
Técnicas de Visualização para Avaliação e Melhoria de Qualidade de Software Livre e Aberto

Rafael Messias Martins

SERVIÇO DE PÓS-GRADUAÇÃO DO ICMC-USP

Data de Depósito: 11/10/2012

Assinatura: _____

Técnicas de Visualização para Avaliação e Melhoria de Qualidade de Software Livre e Aberto

Rafael Messias Martins

Orientador: *Prof. Dr. José Carlos Maldonado*
Coorientadora: *Profa. Dra. Rosane Minghim*

Monografia apresentada ao Instituto de Ciências
Matemáticas e de Computação - ICMC-USP, para
o Exame de Qualificação, como parte dos
requisitos para obtenção do título de Doutor em
Ciências - Ciências de Computação e Matemática
Computacional.

USP – São Carlos
Outubro de 2012

Resumo

O desenvolvimento de software livre e aberto (*Free and Open Source Software – FOSS*) surgiu como um movimento de desenvolvedores voluntários mas, devido ao seu sucesso, vem sendo pesquisado e adotado por grandes organizações de software do mundo todo. Apesar disso, o conceito de qualidade de FOSS e os fatores que levam ao seu sucesso ainda são tópicos recentes na Engenharia de Software, o que causa relutância na adoção dessa nova abordagem e dificulta a seleção de produtos a serem utilizados e integrados. Tal situação motivou o estabelecimento de processos objetivos de compreensão e avaliação de qualidade para, entre outras finalidades, a classificação e comparação de projetos FOSS. Uma forma de obter essa compreensão é a extração e análise de métricas de qualidade – de código, de teste ou de repositório, por exemplo – como foi realizado no projeto Qualipso (*Quality Platform for Open Source Software*), cujo objetivo foi alavancar as práticas atuais de desenvolvimento aberto a níveis de operações industriais reconhecidas e confiáveis, sem perder o frescor e o entusiasmo tradicional da comunidade FOSS. O processo de extração e análise, no entanto, tende a gerar grandes quantidades de dados, difíceis de analisar sem ferramentas apropriadas. Neste trabalho é proposta a aplicação de técnicas de visualização para analisar dados provenientes de características de qualidade de projetos FOSS, com o objetivo de estabelecer um processo e uma plataforma colaborativa para apoiar as tarefas de avaliação, monitoramento e melhoria de qualidade. Como resultado espera-se contribuir com a compreensão geral dos fatores que levam à qualidade e ao sucesso de projetos FOSS e oferecer mecanismos para avaliação objetiva da qualidade dos projetos disponíveis, incentivando ainda mais a adoção de FOSS pela indústria e auxiliando comunidades e organizações no gerenciamento e evolução colaborativa da qualidade de seus esforços.

Abstract

The development of Free and Open Source Software (FOSS) has emerged as a movement of volunteer developers, but due to its success, has been researched and adopted by large software organizations worldwide. Nevertheless, the concept of FOSS quality and the factors that lead to its success are still hot topics in Software Engineering, which causes reluctance in the adoption of this new approach and complicates the selection of products to be integrated and used. This situation motivated the establishment of objective processes for understanding and assessing FOSS quality for, among other purposes, classification and comparison of projects. One way of obtaining this understanding is the extraction and analysis of quality metrics – from code, test or repository, for example – as was done in the Qualipso (*Quality Platform for Open Source Software*) project, which aimed to leverage the current practices of FOSS development to well recognized and reliable industrial levels, without losing the freshness and excitement that are traditional to the FOSS community. The procedure for extraction and analysis, however, tends to generate large amounts of data which are difficult to analyze without proper tools. This work proposes the application of visualization techniques to analyze data from quality characteristics of FOSS projects, aiming to establish a process and a collaborative platform to support the tasks of assessing, monitoring and improving quality. As a result we expect to contribute to the general understanding of the factors leading to quality and success of FOSS projects and provide mechanisms for an objective quality evaluation of available projects, further encouraging the adoption of FOSS by industry and helping communities and organizations in managing and evolving the quality of their collaborative efforts.

Conteúdo

1	Introdução	1
1.1	Justificativa e Motivação	2
1.2	Objetivos	4
1.3	Contexto	5
1.4	Organização	6
2	Qualidade de Software Livre e Aberto	7
2.1	Considerações Iniciais	7
2.2	Modelos de Qualidade e Maturidade	10
2.3	Testes	17
2.4	Considerações Finais	23
3	Visualização de Software	25
3.1	Considerações Iniciais	25
3.2	Conceitos Básicos, Técnicas e Ferramentas	28
3.3	Aspectos Sociais e Múltiplas Perspectivas	32
3.4	Considerações Finais	36
4	Proposta de Trabalho	39
4.1	Considerações Iniciais	39
4.2	Motivação	40
4.3	Detalhes da Proposta e Objetivos	42
4.4	Metodologia	44
4.5	Ferramentas e Dados	46
4.6	Atividades e Cronograma da Pesquisa	47

Lista de Figuras

2.1	Publicações experimentais em FLOSS (Crowston et al., 2012)	9
2.2	Norma ISO/IEC 9126 – Adaptado de (ISO/IEC, 2001)	11
2.3	Modelos de avaliação de qualidade FOSS – 1 ^a geração	13
2.4	Modelos de avaliação de qualidade FOSS – 2 ^a geração	14
2.5	Níveis de práticas do OMM	16
2.6	Estrutura de comunidades para evolução do OMM (Malheiros, 2010) . .	17
2.7	Exemplo de questões do <i>checklist</i> do OSS-TMM (Morasca et al., 2011) .	22
3.1	Exemplos de visualização de dados multidimensionais	27
3.2	Exemplos de visualização de estrutura de software	28
3.3	Exemplos da técnica HEB (Holten, 2006)	29
3.4	Exemplo da ferramenta SolidSX (Reniers et al., 2011)	30
3.5	Exemplo da ferramenta Tarantula (Jones et al., 2002)	31
3.6	Exemplos de visualização de evolução de software	32
3.7	Visualização da rede de comunicação do projeto Apache (Ogawa et al., 2007)	33
3.8	Análise sócio-técnica – CodeSaw (Gilbert e Karahalios, 2007)	34
3.9	Exemplo da ferramenta StarGate (Ogawa e Ma, 2008)	35
3.10	Exemplos de visualização sócio-técnica de software	36
3.11	Exemplos de visualização de software com animação	37
4.1	Metodologia experimental proposta por Mafra et al. (2006)	46

Lista de Tabelas

2.1 Cobertura de testes do projeto LTP (Modak e Singh, 2008)	20
2.2 Análise de cobertura de testes de projetos FOSS (Rincon et al., 2011) . . .	20
4.1 Cronograma de atividades	48

Introdução

Nas últimas décadas, a remoção progressiva de barreiras físicas na comunicação mundial tem levado a um sistema de produção com características diferentes das encontradas na economia industrial que estruturou o pensamento da sociedade durante o último século. O crescimento de esforços cooperativos em grande escala tem se espalhado por todos os domínios da produção de cultura e informação, desde a construção colaborativa de enciclopédias, até notícias, comentários e entretenimento. Um exemplo importante desse novo sistema de produção está no surgimento das comunidades de desenvolvimento de software livre e aberto (*Free and Open Source Software* – FOSS)¹ e seu sucesso econômico e social (Benkler, 2006).

Tal sucesso não passou despercebido no cenário tecnológico mundial. Apesar de ter surgido como um movimento de desenvolvedores e usuários voluntários, o modelo FOSS vem sendo pesquisado e adotado por grandes companhias da indústria de software do mundo todo (Riehle, 2007; Wasserman e Capra, 2007). Dados de pesquisas mostram que o uso de FOSS em empresas está presente em diversas categorias como *back-end*, *middleware*, ferramentas de escritório e aplicações de negócios e que a maioria dos usuários consideram que produtos FOSS alcançaram ou excederam suas expectativas de qualidade (Forrester Consulting, 2008).

¹Também é comum a utilização do termo FLOSS – *Free/Libre/Open Source Software* – para evitar a ambiguidade da palavra *free*, que pode significar “livre” ou “grátis”.

1.1. JUSTIFICATIVA E MOTIVAÇÃO

Apesar das histórias de sucesso de desenvolvimento, uso e integração de FOSS e seu reconhecimento internacional, muitas organizações ainda relutam em adotar esse novo modelo devido a problemas legais ou jurídicos, aspectos comerciais ou falta de confiança nos processos, produtos e suporte do desenvolvimento aberto. Além disso, podem ser encontradas diversas alternativas FOSS disponíveis para a realização de tarefas similares, tornando difícil a seleção por parte de potenciais integradores de software. Por outro lado, sob o ponto de vista do desenvolvedor, muitas vezes não são claros os fatores de qualidade do projeto e seu diferencial em relação às outras alternativas, o que dificulta a divulgação externa e o direcionamento interno do trabalho rumo a um melhor produto final (Groven et al., 2010; Izquierdo-Cortazar et al., 2010).

