetos assinalou esta opção. A explicação plausível é que para esta pergunta alguns projetos consideraram a equipe de uma forma menos rígida, levando-se em consideração participantes menos freqüentes.

Dadas estas ressalvas, a proporção de respostas indica positivamente a natureza dispersa da

população de desenvolvedores. O mais surpreendente destes resultados, na verdade, é o número relativamente alto — superior a 10% — para a opção `(b)`, o que indica que pelo menos uma parte dos projetos é desenvolvida de forma mais convencional.

Experiência do Desenvolvedor: Pouco mais da metade dos projetos participantes afirma ter na sua equipe desenvolvedores com mais de 5 anos de experiência. É interessante notar o quão equilibrado é este resultado se considerado o levantamento BCG. Neste estudo, foi relatada uma experiência média de 11 anos para os participantes. Se esperaria, portanto, uma proporção mais elevada de projetos com desenvolvedores experientes.

Não há dados descrevendo a distribuição de experiência dos participantes do estudo BCG; é possível que os 11 anos de experiência média sejam resultantes de uma parcela pequena de desenvolvedores com vasta experiência, e que a mediana seja mais baixa.

Atribuições: A alta proporção de respostas para o item `(e)` é indicativa de uma propriedade interessante nos projetos: em uma parcela considerável, os membros da equipe participam de múltiplas atividades.

Outros: Um número de participantes indicou o fato de ter apenas um desenvolvedor na equipe como uma justificativa para não assinalar nenhuma opção desta questão. Nas palavras de Bill Weinman, líder do projeto BW Whois: *'N/A – a team of one'*. Outros descreveram a situação de distribuição geográfica, como Ander Lozano Pérez coloca: *'Most people in the team know each other personally but all the work is done through the internet'*.

8.2.7 Requisitos (Questões 3.1 e 1.4)

A questão 3.1 apresenta um conjunto de alternativas que fornecem informações a respeito do processo de definição de requisitos dos projetos. Optou-se por discutir esta questão em conjunto com a questão 1.4, que cobre a existência de um padrão pré-existente. Os resultados das questões estão dispostos nas tabelas 8.7 e 8.8, respectivamente.

Uma análise combinada destes resultados oferece uma visão mais precisa da atividade de engenharia de requisitos para projetos de software livre. Os totais obtidos, e os comentários associados às respostas, oferecem os seguintes pontos para reflexão:

Requisitos Facilitados: É interessante observar como certos domínios abordados pelos projetos de software livre individuais possuem precedentes importantes. A visão combinada das respostas `3.1(a)` e `1.1(a)` é esclarecedora: 36% dos projetos replicam a funcionalidade de outro pacote de software, e 32% dos projetos possuem funcionalidade definida parcial ou totalmente por um padrão prévio. O total de projetos que respondeu a pelo menos uma destas opções é de 274, ou 52%. Em outras palavras, mais da metade dos projetos avaliados possui parte dos seus requisitos

3.1. This question aims to find out how the behavior and functionality of your software were defined. Please mark ALL phrases that apply to your project.

a.	The project's software explicitly mimics (or is significantly based upon) the functionality or behavior of an existing (proprietary or Free Software/Open Source) software package. If so, please specify which?	185	(36%)
b.	A significant amount of effort is spent defining what the software functionality and behavior (the requirements) should be.	222	(43%)
c.	This project has no end-users apart from the project team.	7	(1%)
d.	There have been meetings or discussions with end-users to define significant parts of the software functionality or behavior.	195	(38%)
e.	These meetings/discussions occur frequently, and can be considered an important part of defining the project's software.	97	(19%)
f.	The software functionality is implemented according to what the team thinks is correct, without significant external end-user input.	209	(40%)
g.	I believe much of the expected functionality and behavior is not completely known or understood by the project team as a whole.	50	(10%)
h.	Other	62	(12%)
	(Nenhum selecionado)	7	(1%)

Tabela 8.7: Respostas que descrevem as atividades relacionadas ao processo de requisitos nos projetos de software livre.

já descritos ou implementados. Este fator, em combinação com o alto índice de desenvolvedores usuários, ajuda a explicar melhor a ausência de um processo convencional de requisitos.

Esforço de Requisitos Alto: Mesmo tendo boa parte dos requisitos já descritos, é notável que boa parte dos projetos (43%) afirma despender esforço considerável na determinação de funcionalidade. Embora o real valor desta medida dependa da percepção subjetiva do que é um 'esforço significativo', é possível estabelecer algumas conjecturas que expliquem esta situação.

É possível, por exemplo, que de fato haja muito trabalho adicional para definir mais detalhadamente um padrão ou a funcionalidade de uma aplicação pré-existente. Também é possível que boa parte dos padrões sejam incompletos ou cubram apenas uma parte da definição de funcionalidade, o resto ficando a cargo da equipe do projeto. Por exemplo, Duncan Grisby, autor do OmniORB, afirma: *'The features are partly defined by the CORBA standard. Significant effort is spent in selecting standard features to implement and defining non-standard extensions'*.

Independência da Equipe: Outro resultado importante é o do item `(f)`. Uma parcela significativa dos projetos afirmou determinar internamente qual funcionalidade será implementada. Para um grande número de projetos, este é um aspecto chave: a equipe tem autonomia completa sobre a evolução do código.

Diversos comentários afirmam que isto não significa necessariamente que usuários não participam do processo de definição de funcionalidade: *'Users propose extensions that the development*

1.4. Was there a pre-existing standard or public specification previously available that helped define a significant part of the project? If so, please add a comment stating the most important of them. Example: HTTP (Internet RFCs 1945 and 2616) for Apache.

a.	Yes.	167	(32%)
b.	No.	319	(62%)
c.	Other	27	(5%)
	(Left unanswered)	6	(1%)

Tabela 8.8: Proporção dos projetos que possuíam ou não um padrão pré-existente.

team can accept/reject ' (ntop), *'A lot of our members (especially the leader ;-)) have very strong opinions about certain technologies and those will many times override wishes of end-users. But we also take end-user input very seriously and if it doesn't conflict with philosophy we will not stop anyone from implementing it.*' (Twisted), *"by selecting (d) (e) and (f) I mean the discussions do take place however I reserve for myself the 'veto' right. It rarely goes against a m ajority though ;-) "* (Grace). Confirma esta perspectiva a resposta alternativa oferecida por Juergen Vigna, pelo projeto LyX: *'The software functionality is implemented according to what the team thinks is correct* ***with*** *significant external end-user input'*.

Outros projetos que indicaram esta questão afirmaram não receber sugestões, mesmo interessados em tê-las. Muli Ben-Yehuda, do projeto syscalltrack, oferece uma explicação: '*We are very open to user input and user requests usually get implemented very quickly. Unfortunately we dont get many of those.*'.

Estabilidade de Projetos Maduros: Alguns comentários chamam atenção para o fato de projetos maduros serem aversos a grandes alterações de funcionalidade. Martin Pool afirma, sobre o projeto rsync: *'Because this is a mature and complex product choosing features that can be appropriately supported in the future is a significant part of the core team's work. We reject more patches than we apply for this reason.*' Outro participante, Paul D. Smith, responsável pelo GNU make, afirma ser difícil aceitar uma alteração de funcionalidade em um projeto com uma base ampla de usuários: *"Make receives a large number of 'suggested enhancements'. However I am extremely stingy when making any user-visible changes: I definitely prefer to err on the side of caution. GNU make is used with probably hundreds of thousands of makefiles and is a lynchpin for hundreds of free software projects and I'm very conscious of the need for backward compatibility and stability.*".

Usuários Finais: É notável que praticamente todo projeto afirmou ter usuários finais (opção `(c)`); é possível que em parte isto seja devido ao critério de seleção da amostra.

Evolução: Finalmente, embora seja relacionado mais ao projeto do sistema que seus requisitos, é par-

ticularmente relevante discutir o aspecto evolutivo apontado por uma série de comentários desta questão. Rik van Riel afirma para o núcleo Linux: *'The structure of the system evolves.'*, e elabora no seu comentário: *'Natural selection and criticism between developers quickly weeds out bad code. There is a high barrier of entry for new code and only bad code is just refused until it has been improved. There is no high-level goal for where the system should go in the future; every developer has his/her own goals and a compromise in code is found. '*. Esta colocação reflete muito precisamente a descrição de Moon e Sproull, presente na Seção 5.2.2, a respeito da evolução do núcleo. Este assunto foi recentemente discutido a fundo na lista de discussão (`linux-kernel`) deste projeto, com uma observação de Linus sendo citada[9]:

> "[...] Let's just be honest, and admit that [the Linux kernel] wasn't designed. [...] I know better than most that what I envisioned 10 years ago has *nothing* in common with what Linux is today. There was certainly no premeditated design there.
>
> And I will claim that nobody else 'designed' Linux any more than I did, and I doubt I'll have many people disagreeing. It grew. It grew with a lot of mutations - and because the mutations were less than random, they were faster and more directed than alpha-particles in DNA."

Outros comentários também projetam esta idéia de evolução contínua: *'The project evolves over time based on feedback from both users and developers.'* (Audacity); *'the goals of the project's functionality evolves as the project is developed'* (KStars); *'[...] There are several pieces of innovation or at least creative design in gentoo and the overall design is still evolving freely'* (Gentoo).

8.2.8 Usabilidade (Questão 3.2)

Das respostas da questão relacionada à usabilidade, disponíveis na Tabela 8.9, o item mais assinalado é o que indica ausência de interface com o usuário. A distribuição dos domínios de aplicação, discutidos na Seção 8.1.4, sugere que uma parcela significativa dos projetos avaliados não tem como objetivo atender a usuários finais. É provável que nestes domínios haja menor ênfase no aspecto de usabilidade. Levando isto em consideração, foi elaborada uma tabela alternativa que inclui apenas os projetos que afirmaram ter usuários não-técnicos (opção 1.2`(c)`). O tamanho desta amostra é de 193 projetos.

Mesmo nesta população reduzida, são notáveis os baixos resultados para as primeiras 4 opções, que descrevem aspectos comuns à engenharia de usabilidade: testes, protótipos de interface, e controle de alterações. É difícil avaliar a significância desta resposta além do fato das atividades terem pequeno destaque entre os projetos avaliados. O resultado parece aludir ao fato de parte desta população de

[9] A discussão completa está arquivada em `http://kerneltrap.com/node.php?id=11`

3.2. This question touches the issue of usability: what sort of approach and how much effort is placed upon making the software more usable (i.e. easier, more efficient and more pleasant to use). Please mark ALL that apply.

a.	The user interfaces for the project are designed (or prototyped) and refined before actually being implemented.	144	(28%)
b.	We have conducted serious usability tests and studies on the project's user interfaces.	34	(7%)
c.	Developers are not allowed to implement or change the user interfaces before the implementation/change has been reviewed and approved.	61	(12%)
d.	A part of the team is specifically in charge of UI design.	41	(8%)
e.	We would like to invest more in usability, but we are hampered by lack of documentation and/or team knowledge on the subject.	59	(11%)
f.	This project doesn't have any significant user interface.	156	(30%)
g.	Other	117	(23%)
	(Nenhum selecionado)	50	(10%)

Tabela 8.9: Totais para a questão relacionada à usabilidade, levando-se em consideração a população total de projetos. A significância destes resultados é reduzida considerando que grande parte dos projetos não objetiva usuários não-técnicos, e uma nova tabela foi elaborada, a Tabela 8.10, que descreve estes totais para uma população mais restrita.

desenvolvedores ter pouco domínio da área de Interação Humano-Computador — certamente a opção `(e)` o indica — mas é dificil concluir mais precisamente, mesmo analisando os comentários fornecidos.

Definitivamente, um conjunto de projetos percebe a necessidade de melhorar a interface com o usuário: '*Usability and user interface is a definite weakness.*' (FlightGear), '*The media player and other GUI programs on top get relatively little UI design work but need a lot more.*' (GStreamer), '*We would like to invest more in usability but we are hampered by* ***lack of developers****.*' (GRASS GIS). Mas devido à própria natureza distribuída de um projeto de software livre, é provável que problemas de usabilidade raramente venham a ser reportados, já que normalmente os usuários que possuem maior dificuldade são os com menor conhecimento técnico e portanto, menor disposição para entrar em contato com o autor do software.

Uma parcela pequena dos projetos aparenta tratar usabilidade com destaque entre os seus objetivos. Um exemplo é o Ion, um gerenciador de janelas. Tuomov Valkonen, seu autor, afirma: *'The user interface is the most significant portion of this project the rest is just details. The idea of the project is to design more usable user interfaces from the point of view of the developer and possibly other experienced users not newbies.*'. Owen Taylor, mantenedor da biblioteca gráfica GTK, descreve a situação para seu projeto: *'Another form of prototyping is simply that most UI behaviors in GTK+ are more or less copied from an existing system. Radical innovation is a bad thing in something as fundemental as the UI toolkit'*.

Talvez o comentário mais peculiar para esta questão venha de Larry Wall, autor do Perl (que, iro-

a.	The user interfaces for the project are designed (or prototyped) and refined before actually being implemented.	73	(38%)
b.	We have conducted serious usability tests and studies on the project's user interfaces.	19	(10%)
c.	Developers are not allowed to implement or change the user interfaces before the implementation/change has been reviewed and approved.	29	(15%)
d.	A part of the team is specifically in charge of UI design.	22	(11%)
e.	We would like to invest more in usability, but we are hampered by lack of documentation and/or team knowledge on the subject.	27	(14%)
f.	This project doesn't have any significant user interface.	32	(17%)
g.	Other	48	(25%)
	(Nenhum selecionado)	15	(8%)

Tabela 8.10: Totais para usabilidade, considerados apenas os projetos que afirmaram possuir usuários não-técnicos; o total de projetos desta amostra é de 193.

nicamente, não é uma aplicação, e sim uma linguagem): *'This project is a 15 year experiment in user-interface design.'* . A observação cabível é que para grande parte dos projetos, usabilidade é avaliada sob a perspectiva do desenvolvedor.

8.2.9 Documentação (Questão 3.3)

A Tabela 8.11 descreve os resultados para a questão associada à documentação. Um conjunto de alternativas desta questão tem índices muito altos, o que indica que a comunidade considera importante uma parcela dos tipos de documentação descritos. A respeito destas alternativas, alguns comentários podem ser realizados:

Documentação Funcional: Observa-se das respostas um prevalecente esforço em documentar o software do ponto de vista funcional. Estes dados refletem o grande número de projetos independentes de documentação existente. O maior destes é o LDP (`www.linuxdoc.org`) que centraliza um número grande de documentos descrevendo pacotes de software normalmente incluídos nas distribuições.

É importante ressaltar que existe forte probabilidade da opção `(a)` ter sido interpretada de forma a significar qualquer descrição funcional, inclusive guias para o usuário final. O conceito de um 'documento de requisitos' parece raro entre a comunidade de software livre, embora, na minha experiência, existam especificações formais descrevendo partes restritas de alguns sistemas livres mais complexos. Estas especificações normalmente objetivam garantir interoperabilidade entre projetos, como é o caso dos padrões publicados pelos meta-projetos KDE (KOM, KParts) e GNOME (Bonobo).

Documentação de Projeto: Aproximadamente um terço (30%) dos participantes afirmou possuir al-

3.3. In this question I try to determine information on documentation produced in/for the project. Please mark ALL that apply.

a.	We have produced documents that describe at least some of the expected functionality and behavior (requirements) of our software.	361	(70%)
b.	The project produces and maintains documents for developers that describe how the code is organized (architecture), and/or how parts of it work.	156	(30%)
c.	There is a reasonably complete coding standards guide that is actively followed by the team.	124	(24%)
d.	There is documentation for the end-user available for the project's software (consider also third-party documents available).	403	(78%)
e.	End-user documentation is provided to a large extent by people or groups external to the project team.	61	(12%)
f.	A significant part of the available documentation is frequently updated and revised to be up to date.	284	(55%)
g.	Other	39	(8%)
	(Nenhum selecionado)	12	(2%)

Tabela 8.11: Totais para documentação. Note a alto frequência de respostas, especialmente entre os itens referentes à documentação funcional.

guma documentação de projeto. É impossível, com base apenas nesta afirmação, definir quanta documentação de projeto de fato existe. No entanto, o termo 'UML' não é citado em nenhum dos comentários, e na minha experiência pessoal existe pouca documentação formal de projeto. Andrew Clausen, do projeto GNU Parted, faz uma observação que considero representativa: *'Parted development has been very iterative. The initial requirements and architecture documents didn't look very far into the future. Architectural changes are usually discussed on the list but not documented formally. After they are complete the API documentation is updated.*'. David Gilbert, do projeto JFreeChart, faz um comentário semelhante: *' Most of the time we design by coding (often the code is refined later)'*.

É provável que grande parte do trabalho de descrever o projeto fique a cargo do próprio código-fonte, conforme Barry Warsaw indica para o projeto GNU Mailman: *'I believe the architecture of the system is well documented albeit mostly in lavish code comments.*'. Quando considerando documentação para desenvolvedores, é sempre importante levar em consideração este aspecto peculiar de software livre: o código-fonte passa a ser um recurso valioso para o aprendizado, permitindo que a arquitetura do projeto seja avaliada e compreendida por um conjunto grande de pessoas, o contrário do que ocorre com software proprietário.

Vale notar também a existência de projetos acadêmicos para documentar a arquitetura de alguns projetos através de re-engenharia com base em processamento automatizado de código. Michael W. Godfrey e Eric H. S. Lee identificam a arquitetura do Mozilla em seu trabalho 'Secrets from the Monster: Extracting Mozilla's Software Architecture' [82], e Ivan T. Bowman, Richard C.

Holt e Neil V. Brewster descrevem a arquitetura do núcleo Linux em 'Linux as a Case Study: Its Extracted Software Architecture' [83].

Atualização Frequente: Conforme indicado na tabela, grande parte dos projetos realiza um esforço significativo de atualização da sua documentação. Esta atividade é freqüentemente realizada durante o processo de lançamento público de versões principais, conforme indicado por um número de comentários: *'End user documentation is always updated before issuing a new release containing functional changes'* (KAlarm); *'Documentation is updated only at major releases'* (Guikachu); *'The graft doco and man pages are updated with every release'* (graft).

Outros: Além destes aspectos principais, é notável a grande variabilidade dos comentários em relação à documentação. Enquanto grande parte dos projetos afirma não possuir documentação suficiente; no entanto, alguns projetos possuem literatura extensa: *'There's a shelf-and-a-half of Python books in my local Barnes-and-Noble bookstore.'* (Python), *'There is a forthcoming O'Reilly book ...'* (subversion) e *'There are many books about MySQL in many languages.'* (MySQL) são exemplos.

Como a maior parte das atividades nos projetos de software livre, documentação parece resultar de uma necessidade percebida pela equipe do projeto, mais do que de um padrão ou requisito de processo implícito. No entanto, parece ser uma atividade bastante valorizada, o que explica a existência dos projetos independentes de documentação (e evidenciado em parte pelos 12% que indicaram o item `(e)`). Jeff Freedman, do projeto Exult, exemplifica: *'A couple [of] members have done extensive documentation which I feel is one of the most important contributions.'*.

Em retrospectiva, é possível verificar o quanto alguns dos itens desta questão são vagos. Parte desta falta de definição resultou da dificuldade em elaborar alternativas que pudessem estabelecer de fato que nível de documentação o projeto oferece. Tendo em vista esta dificuldade, uma proposta de tratamento destes dados seria validar este conjunto de respostas através de uma avaliação manual da documentação oferecida por uma amostra selecionada a partir dos projetos participantes deste levantamento. Com base nesta confirmação, seria possível confirmar de fato o quanto as atividades descritas resultam em documentação efetiva.

8.2.10 Garantia de Qualidade (Questão 3.4)

A questão de garantia de qualidade contém um conjunto de opções relacionados a teste, política de integração, e qualidade percebida. As respostas para esta questão estão sumarizadas na Tabela 8.12.

Os resultados desta questão indicam existir muito pouco processo sistemático em relação à qualidade. Paradoxalmente, a maior parte dos projetos indicou lançar publicamente software apenas quando testado extensivamente. Uma discussão mais detalhada destes resultados segue:

3.4. This question covers activities that are performed with the intention to assure or improve the quality of the software produced by your project. Please mark ALL that apply.

a.	There is an [automated] test suite for the project's software, that is used to validate it.	141	(27%)
b.	There is a test plan (a written document describing tests) for the project's software, that is used by the project team.	51	(10%)
c.	Periodic (i.e. nightly, weekly) snapshots of the project's code (or binaries) are distributed and used as a significant means of testing the software.	140	(27%)
d.	There is an active code review process, where code is read by other members of the team.	119	(23%)
e.	The project team members have formal rules for integrating code changes into the main codebase, and review is a strict requirement.	78	(15%)
f.	Usually, an unexpected amount of bugs are discovered after a version has been publically released (this includes public release candidates).	107	(21%)
g.	There is a tendency (or policy) to release a public version only when it has been extensively tested by the team.	289	(56%)
h.	Other	72	(14%)
	(Nenhum selecionado)	42	(9%)

Tabela 8.12: Totais da questão relacionada à qualidade. A opção mais selecionada (`(g)`) descreve a política prevalecente em lançar versões públicas que tenham sido testadas e 'homologadas' pela equipe, mas que as alternativas restantes indicam existir pouco processo sistemático.

Teste: De certa forma, é surpreendente que tão poucos projetos tenham indicado possuirem suítes de teste. Normalmente, espera-se que em projetos de software livre as ferramentas façam boa parte da tarefa de mediar políticas e gerenciar conflitos, como discutido na Seção 5.3.2. No entanto, para o aspecto de teste sistemático, esta propriedade parece não se confirmar.

Observando-se a lista de projetos que afirmaram positivamente possuir suite de testes, verifica-se a reduzida presença de projetos com interface predominantemente gráfica; a maior parte dos projetos que indicaram ter suítes de teste são ferramentas de sistema, linguagens e bibliotecas. É possível que parte da dificuldade em implementar suítes de teste derive do fato de muitos softwares oferecerem interface gráfica, cujo teste sistemático é normalmente mais dificil [84]. De qualquer forma, é evidente que grande parte dos projetos se beneficiaria de melhor automação e cobertura de testes.

A relativa ausência de planos de teste é outro fator a ressaltar. Considerando a postura pouco formal dos projetos de software livre, é de se esperar que um documento tradicional de testes tenha pouco apelo. É possível também que o conceito de um 'plano de testes' seja algo relativamente desconhecido, já que é pouco citado nos comentários.

É possível notar dos comentários a grande confiança em teste funcional realizado pelo usuário final: em outras palavras, no chamado 'teste beta'. Um número de projetos afirmou utilizar como

parte importante do processo de qualidade um sistema de testes pré-lançamento, usando *release candidates* (como descrito na Seção 6.8.4), versões alfa e beta, e CVS público. Jacek Sieka, do projeto DC++, diz *'The public is the beta test team'*.

Revisão de Código: Também foram baixos os resultados para as duas alternativas relacionadas ao controle de alterações. Esta é particularmente inesperada, dado o aparente consenso em torno do preceito de Raymond sobre revisão fornecido na Seção 5.1.1. Aparentemente, projetos de maior porte utilizam de fato este recurso: Berkeley DB, GCC, Apache, Bugzilla, CUPS, Gnumeric; a grande maioria, no entanto, não afirmou ser importante esta atividade para o processo de qualidade.

Uma explicação para este baixo resultado pode estar relacionada à dimensão da equipe – como visto na Seção 8.2.4, a maior parte dos projetos *tem* equipes de dimensão reduzida. No entanto, mesmo observando projetos com equipes de 2 a 5 indivíduos, esta baixa proporção se mantém: destes 239 projetos, apenas 54 projetos, ou 23%, afirmaram fazer revisão de código, e apenas 33, ou 13%, afirmaram utilizar uma política formal de revisão pré-integração.

Qualidade Pré-lançamento: Desta questão, o item mais assinalado afirma que lançamentos públicos ocorrem apenas após teste extensivo, indicado por 56% dos participantes. Dado que as atividades sistemáticas de teste oferecidas no questionário foram assinaladas apenas por uma minoria, esta medida pode refletir apenas duas situações:

- Como a alternativa requer uma avaliação subjetiva do quanto significa 'teste extensivo', é possível que os respondentes tenham interpretações distintas para o termo. Além disso, por fatores psicológicos, pode ser mais atrativa a alternativa que indica a realização de teste.
- Os processos de teste dos projetos não são baseados nas atividades tradicionais de garantia de qualidade; por exemplo, é possível que grande parte do teste realizado seja ad hoc, ou utilizando teste beta maciço.

Se avaliado o resultado do item `(f)`, poucos projetos afirmaram descobrir um número inesperado de erros após o lançamento público. Além disso, os projetos que assinalaram opção `(g)`, apenas 36, ou 12%, indicaram a opção `(f)` também. Novamente, é possível uma interpretação subjetiva deste item; no entanto, sugere que o nível de qualidade esperado corresponde, de fato, ao resultante após o lançamento. Em outras palavras, a equipe do projeto parece ter *boa consciência* da qualidade do seu produto.

Outros: Como citado acima, uma parcela dos projetos afirmou na opção personalizada usar teste alfa e beta como uma forma importante de garantia de qualidade. Alguns exemplos: *"I release Alpha, Beta and Release-Candidates for user testing & bug reporting prior to new 'stable' releases"* (Nmap); *'Public Beta Testing'* (phpBB); *'Lots and lots of betatest versions'* (elvis).

Outra parcela indicou realmente usar um processo pouco formal de testes: *'Testing is usually ad hoc'* (SalStat); *'We only do random tests. Not so many bugs are reported after release.'* (Cannon Smash); *'Our testing is informal although we have *some* JUnit tests defined.'* (JFreeChart).

Em menor proporção, foram feitos comentários sobre a freqüência dos lançamentos públicos (Hibernate: *'Very frequent public releases (weekly or monthly) help us spot bugs early'*), e sobre a estabilidade que certos projetos já alcançaram (rp-pppoe: *'Software is very stable and has not changed much lately'*). Apenas uma minoria citou lançamentos freqüentes, embora seja uma das outras premissas apontadas por Raymond.

É importante destacar a grande variabilidade de resposta para esta questão. Embora a maioria dos projetos tenha realmente indicado possuir reduzido processo sistemático com relação à qualidade, um conjunto pequeno, mas significativo, afirmou realizar boa parte das atividades acima, tendo indicadas todas as opções entre `(a)-(e)` os seguintes: Berkeley DB, Common UNIX Printing System, Compiere, GNU Visual Debugger, HTMLDOC, Jakarta Avalon, Mozilla, Perl.

Os resultados também sugerem que mecanismos sociais e psicológicos tenham impacto significativo na qualidade do software, como discutido nas seções 5.2.4 e 5.3.2. Este dispositivo é exemplificado pelo comentário de Charles Cazabon, mantenedor do getmail, um sistema de transferência de correio eletrônico: *"I will not release a new version of getmail until I have tested it extensively myself. A motto stolen from D. J. Bernstein would be 'Reliability means never having to say you're sorry'. I refuse to risk the loss of others' mail."*.

8.2.11 Ferramentas (Questão 3.5)

Esta questão busca detalhar que tipo de ferramenta o projeto de fato usa para apoiar seu desenvolvimento. Os resultados, exibidos graficamente na Figura 8.7, constituem uma visão do tipo de ferramentas utilizados nos projetos.

Este resultados confirmam as observações de Yamauchi et al., apresentadas na Seção 5.3.3, que discutem o uso de ferramentas simples entre os projetos. Resultam da análise deste gráfico um número de conclusões:

Uso intenso de controle de versão e listas de discussão: Conforme apresentado na literatura, e em particular por Hoek, Asklund e Bendix na Seção 5.3.4, os resultados caracterizam as ferramentas mais utilizadas em software livre: sistemas de controle de versão e as listas de discussão.

O destaque dado aos sites Web requer uma análise maior: embora apenas 348, ou 68%, dos projetos tenha afirmado utilizar um site Web, virtualmente todos os projetos apontaram um endereço Web na etapa de registro do projeto no questionário. Esta aparente discrepância se explica por uma ambiguidade resultante do enunciado da questão, que especifica ferramentas que são usadas para *desenvolvimento*. Em outras palavras, muitos projetos participantes possuíam sites Web mas

3.5. Select ALL of the software or communication tools that you actively use during development (don't select it if is available but rarely or never used):

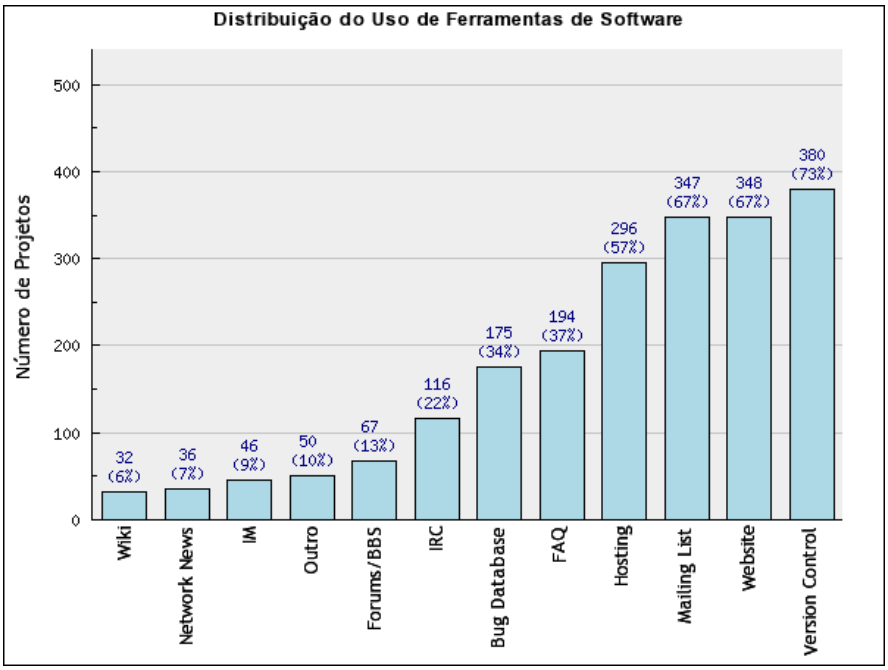


Figura 8.7: Resultados da questão ligada às ferramentas usadas no projeto. Se destaca o uso do CVS e de listas de discussão, que reafirma o descrito na literatura. Os dados que apontam website são pouco válidos, levando-se em consideração virtualmente todos os projetos ter um website próprio.

não o indicaram por não ser importante para o desenvolvimento, como é o caso de um site apenas para distribuição de lançamentos públicos.

Serviços de Hospedagem: Também deve ser destacado o alto índice de projetos que utilizam serviços de hospedagem. Parte deste destaque certamente deriva do fato de uma parcela dos projetos ter sido selecionada do `sourceforge.net`; no entanto, o número indicado é maior do que a quantidade de projetos selecionados deste serviço. Este resultado indica o valor que estes serviços oferecem aos projetos.

Sistemas de Controle de Alterações (Bugs): uma parcela significativa (175, ou 33%) dos projetos afirmou utilizar um sistema de controle de alterações. Em parte isso pode ser explicado pela presença destas ferramentas nos serviços de hospedagem, mas é indicativo de um uso crescente deste tipo de ferramenta. Há cinco anos, poucos sistemas desta natureza existiam em software livre. Recentemente, no entanto, diversos projetos diferentes desta natureza têm surgido; o Bugzilla exemplifica uma ferramenta que está se firmando como um padrão entre a comunidade, sendo usada pelos meta-projetos Mozilla, GNOME, KDE e Apache, e mesmo entre organizações formais como a NASA, o MIT, a Redhat e a Conectiva.

Outros: Há uma notável polarização das respostas desta questão. Embora as ferramentas anteriormente descritas sejam utilizadas com frequência, há um grande número que tem uso bastante restrito. Por exemplo, há baixa frequência de sites Wiki (que permitem autoria colaborativa de documentos Web), e também pouco uso de sistemas de comunicação em tempo real, entre estes predominando o uso do IRC, exemplificado pela rede principal dedicada a software livre `freenode.net`.

Note o pouco uso de correio eletrônico via Usenet (NNTP), que há uma década atrás era um padrão de fato para comunicação técnica.

É essencial reconhecer que a questão reflete ferramentas usadas *para desenvolvimento*, de forma que o fato do participante não selecionar o item não significa que ele não seja usado em absoluto. É possível que a baixa frequência destas ferramentas indique que as ferramentas de fato assinaladas constituam o 'toolbox' do desenvolvedor de software livre: correio eletrônico, controle de versões e possivelmente uma ferramenta de controle de alterações.

Esta questão é uma das mais uniformes entre as do questionário, havendo poucos comentários e respostas personalizadas. A maior parte dos comentários descreve o uso de uma ferramenta particular, ou justifica a sua ausência pela dimensão reduzida da equipe: *'single maintainer so little communication tools needed'* (afio), *'The userbase isnt́ large enough to justify the use of large (costly-to-maintain) systems by myself.'* (Eclipse/PHP).

Serviços de Hospedagem de Projetos

Dado a proporção alta de projetos que indicaram utilizar um serviço de hospedagem de projetos, foi realizado um levantamento simples com base nos endereços Web providos pelo usuários para cada projeto. O resultado dos 5 domínios mais utilizados para os sites Web está exposto na Tabela 8.13:

1	`sourceforge.net`	140	27%
2	`gnu.org`	15	3%
3	`gnome.org`	7	1%
4	`kde.org`	6	1%
5	`free.fr`	5	1%

Tabela 8.13: Um resumo dos domínios Web dos sites utilizados pelos projetos participantes do questionário. Note o número elevado de projetos cuja página é hospedada no `sourceforge.net`

É importante ressaltar que esta avaliação é feita apenas utilizando o domínio da página Web do projeto; muitos projetos possuem domínios próprios e utilizam serviços de hospedagem (um exemplo é o projeto Python, que hospeda seu repositório CVS e registro de bugs no `sourceforge.net`). Além disso, vale lembrar que uma parte da população de projetos identificada foi obtida diretamente das listagens do `sourceforge.net`.

Feitas estas ressalvas, é óbvio pela tabela a importância que este serviço de hospedagem tem para a comunidade. O `free.fr` é um serviço de hospedagem gratuito francês, utilizado por diversos projetos de software livre.

Outros serviços de hospedagem não listados nesta tabela incluem o do Apache (`www.apache.org`), que hospeda o servidor Web e outros projetos importantes, e o Tigris.org (`www.tigris.org`), que hospeda um conjunto de projetos associados a engenharia de software, incluindo o Subversion.

8.3 Agrupamentos e Correlações

Além dos resultados obtidos diretamente das freqüências de resposta, foram realizados agrupamentos entre algumas das grandezas do projeto. Esta análise não é exaustiva, e apenas demonstra algumas correlações interessantes que foram observadas durante a análise principal.

8.3.1 Tamanho da Equipe e Idade

O gráfico da Figura 8.8 demonstra de forma combinada as distribuições de equipe e idade entre os projetos participantes. Novamente, é importante analisar este gráfico considerando o critério de seleção dos projetos, como descrito anteriormente – esta distribuição não é generalizável a todos os projetos de software livre, já que a população-alvo são justamente projetos mais estáveis.

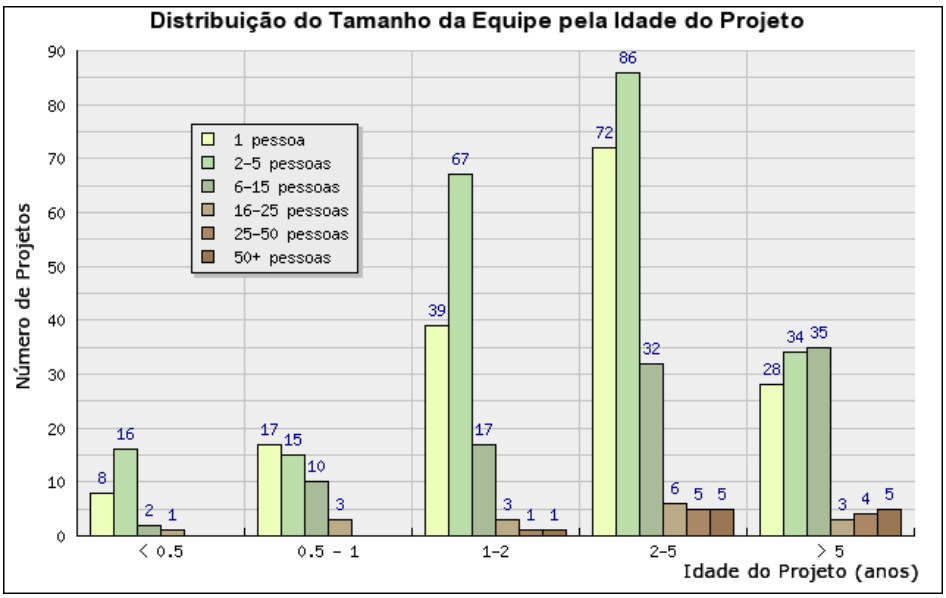


Figura 8.8: Gráfico da distribuição dos projetos de acordo com o tamanho de sua equipe, agrupado pela idade do projeto. (Os dados deste gráfico devem ser observados levando-se em consideração o critério de seleção da população.)

Ainda assim, é interessante observar a forma como se distribuem as dimensões dos projetos. Embo-

ra existam projetos de tamanho médio com menos tempo de existência, virtualmente todos os projetos com mais de 25 pessoas tem mais de 2 anos de existência, e uma boa parcela tem mais de 5.

É possível conjecturar que os projetos, até atingirem um conjunto mínimo de funcionalidade, sobrevivem através do esforço de poucos participantes. Após este período de evolução pública, o projeto pode passar a ter um número maior de desenvolvedores interessados, e esta mudança de proporção se observa no gráfico nos três últimos grupos. Nem todos os projetos atingem esta idade mais avançada, sendo abandonados antes de serem funcionalmente completos; muitas vezes o desenvolvedor principal não possui mais interesse ou tempo disponível para continuar o desenvolvimento.

Uma parcela dos projetos, mesmo com mais tempo de existência, não possui grandes equipes. Observando os projetos individuais e seus comentários, é possível identificar uma das possíveis causas: projetos funcionalmente avançados e com grande base de usuários possuem uma inércia para adotar nova funcionalidade – os comentários de Paul Smith e Martin pool na Seção 8.2.7 descrevem bem a situação. Outro motivo possível é a barreira de entrada maior que projetos com funcionalidade complexa apresentam a novos desenvolvedores: o custo de aprendizado para contribuir é muito alto. Exemplos de projetos com mais de 5 anos e equipes reduzidas incluem GNU make, GNU coreutils, GNU gettext, GNU grep, LILO, INN, ipchains e omniORB.

Vale notar, também, que software livre de uma forma geral é um fenômeno recente, e por este motivo existe um grande número de projetos ainda jovens. Seria interessante repetir este experimento com uma faixa de projetos de um estado de maturidade inferior para verificar em que aspectos divergem dos resultados apresentados neste trabalho.

Linhas de Código por Tamanho da Equipe e Idade do Projeto

Uma análise pertinente é avaliar o quanto uma equipe maior, e tempo de existência mais longo, tem relação com a dimensão do software produzido. Os gráficos na Figura 8.9 permitem visualizar estas propriedades. Ambos os gráficos mostram uma tendência de crescimento do número de linhas de código. Projetos maiores e mais antigos realmente tendem a possuir funcionalidade mais complexa, e portanto base de código maior.

Apesar de esperada, analisando esta tendência é possível conjecturar a respeito de uma propriedade discutida em parte da literatura: um projeto que possui funcionalidade extensa possui uma tendência a atrair usuários, e portanto desenvolvedores. Como descrito por Stefano Mazzocchi em seu trabalho ‘The Stellar Model of Open Source’ [85], é possível fazer uma analogia com a relação física de gravidade: projetos maiores, como possuem funcionalidade mais completa, tendem a atrair mais fortemente os usuários em busca de uma solução naquele domínio.

Buscando maior fundamentação para esta analogia, é possível analisar também a relação entre linhas de código e o domínio. O gráfico da Figura 8.10 demonstra tendências particulares para alguns domínios: emulação, aplicações *desktop* e projetos relacionados ao desenvolvimento de software. Os

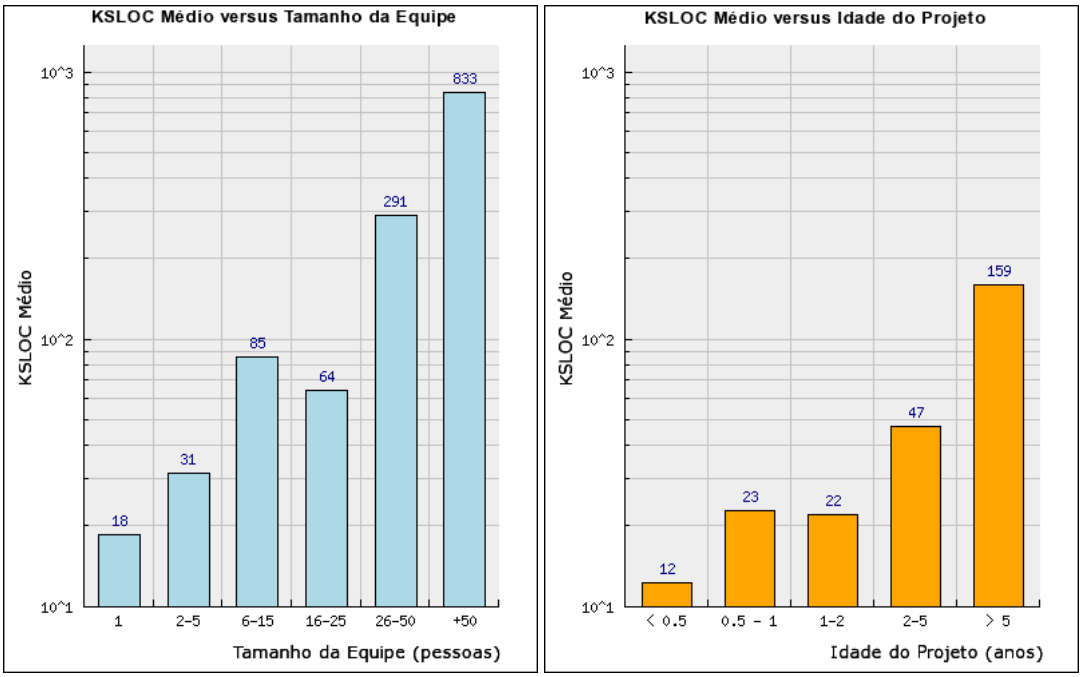


Figura 8.9: Gráficos relacionando linhas de código (em unidades de 1000, logo KSLOC) ao tamanho da equipe e à idade do projeto. Observe a tendência de crescimento clara que ambos apresentam: projetos maiores e mais antigos possuem bases de código maior. Existem leves inflexões nos gráficos, mas é provável que sejam decorrentes do reduzido número de projetos em algumas categorias, como descrito na Seção 8.2.4.

motivos que levam a esta concentração peculiar são dificilmente elicitados, mas algumas hipóteses podem ser estabelecidas.

8.3.2 Linhas de Código por Domínio

Primeiro, é possível que para certos domínios exista muito pouca tendência para a competição. Entre os emuladores, por exemplo, o custo de se iniciar um novo projeto é muito grande, já que envolve na maior parte dos casos engenharia reversa extensa. Boa parte do conhecimento acumulado ao longo dos anos no seu desenvolvimento é detido pelos membros do projeto, e é mais provável que um novo desenvolvedor em busca de uma nova funcionalidade queira contribuir para este projeto do que reiniciar este processo do zero. Em outros domínios, a complexidade e dimensão do projeto é inerentemente maior – aplicações para o usuário final podem apresentar esta característica.

Segundo, para domínios onde há grandes concentrações de desenvolvedores usuários, a situação inversa pode ocorrer. Um grande número de projetos concorrentes pode sobreviver graças à disponibilidade ampla de desenvolvedores, motivados a contribuir para um projeto particular por motivos pessoais – preferência técnica, facilidade de comunicação, percepção da qualidade do projeto.

Existem, por último, tendências sociais que levam o novo participante a contribuir para projetos conhecidos: o desejo de ter seu nome associado a um projeto importante, e a percepção geral de que

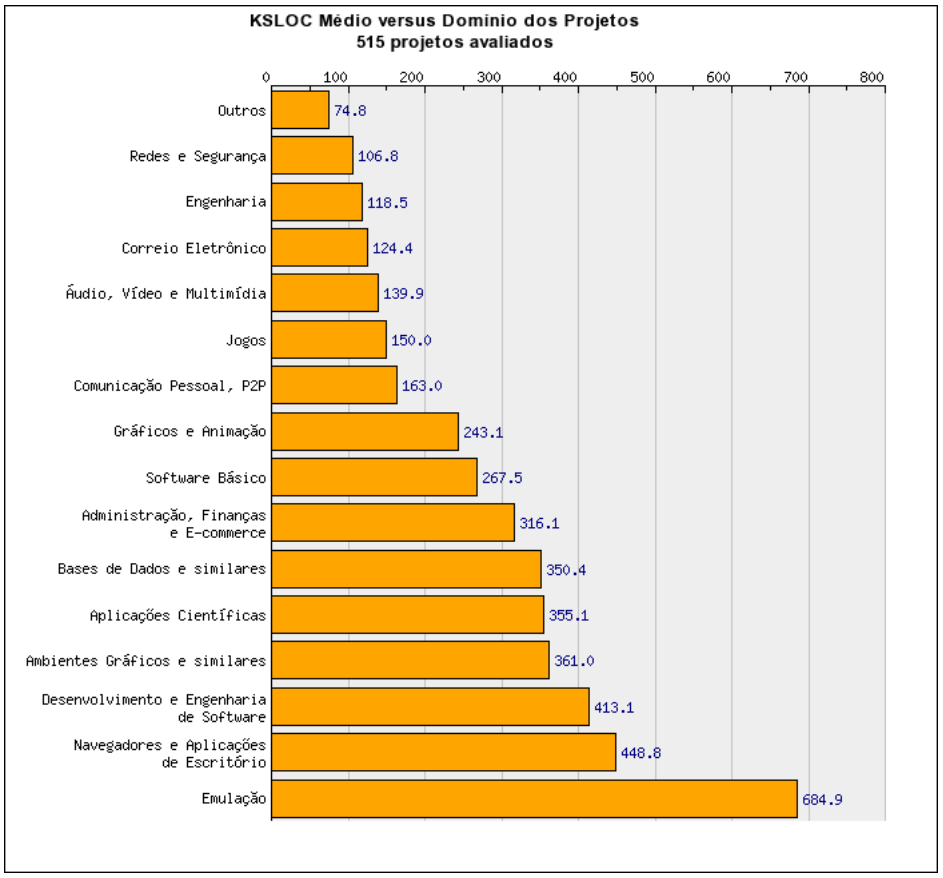


Figura 8.10: Gráfico relacionando numero médio de linhas de código com o domínio da aplicação.

colaborar é melhor do que reescrever ou dividir esforços [86].

8.3.3 Linhas de Código por Índice de Engenharia de Software

A próxima análise conjunta a ser descrita relaciona a dimensão da base de código com o índice de engenharia de software conforme descrito na Seção 6.8.5. Do gráfico scatter disposto na Figura 8.11 é possível observar a grande variação da atividade de engenharia de software entre projetos de todas as dimensões.

Pode-se observar uma concentração nos valores médios para o índice de engenharia de software. Esta concentração sugere que de fato exista um processo comum entre boa parte dos projetos de software livre, mas que não envolve todas as atividades que foram incorporadas no questionário.