1.1 Justificativa e Motivação

Tal situação evidencia a necessidade de um processo de avaliação de qualidade que defina critérios objetivos para a classificação e comparação de projetos FOSS. Muitas vezes essa avaliação é feita informalmente, com a leitura da documentação e análise de opiniões de usuários anteriores, o que nem sempre gera resultados confiáveis. Porém, é interessante observar que, se tratando de projetos FOSS, muitos dados estão disponíveis publicamente tanto sobre o processo de desenvolvimento quanto sobre o produto. Exemplos destes dados são: código-fonte, históricos de evolução do repositório de código, conjuntos de teste, relatórios de erros e suas soluções, trocas de mensagens entre membros do projeto, entre outros. Todas essas informações, se corretamente processadas, podem ser utilizadas para avaliar a qualidade do projeto (Deprez et al., 2008; Izquierdo-Cortazar et al., 2010; Samoladas et al., 2008).

Pode ser constatado que muitas medidas relacionadas às mais diversas características de projetos de software têm sido propostas desde cedo na pesquisa em Engenharia de Software (Mills, 1988). Métricas de código fonte, por exemplo, oferecem a oportunidade de analisar flexibilidade, complexidade e manutenibilidade do código, fatores importantes para a evolução bem sucedida de um produto FOSS (Sato et al., 2007). Por outro lado, medidas de teste estrutural, obtidas a partir da execução monitorada do software com um conjunto específico de entradas, refletem tanto a qualidade dos testes quanto a confiabilidade do software (Lemos et al., 2007), informações cruciais para a avaliação de um projeto. No contexto FOSS são importantes também medidas obtidas por análise de repositório (Voinea e Telea, 2009) e de *bugtrackers* (D'Ambros et al., 2007), levando em consideração o fato de que a interação e participação ativa de comunidades de desenvolvedores e usuários são cruciais para o sucesso de um projeto (Meirelles et al., 2010).

Métricas podem ser usadas para medir características e atributos definidos por modelos de qualidade de software voltados para a análise tanto do processo quanto do produto. Alguns modelos propostos especificamente para FOSS (Deprez e Alexandre, 2008; Petrinja et al., 2009; Spinellis et al., 2009) se baseiam em modelos tradicionais como o CMMI (*Capability Maturity Model Integration*) (Chrissis et al., 2006) e a norma ISO 9126 (ISO/IEC, 2001) mas incluem extensões que consideram especificamente aspectos importantes do desenvolvimento aberto. No entanto, a multiplicidade de indicadores de qualidade que podem ser extraídos de um projeto FOSS, considerando todas as suas diversas perspectivas (principalmente código, testes e repositório), cada um com sua própria interpretação e valores de referência, torna complicada a realização de uma avaliação objetiva e direta. Plataformas de avaliação de qualidade baseadas em conjuntos de ferramentas, como é o caso das plataformas Qualipso, FLOSSMetrics ou Alitheia Core², expõem ao avaliador de qualidade dezenas de valores numéricos relacionados às métricas calculadas. Sem uma apresentação especial esses dados são de pouco valor, pois são numerosos e difíceis de avaliar. Muitas vezes as métricas não são correlacionadas e a formatação da apresentação consiste apenas na aplicação de valores de referência.

Uma possibilidade para a solução desse problema é o uso de visualização de informação – representações visuais e interativas de dados, apoiadas por computador, utilizadas para ampliar a aquisição de conhecimento e apoiar descobertas, tomadas de decisão e explicações a partir de dados complexos (Card et al., 1999). A pesquisa em visualização de indicadores de qualidade de software é um tópico que exige atenção especial devido à grande quantidade de informação disponível e a falta de soluções de visão geral, o que leva o usuário a perder a imagem do todo e dificulta a análise dos dados (Shollo e Pandazo, 2008). Quando se considera a análise da evolução do projeto no tempo o problema é ainda maior, já que diversas versões diferentes são analisadas em paralelo. O uso de visualização de software pode reduzir essa complexidade, pois uma boa apresentação visual permite ao avaliador estudar múltiplos aspectos de um problema complexo ao mesmo tempo (Lanza e Ducasse., 2002).

Atualmente, embora em alguns casos os indicadores sejam apresentados visualmente, as metáforas visuais são simples e os avaliadores são envolvidos por uma quantidade esmagadora de informação e perdem a imagem do todo (Burkhard et al., 2005). Por isso, processos de avaliação devem focar em apresentar os dados coletados de forma a comunicar a visão geral ao invés de apresentar dados brutos em tabelas decorativas. No entanto, a tarefa de selecionar uma técnica de visualização mais relevante para um objetivo ou

²<http://qualipso.org/>; <http://flossmetrics.org/>; <http://sqa-oss.org> (último acesso em 13/10/2012)

1.2. OBJETIVOS

aplicação particular não é trivial, já que nenhuma técnica específica funciona para todos os problemas (Thomas e Cook, 2005).

Além da utilização de visualização de informação para apoiar a avaliação de qualidade e a tarefa de compreender a grande quantidade de medidas obtidas a partir de um projeto, é importante também considerar o papel da comunidade – usuários, desenvolvedores e avaliadores – na qualidade de FOSS. O sucesso de projetos FOSS é alcançado devido às redes de colaboração voluntária de usuários e desenvolvedores que relatam falhas, corrigem problemas e adicionam funções (Meirelles et al., 2010); portanto, um método efetivo de avaliação de qualidade de FOSS deve não só considerar os fatores relacionados à comunidade no projeto como também envolver a participação ativa da mesma na manutenção e atualização dos dados avaliados, evolução da plataforma de avaliação e discussão e validação dos resultados obtidos.

1.2 Objetivos

Tendo em vista o que foi apresentado na seção anterior, os principais objetivos deste trabalho são:

1. Estudar a aplicação de técnicas de visualização em dados provenientes de medidas de FOSS relacionadas a fatores como código, testes e comunidade, visando obter uma melhor compreensão dos indicadores de qualidade e buscando evidências que mostrem as vantagens e a viabilidade de tal aplicação.
2. Estabelecer um processo de avaliação de qualidade com base nas técnicas de visualização estudadas, tornando mais eficientes e eficazes as atividades de desenvolvimento, gerenciamento e integração de projetos FOSS.
3. Implementar uma plataforma colaborativa de avaliação e melhoria de qualidade FOSS que dê suporte completo ao processo estabelecido, incentivando a participação da comunidade em sua validação, utilização e evolução contínua.

A seguir são relacionadas algumas questões de pesquisa que deverão guiar o trabalho rumo aos objetivos definidos:

1. Os modelos, características e medidas disponíveis são adequados para o processo de avaliação de qualidade FOSS? Se não, é possível melhorá-los?
2. Quais técnicas de visualização podem ser utilizadas para compreender melhor os dados obtidos a partir de medidas de qualidade FOSS?

3. A utilização dessas técnicas torna o processo de avaliação de qualidade de FOSS mais eficaz e/ou eficiente? Se sim, quais são os passos necessários?
4. É possível melhorar essas técnicas para que gerem melhores resultados quando aplicadas especificamente no contexto da avaliação de qualidade FOSS? Se sim, como?
5. Como a comunidade FOSS pode interagir com a análise visual em um processo colaborativo de avaliação e melhoria de qualidade?

1.3 Contexto

Um exemplo de iniciativa recente de pesquisa motivada pelo interesse da indústria de software pelo desenvolvimento FOSS foi o projeto Qualipso – *Quality Platform for Open Source Software*, que consistiu em uma aliança global entre membros fundadores na Europa, América do Sul e Ásia, envolvendo participantes da indústria como *Siemens*, *Telefónica* e *Atos Origin*, institutos de pesquisa como *Bull*, *INRIA* e *Fraunhofer*, e universidades como a *Universidade de São Paulo*, a *Universidade Rey Juan Carlos* (Espanha) e a *Universidade de Insubria* (Itália).

O trabalho proposto nesta monografia tem como uma de suas bases as experiências obtidas durante o projeto Qualipso, mantidas vivas e em constante evolução pelos Centros de Competência em Software Livre da USP (CCSL/USP): CCSL/IME e CCSL/ICMC³. Além disso, o trabalho está sendo desenvolvido no contexto do NAPSoL (*Núcleo de Apoio à Pesquisa em Software Livre*), uma parceria entre pesquisadores do ICMC, IME e EAC-H/USP com o objetivo de apoiar a realização de pesquisa científica e tecnológica e a criação de processos, métodos e ferramentas para a produção de FOSS de alta qualidade. O núcleo pretende não só apoiar a pesquisa em computação, mas também outras áreas do conhecimento e outros grupos de pesquisa relacionados à produção de FOSS, facilitando a transferência de tecnologia livre da universidade para a indústria.