Apesar da distribuição esparsa, há uma leve tendência de relação entre os valores. Se observada com cuidado a distribuição, é possível notar que esta cresce suavemente, o que indica que projetos com maior base de código têm uma pequena tendência a realizar maior esforço de engenharia de software.

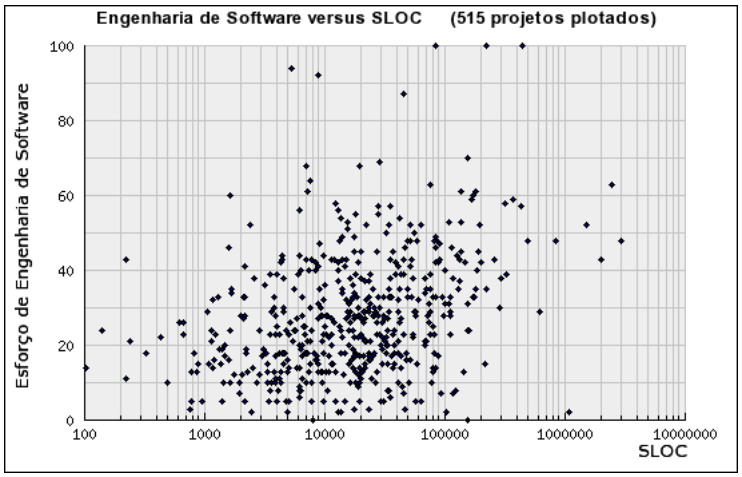


Figura 8.11: Scatter relacionando o índice de engenharia de software com o número de linhas de código de cada projeto. Observe o quanto se espalha a distribuição, comprovando a grande variabilidade das atividades de engenharia de software entre os projetos. Existe uma concentração de projetos nos valores médios do índice, sugerindo que existe de fato um conjunto comum de atividades.

8.3.4 Índices por Idade e Tamanho de Equipe

Para buscar uma melhor definição do padrão apresentado na seção anterior, foram criados um conjunto final de gráficos que relacionam a dimensão da equipe e a idade do projeto com os índices médios de engenharia de software nestes grupos. Estes gráficos estão dispostos na Figura 8.12.

O que os gráficos sugerem é peculiar: que a execução de atividades de engenharia de software está relacionada à dimensão da equipe, mas não ao tempo de existência do projeto. Em outras palavras, projetos com muitos integrantes realizam mais atividades de engenharia de software.

De certa forma, faz sentido esta tendência. Grande parte das atividades de engenharia de software descritas no questionário tem como objetivo coordenar o trabalho da equipe; em projetos com equipes menores, existe menor necessidade de coordenação. Equipes pequenas tem naturalmente uma necessidade menor de revisão de código, discussão de projeto, e documentação para desenvolvedores.

Ainda resta ser melhor analisado o gráfico relacionado à idade do projeto. Seria esperado que projetos ao longo do tempo agregassem novas atividades de engenharia de software, o que caracterizaria uma maturidade maior do seu processo. No entanto, o que se percebe é que os projetos sobrevivem com um conjunto mínimo de atividades de engenharia de software; se realizam outras atividades para sustentar seu crescimento, estas não estão entre as atividades convencionais abordadas no questionário.

8.4 Considerações Finais

Ao longo deste capítulo um grande número de gráficos e tabelas resume os resultados obtidos do levantamento por questionário. Este conjunto não representa o total de informações coletadas, mas uma

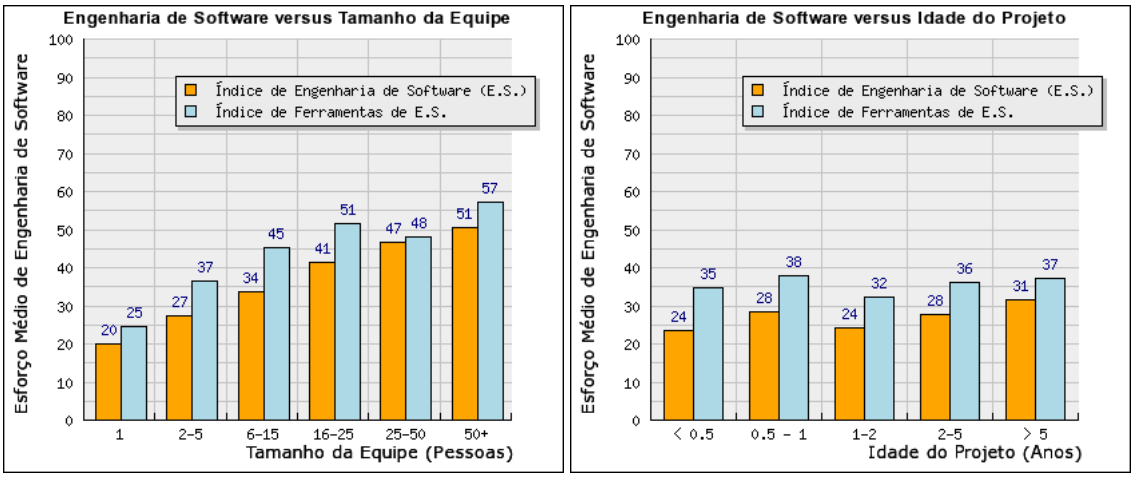


Figura 8.12: Gráficos representando a relação entre os índices calculados para engenharia de software e ferramentas de software, agrupados pela idade do projeto e pelo tamanho da equipe. A tendência crescente do gráfico de tamanho da equipe indica que projetos com equipes maiores realizam mais atividades de engenharia de software.

fração destas já analisada. Existem grande potencial para trabalho futuro com esta base de dados, e as informações estão disponíveis para outros pesquisadores avaliarem e oferecer enfoques alternativos aos resultados aqui apresentados.

Nesta seção, é feita uma série de considerações a respeito do levantamento e da ferramenta utilizada na sua realização.

8.4.1 Melhorias no Questionário

Ao realizar um levantamento desta escala, rapidamente se tornam aparentes as limitações e imprecisões da ferramenta utilizada. Com o objetivo de registrar esta auto-avaliação, e para permitir que outros pesquisadores possam aproveitar esta experiência prática, observações sobre o questionário aplicado seguem.

Forma de Contato

Leo Simons, do projeto Jakarta Avalon, ofereceu a seguinte sugestão:

> ‘for non-hierarchically organised projects (like the ones at apache), it is often more appropriate to contact the development group (through a mailing list, usually) instead of a single developer’

Nesta pesquisa, durante o período de envio do questionário, a opção de enviar a solicitação para uma lista de discussão foi feita apenas quando era impossível obter um endereço de contato pessoal. Acredito que esta técnica resultou em um melhor índice de resposta, porque diminuiu a necessidade

de coordenar entre os assinantes da lista quem responderia o questionário, e porque torna a solicitação mais pessoal. No entanto, poderia ser melhor avaliado o uso de listas de discussão em projetos que possuem um grupo central de desenvolvedores, como o Apache.

Questões Adicionais

Além das questões adicionais sugeridas ao longo deste capítulo, alguns respondentes, após preencher o questionário, enviaram sugestões para questões adicionais. Richard Stallman, como citado anteriormente, ofereceu a seguinte sugestão para a questão de motivação (1.1): *'There are some common motives you didn't mention–for instance, the desire to contribute to computer users' freedom, and gratitude to the free software community'*.

Bruno Haible, que lidera um conjunto de projetos de software livre, também ofereceu sugestões para novas questões relacionadas ao suporte ao usuário final realizado em software livre, que difere radicalmente do suporte usual oferecido para software proprietário:

> 'I really wonder that nearly no question deals with a point that I consider of paramount importance for the quality of free software: the bug report - feedback loop from users to developers. This is really what distinguishes free from commercial software: in free software developers have direct contact with the users (through e-mail), whereas in many companies the hotline calls never reach the developers (on the contrary: they have multi-level support in order to *ensure* that user feedback will be maximally filtered out!). Therefore some good additional questions would be:
>
> - Does the project have a published e-mail address where to send bug reports?
> - What is the average time until a bug report is answered?
> - What is the percentage of bug reports which are not addressed within six months?
> - Are bug reports handled by the developers themselves?'

Questionários futuros com o mesmo tema deverão levar em consideração estas sugestões para uma melhor cobertura do processo nos projetos de software livre.

Questões Vagas

Um número de questões do questionário acabaram tendo relevância reduzida por seu enunciado ser vago ou incompleto. Em particular, alguns itens das questões motivação (Seção 8.2.1), aspectos da equipe (Seção 8.2.6), requisitos (Seção 8.2.7), documentação (Seção 8.2.9), garantia de qualidade (Seção 8.2.10) e ferramentas (Seção 8.2.11) poderiam ter sido escritas de forma mais precisa. Outras ressalvas que foram identificadas durante a medição dos projetos foram apresentadas na Seção 6.8.2.

Uma forma de melhorar a precisão das questões, de uma maneira geral, seria oferecer um conjunto de graus para cada alternativa das questões. Desta maneira, seria possível obter informações de intensidade e relevância para as alternativas individuais; o questionário atual oferece discretização binária apenas - 'existe' ou 'não existe' a atividade.

No entanto, esta forma de avaliação gradativa tem usabilidade menor, e requer maior esforço do participante ao responder. Estas desvantagens merecem ser avaliadas quando realizando outro questionário para a mesma população.

Melhorias na Ferramenta

A ferramenta de questionário, embora apropriada para a função, poderia ter maior flexibilidade na aplicação das questões. Atualmente, apenas duas formas de pergunta são possíveis: múltipla escolha (usando *checkbuttons*) e escolha simples (*usando radiobuttons*). Seria interessante possibilitar um formato mais flexível para algumas das alternativas.

Por exemplo, a questão 1.4, que trata de padrões pré-existentes, pede que o usuário especifique qual. Embora uma parte dos respondentes tenha de fato descrito no campo de comentário que padrões eram relevantes para sua aplicação, seria interessante a existência de um campo texto específico para esta função, disposto em um local mais próximo do item assinalado.

Além desta situação, seria possível proporcionar fluxos alternativos de questões. Para a questão de usabilidade, por exemplo, poderia ser oferecida uma questão anterior para determinar se a aplicação possui interface gráfica ou de console; a escolha desta alternativa poderia alterar o conteúdo da questão subseqüente. Implementar esta variação de forma usável e multiplataforma, no entanto, não é trivial.

Finalmente, note que um balanço precisa ocorrer entre usabilidade e flexibilidade da ferramenta. Como a ferramenta atual é bastante simples, tem como vantagem apresentar facilidade de uso ao respondente se este já conhece o funcionamento geral dos formulários Web, como espera-se ser o caso desta população avaliada.

Satisficing

Existe literatura publicada que critica o uso de perguntas de múltipla escolha, do tipo 'marque todas as opções relevantes' [77, 87]. O motivo é uma tendência a *satisficing*: respondentes, em questões que envolvem esforço mental maior, tendem a preencher as questões por motivos psicológicos e não pelo conteúdo da alternativa. Infelizmente, tive acesso a este material apenas após veiculado o questionário, e por isso não pude avaliar outra opção de organização para as alternativas.

Na literatura, algumas observações são feitas sobre que fatores podem ajudar a reduzir a ocorrência de satisficing:

- alta motivação dos respondentes em fornecer dados acurados,

- facilidade no preenchimento das questões,
- existência de formas para comunicar situações alternativas que não são cobertas pelas alternativas oferecidas.

Neste levantamento, o interesse pessoal demonstrado pelos participantes, e a existência ampla de comentários e respostas customizadas oferecem, neste sentido, base ampla para reduzir a ocorrência de satisficing entre os questionários. Este aspecto poderia também ser confirmado por uma validação do questionário realizado, e este é um trabalho futuro com grande potencial.

8.4.2 Validação Posterior

É importante deixar claro que as análises discutidas neste capítulo são preliminares. O período dedicado à análise foi curto e limitado pelo tempo de realização do mestrado.

Uma importante atividade que deve ser realizada é uma validação dos dados obtidos no questionário. No trabalho FLOSS, descrito na Seção 5.3.6, uma validação posterior dos dados obtidos com o questionário foi realizada e um resumo está disponível na Seção [88]. Uma avaliação desta natureza poderia dar credibilidade extra aos dados obtidos, e oferecer melhor fundamentação para a discussão dos resultados.

Capítulo 9

Conclusões

Este capítulo representa a síntese das atividades realizadas ao longo deste trabalho de mestrado. Estas atividades — revisão bibliográfica, participação ativa, e o levantamento internacional — foram essenciais para estabelecer e quantificar conhecimento a respeito dos projetos de software livre, e os resultados deste processo foram apresentados e analisados nos capítulos anteriores. Nesta conclusão, são discutidas as hipóteses experimentais enunciadas no Capítulo 6, e é apresentada uma visão resumida do ciclo de vida dos projetos de software livre.

De uma maneira geral, o processo de software parece variar substancialmente entre os projetos, e não devem ser feitas generalizações amplas quando se tratando desta população. Vale ressaltar que não foi estabelecida representatividade da amostra do levantamento. Vale ressaltar que todos os resultados apresentados anteriormente, e a discussão das hipóteses e ciclo de vida presentes neste capítulo são referentes à população avaliada, e não são necessariamente válidos para uma população maior de projetos de software livre.

9.1 Avaliação das Hipóteses Experimentais

Na Seção 6.6.1 foi enunciado um conjunto de 13 hipóteses experimentais consideradas ao elaborar o questionário. Com base nos resultados apresentados nos capítulos anteriores, a lista a seguir avalia estas hipóteses e discute sua validade.

H2. O trabalho é realizado por equipes geograficamente dispersas.

Com base nos resultados apresentados na Seção 8.2.6, podemos confirmar que, para a maioria dos projetos, a equipe realmente trabalha de maneira descentralizada, sendo comum trabalharem em conjunto desenvolvedores que nunca se conheceram pessoalmente.

Nos últimos anos, têm havido mudanças que podem impactar neste perfil. Empresas e outras organizações convencionais têm investido em contratar desenvolvedores para projetos de software livre, e mesmo iniciar novos projetos. Projetos com esta característica incluem o MySQL, o

Mozilla e o BerkeleyDB. Além disso, um número de eventos internacionais tem contribuído para aproximar os membros da comunidade; exemplos incluem o FOSDEM, o GUADEC e o Linux Kernel Summit.

H3. O desenvolvedor do software é também seu usuário e contribui efetivamente para determinar grande parte de sua funcionalidade.

Conforme descrito nas seções 8.2.1 e 8.2.2, foi comprovada esta hipótese. Grande parte dos projetos foi criado por motivos pessoais, e a grande maioria dos projetos tem como usuários importantes sua equipe de desenvolvimento.

É esperado que, em conseqüência deste fato, ocorra uma concentração natural em domínios de aplicação onde existam usuários com conhecimento da área de desenvolvimento de software, como sugerem os resultados na Seção 8.1.4.

H4. Todo projeto utiliza um sistema de liderança particular para coordenar e arbitrar disputas, existindo duas variantes principais: o 'ditador benevolente' de Raymond, e o grupo central (ou *core*) de desenvolvedores.

A partir dos resultados da Seção 8.2.5, é possível afirmar que existe de fato, na maior parte dos projetos, um sistema de liderança utilizado. Os resultados indicam que as três formas de liderança propostas – líder com delegação informal, líder com delegação formal, e comitê – são aplicadas com freqüência semelhante entre a população.

Outras formas de liderança podem existir e sua determinação depende de um estudo mais detalhado deste aspecto dos projetos.

H5. Há uso amplo de ferramentas para viabilizar a comunicação entre a equipe geograficamente dispersa.

Segundo as freqüências apresentadas na Seção 8.2.11, existe uma preferência clara por ferramentas de controle de versão e listas de correio eletrônico, e são utilizadas pela grande maioria dos projetos. Uma parcela menor, porém significativa, indicou utilizar sistemas de acompanhamento de alterações/defeitos, o que, na minha opinião, é uma tendência crescente entre os projetos.

Um grande número de projetos utiliza um serviço de hospedagem de projetos. Ferramentas de comunicação em tempo real, como IRC e ICQ, são utilizadas em menor proporção.

H6. Há uso amplo de ferramentas de controle de versão, em particular do CVS.

Conforme descrito no item anterior, a grande maioria dos projetos afirmou utilizar um sistema de controle de versão. Dado que uma parcela significativa dos projetos é hospedado no site `sourceforge.net` (que oferece um serviço de CVS), e a tendência percebida entre a comunidade de projetos de software livre, CVS parece ainda ser a ferramenta mais utilizada. É utilizada mesmo em projetos de grande porte, como o Mozilla e o Python.

O uso de ferramentas como o Subversion e o Bitkeeper tem crescido e, à medida que se tornam mais funcionais e conhecidas, é possível que ocorra uma gradual migração das bases de código de CVS para outras ferramentas.

H7. A maior parte dos projetos possui equipe pequena, e a média do número de indivíduos por equipe não passa de 5.

Os resultados apresentados na Seção 8.2.4 confirmam esta tendência: a grande maioria dos projetos tem entre 2 e 5 desenvolvedores, e uma parcela considerável possui apenas um.

Foram estabelecidas duas outras correlações com a dimensão da equipe: o grau de uso de ferramentas de comunicação de desenvolvimento, e o esforço de engenharia de software que realizam (com relação às atividades observadas no questionário). Estes resultados sugerem que, à medida que novos desenvolvedores ingressam na equipe, a necessidade de gerenciar e apoiar o seu trabalho – através de atividades do processo e ferramentas – cresce.

Embora a existência de um grande número de projetos de equipe pequena indique que uma parcela considerável destes não tenha atingido sucesso notável, é provável que esta grande variedade amplie a diversidade de idéias entre a comunidade, e estimule inovação através da experimentação e concorrência.

H8. A maior parte das equipes possui membros com mais de 5 anos de experiência em desenvolvimento de software.

Não foi possível estabelecer a generalidade desta hipótese: aproximadamente metade dos projetos que participaram do levantamento afirmaram ter em sua equipe membro(s) com mais de 5 anos de experiência na área. É possível que projetos de maior porte, ou de maior sucesso, tenham membros com experiência mais ampla, mas ainda não foram feitas análises com a base de dados do questionário para confirmar este aspecto.

H9. O trabalho de engenharia de requisitos é freqüentemente facilitado por levantamentos anteriores feitos por outras equipes, seja através de padrões estabelecidos e/ou documentados, seja através de código-fonte herdado de outro projeto.

Embora não seja possível estabelecer uma conclusão absoluta para esta hipótese, os resultados das seções 8.2.1 e 8.2.7 sugerem fortemente o uso de conhecimento pré-existente para estabelecer os requisitos dos projetos. Aproximadamente um terço dos participantes afirmou fundamentar seu desenvolvimento em um padrão publicado anteriormente. Também é aproximadamente um terço o número de projetos que afirmou replicar de maneira significativa um outro produto de software.

H10. Existe um compromisso pessoal por parte dos membros da equipe do projeto em garantir a qualidade do software lançado.

Este levantamento não pode estabelecer grandezas subjetivas com precisão: os resultados da questão sobre garantia de qualidade apresentados na Seção 8.2.10 indicam que aproximadamente

metade dos projetos efetua teste extensivo quando lançando uma versão publicamente, o que apenas sugere o nível de compromisso da equipe com a qualidade do software.

H11. Existe uma forte política de controle e revisão relacionada à aceitação, integração e auditoria de contribuições de código.

Conforme discutido no item anterior, e de maneira surpreendente, não foi estabelecida a execução destas atividades entre os projetos; apenas um quinto destes afirmou realizar revisão de código, e uma proporção ainda menor afirmou ter regras formais de revisão e auditoria para integração. Estes resultados apresentam um paradoxo se de fato se confirmar a aparente alta confiabilidade de boa parte dos softwares livres, como introduzido na seção 3.4.4. Embora ocorra de fato teste funcional público, e embora os usuários tenham iniciativa e disposição para informar e auxiliar os desenvolvedores a reparar seu software, não parece justificar a reputação de estabilidade que software livre possui.

Pela experiência obtida nestes três anos de envolvimento, sugiro uma explicação baseada em aspectos sociais da equipe de desenvolvedores. Conforme apresentado por Niels Jørgensen (ver Seção 5.2.4), existe um forte indício de que a atenção pública ao desenvolvimento exerça uma pressão grande sobre o desenvolvedor de software livre. Ter seu trabalho realizado constantemente sobre um potencial olhar público, **mesmo que não ocorra revisão constante**, força o desenvolvedor a encarar sua tarefa com atenção e padrão de qualidade superiores. Em outras palavras, mesmo que o código produzido não seja sempre revisado e auditado, o fato de que *pode* ser avaliado – e publicamente criticado – influencia significativamente a atitude com que o desenvolvedor constrói o software.

H12. O tamanho da base de código do projeto está diretamente relacionado à dimensão da equipe e ao tempo de existência do projeto.

As correlações descritas na Seção 8.3.1 indicam a existência de vínculos entre estas grandezas. Em particular com relação à idade do projeto, no entanto, é importante observar estes resultados de maneira conservadora tendo em vista a não-representatividade da amostra.

H13. A equipe do projeto tende a crescer com o tempo.

Com os dados do questionário aplicado, não é possível concluir a respeito desta hipótese. O gráfico da Figura 8.8 indica a relação das grandezas tamanho da equipe e idade do projeto, mas é uma visão atemporal dos resultados, e não se pode estabelecer um padrão de evolução.

Com alguma relevância para esta questão, foi estabelecida uma leve correlação entre dimensão da base de código e a dimensão da equipe, o que sugere que projetos que atingem um porte maior tendem a possuir uma equipe maior. Não é possível determinar com os dados deste levantamento quais dos fatores são dominantes nesta correlação, mas é provável que o crescimento da equipe seja influenciado por um conjunto maior de fatores subjetivos, incluindo habilidade de liderança,

qualidade da base de código, extensão da funcionalidade implementada, e potencial de evolução do projeto.

H14. Pouca atenção é dada à usabilidade do software.

Conforme discutido por membros importantes da comunidade, e segundo conjecturas presentes na literatura, este levantamento confirma esta tendência: os resultados da seção 8.2.8 indicam não haver grande ênfase neste aspecto do processo de software. Este é provavelmente um dos fatores principais que leva software livre a ter sua reputação de complexo ou difícil de operar.

Desta lista, é possível observar – corretamente – que grande parte das hipóteses experimentais foi confirmada. Vale ressaltar que estas hipóteses foram elaboradas a partir do conteúdo discutido na literatura, e da experiência obtida durante a participação ativa; em outras palavras, foram estabelecidas com bom embasamento teórico. O questionário, neste sentido, teve como função principal *confirmar* estas experiências e impressões obtidas.

9.2 Um Ciclo de Vida para Projetos de Software Livre

Sintetizando o conhecimento obtido em um modelo simples, esta seção apresenta um ciclo de vida resumido para projetos de software livre. Como a maior parte dos modelos de processo, este é um modelo genérico: cada projeto de software livre implementa uma instância do processo de software, que é influenciado primordialmente pela sua cultura interna, seu histórico e seus objetivos.

Como uma observação geral, pode-se afirmar que este processo de software é tipicamente um processo de manutenção: a maior parte da atenção é voltada para os eventos e atividades realizadas *após* seu primeiro lançamento público. De maneira semelhante à discutida por Tomer e Schach em seu trabalho ‘The Evolution Tree’ [89], a visão de desenvolvimento de software proporcionada pelos projetos de software livre é uma de evolução constante. O objetivo do software não é atender a um conjunto de requisitos estável, e sim evoluir continuamente de acordo com as necessidades dos seus usuários (guardando uma ressalva para projetos comerciais). Projetos que atingiram estabilidade funcional tendem a ter equipe reduzida, mesmo quando extensos e importantes, como é o caso do GNU Make e do LILO, discutidos na seção 8.2.7.

O ciclo de vida destes projetos tem o aspecto particular de poder ser descontinuado a qualquer momento, sendo sustentado apenas pela motivação da sua equipe. Se um autor ou comunidade de desenvolvedores decidir desistir de um projeto, seu desenvolvimento pára, e o código-fonte persiste ‘congelado’ em um dos arquivos de software livre – possivelmente para ser adotado por um novo grupo de desenvolvedores no futuro. É raro existir incentivo financeiro ou contratual que garanta desenvolvimento continuado; manter software livre em desenvolvimento resulta do auto-interesse da sua equipe.

Antes de apresentar as etapas do ciclo de vida, é importante ressaltar que poucos projetos seguem este caminho até a maturidade. A maior parte, por fatores naturais, não atingirá dimensão maior, e

se estabilizará em algum ponto intermediário de evolução. Estes fatores podem incluir funcionalidade restrita, ou número de usuários insuficiente para atingir uma massa crítica.

9.2.1 Criação

Grande parte dos projetos será iniciado por um autor único, motivado por fatores pessoais. Freqüentemente sua motivação principal será criar um software para atender a uma necessidade pessoal que não se encontra atendida por nenhum produto existente de software livre.

É raro ocorrer apoio institucional significativo na criação de um produto. No entanto, parece ser uma característica comum entre projetos grandes, o que indica que projetos de sucesso têm oportunidade de atrair este tipo de apoio posteriormente. Também é importante ressaltar que apoio institucional inicial parece ser um fator comum entre projetos de sucesso e crescimento notável.

A maior parte dos projetos será criado com a intenção prévia de licenciar seu software através de uma licença de software livre. A maioria dos projetos, como descrito por Wheeler (ver Seção 5.3.6), opta pela licença GNU GPL (Seção 3.2.2).

Desenvolvimento Interno

O ciclo de vida adotado para esta fase inicial de desenvolvimento não pode ser determinado, já que pode variar radicalmente de projeto para projeto. Boa parte dos projetos descritos na literatura são desenvolvidos de maneira informal. O tempo de desenvolvimento interno pode variar, mas existe um consenso descrito por Raymond (Seção 5.1.1) que o software lançado deve apresentar funcionalidade 'interessante' o suficiente para atrair atenção. Processos mais convencionais de desenvolvimento podem aplicar-se nesta fase.

Uma parcela de projetos se iniciará baseado em código-fonte de outro projeto (os chamados code forks). Estes originam-se por diversos fatores, incluindo desavenças culturais ou pessoais entre desenvolvedores. Ocasionalmente, projetos abandonados serão adotados por uma nova equipe de desenvolvimento; diversos exemplos podem ser estudados no site `unmaintained.sourceforge.net`.

9.2.2 Lançamento Público

É possível que ocorram lançamentos das versões internas para grupos restritos, normalmente usuários diretos com quem o autor inicial tem contato; na minha experiência como desenvolvedor observei esta característica. O primeiro lançamento público é freqüentemente veiculado através de sites como o `freshmeat.net`, e em uma ou mais listas de discussão. Por exemplo, projetos do meta-projeto GNOME são anunciados em uma lista de discussão especial para lançamentos; outros meta-projetos possuem listas semelhantes.

A partir do momento de lançamento público, é caracterizada a fase de manutenção. É normalmente

muito difícil determinar quantos usuários de fato o software tem, já que é comum ocorrerem numerosos downloads de um pacote lançado, e dependendo da forma de distribuição escolhida, estes pacotes podem ser replicados em múltiplos repositórios de arquivos.

Normalmente um software livre lançado possui uma página Web e um endereço de email para contato.

9.2.3 Crescimento e Organização

Esta fase compreende a maior parte de atividades 'normais' realizadas nos projetos de software livre. Projetos desenvolvidos de maneira aberta — em outras palavras, cuja equipe de desenvolvimento tenha interesse em contar com a colaboração de desenvolvedores adicionais[1] — possuem um ciclo de desenvolvimento característico nesta fase.

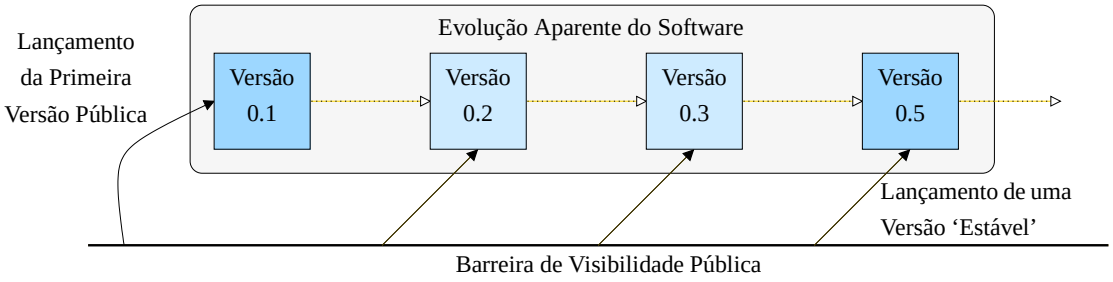


Figura 9.1: Uma visão 'caixa-preta' dos produtos de um projeto de software livre em desenvolvimento público, após seu lançamento inicial. A 'Barreira de Visibilidade Pública' representa o limite externo da comunidade de desenvolvimento, que só pode ser observado através dos produtos de sua interação interna — registros de listas de discussão, por exemplo — ou com participação ativa.

Observado externamente, de uma perspectiva 'caixa-preta', o desenvolvimento aparenta resultar apenas em um sucessão de pacotes de software numerados, conforme descrito na figura 9.1. Esta perspectiva simples, no entanto, não transmite a riqueza da interação interna do projeto, que é predominantemente social.

Dinâmica Social

A maior parte das equipes tem início em um autor único ou em uma equipe reduzida. A partir do lançamento inicial ocorre gradualmente uma formação de um conjunto de pessoas interessadas no produto, freqüentemente chamados de 'comunidade X', onde X é o nome do projeto em questão.

Normalmente, para atender à necessidade de comunicação com seus usuários, um projeto não-trivial estabelecerá uma lista de discussão por correio eletrônico. Poderá, conforme necessidade e disponibili-

[1] Poderia ser feita aqui uma alusão ao Bazar, ou a Open Source; na minha experiência, no entanto, a maior parte dos projetos de fato deseja participação externa. Existem casos onde não ocorre esta participação, mas pelo que compreendo dos comentários, é freqüentemente porque a barreira de ingresso causada pela complexidade do projeto é alta, ou porque há reduzido interesse externo.

dade de recursos — é importante ressaltar que estes serviços requerem ao menos um local para hospedagem e alguma gerência — criar outros mecanismos para apoiar comunicação e desenvolvimento.

Usuários e equipe de desenvolvimento, utilizando estes meios de comunicação, discutirão um conjunto de temas associados ao projeto. Este temas incluem: elogios, novos requisitos e alterações de funcionalidade, problemas encontrados ao usar o software, reparos ocasionais de código, e implementação de novas funcionalidades.

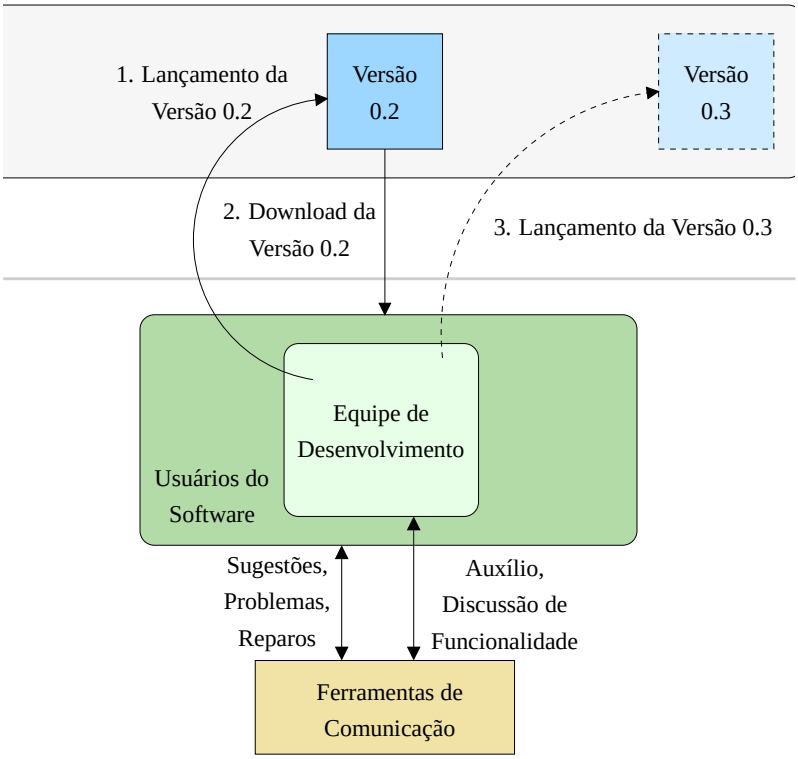


Figura 9.2: Um diagrama descrevendo uma comunidade simples de um projeto de software livre. Este diagrama detalha uma seção individual da evolução apresentada na Figura 9.1: estão representados os usuários, e entre estes, a equipe de desenvolvedores (que freqüentemente é composta de uma única pessoa). Estão ilustrados três passos importantes: o lançamento de uma versão (0.2); o download e uso subseqüente desta versão pelos usuários do software; e o lançamento de uma nova versão (0.3). Note que esta nova versão normalmente incorpora funcionalidade e reparos que foram apresentados e discutidos com os usuários do sistema.

O diagrama da Figura 9.2 descreve uma comunidade de pequeno porte. Como visto na figura, as responsabilidades da equipe de desenvolvimento envolvem produzir e lançar software, mas também estimular e apoiar sua comunidade de usuários. Conforme sugerido por Bruno Haible (Seção 8.4.1), esta característica das comunidades de software livre ajuda a explicar a sua popularidade crescente: os desenvolvedores de fato prestam atenção nos seus usuários. A participação de um usuário ativo se torna recompensadora à medida que percebe que suas solicitações e problemas refletem em resultados práticos por parte da equipe.

Desenvolvimento Concorrente

Da base de usuários, muitas vezes surgirão desenvolvedores potenciais. Isto reflete precisamente minha experiência pessoal nos projetos, e a de diversos participantes com quem convivi durante este período: se inicialmente apenas discute-se o software e sua funcionalidade, quando o participante possui iniciativa própria e algum conhecimento é uma passo natural começar a estudar o código-fonte do projeto. Freqüentemente este interesse surge com o propósito de resolver algum problema específico: o usuário encontra um bug, e antes de notificar o desenvolvedor, busca entender onde e por que ocorre. Desta maneira, parte dos usuários ativos acaba se tornando um desenvolvedor potencial, e assim por diante.

Quando a equipe do projeto é maior do que um ou dois desenvolvedores, é comum utilizar-se um repositório de controle de versões privado. À medida que usuários externos se interessam pelo código, torna-se mais conveniente oferecer um repositório público para que possam acompanhar de maneira mais próxima a evolução da base de código. Neste momento, uma definição da política de integração é feita: usuários externos devem ter acesso de escrita ao repositório, e caso positivo, quais?

Esta política define um grupo de desenvolvimento restrito com maior privilégio — o ***core***. O core é normalmente composto de indivíduos que tenham interesse continuado no projeto. Alguns projetos mais formais, como o Mozilla e o Zope (`www.zope.org`), requerem adicionalmente que um novo 'integrador' envie um formulário por correio que confirme sua intenção de participar mais ativamente do desenvolvimento, e estabeleça um contrato de submissão de código — normalmente envolvendo assuntos de propriedade intelectual. Alguns projetos, como o FreeBSD, diferenciam a atribuição de *core member* da permissão para integrar código; neste caso, existem membros não-core que podem escrever no repositório (*committers*), e o core é um grupo ainda mais restrito que determina políticas e arbitra conflitos.

Desenvolvedores externos sem permissão de integrar código enviam suas alterações aos membros do core para que sejam revisadas, discutidas e eventualmente, integradas. Muitos comentários na questão referente à garantia de qualidade (Seção 8.2.10) afirmam que código submetido por terceiros (não-membros do core) é revisado com bastante atenção, e freqüentemente reescrito. Desenvolvedores que participem do núcleo parecem ter seu trabalho revisado em menor intensidade, mas existe o aspecto social discutido anteriormente junto à hipótese 11 que contribui para incentivar a integração de código de boa qualidade.

Note que alguns projetos não oferecem sistemas de controle de versão públicos. Embora raro, ocorre mesmo entre projetos de grande porte: o Linux, até o inicio de 2002, não possuía um repositório público oficial. Até hoje, toda a integração na árvore principal é feita por uma única pessoa, Linus Torvalds.

Deve-se ressaltar que todo a codificação ocorre de forma bastante simultânea, de maneira semelhante à descrita por Davis e Sitaram no seu modelo concorrente [13] — pelo fato de serem utilizados mecanismos de controle de versão que operam pelo modelo de cópia local, os desenvolvedores podem

trabalhar independentemente e integrar suas mudanças conforme necessário.

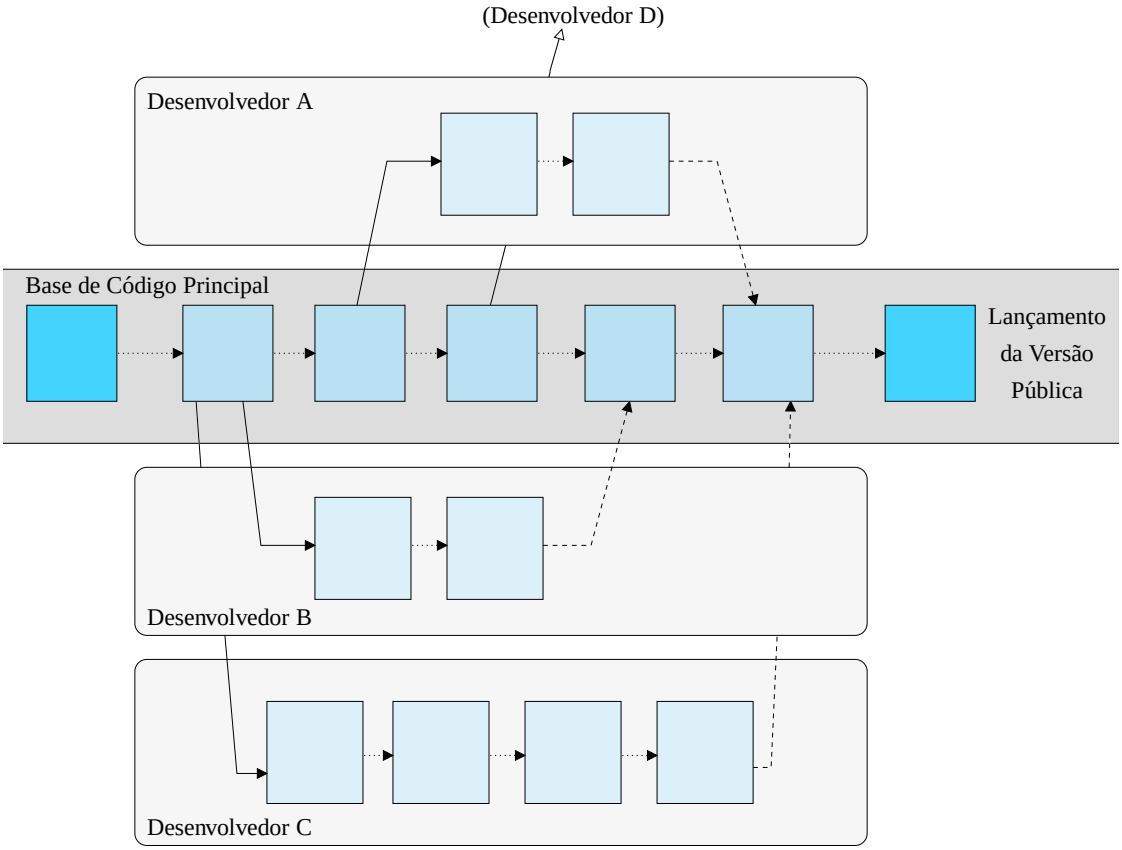


Figura 9.3: Diagrama descrevendo um ciclo de desenvolvimento de uma versão de um software livre. Diversos desenvolvedores obtém cópias do código-fonte, e elaboram suas alterações localmente. As linhas pontilhadas indicam escrita na base de código, que ocorre quando a alteração é considerada 'pronta' para integração. O trabalho culmina no lançamento de uma versão pública. Note que o Desenvolvedor D apenas obteve uma cópia local, não integrou alterações neste ciclo. Na prática, é frequente que desenvolvedores recorrentes já possuam cópias locais; neste caso, apenas atualizam a cópia, ao invés de obter uma nova.

O diagrama da Figura 9.3 evidencia a natureza paralela do desenvolvimento levando ao lançamento de uma versão pública. A concorrência é controlada através dos mecanismos de comunicação do projeto — listas de discussão e sistemas de controle de alteração têm importância neste momento. Na prática, pode ocorrer algum trabalho duplicado; no entanto, é raro e quando ocorre existe a vantagem de se avaliar mais de uma solução possível para o problema.

À medida que o projeto evolui, pode ser definida uma política para branching e versionamento, adaptado ao projeto de acordo com suas necessidades; esta política a descrição feita na Seção 4.3.4.

9.2.4 Maturidade

Grande parte dos projetos permanecerá indefinidamente neste estado de desenvolvimento paralelo constante, bastando apenas o interesse da sua equipe para mantê-lo em desenvolvimento. Outros projetos

estabilizam-se ao ponto de implementar apenas reparos para defeitos, como os projetos GNU Make e LILO, citados na abertura desta seção. Ainda outro conjunto de projetos cresce até outro patamar de organização.

Estes projetos atingem uma participação mais numerosa, e tendem a diversificar-se em termos da sua organização e ferramentas utilizadas. Projetos muito grandes, como é o caso do Mozilla, chegam a escrever suas próprias ferramentas para apoio ao processo: desta maneira surgiu o Bugzilla. Estes projetos maiores geralmente possuem uma estratificação interna maior, conforme apresentado na Figura 9.4.

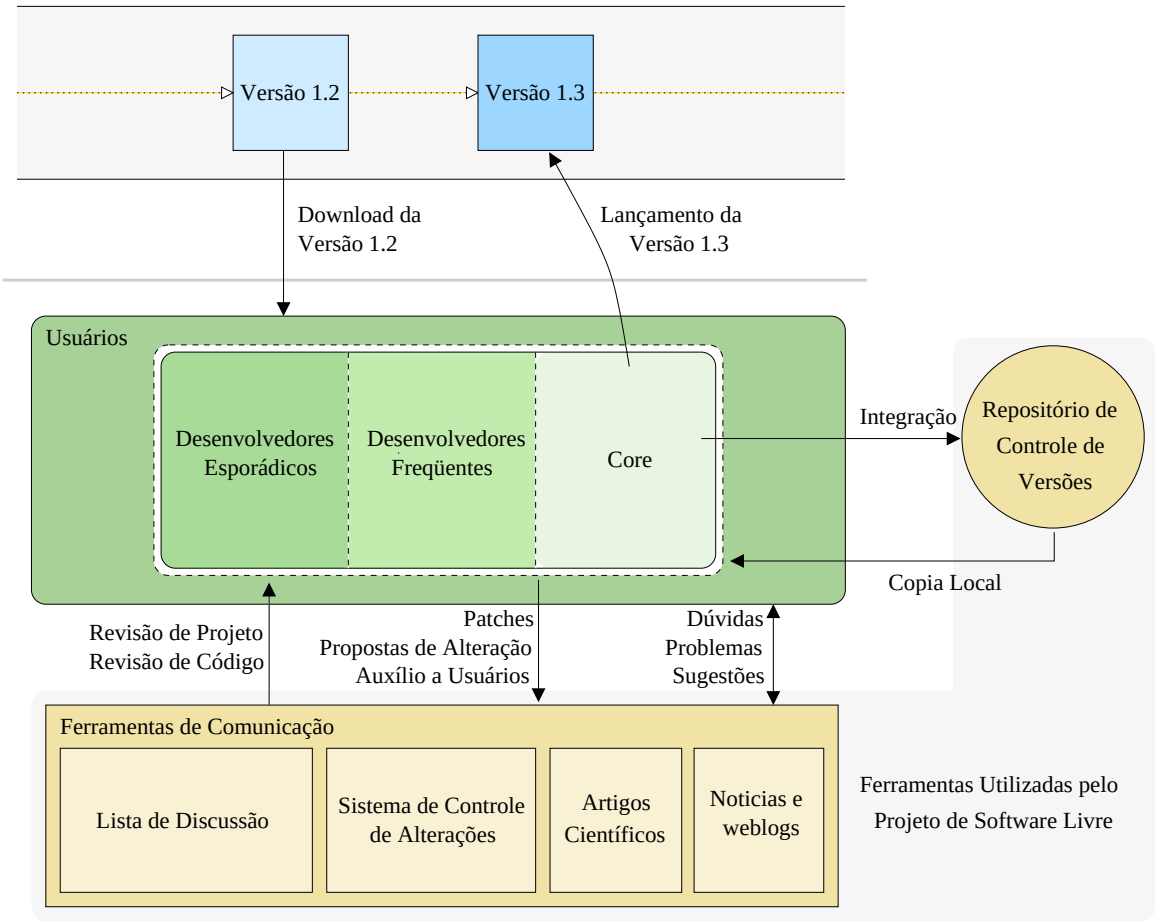


Figura 9.4: Diagrama que representa os três elementos chave de um projeto de software livre de maior porte: o produto, a equipe, e as ferramentas de comunicação e desenvolvimento. Muitos projetos de sucesso possuem uma composição de desenvolvedores mais estratificada, e se apóiam em um conjunto maior de ferramentas. As linhas pontilhadas indicam que há um fluxo entre estes grupos em ambos os sentidos – desenvolvedores perdem interesse e se tornam apenas usuários, por exemplo. Note que *weblogs* e notícias estão indicados no diagrama; recentemente têm havido um grande número de artigos e ensaios discutindo questões técnicas referentes a projetos de software livre, e é provável que provenham destes veículos outras idéias e sugestões para implementação.

Este diagrama sintetiza os elementos principais do processo de software em projetos mais desenvolvidos, observando-se apenas um período curto do desenvolvimento. O lançamento da nova versão

(1.3), como esperado, incorpora sugestões provindas dos usuários e desenvolvedores veiculadas por um conjunto de meios de comunicação, e concretizadas em código-fonte disponível publicamente.

Outros detalhes do processo precisam ainda ser melhor estabelecidos e descritos. Este ciclo de vida apresentado é um resumo bastante simplificado do processo elicitado durante este trabalho, e observa o projeto em um nível mais elevado que a maior parte dos modelos descritos na literatura. No entanto, como um recurso didático, simplifica a compreensão da organização e dinâmica de interação presentes em um projeto típico de software livre.

9.3 Considerações Finais

Este trabalho de mestrado oferece um retrato detalhado de um grande conjunto de projetos de software livre existentes na atualidade. Foi elaborada uma metodologia de pesquisa original, e com base nesta foi possível levantar um grande volume de resultados quantitativos. Destes resultados foram analisadas suas causas e conseqüências, e sintetizada esta análise no ciclo de vida descrito neste capítulo. Minha intenção com esta contribuição é proporcionar uma visão ampla do estado da arte relativo ao processo de software realizado nestes projetos, e ressaltar que aspectos extraordinários merecem maior atenção em estudos futuros.

Chamam a atenção em particular nesta conclusão os aspectos humanos relacionados ao desenvolvimento de software. Da literatura estudada a respeito de engenharia de software, poucos trabalhos discutem o quanto é importante uma equipe com alta competência e motivação; ao invés disso, a maior parte busca estabelecer controles e garantias que possam proteger o investimento das organizações que constroem software. Dado o contexto competitivo que têm, é razoável que estas organizações busquem garantias; no entanto, é importante prestar atenção em formas de desenvolvimento alternativas.