Outro exemplo de projeto internacional de fomento à pesquisa em FOSS também co-financiado pela Comissão Européia foi o SHARE, com o objetivo de facilitar a utilização e o compartilhamento de software de código aberto dentro do domínio de sistemas embarcados críticos, criando um ambiente para o desenvolvimento de aplicações e soluções em *middleware* de código aberto e abrindo caminho para novos modelos de negócios e serviços. No Brasil, um projeto importante é o INCT-SEC⁴ – *Instituto Nacional de Ciência e Tecnologia em Sistemas Embarcados Críticos* – uma criação do Ministério de Ciência

³<http://ccsl.usp.br>; <http://ccsl.ime.usp.br>; <http://ccsl.icmc.usp.br> (último acesso em 13/10/2012)

⁴<http://share-project.eu/>; <http://inct-sec.org/> (último acesso em 13/10/2012)

1.4. ORGANIZAÇÃO

e Tecnologia em parceria com instituições de fomento (CNPq e FAPESP) que constitui uma rede de pesquisa para trabalhos na área de Sistemas Embarcados Críticos, com ênfase em veículos autônomos móveis aéreos e terrestres. Uma das missões do INCT-SEC é o desenvolvimento científico e tecnológico em sistemas embarcados críticos através da integração da academia com a indústria. A avaliação da qualidade dos produtos FOSS desenvolvidos no projeto pode ser essencial para garantir a eficiência desta transferência tecnológica.

1.4 Organização

Os próximos capítulos estão organizados da seguinte forma: no Capítulo 2 são apresentados conceitos e trabalhos relacionados à pesquisa em qualidade FOSS, incluindo modelos de qualidade de processo e produto, que auxiliam na definição, avaliação e predição de atributos de qualidade, e pesquisas sobre o estado atual da atividade de testes no contexto FOSS; no Capítulo 3 é apresentada uma visão geral da área de *Visualização de Software* – o uso de representações visuais de aspectos do software para auxiliar sua compreensão e desenvolvimento – com ênfase em trabalhos que envolvem a análise de projetos FOSS; e no Capítulo 4 é apresentada em detalhes a proposta do trabalho, incluindo sua metodologia, ferramentas de apoio e um cronograma previsto para as atividades.

Qualidade de Software Livre e Aberto

2.1 Considerações Iniciais

Existem hoje milhares de projetos FOSS ativos em diversos segmentos, sendo utilizados na academia e na indústria, englobando áreas como computação científica, entretenimento, eletrônicos de consumo e até sistemas embarcados críticos. Exemplos comuns – devido ao tamanho, sucesso e influência – são o sistema operacional **Linux**, o servidor web **Apache**, o navegador **Mozilla Firefox** e as aplicações de escritório **OpenOffice** e **LibreOffice**. Entre desenvolvedores, ferramentas como o compilador **GCC**, as linguagens de programação **PHP**, **Perl** e **Python** e o banco de dados **MySQL** são importantes para tarefas do dia-a-dia. Até mesmo a infraestrutura da internet é dependente de programas FOSS como **sendmail** e **bind** (Crowston et al., 2012; Midha e Palvia, 2012).

O modelo de desenvolvimento FOSS possibilita a construção de software de qualidade altamente adaptável, com baixo custo e alto reuso. No entanto, o impacto do desenvolvimento aberto não é só técnico, mas também social e econômico. O apoio de organizações governamentais na adoção de soluções abertas oferece mais transparência e acessibilidade a recursos tecnológicos, serviços e dados à população em geral. Estudantes e desenvolvedores iniciantes têm acesso completo a bases de código-fonte de projetos importantes, possibilitando o aprendizado prático com tecnologias efetivamente aplicadas (Kon et al., 2011).

2.1. CONSIDERAÇÕES INICIAIS

De acordo com a *Open Source Initiative* (OSI)¹, software aberto deve ser distribuído sob uma licença que garanta direitos ao usuário (não comuns em licenças tradicionais) como permissão de acesso ao código fonte e de criação e redistribuição de modificações e trabalhos derivados, além de garantir a não inclusão de discriminação na redistribuição e de restrições a tecnologias específicas ou a outros softwares distribuídos em conjunto. No entanto, é importante entender algumas diferenças conceituais importantes entre os termos *software aberto* e *software livre*. Stallman (2009) observa que, embora os dois termos descrevam quase a mesma categoria de software e quase todos os softwares *abertos* sejam *livres*, *software aberto* é um conceito puramente prático que se refere à metodologia de desenvolvimento cujo objetivo é desenvolver software melhor, mais poderoso e mais confiável, enquanto o conceito de *software livre* se refere ao movimento social de defesa dos direitos do usuário. Nesta monografia o termo FOSS engloba projetos dos dois grupos.

O desenvolvimento FOSS tem chamado a atenção da academia por oferecer aos pesquisadores da Engenharia de Software a oportunidade de validarem seus trabalhos com dados reais, abundantes e disponíveis publicamente, tornando-o uma ferramenta importante para a construção de trabalhos científicos extensíveis e reproduutíveis. Como exemplo, na Figura 2.1 é mostrada a evolução da quantidade de publicações experimentais em FOSS até 2005 de acordo com o levantamento de Crowston et al. (2012). Além disso, a compreensão do modelo propriamente dito também se tornou um tópico importante de pesquisa. Diversos exemplos de produtos FOSS complexos e bem sucedidos foram desenvolvidos com metodologias e processos muitas vezes bem diferentes do desenvolvimento “convencional” e a compreensão dessas diferenças (e também das similaridades) pode ajudar a melhorar a prática da Engenharia de Software no geral. Pesquisas neste contexto investigam principalmente as práticas de desenvolvimento e os processos sociais de projetos FOSS, com um recente foco no crescimento das relações entre organizações e projetos voluntários e como os dois lados podem se beneficiar com essa interação (Crowston et al., 2012; Kon et al., 2011).

Independentemente do processo de desenvolvimento utilizado, sistemas de software são produtos complexos, difíceis de construir e testar; muitas vezes podem se comportar de forma inesperada e indesejada, até mesmo causando problemas ou danos graves. A qualidade de um sistema é um fator determinante de seu sucesso (ou falha) no mercado, o que torna o estudo da qualidade um dos tópicos mais importantes da Engenharia de Software (Fuggetta, 2000; da Rocha et al., 2001; Tian, 2004).

A crescente adoção de FOSS por usuários e organizações em produtos, sistemas e infraestruturas de software tem levantado a questão da avaliação de qualidade neste contexto. Projetos FOSS oferecem funcionalidades valiosas sem custo de licença e com independê-

¹<http://opensource.org/> (último acesso em 14/10/2012)

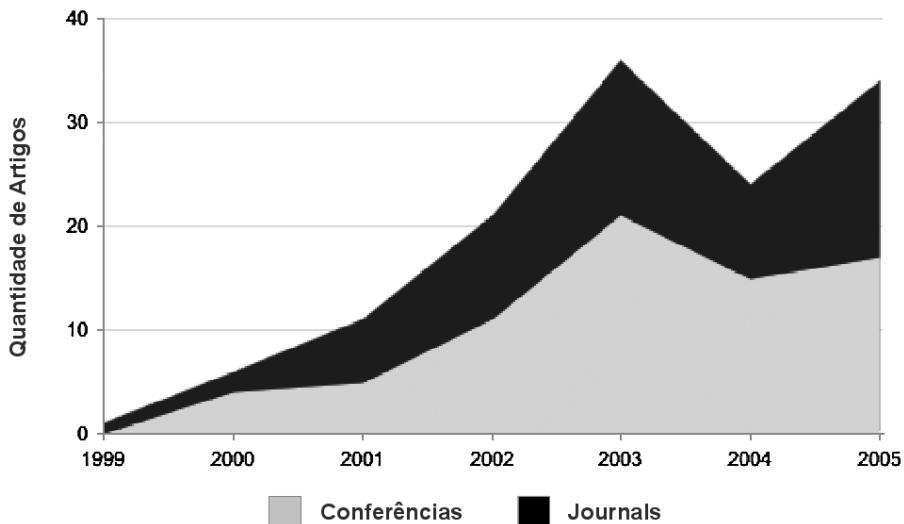


Figura 2.1: Publicações experimentais em FLOSS até o final de 2005 (Crowston et al., 2012)

cia do fornecedor; sua utilização em uma organização ou sua incorporação em sistemas de grande escala, no entanto, não são grátis e acompanham de riscos e incertezas (Soto e Ciolkowski, 2009).

O aumento do impacto econômico do modelo FOSS, sua crescente utilização em aplicações críticas e a grande disponibilidade de projetos FOSS de sucesso nas mais diversas áreas de atuação tornam necessário o desenvolvimento de abordagens de avaliação de qualidade e seleção que diferenciem projetos candidatos de acordo com critérios técnicos, funcionais e estratégicos, dos pontos de vista do processo e do produto. Por um lado, características técnicas como complexidade do código, performance e segurança mostram a qualidade imediata do produto; por outro lado, também devem ser consideradas as chances de um projeto FOSS continuar sendo mantido e suportado no futuro, características inerentes à comunidade de desenvolvimento. A disponibilidade de artefatos como código fonte, dados do sistema de controle de versão, bancos de dados de relatórios de *bugs* e listas de discussão permitem que a avaliação de qualidade e seleção sejam realizadas de forma transparente (Soto e Ciolkowski, 2009; Spinellis et al., 2009).

Existe um interesse substancial em adotar e aprender com o desenvolvimento FOSS para fomentar e repetir seus sucessos, mas para isso é necessário compreender melhor esse fenômeno e saber determinar o que leva ao seu sucesso (Crowston et al., 2006; Samoladas et al., 2008). Neste capítulo são apresentados conceitos e trabalhos de pesquisa relacionados à qualidade de software livre, organizados da seguinte forma: na Seção 2.2 são apresentados modelos que definem características de qualidade e maturidade e auxiliam na avaliação e predição de qualidade de processos e produtos FOSS; na Seção 2.3 são descritos alguns trabalhos que apresentam o estado atual da atividade de testes em FOSS; e na Seção 2.4 são apresentadas as considerações finais sobre o capítulo.

2.2 Modelos de Qualidade e Maturidade

De forma geral, o conceito de *qualidade de software* se refere a um conjunto de características que devem ser alcançadas para satisfazer requisitos explícitos e implícitos dos diferentes interessados no software (ISO/IEC, 2001; Pressman, 2004). As características de qualidade dependem do contexto e domínio do projeto e das tecnologias utilizadas no seu desenvolvimento, e podem ser descritas em diferentes níveis de detalhamento. Por esse motivo é comum a utilização de modelos que organizam atributos de qualidade e permitem uma avaliação quantitativa e objetiva, ao longo do tempo, auxiliando na predição de qualidade e na identificação de áreas problemáticas específicas dentro de um projeto (da Rocha et al., 2001; Tian, 2004).

Muitos modelos de qualidade de software voltados à análise do produto foram propostos desde cedo na Engenharia de Software. Exemplos pioneiros e influentes são os trabalhos de Boehm et al. (1976) e McCall et al. (1977), desenvolvidos com o objetivo de fornecer a gerentes de aquisição de software um mecanismo para especificação e medição quantitativa do nível de qualidade desejado dos produtos de software a serem adquiridos por organizações. Outros exemplos são: o modelo de Dromey (1995), o modelo FURPS (*Functionality, Usability, Reliability, Performance, Suitability*) (Grady e Caswell, 1987) e a norma ISO 9126 (ISO/IEC, 2001), que surgiu como tentativa de consolidar as diferentes visões de qualidade estudadas nos modelos anteriores em um único modelo e padrão internacional².

Como exemplo de um modelo de qualidade tradicional, na Figura 2.2 é mostrada a estrutura do modelo de características internas e externas da norma ISO/IEC 9126. Cada uma das seis características (*Funcionalidade, Confiabilidade, Usabilidade, Eficiência, Manutenibilidade e Portabilidade*) se desdobra em subcaracterísticas (num total de 27) que, por sua vez, se desdobram em atributos de qualidade, medidos com a aplicação de métricas internas e externas. Utilizando um método de agregação, os valores das subcaracterísticas são combinados em um único valor composto que determina o nível de cada característica. A norma ISO/IEC 9126 ainda conta com um modelo de qualidade em uso, que visa caracterizar a capacidade do produto de software em permitir a usuários atingir metas de efetividade, produtividade, segurança e satisfação (Jung et al., 2004).

Um modelo de qualidade por si só não é o bastante para a realização da avaliação de um produto de software; é necessário também um processo que descreva a utilização do modelo

²Em 2011 a norma ISO/IEC 25010 foi lançada para atualizar a norma ISO/IEC 9126. Visto que a norma original vem sendo utilizada com sucesso durante muitos anos e ainda não foi substituída completamente, além de não serem encontrados trabalhos suficientes com dados experimentais sobre a efetividade das características introduzidas pelo novo modelo, a norma original foi mantida como referência neste trabalho.

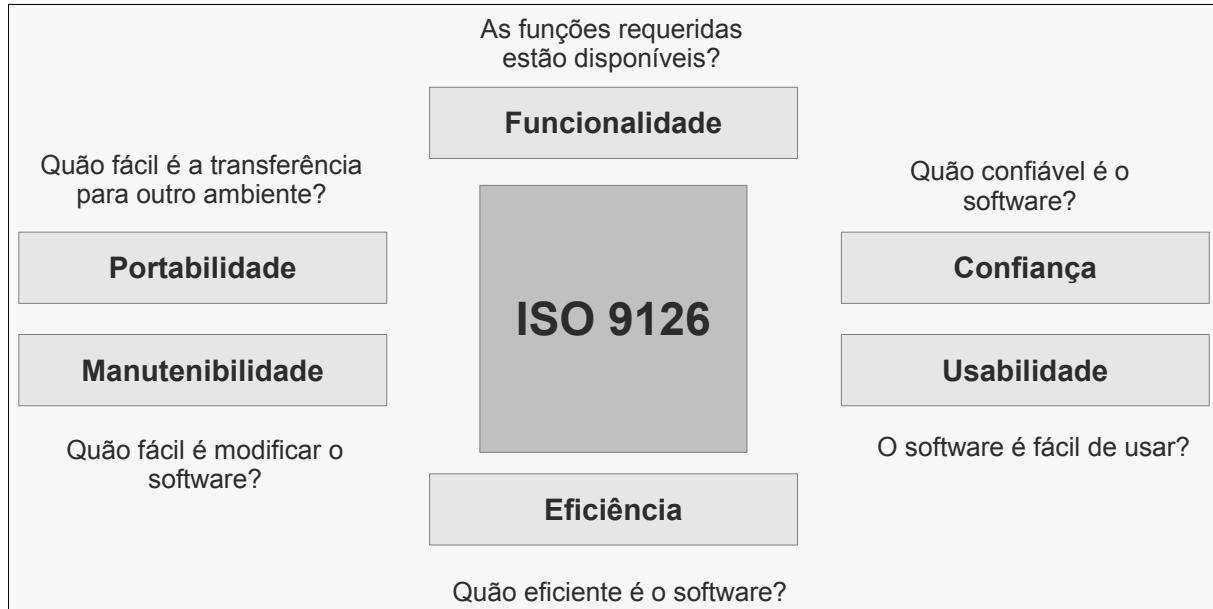


Figura 2.2: Norma ISO/IEC 9126 – Adaptado de (ISO/IEC, 2001)

e guie sua aplicação efetiva. A norma ISO/IEC 14598, projetada para ser utilizada em conjunto com o modelo apresentado, descreve todo o processo de planejamento, execução e documentação da avaliação de qualidade de produtos de software, considerando diversos pontos de vista como desenvolvedores, adquirentes e avaliadores externos (da Rocha et al., 2001).

Na última década, pesquisadores têm focado na discussão de como extensões de modelos como a norma ISO/IEC 9126 devem ser aplicadas em projetos FOSS. Algumas características comuns no contexto FOSS – como o livre acesso ao código fonte, repositórios de artefatos compartilhados, revisão de código, desenvolvimento global assíncrono e a falta de processos formais – não são consideradas em modelos tradicionais, o que pode indicar que os mesmos não são suficientes (Samoladas et al., 2008). Em particular, modelos tradicionais não consideram a importância da estrutura e do comportamento da comunidade de desenvolvimento na qualidade final do produto. A mistura de desenvolvedores voluntários e contratados, trabalhando com diferentes motivações, objetivos e intensidades, além da distribuição entre desenvolvedores do núcleo, que tomam a maioria das decisões e contribuem com o maior esforço, e desenvolvedores periféricos ou mesmo casuais, que participam pouco do desenvolvimento mas auxiliam na detecção de *bugs*, podem causar cenários bem diferentes dos previstos em modelos tradicionais (Groven et al., 2010; Izquierdo-Cortazar et al., 2010).

Diversos modelos de qualidade foram propostos tendo em vista características específicas do desenvolvimento FOSS. A primeira geração desses modelos se baseou nos modelos tradicionais, incluindo pequenas adaptações e extensões para considerar fatores relativos

2.2. MODELOS DE QUALIDADE E MATURIDADE

às comunidades. Dentre os modelos da primeira geração, exemplos importantes são o OpenBRR – *Business Readiness Rating for Open Source* (OpenBRR, 2005) e o QSOS – *Qualification and Selection of Open Source Software* (Atos Origin, 2006), descritos a seguir.

O modelo OpenBRR, mostrado na Figura 2.3 (a), apresenta uma estrutura hierárquica de categorias (como *Funcionalidade*, *Usabilidade* e *Suporte*) e métricas de qualidade similar à norma ISO/IEC 9126; os valores obtidos são ajustados com pesos de acordo com o domínio de aplicação e o tipo de projeto e agregados em uma pontuação final que varia 1 a 5. Foi um modelo pioneiro por incluir fatores da comunidade no processo de avaliação, mas algumas críticas incluem o fato de considerar apenas características de alto nível e do ponto de vista do usuário, além de ser uma avaliação altamente subjetiva (sem um processo ou método de avaliação bem definido) e gerar um indicador final que dificilmente representa todo o conjunto complexo de qualidades de um produto de software (Samoladas et al., 2008; Taibi et al., 2007).