Tom DeMarco vem há mais de uma década alertando para a importância de valorizar os fatores humanos que influenciam produtividade e qualidade nas equipes de desenvolvimento de software [90]. James Bach, há 8 anos, em seu artigo ‘Enough About Process: What We Need Are Heroes’ [91] deixou clara sua opinião a este respeito:

> ‘Process is useful, but it is not central to successful software projects. The central issue is the human processor — the hero who steps up and solves the problems that lie between a need expressed and a need fulfilled[2].’

Se observamos a forma como se organizam e progridem os projetos de software livre, é impressionante como sua estrutura social valoriza precisamente estes recursos tão valiosos: o usuário, e o

[2] ‘Processo é útil, mas não é o elemento fundamental do sucesso dos projetos de software. O elemento fundamental é o processador humano — o herói que resolve os problemas que existem entre uma necessidade expressada e uma necessidade implementada.’

desenvolvedor. Se o desenvolvedor de software livre conjura a imagem do hacker solitário codificando intensamente, este estereótipo pode pelo menos sugerir uma verdade maior: que o trabalho é auto-gratificante, e que este grupo de desenvolvedores é motivado — internamente, deve-se notar — o suficiente para construir e manter grandes e complexos projetos de software.

É gratificante perceber a forma como o usuário é destacado entre os projetos. Recentemente, participei de uma discussão para definir um recurso na base de dados ZODB, utilizada no projeto Zope. Não contribuí com código-fonte; apenas descrevi detalhadamente uma necessidade e analisei com os desenvolvedores alternativas para implementação. No entanto, em pouco mais de uma semana havia sido integrado este código em um branch experimental, e na mensagem associada ao commit havia uma nota dizendo 'Thanks to Christian Reis for championing this feature.'

Esta atenção e reconhecimento tornam muito positiva a experiência para o usuário ao lidar com desenvolvimento de software. Em termos mais convencionais, substitui-se a palavra 'usuário' por 'cliente', e vem a realização de que os projetos fazem exatamente o que as cartilhas de marketing afirmam há décadas — ou, em termos mais recentes, o que Raymond pretende quando afirma 'and listen to your customers' (Seção 5.1.1). Não é surpreendente que a equipe de teste beta dos projetos de software livre tenha tanta dedicação e disposição para incorrer em esforço ao informar problemas; sabem que sua contribuição será útil, e não perdida entre camadas de atendimento e suporte técnico tão freqüentes entre os fornecedores de software de larga escala.

Além disso, os projetos oferecem uma fonte de inspiração para a 'crise de software' [9]. Trata-se de um meio onde parece abundar honestidade e disposição para retornar e reorganizar código mal-escrito. Os projetos mostram uma atitude saudável em relação à base de código existente; ao invés de ser um ônus, é vista como um recurso com grande potencial. Têm uma visão extremamente positiva com relação aos problemas que enfrentam durante o desenvolvimento: Sempre há chance de voltar e consertar o código.

As motivações para participar, apesar das ressalvas de Robert Glass (Seção 5.1.4), parecem claras: os projetos oferecem uma chance de cooperar com pessoas com quem não se teria contato normalmente, trabalhando para um objetivo prático, e com resultados concretos. Esta forma de colaboração herda do processo científico, agregando o ímpeto de desenvolver e evoluir um produto pelo prazer inerente à tarefa.

9.3.1 Trabalhos Futuros

Ressalto que esta é uma área muito especial para o pesquisador da engenharia de software, que tem a chance de observar o funcionamento de uma comunidade de desenvolvimento em autêntica atividade. Projetos de software livre oferecem oportunidades para compreender melhor as dinâmicas tecnológicas e sociais que influenciam o processo de software, avaliando seu impacto sobre a qualidade do produto construído.

Existem duas grandes áreas a destacar para trabalhos futuros. A primeira e mais objetiva representa a continuidade do processo de pesquisa iniciado com este trabalho de mestrado. Os dados coletados e a metodologia estabelecida podem ser reutilizados livremente, e aqui são resumidos trabalhos potenciais já sugeridos ao longo do texto:

- Como citado na Seção 8.4.2, uma atividade que resta por fazer é validar o questionário e avaliar a representatividade da amostra de projetos selecionada.
- Além disso, uma série de analises ainda resta a ser realizada com base nos dados obtidos, como descrito em diversas instâncias no capítulo 8.
- Este questionário poderia ser reaplicado a outras populações de projeto. Poderia também ser repetido, após um intervalo de tempo, para a mesma amostra, determinando desta forma um perfil de variação no seu processo e produto.
- Finalmente, o modelo descrito neste capítulo pode ser elaborado e estendido para outros aspectos do processo de software. Por exemplo, a fase de integração de modificações de código agrega mecanismos interessantes, incluindo revisão e reescrita em alguns casos. Estes mecanismos poderiam ser sintetizados de forma a facilitar sua compreensão.

A segunda área para trabalhos futuros a mencionar é relacionada às contribuições que podemos oferecer aos projetos de software livre. Apesar de suas qualidades, existem limitações e problemas a serem solucionados com relação ao seu processo e sua comunidade; para ajudá-los, é preciso determinar primeiro *como* podem ser ajudados. Neste trabalho identificamos algumas deficiências que merecem abordagem direta:

- É necessário um conjunto maior de estudos para verificar a confiabilidade e qualidade geral do software livre produzido. Trabalhos como o de Barton P. Miller et al. [31] têm grande valor, e devem ser estimulados e replicados.
- Como aparente na seção 8.2.8, Usabilidade é uma área potencial para grandes avanços entre os projetos de engenharia de software. É necessário trabalho de padronização e educação entre os desenvolvedores para incentivar a produção de software mais usável.
- Como descrito em diversos pontos nesta dissertação, o processo aplicado em projetos de software livre é fortemente apoiado por ferramentas. Em particular, existe necessidade de melhoria nas áreas de teste sistemático — uma ferramenta para apoiar teste de interface gráfica, por exemplo. Outras áreas incluem controle de versão e gerência de projeto. Os projetos Bugzilla e Subversion oferecem alguns pontos de entrada para colaboradores nestas áreas.

  Como um exemplo muito positivo desta classe de iniciativa, podem ser citadas as ferramentas `checker` de Yichen Xie e Dawson Engler [48], que realizam análises de código morto e duplicado. Os pesquisadores utilizaram o núcleo Linux como base de código para testes, e trabalharam

bem integrados à equipe de desenvolvimento, informando problemas encontrados e confirmando reparos.

- Por último, resta descobrir como incentivar a escrita de software livre direcionado a outros domínios onde não está presente atualmente. É importante prover requisitos e padrões que possam auxiliar os desenvolvedores, mas realmente essencial é capacitar os usuários para que possam participar mais ativamente do processo de desenvolvimento. Neste contexto de participação mais global, também faz sentido incentivar a participação de países e segmentos da sociedade que ainda não têm presença definida neste universo, proporcionando benefícios em maior escala.

Bibliografia

[1] International Organization for Standardization (ISO). *Information technology – Software life cycle processes*: ISO/IEC 12207:1995. Genebra, Suiça, 1995.

[2] PFLEEGER, S. L. *Software Engineering: Theory and Practice*. 2nd ed. Upper Saddle River, New Jersey: Prentice-Hall, 2001.

[3] BERNSTEIN, D. J. Internet host SMTP server survey. 2003. Disponível em: <http://cr.yp.to/surveys/smtpsoftware6.txt>.

[4] KENWOOD, C. A. *A Business Case Study of Open Source Software*. The MITRE Corporation, 2001. Disponível em: <http://www.mitre.org/support/papers/tech_papers_01/kenwood_software/>.

[5] HECKER, F. Setting Up Shop: The Business of Open-Source Software. junho 2000. Disponível em: <http://www.hecker.org/writings/setting-up-shop.html>.

[6] GHOSH, R. A. et al. Free/Libre and Open Source Software: Survey and Study: Part IV Survey of Developers. junho 2002. Disponível em: <http://www.infonomics.nl/FLOSS/report/FLOSS_Final4.pdf>.

[7] GHOSH, R. A.; PRAKASH, V. V. The Orbiten Free Software Survey. 2000. Disponível em: <http://orbiten.org/ofss/01.html>.

[8] LAKHANI, K.; WOLF, B. The Boston Consulting Group/OSDN Hacker Survey. OSDN, julho 2002. Disponível em: <http://www.osdn.com/bcg/>.

[9] PRESSMAN, R. S. *Software Engineering: A Practitioner's Approach*. 4th ed. [S.l.]: McGraw-Hill, 1997. 16 p.

[10] SOMMERVILLE, I. *Software Engineering*. 5th ed. [S.l.]: Addison-Wesley, 1995. 7 p.

[11] FOWLER, M. *UML Distilled: A Brief Guide to the Standard Object Modeling Language*. 2nd ed. Reading, Massachussets: Addison-Wesley, 1999.

[12] BECK, K. *Extreme Programming Explained*. Reading, Massachussets: Addison-Wesley, 2000.

[13] DAVIS, A. M.; SITARAM, P. A Concurrent Process Model of Software Development. *ACM SIGSOFT Software Engineering Notes*, v. 9, n. 2, p. 38–51, abril 1994.

[14] ROYCE, W. W. Managing the development of large software systems. In: *Proceedings of IEEE WESCON*. [S.l.: s.n.], 1970. p. 1–9.

[15] NEFF, G.; STARK, D. Permanently Beta: Responsive Organization in the Internet Era. In: HOWARD, P.; JONES, S. (Ed.). *The Internet and American Life*. 1st ed. Thousand Oaks, California: Sage, 2002. Disponível em: <http://www.coi.columbia.edu/pdf/neff_stark_pb.pdf>.

[16] ENGELFRIET, A. Crash course on copyright. 2002. Disponível em: <http://www.iusmentis.com/copyright/crashcourse/>.

[17] KAMINSKI, O. Contrafação na Web - Devemos limitar o acesso a obras protegidas? *Direito na WEB.adv.br*, 2002. Disponível em: <http://www.direitonaweb.adv.br/doutrina/dinfo/Omar_Kaminski_(DINFO_0005).htm>.

[18] NOBUO, I. The Open-Source Software as a Mechanism to Allocate Attention. *MITI International Conference*, 1998. Disponível em: <http://www.glocom.ac.jp/users/ikeda/oss.html>.

[19] ENGELFRIET, A. Choosing a Software License. 2003. Disponível em: <http://www.iusmentis.com/computerprograms/licenses/pcactive0203/>.

[20] The Free Software Foundation. Categories of Free and Non-Free Software. 2001. Última visita em Fevereiro 2001. Disponível em: <http://www.gnu.org/philosophy/categories.html>.

[21] The Free Software Foundation. What is Free Software? 2001. Disponível em: <http://www.gnu.org/philosophy/free-sw.html>.

[22] The GNU Project. Various Licenses and Comments about Them. 2001. Disponível em: <http://www.gnu.org/philosophy/license-list.html>.

[23] GACEK, C.; LAWRIE, T.; ARIEF, B. *The Many Meanings of Open Source*. School of Computing Science, University of Newcastle upon Tyne, 2001. CS-TR-737. Disponível em: <http://www.dirc.org.uk/publications/papers/12.pdf>.

[24] Debian GNU/Linux. What Does Free Mean? 2002. Disponível em: <http://www.debian.org/intro/free>.

[25] LEVY, S. *Hackers: Heroes of the Computer Revolution*. 2nd ed. New York: Penguin Putnam, 1994.

[26] MCKUSICK, M. K. Twenty years of Berkeley Unix. In: *Open Sources*. 1st ed. Sebastopol: O'Reilly and Associates, 1999. p. 31–46.

[27] RAYMOND, E. S. A Brief History of Hackerdom. In: *The Cathedral and The Bazaar*. 1st ed. Sebastopol: O'Reilly and Associates, 1999. p. 5–26.

[28] STALLMAN, R. M. The GNU Operating System and the Free Software Movement. In: *Open Sources*. 1st ed. Sebastopol: O'Reilly and Associates, 1999. p. 53–70.

[29] Thomas Schenk. Linux: Its History and Current Distributions. *IBM Developerworks*, maio 2001. Disponível em: <http://www-106.ibm.com/developerworks/library/it-schenk1/schenk1.html>.

[30] GANCARZ, M. *Unix Philosophy*. Reading, Massachussets: Prentice-Hall, 1995.

[31] MILLER, B. P. et al. *Fuzz Revisited: A Re-examination of the Reliability of Unix Utilities and Services*. University of Wisconsin, Computer Science Department, novembro 1995. Disponível em: <ftp://grilled.cs.wisc.edu/technical_papers/fuzz-revisited.ps>.

[32] RAYMOND, E. S. The Cathedral and The Bazaar. In: *The Cathedral and The Bazaar*. 1st ed. Sebastopol: O'Reilly and Associates, 1999. p. 27–78. Disponível em: <http://www.tuxedo.org/ esr/writings/cathedral-bazaar/>.

[33] BERLINER, B. CVS II: Parallelizing software development. In: *Proceedings of the USENIX Winter 1990 Technical Conference*. Berkeley, CA: USENIX Association, 1990. p. 341–352. Disponível em: <http://www.fnal.gov/docs/products/cvs/cvs-paper.ps>.

[34] HOEK, A. van der. Configuration Management and Open Source Projects. In: *Proceedings of the 3rd International ICSE Workshop: Software Engineering over the Internet*. IEEECSP, 2000. Disponível em: <http://sern.ucalgary.ca/ maurer/icse2000ws/submissions/Hoek.pdf>.

[35] VAUGHN, G. et al. *GNU Autoconf, Automake, and Libtool*. 1st ed. Indianapolis, Indiana: New Riders, 2000.

[36] MOON, J. Y.; SPROULL, L. Essence of Distributed Work: The Case of the Linux Kernel. *First Monday*, v. 5, n. 11, novembro 2000. Disponível em: <http://www.firstmonday.dk/issues/issue5_11/moon/>.

[37] IEEE. Portable Operating Systems Interface (POSIX). 1995. Disponível em: <http://standards.ieee.org/catalog/posix.html>.

[38] TORVALDS, L. The Linux Edge. In: *Open Sources*. 1st ed. Sebastopol: O'Reilly and Associates, 1999. p. 101–111.

[39] BROOKS, F. P. The Mythical Man-Month. In: *The Mythical Man-Month*. 2nd ed. Reading, Massachussets: Addison-Wesley, 1995.

[40] BEZROUKOV, N. A Second Look at the Cathedral and the Bazaar. *First Monday*, v. 4, n. 12, dezembro 1999. Disponível em: <http://www.firstmonday.dk/issues/issue4_12/bezroukov/>.

[41] HERBSLEB, J. D.; GRINTER, R. E. Splitting the Organization and Integrating the Code: Conway's Law Revisited. In: *Proceedings of ICSE*. Los Angeles: IEEECSP, 1999. p. 85–95.

[42] KRISHNAMURTHY, S. Cave or Community? An Empirical Examiniation of 100 Mature Open Soruce Projects. *First Monday*, v. 7, n. 6, junho 2002. Disponível em: <http://www.firstmonday.dk/issues/issue7_6/krishnamurthy/>.

[43] MCCONNELL, S. C. Open Source Methodology: Ready for Prime Time? *IEEE Software*, v. 16, n. 4, p. 6–8, julho/agosto 1999.

[44] GLASS, R. L. The Sociology of Open Source: Of Cults and Cultures. *IEEE Software*, v. 17, n. 3, p. 104–105, maio/junho 2000.

[45] WILSON, G. Is the Open-Source Community Setting a Bad Example? *IEEE Software*, v. 16, n. 1, p. 23–25, janeiro/fevereiro 2000.

[46] MOCKUS, A.; FIELDING, R.; HERBSLEB, J. A Case Study of Open Source Software Development: The Apache Server. In: *Proceedings of ICSE 2000*. Limerick, Ireland: ACM Press, 2000. p. 263–272.

[47] FIELDING, R. T. Shared leadership in the Apache Project. *Communications of the ACM*, v. 42, n. 4, p. 42–43, abril 1999.

[48] XIE, Y.; ENGLER, D. Using redundancies to find errors. In: *Proceedings of the 10th ACM SIGSOFT Symposium on Foundations of Software Sngineering*. Charleston, South Carolina, USA: [s.n.], 2002. p. 51–60. Disponível em: <citeseer.nj.nec.com/xie02using.html>.

[49] KUWABARA, K. Linux: A Bazaar at the Edge of Chaos. *First Monday*, v. 5, n. 3, março 2000. Disponível em: <http://www.firstmonday.dk/issues/issue5_3/kuwabara/>.

[50] GODFREY, M. W.; TU, Q. Evolution in Open Source Software: A Case Study. In: ICSM-00. *Proceedings of the 2000 Intl. Conference on Software Maintenance*. San Jose, California, 2000. p. 131–142. Disponível em: <http://plg.uwaterloo.ca/ migod/papers/icsm00.pdf>.

[51] LEHMAN, M. M. et al. Metrics and laws of software evolution: The nineties view. In: *Proceedings of ISMS (Metrics'97)*. Albuquerque: IEEE-CS, 1997.

[52] SCHACH, S. R. et al. Maintainability of the Linux Kernel. *IEE Proceedings: Special Issue on Open Source Software Engineering*, 2002. Disponível em: <http://www.isse.gmu.edu/faculty/ofut/rsrch/papers/linux-maint.pdf>.

[53] REIS, C.; MATTOS FORTES, R. P. de. An Overview of the Software Process and Tools in the Mozilla Project. In: *Proceedings of the Open Source Software Development*

Workshop. Newcastle Upon Tyne, United Kingdom: [s.n.], 2002. p. 155–175. Disponível em: <http://www.async.com.br/ kiko/mozse/>.

[54] JØRGENSEN, N. Putting it All in the Trunk: Incremental Software Development in the FreeBSD Open Source Project. *Information Systems Journal: Special Issue on Open Source Software*, v. 11, n. 4, p. 321–336, outubro 2001. Disponível em: <http://www.dat.ruc.dk/ nielsj/research/papers/freebsd.pdf>.

[55] MASSEY, B. Where Do Open Source Requirements Come From (And What Should We Do About It)? In: *Proceedings of the 2nd Workshop On Open-Source Software Engineering*. Orlando, FL: [s.n.], 2002. Disponível em: <http://www.cs.pdx.edu/ bart/papers/os-req.pdf>.

[56] ZHAO, L.; ELBAUM, S. Survey on Quality Related Activities in Open Source. *ACM SIGSOFT Software Engineering Notes*, v. 25, n. 3, p. 54–57, maio 2000.

[57] HALLORAN, T.; SCHERLIS, W. L. High Quality and Open Source Software Practices. In: *Proceedings of the 2nd Workshop On Open-Source Software Engineering*. Orlando, FL: [s.n.], 2002. Disponível em: <http://opensource.ucc.ie/icse2002/HalloranScherlis.pdf>.

[58] YAMAUCHI, Y. et al. Collaboration with Lean Media: How Open Source Succeeds. In: *Proceedings of CSCW*. ACM Press, 2000. p. 329–338. Disponível em: <http://www.bol.ucla.edu/ yutaka/papers/yamauchi_cscw2000.pdf>.

[59] ASKLUND, U.; BENDIX, L. A Study on Configuration Management in Open Source Software Projects. *IEE Proceedings – Software*, v. 149, n. 1, p. 40–46, fevereiro 2002. Disponível em: <http://www.lucas.lth.se/lucas-dagar/publications/CM4OSS.pdf>.

[60] DEMPSEY, B. J. et al. *A Quantitative Profile of a Community of Open Source Linux Developers*: SILS Tech. Rep. TR-1999-05. School of Information and Library Science, University of North Carolina at Chapel Hill, outubro 1999. Disponível em: <http://metalab.unc.edu/osrt/develpro.html>.

[61] HEALY, K.; SCHUSSMAN, A. The Ecology of Open-Source Software Development. Department of Sociology, University of Arizona, 2003. Disponível em: <http://opensource.mit.edu/papers/healyschussman.pdf>.

[62] NAKAKOJI, K. et al. Evolution patterns of open-source software systems and communities. In: *Proceedings of the International Workshop on Principles of Software Evolution*. Orlando, Florida: ACM Press, 2002. p. 76–85.

[63] WHEELER, D. A. Estimating Linux's Size. julho 2001. Disponível em: <http://www.dwheeler.com/sloc/redhat62-v1/redhat62sloc.html>.

[64] WHEELER, D. A. More Than a Gigabuck: Estimating GNU/Linux's Size. julho 2002. Disponível em: <http://www.dwheeler.com/sloc/redhat71-v1/redhat71sloc.html>.

[65] BOEHM, B. W. et al. *Cost Models for Future Software Life Cycle Processes: COCOMO 2.0*: USC-CSE-95-508. Center for Software Engineering, University of Southern California, 1995. Disponível em: <http://sunset.usc.edu/publications/TECHRPTS/1995/usccse95-508/usccse95-508.pdf>.

[66] GONZÁLEZ-BARAHONA, J. M. et al. Counting potatoes: The size of Debian 2.2. *Upgrade Magazine*, v. 2, n. 6, dezembro 2001. Disponível em: <http://people.debian.org/~jgb/debian-counting/counting-potatoes/>.

[67] GHOSH, R. A.; ROBLES, G.; GLOTT, R. Free/Libre and Open Source Software: Survey and Study: Part V Software Source Code Survey. junho 2002. Disponível em: <http://www.infonomics.nl/FLOSS/report/FLOSS_Final5all.pdf>.

[68] TRAUTH, E.; O'CONNOR, B. A study of the interaction between information technology and society: An illustration of combined qualitative research methods. In: *Information Systems Research: Contemporary Approaches & Emergent Traditions*. Amsterdam: North-Holland, 1991. p. 131–144.

[69] HAMERLY, J.; PAQUIN, T.; WALTON, S. Freeing the Source: The Story of Mozilla. In: *Open Sources*. 1st ed. Sebastopol: O'Reilly and Associates, 1999. p. 197–206.

[70] REIS, C. Developing applications with Kiwi. novembro 2002. Disponível em: <http://www.async.com.br/projects/kiwi/howto/>.

[71] THOMAS, M. Why Free Software Usability Tends to Suck. 2002. Disponível em: <http://mpt.phrasewise.com/discuss/msgReader$173>.

[72] NICHOLS, D. M.; TWIDALE, M. B. The Usability of Open Source Software. *First Monday*, v. 8, n. 1, janeiro 2003. Disponível em: <http://www.firstmonday.dk/issues/issue8_1/nichols/>.

[73] NICHOLS, D. M.; THOMSON, K.; YEATES, S. A. Usability and open-source software development. In: E.KEMP et al. (Ed.). *Proceedings of the Symposium on Computer Human Interaction*. ACM SIGCHI New Zealand, 2001. p. 49–54. Disponível em: <http://www.cs.waikato.ac.nz/ say1/pubs/oss.pdf>.

[74] VIXIE, P. Software Engineering. In: *Open Sources*. 1st ed. Sebastopol: O'Reilly and Associates, 1999. p. 91–100. Disponível em: <http://www.oreilly.com/catalog/opensources/book/vixie.html>.

[75] BÄCKSTRÖM, C.; NILSSON, C. Mixed Mode: Handling method-differences between paper and web questionnaires. 2002. Disponível em: <http://gathering.itm.mh.se/modsurvey/pdf/MixedMode-MethodDiff.pdf>.

[76] DILLMAN, D. A.; TORTORA, R. D.; CONRADT, J. Influence of plain vs. fancy design on response rates for web surveys. In: *Joint Statistical Meetings*. Dallas, Texas: [s.n.], 1998. Disponível em: <http://survey.sesrc.wsu.edu/dillman/papers/asa98ppr.pdf>.

[77] DILLMAN, D. A.; TORTORA, R. D.; BOWKER, D. *Principles for Constructing Web Surveys*: SESRC Technical Report 98-50. Pullman, Washington, 1988. Disponível em: <http://survey.sesrc.wsu.edu/dillman/papers/websurveyppr.pdf>.

[78] FREITAS, H. et al. O método de pesquisa survey. *Revista de Administração*, São Paulo, v. 35, n. 3, p. 105–112, julho/setembro 2000.

[79] PAULK, M. C.; CURTIS, B.; CHRISSIS, M. B. Capability Maturity Model, Version 1.1. *IEEE Software*, v. 10, n. 4, p. 18–26, julho/agosto 1993.

[80] FENTON, N. Software Measurement: a Necessary Scientific Basis. *IEEE Transactions on Software Engineering*, v. 20, n. 3, p. 199–206, 1994.

[81] REIS, C. Uma Visão Geral do Bugzilla, uma Ferramenta de Acompanhamento de Alterações. In: *XVI Simpósio Brasileiro de Engenharia de Software*. Gramado, RS: SBES, 2002. Disponível em: <http://www.async.com.br/˜kiko/bugzilla_sbes.ps.gz>.

[82] GODFREY, M. W.; LEE, E. H. S. Secrets from the Monster: Extracting Mozilla's Software Architecture. In: CoSET-00. *Proceedings of the Second International Symposium on Constructing Software Engineering Tools*. Limerick, Ireland, 2000. Disponível em: <http://plg.uwaterloo.ca/ migod/papers/coset00.pdf>.

[83] BOWMAN, I. T.; HOLT, R. C.; BREWSTER, N. V. Linux as a Case Study: Its Extracted Software Architecture. In: ICSE-99. *Proceedings of the 21st International Conference on Software Engineering*. Los Angeles, California, 1999. Disponível em: <http://plg.uwaterloo.ca/ itbowman/pub/icse99.ps>.

[84] MYERS, B. A. *Why are Human-Computer Interfaces Difficult to Design and Implement?* Carnegie Mellon University School of Computer Science, julho 1993. Technical Report CMU-CS-93-183. Disponível em: <http://www-2.cs.cmu.edu/afs/cs.cmu.edu/project/garnet/doc/papers/whyhardTR.ps>.

[85] MAZZOCCHI, S. The Stellar Model of Open Source. Java-Apache Project, fevereiro 1999. Disponível em: <http://bioinformatics.weizmann.ac.il/software/apache.org/java/framework/stellar.html>.

[86] RAYMOND, E. S. Homesteading the Noosphere. In: *The Cathedral and The Bazaar*. 1st ed. Sebastopol: O'Reilly and Associates, 1999. p. 79–135.

[87] KROSNICK, J.; NARAYAN, S.; SMITH, W. Satisficing in Surveys: Initial Evidence. *New Directions in Evaluation: Advances in Survey Research*, Jossey-Bass, n. 70, p. 29–44, junho 1998.

[88] GHOSH, R. A.; ROBLES, G.; GLOTT, R. Free/Libre and Open Source Software: Survey and Study: Part IVa Survey of Developers - Annexure on validation and methodology. outubro 2002. Disponível em: <http://www.infonomics.nl/FLOSS/report/FLOSS-Final4a.pdf>.

[89] AMIR TOMER, S. R. S. The Evolution Tree: A Maintenance-Oriented Software Development Model. In: *Proceedings of the Conference on Software Maintenance and Reengineering*. Zurich, Switzerland: IEEE Computer Society, 2000.

[90] DEMARCO, T. Mad about Measurement (1995). In: *Why does Software Cost so Much?* 1st ed. New York: Dorset House, 1995.

[91] BACH, J. Enough About Process: What we need are Heroes. *IEEE Software*, v. 12, n. 2, p. 96–98, março 1994.

Apêndice A

Primeira Versão do Questionário

The Free Software Engineering Survey

Answer the Survey

Please take your time to answer the survey. We would really appreciate it if you could add a comment after some of the answers, with an explanation of any interesting details related to the question. The current results of the survey are available online, as will the final report once it is ready.

On the next step you will have the option of making your comments and survey confidential. This means we will count it in the public statistics, but won't display the actual answers or comments publically.

1. General Questions

1.1. Is the project based on pre-existing source code (code fork, abandonware, etc.)? (help)

- a. Yes
- b. No
- c. Other:

Comment:

1.2. How long ago was the first version of the project's software released? (help)

- a. Less than 6 months
- b. 6 months to a year
- c. 1-2 years
- d. 2-5 years
- e. Over 5 years
- f. Other:

Comment:

1.3. What is the general domain of the project's software? (help)

- a. Base System (Kernel, Driver, OS utilities, Administration Tool)

- b. Development Application (Compiler, Interpreter IDE, Debugger)
- c. Code Library or Development Infrastructure (lib*, CORBA, Database lib)
- d. Server Application (Database, Webserver, Network application server)
- e. End-User Application (Word Processor, Browser, Game, Web Application, Network client)
- f. Other:

Comment:

1.4. Was there already a standard, specification or extensive documentation available that defined a significant part of the project? If so, please add a comment stating *the most important of them*.

Example: HTTP (Internet RFCs 1945 and 2616) for Apache. (help)

- a. Yes.
- b. No.
- c. Other:

Comment:

1.5. What was the *main* motivation for creating the project? (help)

- a. Personal need ('scratch the itch')
- b. Interesting challenge, or new language/technology
- c. Started as a project for business purposes
- d. Other:

Comment:

2. Initial Development

This section has questions about the initial development of the project; in other words, the time that happened before you did the first public announcement of the software code itself. This does not include the "call for participation" type of announcement -- only real software announcements count as leaving the "initial" phase.

2.1. How many developers worked on the project before it's first PUBLIC release? By ''worked on'', I mean people that considered

themselves part of the project team, not one-off contributors. (help)

- a. 1
- b. 2-5
- c. 5-10
- d. 10-50
- e. More than 50
- f. Other:

Comment:

2.2. How would you rate the programming and architectural expertise of the developer group at start? (help)

- a. The developers were a very experienced group (you could say 'wizards').
- b. Most developers were proficient in software programming and architecture.
- c. Most developers had some experience with coding software projects.
- d. Most developers had programmed before, but that was it.
- e. Most developers had never programmed before at all.
- f. Other:

Comment:

2.3. Did internal, non-public releases happen? For example, was there a 0.1.2 version that was never announced publically beyond the private group of developers? (help)

- a. Yes.
- b. No.
- c. Other:

Comment:

2.4. Did you document or discuss requirements for the software during the initial phase? (help)

- a. We developed and/or discussed explicit requirements specifications.
- b. We discussed requirements informally, but never produced or reviewed specifications.

- c. No, we assumed we understood the requirements.
- d. Other:

Comment:

2.5. Did you document and discuss the code design (APIs, data structures, etc.) during the initial phase? (help)

- a. We devised and/or discussed detailed design documentation and/or diagrams.
- b. We discussed design informally, but never produced or reviewed design documents.
- c. No, we basically designed as we coded.
- d. Other:

Comment:

2.6. Did you design and discuss the user interfaces during this phase? (help)

- a. Usability studies were conducted; interface design was a major activity
- b. The interfaces were prototyped and/or sketched out explicitly
- c. The user interface was discussed informally between developers
- d. There was no interface design done, they were put together without any specific process
- e. The software product has no significant user interface
- f. Other:

Comment:

2.7. How would you describe testing during this initial phase? (help)

- a. Great importance was given to testing the software, and it was a major activity in this phase.
- b. Some importance was given to testing, but it wasn't a major activity.
- c. There was no significant testing performed.
- d. Other:

Comment:

2.8. How would you characterize the documentation effort done during the initial phase of the project? (help)

- a. Documentation was a priority: we produced documentation files and/or manuals for both users and developers
- b. There was both user and developer documentation, but not a lot
- c. There was some user documentation provided
- d. There was some developer documentation provided
- e. Documentation was basically provided by comments in the code.
- f. There was no documentation at all in this phase.
- g. Other:

Comment:

2.9. Was there peer review of the code during this initial phase? In other words, before accepting and integrating code, did you read, understand and approve it? (help)

- a. Yes, there was very formal review with an official 'reviewed stamp'.
- b. Yes, but review was informal - we just read patches and if they looked okay, accepted them.
- c. No, there was no real review.
- d. Other:

Comment:

2.10. Select ALL of the software or communication tools that you actively used during the develoment of the initial version of the software (don't select it if it was installed but never used): (help)

- a. A project hosting site like Sourceforge.net (mark all others applicable as well)
- b. A version control tool such as CVS or Bitkeeper
- c. A test suite
- d. A bug database such as GNATS or Bugzilla
- e. One or more mailing lists
- f. Network news (NNTP)
- g. Direct email to registered or interested users
- h. A Wiki variant (swiki, phpwiki, zwiki, etc.)
- i. A website

☐ j. Other:

Comment:

3. Release Time

These questions refer to the time when the initial public announcement was done that offered a free software product for download and consideration. This is not the first stable (or 1.0) version of the software, but the first time the software was announced publically.

3.1. How long did it take for an initial public release to be produced? Count time as beginning when coding was actually started. (help)

- ○ a. Up to 1 week
- ○ b. From 1 week to 1 month
- ○ c. From 1 to 6 months
- ○ d. Over 6 months
- ○ e. Other:

Comment:

3.2. For you, how prone to a crash was the software package at release time? (help)

- ○ a. Rock-solid stability
- ○ b. Acceptable stability; crashed occasionally
- ○ c. Crashed frequently, but was still usable
- ○ d. Very flaky and unstable: crashed more than it worked
- ○ e. Other:

Comment:

3.3. In your opinion, how feature complete was the software upon its first release? (help)

- ○ a. Very complete, ready for general public use
- ○ b. Reasonably well developed but lacking some important features
- ○ c. Bare minimum features to be publically useful
- ○ d. It ran, but didn't do anything useful
- ○ e. Other:

Comment:

3.4. Select below ALL the ways the initial release was announced: (help)

- [ ] a. Freshmeat.net
- [ ] b. Project website
- [ ] c. Project mailing list
- [ ] d. A general announcements list, such as gnome-announce
- [ ] e. A newsgroup, such as C.O.L.A.
- [ ] f. Email to registered people
- [ ] g. Other:

Comment:

4. Present

This section has questions that refer to the current state of the development process.

4.1. If you were to choose, what sort of leadership model best describes the project today? (help)

- ( ) a. The project has a single leader, which delegates no responsibility to others
- ( ) b. The project has a single leader, which delegates some responsibility to others
- ( ) c. The project has a main leader and a number of people that are responsible for certain parts of it
- ( ) d. The project has a core group or committee that is responsible for making decisions about the project
- ( ) e. There is no real leadership structure
- ( ) f. This is a single-person project
- ( ) g. Other:

Comment:

4.2. How many developers are there currently in the project? (help)

- ( ) a. None
- ( ) b. 1

- c. 2-5
- d. 5-10
- e. 10-50
- f. More than 50
- g. Other:

Comment:

4.3. Have the developers met personally to work on the project? Consider the whole group when answering this question. (help)

- a. Yes, the developers work in direct personal contact frequently
- b. Yes, but only occasionally (for example, at conferences or similar events)
- c. The developers have never met personally
- d. Other:

Comment:

4.4. What would you consider the programming and architectural expertise of the current developer group? (help)

- a. The developers are a very experienced group, wizards you could say.
- b. Most developers are proficient in software programming and architecture.
- c. Most developers have some experience with coding software projects.
- d. Most developers have programmed before, but that's it.
- e. Most developers have never programmed before at all.
- f. Other:

Comment:

4.5. Do you currently produce or discuss the requirements for the software before coding? (help)

- a. We develop and/or discuss explicit requirements specifications.
- b. We discuss requirements informally, but never produce or review specifications.
- c. No, we assume we understand the requirements.
- d. Other:

Comment:

4.6. Do you currently produce or discuss the software design before coding functionality? (help)

- a. We devise and/or discuss detailed design documentation and/or diagrams.
- b. We discuss design informally, but never produce or review design documents.
- c. No, we basically design as we code.
- d. Other:

Comment:

4.7. How would you describe testing during the current phase? (help)

- a. There are one or more developers charged specifically with testing the software.
- b. We informally test our code, running and verifying it ourselves.
- c. There is no testing performed.
- d. Other:

Comment:

4.8. Is there, today, peer review of the code? In other words, before accepting and integrating code, is it read, understood and approved of? (help)

- a. Yes, there is very formal review with an official 'reviewed stamp'.
- b. Yes, but review is informal - we just read patches and if they look okay, accepted them.
- c. No, there is no real review.
- d. Other:

Comment:

4.9. Select ALL of the software or communication tools that you currently use during development (don't select it if you installed it but never used it): (help)

- [ ] a. A project hosting site like Sourceforge.net (mark all others applicable as well)
- [ ] b. A version control tool such as CVS or Bitkeeper
- [ ] c. A test suite
- [ ] d. A bug database such as GNATS or Bugzilla
- [ ] e. One or more mailing lists
- [ ] f. Network news (NNTP)
- [ ] g. Direct email to registered or interested users
- [ ] h. A Wiki variant (swiki, phpwiki, zwiki, etc.)
- [ ] i. A website
- [ ] j. Other:

Comment:

4.10. How do you feel development speed changed after the public release? (help)

- ( ) a. Development became faster
- ( ) b. Development became slower
- ( ) c. There wasn't a significant difference in development speed
- ( ) d. Other:

Comment:

When you are done: Step 2: check answers

---

christian reis <kiko@async.com.br>

Apêndice B

Versão Final do Questionário

The Free Software Engineering Survey

Answer the Survey

Please take your time to answer the survey. We would really appreciate it if you could add a comment after some of the answers, with an explanation of any interesting details related to the question. The current results of the survey are available online, as will the final report once it is ready.

Note that you are free to leave a difficult question unanswered; just leave it blank and proceed normally. You can come back later and change any answers you like once you have submitted the initial answers.

1. General Questions

1.1. With respect to the *initial motivations* for starting this project, please mark ALL the sentences below that are true. (help)

- ☐ a. The project was started *primarily* for personal reasons (had desire to learn, found technology interesting, etc).
- ☐ b. The project was started *primarily* to produce software to support another Open Source/Free Software project
- ☐ c. The project was started (or sponsored) by a company or organization.
- ☐ d. The project was started as part of academic (university) work or research.
- ☐ e. The project's intention *from the beginning* was to produce at least part of its software as Open Source/Free Software.
- ☐ f. The project began based on pre-existing Open Source/Free Software (code fork, etc.).
- ☐ g. The project began based on pre-existing proprietary/closed source software.
- ☐ h. Other: _____

Comment: _____

1.2. Who are the main users for your project's software? Please mark ALL that apply. (help)

- [ ] a. Yourself.
- [ ] b. The project team
- [ ] c. Users that are not computer-proficient or non-technical.
- [ ] d. Users in specific fields of expertise (e.g. amateur radio enthusiasts, mathematicians, engineers, software developers). *Please specify which in the comment box below.*
- [ ] e. Users in a particular company, site or organization
- [ ] f. One or more specific commercial customers (sponsors/paying).
- [ ] g. The Free Software/Open Source community.
- [ ] h. The computing community.
- [ ] i. Other:

Comment:

1.3. How long ago was the first public version of the project's software released? (help)

- ( ) a. Less than 6 months
- ( ) b. 6 months to a year
- ( ) c. 1-2 years
- ( ) d. 2-5 years
- ( ) e. Over 5 years
- ( ) f. Other:

Comment:

1.4. Was there a pre-existing standard or public specification previously available that helped define a significant part of the project? If so, please add a comment stating *the most important of them.*
Example: HTTP (Internet RFCs 1945 and 2616) for Apache. (help)

- ( ) a. Yes.

- b. No.
- c. Other:

Comment:

2. Project Team and Community

In this section I am trying to understand how big the team on the project is, what it is like, and how members are involved in the project activities.

2.1. How many people would you estimate make up the project team? If in doubt, consider the project team is made up of the frequent and active contributors (including non-code contributors). (help)

- a. 1
- b. 2-5
- c. 6-15
- d. 16-25
- e. 26-50
- f. More than 50
- g. Other:

Comment:

2.2. If you were to choose, what sort of leadership model best describes the project today? (help)

- a. The project has a single leader and a number of people that are formally responsible for parts of it.
- b. The project has a single leader, which delegates responsibility occasionally to others.
- c. The project is led by a core group or committee of people that is responsible for making decisions; there is no single leader.
- d. There is no effective leadership structure.
- e. This is a single-person project.
- f. Other:

Comment:

2.3. With relation to this project's team and community, please mark ALL phrases below that apply. (help)

- a. Most people in the team know each other only through the Internet, and have never met physically/personally.
- b. Most people in the team work physically close, and meet personally with some frequency.
- c. The team includes people that have more than 5 years experience in serious software development.
- d. The project provides a high barrier of entry to new participants, even to those that are skilled in software development.
- e. A contributor to the project will often participate in more than one project activity: functionality definition, architectural design, designing user interfaces , coding, testing, project management, etc.
- f. Code contributors tend to work only on one specific language or technology used in the project (the one which they are most familiar with).
- g. Other:

Comment:

3. Process Activities and Tools

These questions refer to the type of activities that people in the project team actually do. This includes coding, of course, but along with coding I would like to understand how other important (from the engineering viewpoint) activities are actually done in your project.

3.1. This question aims to find out how the behavior and functionality of your software were defined. Please mark ALL phrases that apply to your project. (help)

- a. The project's software explicitly mimics (or is significantly based upon) the functionality or behavior of an existing (proprietary or Free Software/Open Source) software package. *If so, please specify which?*

- [ ] b. A significant amount of effort is spent defining what the software functionality and behavior (the requirements) should be.
- [ ] c. This project has no end-users apart from the project team.
- [ ] d. There have been meetings or discussions with end-users to define significant parts of the software functionality or behavior.
- [ ] e. These meetings/discussions occur frequently, and can be considered an important part of defining the project's software.
- [ ] f. The software functionality is implemented according to what the team thinks is correct, without significant external end-user input.
- [ ] g. I believe much of the expected functionality and behavior is not completely known or understood by the project team as a whole.
- [ ] h. Other:

Comment:

3.2. This question touches the issue of usability: what sort of approach and how much effort is placed upon making the software more usable (i.e. easier, more efficient and more pleasant to use). Please mark ALL that apply. (help)

- [ ] a. The user interfaces for the project are designed (or prototyped) and refined before actually being implemented.
- [ ] b. We have conducted serious usability tests and studies on the project's user interfaces.
- [ ] c. Developers are not allowed to implement or change the user interfaces before the implementation/change has been reviewed and approved.
- [ ] d. A part of the team is specifically in charge of UI design.
- [ ] e. We would like to invest more in usability, but we are hampered by lack of documentation and/or team knowledge on the subject.
- [ ] f. This project doesn't have any significant user interface.
- [ ] g. Other:

Comment:

3.3. In this question I try to determine information on documentation produced in/for the project. Please mark ALL that apply. (help)

- [ ] a. We have produced documents that describe at least some of the expected functionality and behavior (requirements) of our software.
- [ ] b. The project produces and maintains documents for developers that describe how the code is organized (architecture), and/or how parts of it work.
- [ ] c. There is a reasonably complete coding standards guide that is actively followed by the team.
- [ ] d. There is documentation for the end-user available for the project's software (consider also third-party documents available).
- [ ] e. End-user documentation is provided to a large extent by people or groups external to the project team.
- [ ] f. A significant part of the available documentation is frequently updated and revised to be up to date.
- [ ] g. Other:

Comment:

3.4. This question covers activities that are performed with the intention to assure or improve the quality of the software produced by your project. Please mark ALL that apply. (help)

- [ ] a. There is an [automated] test suite for the project's software, that is used to validate it.
- [ ] b. There is a test plan (a written document describing tests) for the project's software, that is used by the project team.
- [ ] c. Periodic (i.e. nightly, weekly) snapshots of the project's code (or binaries) are distributed and used as an significant means of testing the software.
- [ ] d. There is an active code review process, where code is read by other members of the team.
- [ ] e. The project team members have formal rules for integrating code changes into the main codebase, and review is a strict requirement.
- [ ] f. Usually, an unexpected amount of bugs are discovered after a version has been publically released (this includes public release candidates).
- [ ] g. There is a tendency (or policy) to release a public version only when it has been *extensively* tested by the team.
- [ ] h. Other:

Comment:

3.5. Select ALL of the software or communication tools that you *actively* use during development (don't select it if is available but rarely or never used): (help)

- [ ] a. A project hosting site such as Sourceforge.net, Savannah or Collab.net (mark all others applicable as well)
- [ ] b. One or more Web sites
- [ ] c. A WikiWikiWeb site (SWiki, TWiki, PHPWiki, ZWiki, etc.)
- [ ] d. A Frequently Asked Questions document
- [ ] e. A version control tool such as CVS, RCS, Subversion or Bitkeeper
- [ ] f. A bug database such as GNATS or Bugzilla
- [ ] g. One or more mailing lists
- [ ] h. Network news (NNTP)
- [ ] i. User forums/BBS
- [ ] j. IRC
- [ ] k. An instant messaging system like ICQ, Jabber or AIM
- [ ] l. Other:

Comment:

On the next step you will have the option of making your comments and survey confidential. This means we will count it in the public statistics, but won't display the actual answers or comments publically.

Next: Check answers

christian reis <kiko@async.com.br>

Apêndice C

Mensagem de Convocação Enviada aos Participantes

Hello there,

I'm a developer from Brazil doing an MSc project on Open Source/Free Software engineering processes, and as the final part of it, I'm conducting a survey of the most important free software projects. I'm sending out a request to the maintainer for each project, and so far we have over 420 surveys filled out, including some very interesting ones (Apache, Perl, Python, Sendmail, Exim, Berkeley DB, GNU Make, Vim, Nethack, Bugzilla, sed, MySQL, ntop, omniORB, Ogg, Konqueror, Nautilus, iptables, Freeciv, SpamAssasin, CUPS to name a famous few); http://www.async.com.br/~kiko/fsp/newproj.php lists all registered projects.

Since [Project Name] is an important free software/open source project, I would be really grateful if you could participate in the survey by answering for it, and for other projects you lead (that you find are relevant to the study). The survey is Web-based (I can send an email version of it to you, of course), and it's online at:

    http://www.async.com.br/~kiko/fsp/

IMHO the survey is quite unique, and the data and comments gathered form a very interesting display of the different projects. The data will be published and offered freely when the survey is finished, of course.

This is the only email I will send you with a request, and if you don't have the time, kindly ignore it (use "d" in mail/mailx and "D" in mutt/pine :-). Thanks if you do take the time, too.

I've pasted below some standard details about the survey.

--- Survey details start here ----------

The Survey

        http://www.async.com.br/~kiko/fsp/

    It has 12 questions, and shouldn't take more than 15 minutes to answer (we did some testing and most people finish in about 7). If you would like someone else to answer, please send me an email address and I'll ask them directly. If you would rather the survey came in through email, let me know too and I'll send an email-ready version.

    The survey is usable on any browser (including text browsers like lynx and w3m), and it contains *no* graphics, so it should be quite fast. If you use Mozilla or another DOM-compliant browser the help boxes show in the same page, which is a bit nicer than the default behaviour. Apart from that it shouldn't be too much of a bother to fill it out, and it will really help us get valid research findings.

    If you want to know more about the survey, there is a FAQ on the site, and I am available to answer any questions. A short text follows describing it.

About the Research

    The MSc project is an attempt to characterize open source/free software projects with respect to their software processes. This means how the project organizes, how code is designed, produced and kept track of, and other issues relating to the project's approach to development.

    The qualification paper (in portuguese, i'm afraid) is available at http://www.async.com.br/~kiko/quali/ - Google does a [rather poor, and my surname isn't Kings] translation of it if you enter the URL in the search box.

    The results of this survey will be made publically available, and the report will be also published in English, to make sure that the

largest number of people and organizations can benefit from the data. If you are interested I can offer data in any format that's considered convenient.

About me

I'm an MSc student at ICMC USP, Brazil, and my MSc work involves describing and characterizing software process models for open source projects. As part of my research, I've done a study of a number of projects, and this year I've published an article on the Mozilla Software Process at the OSS Workshop in Newcastle, UK. It's available online at http://www.async.com.br/~kiko/papers.html

I'm also a member of the Kiwi, Bugzilla and PyGTK development teams, and I work mainly on an open source project to develop a free point of sales and retail management software, called Stoq.