Apesar de também definir categorias e medidas de qualidade, o objetivo principal do QSOS foi a criação um método para qualificar, selecionar e comparar FOSS de forma clara e objetiva, utilizando o processo iterativo mostrado na Figura 2.3 (b). Os quatro passos do método são: (i) *Definição*, onde são definidos e refinados conceitos de referência sobre famílias de projetos, tipos de licenças e de comunidades; (ii) *Avaliação*, onde é construído o cartão de identidade do projeto e preenchida sua planilha de avaliação a partir da análise das medidas do modelo; (iii) *Qualificação*, onde as informações anteriores são ajustadas de acordo com o contexto e necessidades do usuário; e (iv) *Seleção*, onde projetos da mesma família são comparados e selecionados de acordo com as necessidades do usuário. Em cada iteração o método é realimentado e evoluído pela própria comunidade, melhorando a qualidade de futuras avaliações. Exemplos de medidas relacionadas à comunidade (aplicadas no passo ii) são: *número de desenvolvedores do núcleo*, *taxa de resposta a relatórios de bugs* e *estilo de gerenciamento*.

As duas abordagens são simples, leves e auxiliam usuários a alcançar o objetivo de selecionar projetos FOSS a partir de uma lista de candidatos. No entanto, apenas validações parciais são encontradas na literatura, onde os métodos são aplicados em um pequeno número de estudos de caso (Deprez e Alexandre, 2008; Petrinja et al., 2010). Além disso, vale a pena notar que nenhum dos dois trabalhos apresenta uma metodologia específica ou apoio ferramental para a obtenção das métricas quantitativas e qualitativas que determinam os valores de atributos de qualidade. Tais ferramentas muitas vezes não existem ou não são compatíveis com os modelos e a obtenção das métricas é realizada manualmente, o que torna o processo demorado, informal e difícil de ser aplicado ou repetido (Izquierdo-Cortazar et al., 2010).

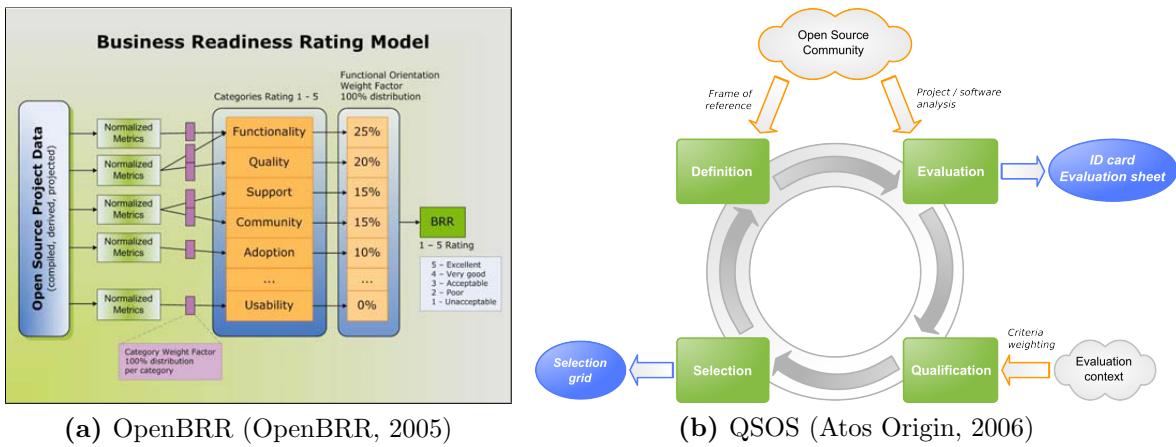


Figura 2.3: Modelos de avaliação de qualidade FOSS – 1^a geração

A segunda geração de modelos de avaliação de qualidade de produtos FOSS surgiu como uma tentativa de resolver esses problemas. Baseados nos sucessos e falhas da geração anterior, os novos modelos atacam principalmente dois problemas: (i) *a dificuldade de aplicação e reprodução dos resultados* dos modelos anteriores, apresentando um extenso apoio ferramental para a extração e cálculo de métricas; e (ii) a *limitação do escopo* dos modelos anteriores, estendendo os atributos de qualidade de dezenas para centenas de métricas possíveis (Groven et al., 2010). Os principais representantes da segunda geração vêm de projetos de pesquisa financiados pela Comissão Européia nos últimos anos, como o SQO-OSS – *Software Quality Observatory for Open Source Software* (Samoladas et al., 2008) e o QualOSS – *Quality in Open Source Software* (Deprez et al., 2008).

Os modelos SQO-OSS e QualOSS possuem objetivos similares: apoiar um processo de avaliação semi-automatizada de qualidade de projetos FOSS, orientado a métricas e com mínima intervenção humana na extração de dados e que possa ser aplicado regularmente e em uma grande variedade de projetos. Em ambos os modelos são considerados aspectos do produto (código fonte) e da comunidade (no SQO-OSS esses atributos são ajustados com pesos pelo avaliador, a partir de alguns perfis pré-definidos). De forma similar aos modelos anteriores, os valores de métricas são categorizados e agregados em atributos e sub-atributos de qualidade até ser obtida uma pontuação final de qualidade (Deprez et al., 2008; Izquierdo-Cortazar et al., 2010; Samoladas et al., 2008). Na Figura 2.4 são apresentadas as hierarquias de atributos dos dois modelos.

Processo

Os modelos apresentados até aqui consideram qualidade especificamente sob a perspectiva do produto; no entanto, a qualidade de produtos de software está diretamente relacionada também à qualidade do seu processo de desenvolvimento (Fuggetta, 2000). Abordagens

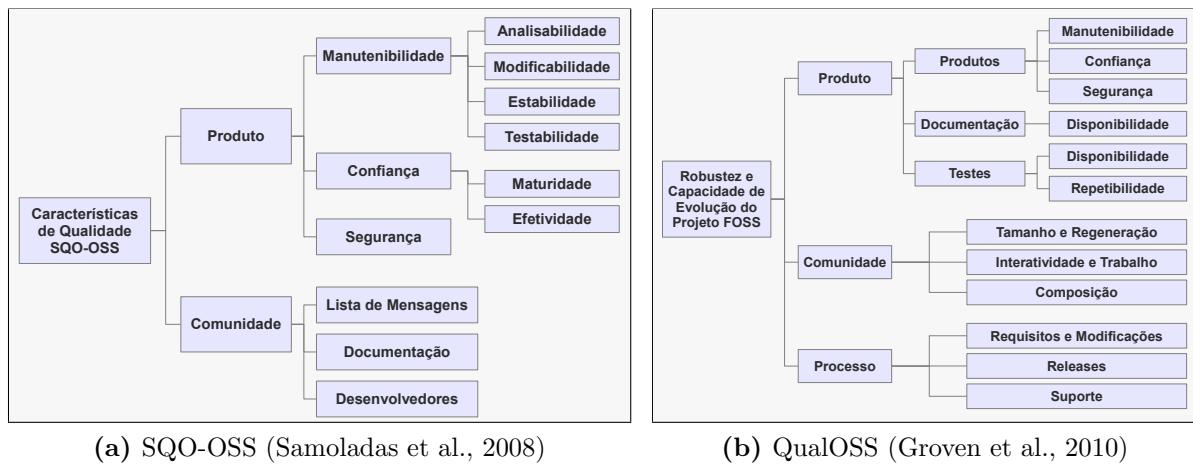


Figura 2.4: Modelos de avaliação de qualidade FOSS – 2^a geração

como as normas ISO/IEC 12207 (ISO/IEC, 2008) e ISO/IEC 15504 (também conhecida como SPICE – *Software Process Improvement and Capability Determination*) (ISO/IEC, 2012) e o modelo CMMI (*Capability Maturity Model Integration*) (Chrissis et al., 2006) definem modelos de avaliação de maturidade de processos e sugerem que melhorando o processo de desenvolvimento também é melhorada a qualidade do produto (da Rocha et al., 2001).

Alguns trabalhos também consideram importante a análise do processo de desenvolvimento na avaliação de qualidade de projetos FOSS. Petrinja et al. (2010) sugerem que os dois aspectos – processo e produto – devem ser levados em consideração em conjunto, visto que integradores podem estar interessados, por exemplo, na disponibilidade da documentação do processo e de quão fácil é integrar essa documentação num processo interno já existente.

As pesquisas em avaliação de qualidade de processos FOSS estendem o modelo CMMI para identificar, a partir do conjunto de objetivos desse modelo, um subconjunto relevante para o contexto FOSS. Um trabalho inicial nesse sentido foi o OSMM – *Open Source Maturity Model* (Duijnhouwer e Widdows, 2003), com o objetivo de facilitar a avaliação e adoção de FOSS com base no princípio de que a qualidade FOSS é proporcional à maturidade do seu processo de desenvolvimento. Sua metodologia define um conjunto de indicadores para avaliar a maturidade global de um produto FOSS com 12 características relacionadas ao produto e 15 características relacionadas às necessidades do usuários. Apesar de ser um modelo simples e de fácil aplicação, muitas características importantes do produto – como atributos do código fonte – e da comunidade não são consideradas na avaliação. Além disso, seu uso é limitado e sujeito a licenças privadas (Petrinja et al., 2009; Samoladas et al., 2008; Taibi et al., 2007).

Mais recentemente foi proposto o OMM – *OpenSource Maturity Model* (Wittmann et al., 2008) como um dos resultados do projeto QualiPSO – *Quality Platform for Open Source Software*, uma aliança global com 18 membros fundadores na Europa, América do Sul e Ásia, envolvendo participantes da indústria como *Siemens*, *Telefonica* e *Atos Origin*, institutos de pesquisa como *Bull*, *INRIA* e *Fraunhofer*, e universidades como a *Universidade de São Paulo*, a *Universidade Rey Juan Carlos* (Espanha) e a *Universidade de Insubria* (Itália). O OMM é um modelo de maturidade que identifica um conjunto de práticas a serem aplicadas para alcançar um desenvolvimento de software livre confiável. Seus objetivos são (i) prover para a comunidade de software livre a base para desenvolver produtos de maneira eficiente e (ii) prover aos integradores de software livre uma base para avaliar o processo utilizado por uma comunidade ou organização de software livre.

O OMM foi definido com base em pesquisas sobre o processo de desenvolvimento em comunidades FOSS e companhias ativas nesse contexto, além de artigos publicados sobre o assunto. A partir desse conhecimento foram identificados elementos de confiança (*Trustworthy Elements* – TWE), ou seja, características de qualidade de processo ou produto, agrupados em 3 níveis de maturidade: básico, intermediário e avançado. Cada nível é construído sobre o anterior, possibilitando uma melhoria de qualidade incremental. Na Figura 2.5 são mostrados os níveis do OMM e os TWEs que compõem cada nível, como por exemplo: *Documentação do Produto* (PDOC), *Popularidade do Produto* (REP), *Uso de Padrões Estabelecidos* (STD), *Disponibilidade de Roadmap* (RDMP), *Qualidade do Plano de Testes* (QTP), *Relacionamento entre Partes Interessadas* (STK), *Licenças* (LCS), *Ambiente Técnico* (ENV), *Número de Commits e Relatórios de Bugs* (DFCT), *Manutenibilidade e Estabilidade* (MST), *Contribuição de Empresas* (CONT) e *Resultados de Avaliações por Companhias Externas* (RASM).

Visto que projetos FOSS nem sempre se encaixam em um mesmo padrão, o modelo OMM não define uma forma específica para a implementação dos vários TWEs. No entanto, quando um projeto FOSS se inicia, seus desenvolvedores devem ao menos planejar detalhes sobre as atividades do nível básico do OMM, como: ferramentas a serem usadas no ambiente de desenvolvimento (plataforma, compiladores, bancos de dados, etc.), padrões a serem seguidos pelo time de desenvolvimento e de documentação, seleção de licenças e atividades relacionadas à verificação de qualidade (Petrinja et al., 2009).

Além do modelo propriamente dito, outro resultado importante do projeto Qualipso foi um conjunto de ferramentas recomendadas para apoiar a avaliação de qualidade baseada no OMM. No entanto, embora uma avaliação automatizada de qualidade seja um fator positivo devido à facilidade, rapidez de uso e possível escopo da avaliação, não é possível realizar uma avaliação completamente automatizada baseada no OMM. Alguns aspectos relacionados ao produto ou outros aspectos quantificáveis podem ser automatiza-

2.2. MODELOS DE QUALIDADE E MATURIDADE

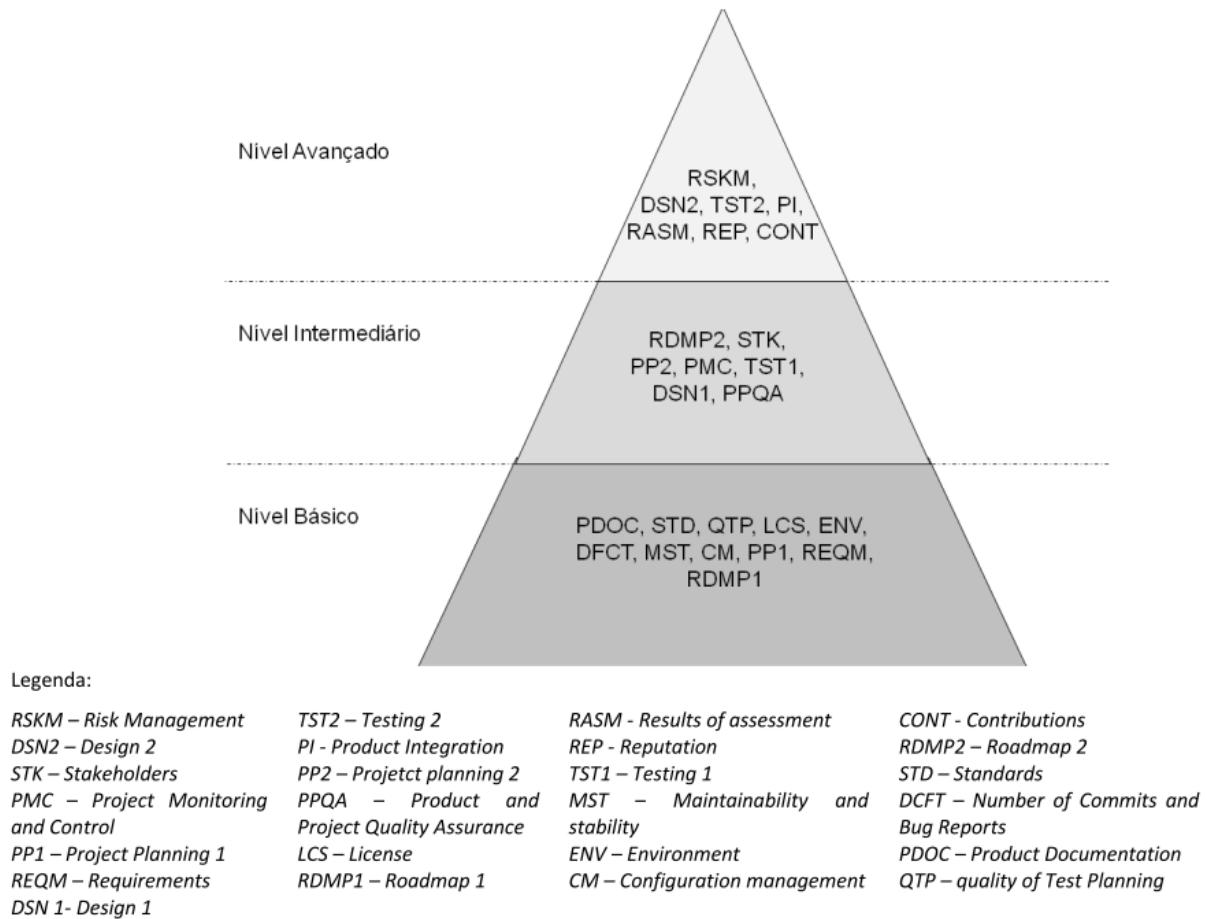


Figura 2.5: Níveis de práticas do OMM

dos, mas alguns elementos relacionados especificamente aos usuários necessitam entradas manuais durante o processo. Portanto a implementação do OMM envolve tanto medidas automáticas quanto o preenchimento de formulários (Petrinja et al., 2009).

Modelos de maturidade como o OMM indicam quais atividades devem fazer parte de um processo de desenvolvimento e quais áreas devem ser melhoradas para que o processo atinja um nível adequado de maturidade; no entanto, não oferecem guias de como essa melhoria pode ser alcançada. Para que a Melhoria de Processos de Software (MPS) ocorra efetivamente, é importante que seja definida uma estratégia para implementar com sucesso esses padrões e modelos. Nesse contexto foi proposta a ColabSPI (Malheiros, 2010), uma estratégia colaborativa e distribuída para a MPS com os seguintes objetivos: (a) melhorar a comunicação e colaboração entre as partes interessadas na MPS; (b) melhorar a participação dos desenvolvedores na MPS; e (c) apoiar a gestão de iniciativas de MPS.

Esse tipo de iniciativa pode ser útil também para comunidades FOSS pois, em essência, o desenvolvimento FOSS acontece com equipes geograficamente distribuídas. Como exemplo dessa possibilidade, a ColabSPI foi adaptada para apoiar a estratégia de evolução do OMM, fornecendo suporte para as comunidades interessadas no OMM administrarem,

influenciarem e colaborarem com a evolução do modelo. A estrutura de organização dessas comunidades é mostrada na Figura 2.6 e é composta pelos seguintes grupos: (i) o *OPG*, grupo direcionador do modelo, responsável por administrar sua evolução; (ii) *Grupos Especialistas* têm o direito de alterar o OMM, subsidiando as novas versões do modelo; (iii) *Contribuidores e Usuários* reportam experiências no uso do modelo, propõem melhorias e encontram e reportam inconsistências no modelo; e (iv) *Patrocinadores OMM* podem oferecer recursos financeiros para a evolução do modelo.