Apêndice D

Lista de Projetos Registrados

The Free Software Engineering Survey

Browse answered surveys

You may view the survey for any project by selecting it from the alphabetically-ordered list below (as long as the project wasn't marked as private).

- AbiWord
- Adaptive Website Framework (AWF)
- afio *
- Agnubis
- AIME
- alien
- AlsaPlayer
- Apache::ASP
- Apollo
- apt for RPM
- ardour *
- Ariadne
- Atlantik *
- atool
- Audacity *
- AutoGen
- avidemux *
- bash completion *
- BBRun
- bc *
- Berkeley DB *
- BibleTime
- bidwatcher
- BINS
- Blackbox
- BlackHole
- Blender *
- Blitzed Open Proxy Monitor/LIBOPM
- Bluefish
- bnetd *
- Boa
- GNU Parted *
- GNU Pascal
- GNU Smalltalk
- GNU Visual Debugger
- gnucash *
- Gnucleus
- Gnumeric *
- GNUMP3d *
- GNUstep *
- GProFTPD
- gq
- Grace *
- graft *
- Graphviz *
- GRASS GIS *
- GStreamer *
- GSwitchIt
- GTK+ *
- GtkBalls
- Guarddog *
- Guikachu
- Guile
- Gwenview
- Hackbot *
- Halflife Admin Mod *
- Hibernate
- Hounddog
- HTML::Mason
- HTML::Template
- HTMLDOC
- ickle
- OpenVPN
- oprofile
- ORBit
- osCommerce *
- oSIP *
- PAI
- Pauker *
- Peep *
- Perl *
- PgMarket
- Pho
- PHP Layers Menu
- phpBB
- phpPgAds
- PIGALE
- Pike *
- PMD *
- Pngcrush
- Poptop
- Portslave *
- Postal
- Procmail Email Sanitizer
- ProFTPPD
- psad *
- PTlink Software
- Pure FTP Server *
- Puto Amo Window Manager
- pygame
- pygtk
- PyQt
- pyslsk *

- bonnie++
- Boot Scriptor *
- Bubblemon
- Bugzilla
- BW Whois *
- BZFlag
- Calamaris *
- Cannon Smash
- Cantus
- Caudium WebMail (CAMAS) *
- Caudium WebServer
- Ccl *
- cdbakeoven
- Cdrdao
- Celestia *
- centericq
- cgi-util
- Chronos
- chrony *
- Circus Linux!
- Clam AntiVirus
- clipper/xBase compatible compiler
- Common UNIX Printing System
- Compiere
- Cricket *
- curl
- DansGuardian
- Date::Calc *
- DbDesigner
- DBPrism
- DC++ *
- Dial-Up Downloader
- DiaSCE
- distcc
- dopewars *
- doxygen
- Driftnet *
- Drip
- Drivel
- Druid the database manager *
- DScaler
- Dump/Restore
- imgv
- IndexedCatalog
- Inflex
- INN
- Integratis
- Ion *
- IP Sorcery
- IP Tables State *
- ipband
- ipchains
- IPCop Firewall
- iptables
- IRC Services *
- IRMP3
- ispell *
- iSQL-Viewer *
- ivtools *
- j
- Jabber *
- Jakarta Avalon *
- Jetty
- JFreeChart *
- jmp *
- John the Ripper *
- JpGraph
- JSwat *
- JSX *
- K-3D
- KAlarm *
- Karbon14 *
- KBabel *
- kbarcode
- KBiff *
- KCDLabel
- KDirStat
- KGoodStuff *
- kio_fish
- Kismet
- Kiwi
- KMameRun *
- Knetfilter *
- Knewspost
- knoda
- Konqueror *
- kover
- Kpl
- python *
- python-ldap
- Q *
- QCad
- qsubst *
- RawWrite for Windows
- rdesktop
- Report Manager
- RIMPS *
- ripMIME
- ripperX
- Rockbox
- rp-pppoe
- RPL/2
- rsync *
- rubrica
- Ruby
- SalStat *
- samba
- SAMPEG
- sane
- Scintilla *
- Scribus
- ScummVM *
- SDCC
- SDL Sopwith
- sendip
- SETEdit *
- Sherpa
- Show Me The Money (smtm) *
- Sigit
- Sinek
- SIPAG Is a Photo Album Generator
- skylendar *
- small window manager
- SoundTracker
- SpamAssassin
- SpeedyCGI *
- Spreadsheet::WriteExcel
- Squid HTTP Proxy *
- Stellation *
- Straw *
- stunnel
- subversion *

- DUnit *
- dvd::rip
- DVDview *
- e3
- EasyTAG
- Eclipse (PHP) *
- ECLiPt Roaster *
- eDonkey2000 GUI *
- EffecTV
- ELOG
- elvis
- EPM
- epydoc
- Eraser *
- ERW *
- Eterm
- Etherboot *
- Euler
- Exim *
- Explore2fs
- Exuberant Ctags
- Exult *
- Eye of Gnome *
- FAM
- fastUtil
- FetchYahoo
- FileZilla
- fireparse
- Firestarter *
- FLAC
- FlightGear
- FLTK
- fluxbox
- fnord *
- Freecell Solver
- Freeciv *
- FreeLords
- FreeMarker
- Freeside *
- Freevo
- Fresco *
- fslint
- ftpcopy
- Ftpcube *
- fwlogwatch
- g-page
- KPSK *
- krfb
- KRunning
- KSendFax *
- KStars *
- KuickShow *
- KVIrc
- lcrzoex
- ldapdns
- LibCGI *
- libglade
- libicq2000
- libmng *
- libpng *
- lilypond
- LinEAK
- Links
- Linux Kernel *
- LInux LOader (LILO) *
- LinuxTrade
- Liquid War *
- LIRC *
- lyskom-server *
- LyX the Document Processor *
- magicfilter
- MailScanner
- Mandrake Update Robot
- MaraDNS *
- Maui Scheduler
- Memtest86
- memwatch
- MID
- Mig
- mkCDrec
- Mod_Survey
- Moleskine
- MonMotha's IPTables Firewall
- motion
- motor
- Mozilla *
- Mp3 Database
- MPlayer
- Sylpheed
- syn Text Editor
- synergy *
- syscalltrack
- Tamber
- Tapestry: Java Web Components *
- Taxipilot *
- Tcl
- tgif *
- The Fish *
- The Lua Language *
- The Open For Business Project *
- The RoboCup Soccer Simulator
- Thor
- TightVNC
- TiLP *
- tin
- TkCVS *
- TkSql *
- tomsrtbt
- Tornado *
- tree.hh
- Turbo Vision *
- Tux Paint *
- TuxCards *
- TWiki
- twin *
- Twisted *
- TYPO3
- UebiMiau
- Umbrello UML Modeller
- UML2EJB code generator *
- UniGNUPlot -Part of UniCalculus
- Unison *
- unixODBC
- UnZip *
- Video4Linux Grab *
- VietPad
- Vim
- vpopmail *
- Waimea *

- Gael
- Gallery
- Gammu
- gcombust
- gcompris *
- Gem Drop X *
- gentoo *
- Getleft
- getmail *
- gFTP
- GL-117
- GlobeCom Jukebox
- Gnome MlView *
- GNOME-PIM *
- GnomeMeeting *
- gnozip
- Gnu awk *
- GNU Backgammon
- GNU coreutils *
- GNU Grep *
- GNU jwhois
- GNU Mailman *
- GNU make *
- MRTG
- MTX *
- Mutella
- MySQL *
- naim *
- Narval
- Natural Language Toolkit
- Nautilus
- ne
- Netatalk *
- NetHack *
- Nmap *
- Nomad II Utils
- ntop
- Numerical Python *
- nut *
- OfflineIMAP *
- ogle
- omniORB *
- Open Webmail
- Openbox
- OpenOffice Draw/Impress
- washerDryer
- webcalendar
- WebGUI *
- WebMail/Java *
- Wherever Change Directory
- WIMS
- Window Maker *
- Wine *
- wxWindows *
- XBindKeys
- xdrawchem *
- XFree86 *
- xlog
- xmame *
- xmcd *
- xmmix
- XOSD
- xpad
- Xplanet
- xrdesktop *
- xrmap
- XWine
- z
- zsh

---

christian reis <kiko@async.com.br>

Apêndice E

Lista de Resultados Globais do Questionário

The Free Software Engineering Survey

Global Survey Results

We list here the results obtained in the survey up to now. If you would like this data in a convenient format, please let me know.

Totals

- Number of requests sent out: 1102
- Number of registered projects: 595 (54% of sent)
- Number of projects which completed surveys: 570 (95.8% of registered)
- Number of projects with invalid surveys: 22 (3.7% of registered)
- Number of responses that covered distributions or metaprojects: 31 (5.2% of registered)
- Total number of projects regarded in global counts: 519 (87.2% of registered)

- Number of completed surveys with valid tarballs: 521 (87.6% of registered)
- Number of completed surveys with valid tarballs and LOC count: 521 (87.6% of registered)

Project Counts by Site

(Yeah, add up sf.net and sourceforge.net)

- other: 400,
- sourceforge.net: 98,
- sf.net: 42,
- gnu.org: 15,
- gnome.org: 7,
- kde.org: 6,
- com.au: 6,
- free.fr: 5,
- newbreedsoftware.com: 4,
- freshmeat.net: 4,
- com.br: 4,

Project Counts by Internet Area

(44 different domains in survey)

com: 128, org: 119, net: 83, de: 42, edu: 23, uk: 14, fr: 14, au: 11, se: 10, nl: 9, it: 8, ca: 5, ch: 5, fi: 4, ru: 3, nu: 3, at: 3, il: 3, ar: 3, hu: 3, br: 3, es: 3, no: 3, pl: 3, cz: 2, be: 2, us: 2, ie: 2, jp: 2, dk: 2, li: 1, CA: 1, : 1, tw: 1, za: 1, ro: 1, gr: 1, sk: 1, mx: 1, am: 1, tr: 1, NET: 1, cx: 1, nz: 1,

Project Counts by Domain

- Audio, Video and Media Recording: 54
- Basic System and Unix Software, Printing, Backup: 52
- Browsers, Viewers, Editors and Office Applications: 50
- Business, Finance and E-Commerce: 8
- Databases and supporting applications: 16
- Electronic Mail and News: 34
- Emulation: 8
- Games: 26
- Graphics, Animation and Imaging: 25
- Mechanical, Civil and Electrical Engineering: 3
- Miscellaneous: 28
- Networking and Security: 78
- Personal Messenging and Peer to Peer Systems: 25
- Scientific Applications: 12
- Software Engineering, Development and Middleware: 106
- Window Systems and supporting applications: 23

Average Scores

- Software Process: 27
- Tools: 35
- Software Engineering Tools: 41
- Redcarpet: 38
- Institutional Support: 12
- Requirements Effort: 37
- Usability Effort: 12
- Documentation Effort: 37
- Quality Assurance Effort: 25

Summary of project sourcecode data

- Average size of project tarball: 1717.873 KBytes
- Average time since last public release: 216 days
- Average number of languages per project: 3.188 languages
- Average of total LOC per project: 60.206 KSLOC

Survey answers

1.1: Motivations	Multiple choices possible
a. The project was started *primarily* for personal reasons (had desire to learn, found technology interesting, etc).	369 (71.1%)
b. The project was started *primarily* to produce software to support another Open Source/Free Software project	80 (15.4%)
c. The project was started (or sponsored) by a company or organization.	69 (13.3%)
d. The project was started as part of academic (university) work or research.	62 (11.9%)
e. The project's intention *from the beginning* was to produce at least part of its software as Open Source/Free Software.	304 (58.6%)
f. The project began based on pre-existing Open Source/Free Software (code fork, etc.).	113 (21.8%)
g. The project began based on pre-existing proprietary/closed source software.	31 (6.0%)
h. Other	59 (11.4%)
(Did not choose any)	5 (1.0%)

1.2: User Base	Multiple choices possible
a. Yourself.	390 (75.1%)
b. The project team	214 (41.2%)
c. Users that are not computer-proficient or non-technical.	193 (37.2%)
d. Users in specific fields of expertise (e.g. amateur radio enthusiasts, mathematicians, engineers, software developers). *Please specify which in the comment box below.*	220 (42.4%)
e. Users in a particular company, site or organization	75 (14.5%)
f. One or more specific commercial customers (sponsors/paying).	66 (12.7%)
g. The Free Software/Open Source community.	323 (62.2%)
h. The computing community.	216 (41.6%)
i. Other	62 (11.9%)
(Did not choose any)	2 (0.4%)

1.3: Project age	Single choice
a. Less than 6 months	27 (5.2%)
b. 6 months to a year	45 (8.7%)
c. 1-2 years	128 (24.7%)
d. 2-5 years	207 (39.9%)
e. Over 5 years	109 (21.0%)
Other	3 (0.6%)

1.4: Pre-existing standard	Single choice
a. Yes.	167 (32.2%)
b. No.	319 (61.5%)

Other	27 (5.2%)
(Left unanswered)	6 (1.2%)

2.1: Team size	Single choice
a. 1	164 (31.6%)
b. 2-5	220 (42.4%)
c. 6-15	97 (18.7%)
d. 16-25	16 (3.1%)
e. 26-50	10 (1.9%)
f. More than 50	11 (2.1%)
Other	1 (0.2%)

2.2: Leadership model	Single choice
a. The project has a single leader and a number of people that are formally responsible for parts of it.	90 (17.3%)
b. The project has a single leader, which delegates responsibility occasionally to others.	119 (22.9%)
c. The project is led by a core group or committee of people that is responsible for making decisions; there is no single leader.	87 (16.8%)
d. There is no effective leadership structure.	13 (2.5%)
e. This is a single-person project.	183 (35.3%)
Other	27 (5.2%)

2.3: General team aspects	Multiple choices possible
a. Most people in the team know each other only through the Internet, and have never met physically/personally.	323 (62.2%)
b. Most people in the team work physically close, and meet personally with some frequency.	57 (11.0%)
c. The team includes people that have more than 5 years experience in serious software development.	290 (55.9%)
d. The project provides a high barrier of entry to new participants, even to those that are skilled in software development.	104 (20.0%)
e. A contributor to the project will often participate in more than one project activity: functionality definition, architectural design, designing user interfaces , coding, testing, project management, etc.	208 (40.1%)
f. Code contributors tend to work only on one specific language or technology used in the project (the one which they are most familiar with).	117 (22.5%)
g. Other	46 (8.9%)
(Did not choose any)	35 (6.7%)

3.1: Defining Functionality	Multiple choices possible
a. The project's software explicitly mimics (or is significantly based upon) the functionality or behavior of an existing (proprietary or Free Software/Open Source) software package. *If so, please specify which?*	185 (35.6%)

b. A significant amount of effort is spent defining what the software functionality and behavior (the requirements) should be.	222 (42.8%)
c. This project has no end-users apart from the project team.	7 (1.3%)
d. There have been meetings or discussions with end-users to define significant parts of the software functionality or behavior.	195 (37.6%)
e. These meetings/discussions occur frequently, and can be considered an important part of defining the project's software.	97 (18.7%)
f. The software functionality is implemented according to what the team thinks is correct, without significant external end-user input.	209 (40.3%)
g. I believe much of the expected functionality and behavior is **not** completely known or understood by the project team as a whole.	50 (9.6%)
h. Other	62 (11.9%)
(Did not choose any)	7 (1.3%)

3.2: Usability	Multiple choices possible
a. The user interfaces for the project are designed (or prototyped) and refined before actually being implemented.	144 (27.7%)
b. We have conducted serious usability tests and studies on the project's user interfaces.	34 (6.6%)
c. Developers are not allowed to implement or change the user interfaces before the implementation/change has been reviewed and approved.	61 (11.8%)
d. A part of the team is specifically in charge of UI design.	41 (7.9%)
e. We would like to invest more in usability, but we are hampered by lack of documentation and/or team knowledge on the subject.	59 (11.4%)
f. This project doesn't have any significant user interface.	156 (30.1%)
g. Other	117 (22.5%)
(Did not choose any)	50 (9.6%)

3.3: Documentation	Multiple choices possible
a. We have produced documents that describe at least some of the expected functionality and behavior (requirements) of our software.	361 (69.6%)
b. The project produces and maintains documents for developers that describe how the code is organized (architecture), and/or how parts of it work.	156 (30.1%)
c. There is a reasonably complete coding standards guide that is actively followed by the team.	124 (23.9%)
d. There is documentation for the end-user available for the project's software (consider also third-party documents available).	403 (77.6%)
e. End-user documentation is provided to a large extent by people or groups external to the project team.	61 (11.8%)
f. A significant part of the available documentation is frequently updated and revised to be up to date.	284 (54.7%)
g. Other	39 (7.5%)
(Did not choose any)	12 (2.3%)

3.4: Quality Assurance	Multiple choices possible

a. There is an [automated] test suite for the project's software, that is used to validate it.	141 (27.2%)
b. There is a test plan (a written document describing tests) for the project's software, that is used by the project team.	51 (9.8%)
c. Periodic (i.e. nightly, weekly) snapshots of the project's code (or binaries) are distributed and used as an significant means of testing the software.	140 (27.0%)
d. There is an active code review process, where code is read by other members of the team.	119 (22.9%)
e. The project team members have formal rules for integrating code changes into the main codebase, and review is a strict requirement.	78 (15.0%)
f. Usually, an unexpected amount of bugs are discovered after a version has been publically released (this includes public release candidates).	107 (20.6%)
g. There is a tendency (or policy) to release a public version only when it has been *extensively* tested by the team.	289 (55.7%)
h. Other	72 (13.9%)
(Did not choose any)	42 (8.1%)

3.5: Tools	Multiple choices possible
a. A project hosting site such as Sourceforge.net, Savannah or Collab.net (mark all others applicable as well)	296 (57.0%)
b. One or more Web sites	348 (67.1%)
c. A WikiWikiWeb site (SWiki, TWiki, PHPWiki, ZWiki, etc.)	32 (6.2%)
d. A Frequently Asked Questions document	194 (37.4%)
e. A version control tool such as CVS, RCS, Subversion or Bitkeeper	380 (73.2%)
f. A bug database such as GNATS or Bugzilla	175 (33.7%)
g. One or more mailing lists	347 (66.9%)
h. Network news (NNTP)	36 (6.9%)
i. User forums/BBS	67 (12.9%)
j. IRC	116 (22.4%)
k. An instant messaging system like ICQ, Jabber or AIM	46 (8.9%)
l. Other	50 (9.6%)
(Did not choose any)	2 (0.4%)

Note: for multiple-choice answers, the totals for the different percentages may surpass 100%. This is by design, since a single project can have more than one option selected.

Apêndice F

Questionário Exemplo: Berkeley DB

The Free Software Engineering Survey

View survey for Berkeley DB (bostic)

<< Prev Browse other surveys Next >>

Domain: Databases and supporting applications		
Last release date: 2002-09-13 (167 days ago)	Tarball size:	3091 K
SLOC Data:	ansic:	95626 LOC
	tcl:	30484 LOC
	perl:	11602 LOC
	java:	7653 LOC
	cpp:	5571 LOC
	sh:	3264 LOC
	awk:	1338 LOC
	sed:	772 LOC
	cs:	505 LOC
	asm:	14 LOC
	Total:	156829 LOC

1.1: Motivations Multiple choices possible

a. The project was started *primarily* for personal reasons (had desire to learn, found technology interesting, etc).

d. The project was started as part of academic (university) work or research.

e. The project's intention *from the beginning* was to produce at least part of its software as Open Source/Free Software.

Comment:
The project began as part of the work to remove AT&T proprietary code from the University of California Berkeley's 4.4BSD distribution.

1.2: User Base Multiple choices possible

f. One or more specific commercial customers (sponsors/paying).

g. The Free Software/Open Source community.

Comment:
Berkeley DB is heavily used in the Open Source community but we have a large commercial base as well.

1.3: Project age — Single choice

e. Over 5 years

Comment:
First public release was probably around 1990.

1.4: Pre-existing standard — Single choice

b. No.

2.1: Team size — Single choice

c. 6-15

2.2: Leadership model — Single choice

c. The project is led by a core group or committee of people that is responsible for making decisions; there is no single leader.

2.3: General team aspects — Multiple choices possible

c. The team includes people that have more than 5 years experience in serious software development.

d. The project provides a high barrier of entry to new participants, even to those that are skilled in software development.

e. A contributor to the project will often participate in more than one project activity: functionality definition, architectural design, designing user interfaces , coding, testing, project management, etc.

Comment:
Berkeley DB doesn't have a public community -- the barrier to entry is so high that we rarely get outside contributions. So the "community" is Sleepycat Software employees.

3.1: Defining Functionality — Multiple choices possible

b. A significant amount of effort is spent defining what the software functionality and behavior (the requirements) should be.

d. There have been meetings or discussions with end-users to define significant parts of the software functionality or behavior.

e. These meetings/discussions occur frequently, and can be considered an important part of defining the project's software.

Comment:
Our "end-users" include many paying customers so they tend to drive the features/functionality of the product.

3.2: Usability — Multiple choices possible

f. This project doesn't have any significant user interface.

3.3: Documentation Multiple choices possible

a. We have produced documents that describe at least some of the expected functionality and behavior (requirements) of our software.

c. There is a reasonably complete coding standards guide that is actively followed by the team.

d. There is documentation for the end-user available for the project's software (consider also third-party documents available).

f. A significant part of the available documentation is frequently updated and revised to be up to date.

Comment:
We spend a lot of time on documentation; the product is written for engineers so we have a complete API guide for multiple languages and a complete Reference Guide with tutorial.

3.4: Quality Assurance Multiple choices possible

a. There is an [automated] test suite for the project's software, that is used to validate it.

b. There is a test plan (a written document describing tests) for the project's software, that is used by the project team.

c. Periodic (i.e. nightly, weekly) snapshots of the project's code (or binaries) are distributed and used as an significant means of testing the software.

d. There is an active code review process, where code is read by other members of the team.

e. The project team members have formal rules for integrating code changes into the main codebase, and review is a strict requirement.

g. There is a tendency (or policy) to release a public version only when it has been *extensively* tested by the team.

Comment:
We have 50000 lines of Tcl test script -- tests take 2-3 days to run on the fastest machines we can buy.

3.5: Tools Multiple choices possible

b. One or more Web sites

e. A version control tool such as CVS, RCS, Subversion or Bitkeeper

f. A bug database such as GNATS or Bugzilla

g. One or more mailing lists

Scores:

- Software Process: 70
- Tools: 36
- Software Engineering Tools: 57
- Redcarpet: 23
- Institutional Support: 45
- Requirements Effort: 95

- Usability Effort: 0
- Documentation Effort: 62
- Quality Assurance Effort: 100

Browse other surveys

---

the free software engineering survey : June 2002

---

christian reis <kiko@async.com.br>

Apêndice G

Questionário Exemplo: GNU Make

The Free Software Engineering Survey

View survey for GNU make (psmith)

<< Prev Browse other surveys Next >>

Domain: Software Engineering, Development and Middleware		
Last release date: 2002-10-04 (146 days ago)	Tarball size:	1212 K
SLOC Data:	ansic:	22270 LOC
	perl:	991 LOC
	sh:	524 LOC
	sed:	16 LOC
	Total:	23801 LOC

1.1: Motivations Multiple choices possible

b. The project was started *primarily* to produce software to support another Open Source/Free Software project

e. The project's intention *from the beginning* was to produce at least part of its software as Open Source/Free Software.

h. Other: An integral component of the FSF's GNU project to produce a UNIX workalike

1.2: User Base Multiple choices possible

a. Yourself.

d. Users in specific fields of expertise (e.g. amateur radio enthusiasts, mathematicians, engineers, software developers). *Please specify which in the comment box below.*

g. The Free Software/Open Source community.

h. The computing community.

Comment:
Primarily software developers but GNU make is used by webmasters document writers etc. etc.

Up there with Emacs and GCC as the GNU tools probably most widely used across all types of development (free software commercial proprietary etc.)