Figura 2.6: Estrutura de comunidades para evolução do OMM (Malheiros, 2010)

Malheiros (2010) também define uma plataforma de evolução do OMM, permitindo a integração de todas as informações relacionadas ao OMM e a edição do modelo, incluindo funcionalidades de controle de versão e de comunicação e coordenação para administrar a evolução do modelo. A plataforma EPF Composer³ foi utilizada como base por ser uma solução difundida na comunidade de software livre, que mantém o modelo, além de permitir a implementação de meta-modelos de processos de engenharia.

2.3 Testes

Para garantir que tanto o modo pelo qual o software está sendo construído quanto o produto em si estejam em conformidade com o especificado são aplicadas ao longo de todo o processo atividades de garantia de qualidade coletivamente chamadas de *Verificação, Validação e Teste* (VV&T). O teste de software é uma atividade dinâmica (se baseia na execução de um programa ou modelo) cujo intuito é executar o programa ou modelo utilizando um conjunto de teste finito, devidamente selecionado do domínio de execuções, e verificar se o comportamento apresentado está de acordo com o esperado (Delamaro et al., 2007; IEEE, 2008).

De uma forma geral, a atividade de testes pode ser dividida em três fases com objetivos distintos: (i) o *teste de unidade*, com foco nas menores unidades de um programa, como funções, procedimentos, métodos ou classes; (ii) o *teste de integração*, com ênfase

³<http://www.eclipse.org/epf/> (último acesso em 08/10/2012)

2.3. TESTES

na construção da estrutura do sistema após o teste de unidade; e (iii) o *teste de sistemas*, cujo objetivo é verificar se as funcionalidades especificadas durante o projeto estão corretamente implementadas. Idealmente, para garantir que o programa testado não possui defeitos, ele teria que ser testado com todas as entradas possíveis; na prática, no entanto, tal abordagem é infactível. A questão de como identificar subdomínios para a seleção de casos de teste é comumente resolvida com a definição de requisitos de teste como, por exemplo, a execução de determinadas estruturas do programa. Dependendo da regra utilizada para a definição dos requisitos são obtidos diferentes conjuntos de casos de teste; tais regras são chamadas “critérios de teste” e podem ser funcionais, estruturais ou baseados em defeitos (ou erros) (Delamaro et al., 2007).

Para ser efetiva na garantia de qualidade, a atividade de teste deve ser conduzida ao longo de todo processo de desenvolvimento, de maneira contínua, sistemática e organizada, com um processo definido para acompanhar e controlar o desenvolvimento do software. Um processo de teste pode ser descrito como um processo usado para revelar defeitos no software, e para estabelecer o grau de qualidade que o software obteve em relação aos atributos de qualidade definidos (Burnstein, 2002; Lewis, 2008).

Crespo et al. (2010) identifica algumas etapas importantes em processos de teste: (i) *Planejamento*, onde é determinado o que deve (ou não) ser testado, quais abordagens de teste serão seguidas e quantos recursos serão alocados à esta atividade; (ii) *Projeto*, onde são revisados os requisitos do ambiente de teste e especificados os procedimentos e casos de teste; (iii) *Execução*, onde ocorre a aplicação dos testes planejados e projetados nas etapas anteriores; (iv) *Acompanhamento*, onde é feita a organização e a consolidação de todas as informações relativas aos testes realizados; e (v) *Finalização*, onde todas as informações sobre o teste são consolidadas em um documento.

As vantagens de se utilizar um processo bem definido e gerenciado estão nas possibilidades de repetição do processo em diversos projetos, avaliação do processo usando uma variedade de métricas e realização de ações para melhorar o processo visando a alcançar melhores resultados (Burnstein, 2002; Lewis, 2008). Assim como acontece no processo geral de desenvolvimento de software, modelos de maturidade de processos de teste apresentam as melhores práticas da área e podem fornecer orientações do que deve ser realizado em um processo bem definido, além de auxiliarem na avaliação do processo vigente e na obtenção de um entendimento do estado atual do processo, servindo como referência para a evolução do processo (Hohn, 2011).

Um exemplo de modelo de maturidade para processos de teste é o TMMi (*Test Maturity Model Integration*) (van Veenendaal, 2009), que fornece um roteiro geral para melhoria do processo de teste. Uma de suas vantagens é a semelhança com a estrutura ao CMMI, tornando-o mais compreensível para a maioria das empresas que já possuem algum modelo

de maturidade para processo de desenvolvimento. De acordo com os autores o modelo foi experimentalmente validado e se fundamenta em normas e padrões bem estabelecidos como IEEE 829 e CMMI, além de ser complementar às atividades de verificação e validação do CMMI versão 1.2 (Hohn, 2011).

A estrutura do TMMi é análoga à estrutura do CMMI e possui cinco níveis de maturidade: *Inicial*, *Gerenciado*, *Definido*, *Gerenciamento e Medição* e *Otimização*. Em cada nível são definidas características que podem ser usadas para avaliar o estado atual do processo de teste de uma organização e também para sugerir áreas em que o processo deve ser melhorado. A evolução de um processo para níveis superiores indica que a organização passou de um processo caótico e mal definido com falta de recursos, ferramentas e testadores bem preparados, para um processo maduro e controlado, que tem a prevenção de defeitos como seu principal objetivo (Hohn, 2011; van Veenendaal, 2009).

Os resultados relatados em trabalhos de pesquisa relacionados a processos de teste no desenvolvimento FOSS são poucos, variados e dependem do projeto analisado (Crowston et al., 2012). De acordo com o estudo de Acuna et al. (2012), a atividade de testes e a condução de revisões são as atividades de VV&T consideradas mais importantes na literatura em processos de desenvolvimento FOSS. No geral é destacado o papel importante dos usuários de FOSS como testadores *beta*, detectando regularmente *bugs* e, quando não os resolvem com acesso direto ao código, levam o problema rapidamente à atenção da comunidade (Erdogmus, 2009).

Um exemplo de projeto FOSS importante que adota estratégias de teste mais formais é o *Linux Test Project* (LTP)⁴. Seu objetivo primário é fornecer uma suíte de testes funcionais e de regressão para a comunidade Linux que auxilie a validar a confiabilidade, robustez e estabilidade do *kernel* Linux. Antes da criação do projeto não havia metodologia formal de testes para desenvolvedores Linux; apesar de testes unitários serem implementados em algumas situações, a implantação de testes sistemáticos de integração era uma tarefa considerada impossível na prática (Modak e Singh, 2009). Modak e Singh (2008) apresentaram as coberturas de testes mostradas na Tabela 2.1 para um subconjunto dos 3000 casos de testes que compunham o LTP na época do trabalho. Pode ser notado que, de 10 subsistemas, apenas 2 apresentam menos de 50% de cobertura.

No entanto, resultados como esses não são encontrados consistentemente na literatura FOSS; a automatização e a adoção de estratégias sistemáticas ou processos de testes é mencionada em poucos trabalhos (Martin e Hoffman, 2007; Potdar e Chang, 2004) e as coberturas de teste obtidas em alguns estudos de caso (Rincon et al., 2011; Rocha et al., 2010; Tahir e Chaim, 2008) indicam uma baixa maturidade dos processos de teste em FOSS. Como exemplo, na Tabela 2.2 são mostrados os resultados obtidos por Rincon et

⁴<http://ltp.sourceforge.net/> (último acesso em 08/10/2012)

2.3. TESTES

Tabela 2.1: Cobertura de testes do projeto LTP (Modak e Singh, 2008)

Diretório	Cobertura	
fs	52.9%	10778/20367 linhas
include/asm	50.9%	613/1204 linhas
include/linux	60.0%	2283/3812 linhas
include/net	57.6%	1015/1762 linhas
ipc	56.4%	1539/2729 linhas
kernel	39.1%	10097/25837 linhas
lib	43.2%	2159/4992 linhas
mm	52.7%	7066/13396 linhas
net	65.7%	633/964 linhas
security	51.9%	666/1283 linhas

al. (2011) para cada projeto analisado. As colunas da tabela são: **PK** – pacotes; **NC** – classes; **NB** – blocos de código; **CC ≠ 0** – classes com cobertura diferente de zero; **BC ≠ 0** – blocos com cobertura diferente de zero; **BC ≥ th** – blocos com cobertura superior ao valor de referência ($th = 80\%$); **CC = 0** – classes sem nenhuma cobertura; e **BC = 0** – blocos sem nenhuma cobertura.