1.3: Project age Single choice

e. Over 5 years

1.4: Pre-existing standard Single choice

a. Yes.

Comment:
POSIX (IEEE 1003.1)

2.1: Team size Single choice

a. 1

Comment:
There are probably 2-3 other people who are semi-active; supporting Windows ports etc. But I'm the only one with code commit privileges.

2.2: Leadership model Single choice

b. The project has a single leader, which delegates responsibility occasionally to others.

Comment:
I delegate all the non-UNIX ports to others and integrate their work; I pretty much run the UNIX side although I often accept patches (many times I rewrite some or all however).

2.3: General team aspects Multiple choices possible

a. Most people in the team know each other only through the Internet, and have never met physically/personally.

c. The team includes people that have more than 5 years experience in serious software development.

f. Code contributors tend to work only on one specific language or technology used in the project (the one which they are most familiar with).

Comment:
I thought about checking (d) but make is a grab-bag: some parts are quite accessible to newcomers but much of the guts are fairly impenetrable.

3.1: Defining Functionality Multiple choices possible

a. The project's software explicitly mimics (or is significantly based upon) the functionality or behavior of an existing (proprietary or Free Software/Open Source) software package. *If so, please specify which?*

f. The software functionality is implemented according to what the team thinks is correct, without significant external end-user input.

Comment:
Make receives a large number of "suggested enhancements". However I am extremely stingy when making any user-visible changes: I definitely prefer to err on the side of caution. GNU make is used with probably hundreds of thousands of makefiles and is a lynchpin for hundreds of free software projects and I'm very concious of the need for backward compatibility and stability.

3.2: Usability Multiple choices possible

f. This project doesn't have any significant user interface.

Comment:
You could consider makefiles "user interfaces"; as mentioned above I'm extremely conservative with allowing user-visible changes. Where they are allowed they almost always must be backward compatible. Where things are not backward compatible they must be "edge cases" which are deemed to be rarely ocurring "in the wild".

3.3: Documentation Multiple choices possible

a. We have produced documents that describe at least some of the expected functionality and behavior (requirements) of our software.

c. There is a reasonably complete coding standards guide that is actively followed by the team.

d. There is documentation for the end-user available for the project's software (consider also third-party documents available).

f. A significant part of the available documentation is frequently updated and revised to be up to date.

Comment:
The GNU make manual is considered to be one of the better manuals available.

3.4: Quality Assurance Multiple choices possible

a. There is an [automated] test suite for the project's software, that is used to validate it.

e. The project team members have formal rules for integrating code changes into the main codebase, and review is a strict requirement.

g. There is a tendency (or policy) to release a public version only when it has been *extensively* tested by the team.

Comment:
I'm also conservative about the number of releases. GNU make provides a fairly extensive suite of unit tests (run with "make check"). For feature and system testing I typically build make on a number of systems with different compilers and attempt to build a number of large software packages like Linux Emacs GCC etc. with the new version of make.

3.5: Tools Multiple choices possible

a. A project hosting site such as Sourceforge.net, Savannah or Collab.net (mark all others applicable as well)

b. One or more Web sites

e. A version control tool such as CVS, RCS, Subversion or Bitkeeper

f. A bug database such as GNATS or Bugzilla

g. One or more mailing lists

h. Network news (NNTP)

Comment:
GNU make is hosted on Savannah. For bugs users can report them via the bug tracker in Savannah which is not really a GNATS or Bugzilla. Probably 95% of bug reports are simply sent to the bug-make mailing list however and not recorded in Savannah.

Scores:

- Software Process: 37
- Tools: 55
- Software Engineering Tools: 71

- Redcarpet: 51
- Institutional Support: 0
- Requirements Effort: 25
- Usability Effort: 0
- Documentation Effort: 56
- Quality Assurance Effort: 56

Browse other surveys

---

the free software engineering survey : June 2002

---

christian reis <kiko@async.com.br>

Apêndice H

Questionário Exemplo: Gnumeric

The Free Software Engineering Survey

View survey for Gnumeric (jody)

<< Prev Browse other surveys Next >>

Domain: Browsers, Viewers, Editors and Office Applications		
Last release date: 2002-11-15 (104 days ago)	Tarball size:	12968 K
SLOC Data:	ansic:	195908 LOC
	yacc:	1034 LOC
	lisp:	200 LOC
	perl:	114 LOC
	python:	73 LOC
	Total:	197329 LOC

1.1: Motivations Multiple choices possible

b. The project was started *primarily* to produce software to support another Open Source/Free Software project

Comment:
My understanding of Miguel's intentions in starting Gnumeric is that his wanted to build a non-trivial application to use as a test bed for the GNOME projects libraries.

1.2: User Base Multiple choices possible

d. Users in specific fields of expertise (e.g. amateur radio enthusiasts, mathematicians, engineers, software developers). *Please specify which in the comment box below*.

Comment:
The main user group is not surprisingly 'spreadsheet users'. I don't mean to be self referential there but the set of people who use spreadsheets is in itself somewhat limited compared to other productivity applications. There is a strong representation from the financial community and sciences.

1.3: Project age Single choice

d. 2-5 years

Comment:
The first documented release was in Aug 1998

1.4: Pre-existing standard Single choice

b. No.

Comment:
Although there is no official or documented standard Gnumeric is compared against its proprietary competators. As a result it's target is a super set of MS Excel Quattro Lotus Applix and OpenOffice. That applies to both functionality and for file formats.

2.1: Team size Single choice

c. 6-15

Comment:
There is a significant user base and community of commentators and bug reporters. However the number of active contributors for code docs or testing is limited to < 15. That set could potentially be increased if the general translator pool is included.

2.2: Leadership model Single choice

Other: There is a single leader with several trusted general contributors

Comment:
with a large base of occasional contributors whose work requires review by the primary team.

2.3: General team aspects Multiple choices possible

a. Most people in the team know each other only through the Internet, and have never met physically/personally.

e. A contributor to the project will often participate in more than one project activity: functionality definition, architectural design, designing user interfaces , coding, testing, project management, etc.

Comment:
The primary team is probably one of the oldest around. Most of us are at least 30 with some significantly older.

3.1: Defining Functionality Multiple choices possible

a. The project's software explicitly mimics (or is significantly based upon) the functionality or behavior of an existing (proprietary or Free Software/Open Source) software package. *If so, please specify which?*

d. There have been meetings or discussions with end-users to define significant parts of the software functionality or behavior.

e. These meetings/discussions occur frequently, and can be considered an important part of defining the project's software.

Comment:
Spreadsheets are extremely complex applications. They bridge the divide between developer tools and raw user applications. As such there is a rich collection of behaviors and interations which are not well defined. In some cases it is reasonable to use an existing Application as a template in others we attempt to poll the user base or a specialist community (like the gnome usability team) for input on what they would expect something to do.

3.2: Usability Multiple choices possible

e. We would like to invest more in usability, but we are hampered by lack of documentation and/or team knowledge on the subject.

g. Other: The user interface evolves

Comment:
Our main constraints on usability are feature set limitations in comparison to MS Excel. Thankfully their interface is not terribly user friendly so we don't have to go to far to be more usable. However people do expect us to implement all of their pet features from MS Excel.

3.3: Documentation Multiple choices possible

a. We have produced documents that describe at least some of the expected functionality and behavior (requirements) of our software.

b. The project produces and maintains documents for developers that describe how the code is organized (architecture), and/or how parts of it work.

c. There is a reasonably complete coding standards guide that is actively followed by the team.

f. A significant part of the available documentation is frequently updated and revised to be up to date.

g. Other: The majority of the function documentation has been translated

3.4: Quality Assurance Multiple choices possible

a. There is an [automated] test suite for the project's software, that is used to validate it.

d. There is an active code review process, where code is read by other members of the team.

e. The project team members have formal rules for integrating code changes into the main codebase, and review is a strict requirement.

Comment:
(e) is only true for the stable branch.
The development tree is intentionally more open to change. Howver contributions from outside the primary team must be reviewed and sponsored before inclusion.

3.5: Tools Multiple choices possible

b. One or more Web sites

e. A version control tool such as CVS, RCS, Subversion or Bitkeeper

f. A bug database such as GNATS or Bugzilla

g. One or more mailing lists

j. IRC

l. Other: Various GNOME support teams (translation usability documentation)

Scores:

- Software Process: 52
- Tools: 45
- Software Engineering Tools: 62
- Redcarpet: 23
- Institutional Support: 0
- Requirements Effort: 60
- Usability Effort: 0
- Documentation Effort: 56
- Quality Assurance Effort: 56

Apêndice I

Questionário Exemplo: Linux

The Free Software Engineering Survey

View survey for Linux Kernel (riel)

<< Prev Browse other surveys Next >>

Domain: Basic System and Unix Software, Printing, Backup		
Last release date: 2002-08-03 (208 days ago)	Tarball size:	32220 K
SLOC Data:	ansic:	2781478 LOC
	asm:	154562 LOC
	sh:	3170 LOC
	perl:	2191 LOC
	yacc:	1573 LOC
	cpp:	757 LOC
	lex:	748 LOC
	tcl:	577 LOC
	awk:	251 LOC
	lisp:	218 LOC
	sed:	72 LOC
	Total:	2945597 LOC

1.1: Motivations Multiple choices possible

a. The project was started *primarily* for personal reasons (had desire to learn, found technology interesting, etc).

Comment:
Documented in various places. IIRC Linus wanted to learn about x86 task switching and protected mode.

1.2: User Base Multiple choices possible

h. The computing community.

Comment:
Popular system getting used by all kinds of people and organisations.

1.3: Project age Single choice

e. Over 5 years

1.4: Pre-existing standard Single choice

a. Yes.

Comment:
POSIX.1 / Single Unix Standard. Note that this only defines part of the API it doesn't define any of the internals of the system.

2.1: Team size Single choice

f. More than 50

2.2: Leadership model Single choice

Other: Single gate-keeper trying to fight off a storm of patches.

Comment:
There is a huge community of developers and an excess of new code and new ideas. Linux maintainers are mostly in the business of refusing new code in order to make sure the few things that do get in are of high enough quality to maintain for the following years to come.

2.3: General team aspects Multiple choices possible

c. The team includes people that have more than 5 years experience in serious software development.

e. A contributor to the project will often participate in more than one project activity: functionality definition, architectural design, designing user interfaces , coding, testing, project management, etc.

3.1: Defining Functionality Multiple choices possible

b. A significant amount of effort is spent defining what the software functionality and behavior (the requirements) should be.

e. These meetings/discussions occur frequently, and can be considered an important part of defining the project's software.

g. I believe much of the expected functionality and behavior is **not** completely known or understood by the project team as a whole.

h. Other: The structure of the system evolves

Comment:
Natural selection and criticism between developers quickly weeds out bad code. There is a high barrier of entry for new code and only bad code is just refused until it has been improved. There is no high-level goal for where the system should go in the future; every developer has his/her own goals and a compromise in code is found.

3.2: Usability Multiple choices possible

f. This project doesn't have any significant user interface.

Comment:
The API is about it not really an end user thing...

3.3: Documentation Multiple choices possible

a. We have produced documents that describe at least some of the expected functionality and behavior (requirements) of our software.

b. The project produces and maintains documents for developers that describe how the code is organized (architecture), and/or how parts of it work.

c. There is a reasonably complete coding standards guide that is actively followed by the team.

d. There is documentation for the end-user available for the project's software (consider also third-party documents available).

e. End-user documentation is provided to a large extent by people or groups external to the project team.

3.4: Quality Assurance Multiple choices possible

a. There is an [automated] test suite for the project's software, that is used to validate it.

c. Periodic (i.e. nightly, weekly) snapshots of the project's code (or binaries) are distributed and used as an significant means of testing the software.

d. There is an active code review process, where code is read by other members of the team.

g. There is a tendency (or policy) to release a public version only when it has been *extensively* tested by the team.

h. Other: Never enough testing possible

Comment:
After all kernel hackers are happy with a kernel version the rest of the world quickly comes up with workloads that behave differently and break everything ;)

3.5: Tools Multiple choices possible

e. A version control tool such as CVS, RCS, Subversion or Bitkeeper

g. One or more mailing lists

j. IRC

Scores:

- Software Process: 48
- Tools: 27
- Software Engineering Tools: 33
- Redcarpet: 23
- Institutional Support: 0
- Requirements Effort: 70
- Usability Effort: 0
- Documentation Effort: 38
- Quality Assurance Effort: 63

Browse other surveys

Apêndice J

Questionário Exemplo: Perl

The Free Software Engineering Survey

View survey for Perl (larry)

<< Prev Browse other surveys Next >>

Domain: Software Engineering, Development and Middleware		
Last release date: 2002-07-18 (224 days ago)	Tarball size:	11023 K
SLOC Data:	perl:	257463 LOC
	ansic:	142628 LOC
	sh:	28251 LOC
	pascal:	6973 LOC
	lisp:	5992 LOC
	cpp:	2035 LOC
	yacc:	1018 LOC
	java:	23 LOC
	Total:	444383 LOC

1.1: Motivations Multiple choices possible

a. The project was started *primarily* for personal reasons (had desire to learn, found technology interesting, etc).

e. The project's intention *from the beginning* was to produce at least part of its software as Open Source/Free Software.

Comment:
Perl was started at a company in support of a project the company was doing for the government but it was not really sponsored by the company.

1.2: User Base Multiple choices possible

a. Yourself.

b. The project team

c. Users that are not computer-proficient or non-technical.

d. Users in specific fields of expertise (e.g. amateur radio enthusiasts, mathematicians, engineers, software developers). *Please specify which in the comment box below.*

g. The Free Software/Open Source community.

h. The computing community.

i. Other: Everyone who uses the Web

Comment:
It is difficult to find any segment of the computing community that is not using Perl (unless of course you classify by which language is preferred :-). But in particular Perl is heavily used by system administrators by Web programmers (and indirectly by those who use the Web) as well as by bioinformaticists and by financial analysts.

1.3: Project age Single choice

e. Over 5 years

Comment:
Perl was released in 1987.

1.4: Pre-existing standard Single choice

Other: Programming Perl (The Camel Book) O'Reilly & Associates

Comment:
But the manpages are more up-to-date.

2.1: Team size Single choice

f. More than 50

Comment:
Gobs. It really depends on how you count but there are hundreds of contributors just to Perl 5 not counting Perl 6.

2.2: Leadership model Single choice

a. The project has a single leader and a number of people that are formally responsible for parts of it.

Comment:
There are also lots of people who are informally responsible for parts of it. I was also strongly tempted to pick "d" since I'm a singularly ineffective leader.

2.3: General team aspects Multiple choices possible

c. The team includes people that have more than 5 years experience in serious software development.

d. The project provides a high barrier of entry to new participants, even to those that are skilled in software development.

e. A contributor to the project will often participate in more than one project activity: functionality definition, architectural design, designing user interfaces , coding, testing, project management, etc.

g. Other: Most people in the team know each other primarily through the Internet and meet physically every now and then at conferences.

Comment:
"a" was true before the first Perl Conference.

3.1: Defining Functionality Multiple choices possible

b. A significant amount of effort is spent defining what the software functionality and behavior (the requirements) should be.

d. There have been meetings or discussions with end-users to define significant parts of the software functionality or behavior.

e. These meetings/discussions occur frequently, and can be considered an important part of defining the project's software.

f. The software functionality is implemented according to what the team thinks is correct, without significant external end-user input.

g. I believe much of the expected functionality and behavior is **not** completely known or understood by the project team as a whole.

Comment:
I realize that some of these are contradictory. Nonetheless they're all true.

3.2: Usability Multiple choices possible

a. The user interfaces for the project are designed (or prototyped) and refined before actually being implemented.

b. We have conducted serious usability tests and studies on the project's user interfaces.

c. Developers are not allowed to implement or change the user interfaces before the implementation/change has been reviewed and approved.

d. A part of the team is specifically in charge of UI design.

f. This project doesn't have any significant user interface.

g. Other: This project is a 15 year experiment in user-interface design.

Comment:
Some of these are a little odd because a computer language doesn't have a user interface it *is* a user interface.

3.3: Documentation Multiple choices possible

a. We have produced documents that describe at least some of the expected functionality and behavior (requirements) of our software.

b. The project produces and maintains documents for developers that describe how the code is organized (architecture), and/or how parts of it work.

c. There is a reasonably complete coding standards guide that is actively followed by the team.

d. There is documentation for the end-user available for the project's software (consider also third-party documents available).

e. End-user documentation is provided to a large extent by people or groups external to the project team.

f. A significant part of the available documentation is frequently updated and revised to be up to date.

g. Other: We even have our own documentation format POD.

Comment:
We think we have spectacularly good documentation compared to most projects. Nevertheless it could be greatly improved. And there's not much we can do about the problem of having *too much* documentation...

3.4: Quality Assurance Multiple choices possible

a. There is an [automated] test suite for the project's software, that is used to validate it.

b. There is a test plan (a written document describing tests) for the project's software, that is used by the project team.

c. Periodic (i.e. nightly, weekly) snapshots of the project's code (or binaries) are distributed and used as an significant means of testing the software.

d. There is an active code review process, where code is read by other members of the team.

e. The project team members have formal rules for integrating code changes into the main codebase, and review is a strict requirement.

g. There is a tendency (or policy) to release a public version only when it has been *extensively* tested by the team.

h. Other: Development releases are expected to be tested by the wide world.

Comment:
Perl was one of the first open source projects with an extensive regression test suite. At the moment there are over 65000 tests in it. Point "g" is true only if you take "h" into account. We use an odd/even scheme for development/stable releases.

3.5: Tools Multiple choices possible

b. One or more Web sites

d. A Frequently Asked Questions document

e. A version control tool such as CVS, RCS, Subversion or Bitkeeper

f. A bug database such as GNATS or Bugzilla

g. One or more mailing lists

h. Network news (NNTP)

i. User forums/BBS

j. IRC

l. Other: Personal email telephone conferences summit meetings of the "inner ringers"

Comment:
Actually "h" is not so true any more but was vital in the early days.

Scores:

- Software Process: 100
- Tools: 73
- Software Engineering Tools: 76
- Redcarpet: 35
- Institutional Support: 0
- Requirements Effort: 100
- Usability Effort: 100
- Documentation Effort: 91
- Quality Assurance Effort: 100

Browse other surveys

christian reis <kiko@async.com.br>

Apêndice K

Questionário Exemplo: XFree86

The Free Software Engineering Survey

View survey for XFree86 (keithp)

<< Prev Browse other surveys Next >>

Domain: Window Systems and supporting applications		
Last release date: 2002-09-12 (168 days ago)	Tarball size:	58496 K
SLOC Data:	ansic:	1913424 LOC
	cpp:	37081 LOC
	sh:	34486 LOC
	asm:	14700 LOC
	pascal:	13773 LOC
	tcl:	9182 LOC
	yacc:	3397 LOC
	python:	2383 LOC
	perl:	2292 LOC
	objc:	1710 LOC
	lex:	1643 LOC
	awk:	933 LOC
	lisp:	304 LOC
	csh:	58 LOC
	Total:	2035366 LOC

1.1: Motivations Multiple choices possible

f. The project began based on pre-existing Open Source/Free Software (code fork, etc.).

Comment:
XFree86 started much like Apache. A small group of people patching X independently banded together to pool patches. While it's nominally an X fork XFree86 started in reaction to a closed source X fork X386.

1.2: User Base — Multiple choices possible

- **a.** Yourself.
- **b.** The project team
- **c.** Users that are not computer-proficient or non-technical.
- **g.** The Free Software/Open Source community.
- **h.** The computing community.

1.3: Project age — Single choice

- **e.** Over 5 years

1.4: Pre-existing standard — Single choice

- **a.** Yes.

Comment:
XFree86 has followed the X consoritum standards for many years. In the past year we have started publishing new standards based on our own work now that XFree86 drives the X community.

2.1: Team size — Single choice

- **f.** More than 50

Comment:
The XFree86 core team is a small group each spending a significant amount of time working on X related activities beyond that a wide community of less focused developers do work generally in localized areas. People making significant contributions over a long period of time and who are interested may be asked to join the core team.

2.2: Leadership model — Single choice

- **a.** The project has a single leader and a number of people that are formally responsible for parts of it.

Comment:
David Dawes is the XFree86 president and project leader. The XFree86 core team works together to define project goals and help coordinate minor efforts but David sets the overall tone and direction.

2.3: General team aspects — Multiple choices possible

- **a.** Most people in the team know each other only through the Internet, and have never met physically/personally.
- **c.** The team includes people that have more than 5 years experience in serious software development.
- **d.** The project provides a high barrier of entry to new participants, even to those that are skilled in software development.
- **e.** A contributor to the project will often participate in more than one project activity: functionality definition, architectural design, designing user interfaces , coding, testing, project management, etc.

f. Code contributors tend to work only on one specific language or technology used in the project (the one which they are most familiar with).

g. Other: The core team has arranged several meetings and will continue to do so.

3.1: Defining Functionality Multiple choices possible

a. The project's software explicitly mimics (or is significantly based upon) the functionality or behavior of an existing (proprietary or Free Software/Open Source) software package. *If so, please specify which?*

b. A significant amount of effort is spent defining what the software functionality and behavior (the requirements) should be.

d. There have been meetings or discussions with end-users to define significant parts of the software functionality or behavior.

e. These meetings/discussions occur frequently, and can be considered an important part of defining the project's software.

Comment:
XFree86 develops and implements many standards new standards are developed in cooperation with various desktop projects (Gnome KDE FLTK etc). Rearchitecture implementing existing standards is largely done without significant involvement beyond the XFree86 community.

3.2: Usability Multiple choices possible

f. This project doesn't have any significant user interface.

3.3: Documentation Multiple choices possible

a. We have produced documents that describe at least some of the expected functionality and behavior (requirements) of our software.

b. The project produces and maintains documents for developers that describe how the code is organized (architecture), and/or how parts of it work.

d. There is documentation for the end-user available for the project's software (consider also third-party documents available).

e. End-user documentation is provided to a large extent by people or groups external to the project team.

Comment:
End user documentation for XFree86 is a cottage industry all it's own. The XFree86 team doesn't significantly interact with publishers or authors in this area.

Large parts of XFree86 are undocumented or have only ancient documentation which isn't entirely accurate anymore. Code inherited from the X consortium is occasionally modified without updating the associated documetation.

Some of the XFree86 additions developed some time ago aren't well documented either many of those are finally now gaining appropriate documentation .

3.4: Quality Assurance Multiple choices possible

a. There is an [automated] test suite for the project's software, that is used to validate it.

d. There is an active code review process, where code is read by other members of the team.

g. There is a tendency (or policy) to release a public version only when it has been *extensively* tested by the team.

Comment:
The XFree86 CVS repository is writable by a small group of people. Those people have essentially no review of their changes unless they explicitly requiest it. People not on this list submit patches which are carefully reviewed before being incorporated into the source pool. A more proactive review process for those with direct commit access would probably improve code quality at the expense of significant time which no one is willing to spend at the current time. Even still XFree86 is almost always usable directly from CVS which is as much a testament to strong interfaces as careful source code mangement.

3.5: Tools Multiple choices possible

a. A project hosting site such as Sourceforge.net, Savannah or Collab.net (mark all others applicable as well)

b. One or more Web sites

d. A Frequently Asked Questions document

e. A version control tool such as CVS, RCS, Subversion or Bitkeeper

g. One or more mailing lists

j. IRC

Comment:
The XFree86 IRC channels (OPN #xfree86) is used by a portion of the community but not by most of the core team. I think it would help bring new contributors up to speed more rapidly but it is a significant time sink. Many people have also requested that XFree86 adopt a bug tracking tool but the overhead for developers required to maintain such a system is higher than people are willing to endure.

Scores:

- Software Process: 43
- Tools: 55
- Software Engineering Tools: 52
- Redcarpet: 65
- Institutional Support: 0
- Requirements Effort: 85
- Usability Effort: 0
- Documentation Effort: 41
- Quality Assurance Effort: 46

Browse other surveys

christian reis <kiko@async.com.br>