Tabela 2.2: Análise de cobertura de testes de projetos FOSS (Rincon et al., 2011)

Projeto	PK	NC	NB	CC ≠ 0	BC ≠ 0	BC ≥ th	CC = 0	BC = 0
Canoo	29	286	40927	243 (85%)	27332 (67%)	10131 (24,75%)	43 (15%)	13595 (33%)
HttpUnit	11	330	43333	317 (96%)	36105 (83%)	33151 (76,50%)	13 (4%)	7228 (17%)
JFreeChart	40	514	209564	369 (72%)	97529 (47%)	61 (0,03%)	145 (28%)	112035 (53%)
JMeter	94	836	174011	667 (80%)	80546 (46%)	10080 (5,79%)	169 (20%)	93465 (54%)
Log4j	18	245	39216	88 (36%)	14971 (38%)	0 (0%)	157 (64%)	24245 (62%)
Mondrian	26	1605	283245	632 (39%)	64941 (23%)	2306 (0,81%)	973 (61%)	218304 (77%)
Poi	145	2583	856179	873 (34%)	84256 (10%)	29632 (3,46%)	1709 (66%)	771923 (90%)
Velocity	25	221	56031	189 (86%)	34448 (61%)	3150 (5,62%)	32 (14%)	21583 (39%)
Weka	93	2244	874351	1099 (49%)	326494 (37%)	38544 (4,41%)	1145 (51%)	547857 (63%)
Xerces2	82	1050	359941	483 (46%)	124071 (34%)	11800 (3,28%)	567 (54%)	235870 (66%)

A partir dos dados apresentados na Tabela 2.2, podemos concluir que a maioria dos projetos estudados apresentou menos de 6% de cobertura acima do valor de referência utilizado e mais de 50% dos blocos sem nenhuma cobertura de testes. Apenas o projeto **HttpUnit** apresentou resultados satisfatórios, com mais de 75% de cobertura acima do valor de referência definido e mais de 95% de classes com cobertura. Visando melhorar esse cenário, Rincon et al. (2011) propõem uma estratégia de testes envolvendo o mesmo processo distribuído adotado para o desenvolvimento FOSS, registrando, coletando e enviando informações de testes realizados remotamente por membros ou usuários do projeto.

Em outro estudo relacionado, del Bianco et al. (2008) aplicaram um questionário a 151 interessados em FOSS (90% de desenvolvedores, contribuidores e gerentes e 10% de usuários) com perguntas relacionadas à importância de alguns fatores na adoção de componentes e produtos FOSS. As respostas mostraram que o fator “existência de suítes de

teste que garantem a qualidade do produto” teve baixa importância. Os autores argumentam que essa percepção não vem do fato da atividade de teste não ser importante, mas sim do fato de que muitas vezes essas suítes não existem; por isso as partes interessadas muitas vezes não percebem esse fator como importante (Morasca et al., 2011). Essa informação se confirma com o estudo de Tosi e Tahir (2010), onde foram analisados os portais web de 33 produtos FOSS importantes com os seguintes resultados: apenas 6% apresentam a disponibilidade de suítes de teste; apenas 3% deixam explícito o uso de um *framework* para apoiar as atividades de teste; e apenas 18% mostram os resultados completos da execução de suítes de teste.

Morasca et al. (2009) discutem a falta de processos maduros de teste no desenvolvimento FOSS – até mesmo em projetos grandes como Apache e GCC – e enumeram algumas diferenças entre processos de teste de software aberto e fechado que podem ser as causas desse cenário: (i) técnicas bem estabelecidas para software fechado podem não se aplicar diretamente no desenvolvimento FOSS, levando a um grande esforço para projetar novas soluções de teste específicas para o desenvolvimento aberto; (ii) a atividade de testes em FOSS, quando presente, é menos estruturada e raramente segue paradigmas tradicionais da Engenharia de Software; (iii) o planejamento e monitoração do processo de testes em FOSS raramente segue modelos tradicionais; e (iv) a atividade de testes em projetos FOSS é altamente dependente da contribuição dos usuários, por isso os ciclos de desenvolvimento em FOSS são menores e arriscam na distribuição de versões potencialmente problemáticas, contando com o *feedback* dos usuários para a detecção e correção dos problemas. Os autores concluem que, devido a essas diferenças, é necessário redefinir os métodos e modelos que formam as bases dos processos de teste a serem utilizados no desenvolvimento aberto considerando as seguintes características:

- *Visibilidade*: a completa visibilidade do funcionamento interno de programas abertos permite tanto a usuários quanto desenvolvedores exercitar e testar completamente o comportamento dos sistemas. Tal característica torna importantes as técnicas de teste estrutural, incluindo fluxo de controle e de dados, e inspeção de código para preencher os requisitos impostos pelo teste de projetos FOSS.
- *Análise de Sistemas Iterativa*: projetos abertos em geral não incluem atividades prévias de análise e projeto. Assim como em métodos ágeis, o sistema é implementado e refatorado iterativamente, conforme surgem as necessidades dos desenvolvedores e usuários. Técnicas de teste baseadas em especificações (como testes baseados em requisitos ou modelos) podem não ser aplicáveis nesse contexto; além disso, a definição de um plano prévio de testes é inviável. Tal característica torna necessária uma atividade de teste contínua e evolucionária durante todo o desenvolvimento.

- *Processo de desenvolvimento:* devido ao formato colaborativo do desenvolvimento aberto, com a distribuição rápida de novas versões e a participação de usuários como testadores do sistema, projetos FOSS tendem a ser mais rápidos na detecção e correção de defeitos. O compartilhamento de informações de teste entre usuários e desenvolvedores pode levar à aplicação de técnicas de teste baseadas em defeitos (Morell, 1990).

Com base nessas características, Morasca et al. (2011) criaram o OSS-TMM – *Open Source Software Testing Maturity Model*, um método para a avaliação da maturidade de processos de teste de software sob duas perspectivas: (i) para o *desenvolvedor*, o método pretende auxiliar na identificação de estratégias de teste que melhor se encaixam com as características do projeto; e (ii) para o *usuário*, o método pretende auxiliar na classificação e seleção de projetos FOSS considerando a maturidade do processo de teste como um indicador da qualidade geral do produto.

Um componente importante do método é o *checklist* de avaliação de projetos, composto de diversas questões relativas à qualidade do projeto FOSS em questão e dividido em 5 seções (chamadas de *Issues*): (I1) visibilidade da lógica e estrutura do código; (I2) aspectos relacionados à análise e projeto do sistema; (I3) conceitos de desenvolvimento colaborativo e distribuído; (I4) crescimento do projeto; e (I5) documentação. Cada questão deve ser analisada e respondida com “Sim”, “Não” ou “Parcialmente”; grupos de questões são associados a diretrizes para seleção de estratégias de teste que se encaixam com as respostas obtidas (como exemplificado na Figura 2.7).

	Y	P	N	Guidelines
I1 – Code Visibility				
I1.1 Is the source code available via Versioning Systems?				IF Y: regression testing is required for versioned projects to avoid the risk of introducing new bugs from a release to another; white-box unit and integration testing is required for projects structured in components, units of code, packages to test each unit of code in isolation or in integration and provide evidence that the functionality of each unit is implemented as specified.
I1.2 Is the project structured in folders: source, binary, libraries, docs?				
I1.3 Is information about releases (date, number, change-log) visible?				
I1.4 Is information about code revision (author, number, description) visible?				(IEEE Std 1008, 1986; Leung & White, 1990; Pezzè & Young, 2007)

Figura 2.7: Exemplo de questões do *checklist* do OSS-TMM (Morasca et al., 2011)

O método OSS-TMM propriamente dito consiste em 5 passos (*Steps*): (S1) aplicação sequencial do *checklist* e identificação das diretrizes de teste correspondentes ao projeto; (S2) definição das melhores práticas de teste (*Best Test Practices – BTP*) a partir do resultado do passo S1; (S3) identificação das práticas de teste atuais do projeto (*Available*

Testing Practices – ATP); (S4) comparação do conjunto de técnicas BTP com o conjunto ATP, identificando a intersecção entre eles para estimar o nível de maturidade do projeto; e (S5) a partir do nível de maturidade obtido, usuários podem avaliar a qualidade do produto e desenvolvedores podem identificar se é necessário melhorar o ATP do projeto com base no BTP identificado. No OSS-TMM os níveis de maturidade variam entre ML1 – *Sem definição e estrutura* e ML4 – *Bem planejado, monitorado e otimizado*.

Para fornecer evidências da real utilidade do método, os autores aplicaram o OSS-TMM em três projetos FOSS – BusyBox, Apache HTTPD e TPTP – e compararam os níveis de maturidade obtidos com informações do sistema de relatórios de *bugs* dos três projetos. Os resultados do estudo de caso mostraram que um alto nível de maturidade tem relação tanto com a habilidade de detectar *bugs* quanto com a habilidade de resolvê-los.

A qualidade do processo de teste também é um aspecto importante do modelo OMM (descrito na Subsubseção 2.2). Atividades de teste são previstas como TWEs nos três níveis do modelo:

- Nível básico – **Qualidade do Plano de Testes** (QTP): inclui itens como o planejamento dos recursos a serem utilizados, a ordem na qual os testes serão executados, as responsabilidades no projeto e como os resultados serão analisados.
- Nível intermediário – **Teste 1** (TST1): o projeto deve alcançar os requisitos especificados. Inclui o planejamento de atividades de verificação como revisão em pares (*peer review*).
- Nível avançado – **Teste 2** (TST2): o projeto ou componente FOSS deve garantir a satisfação de seus requisitos no ambiente onde ele será utilizado. Inclui o planejamento de atividades de validação.

2.4 Considerações Finais

Neste capítulo foi apresentada uma visão geral de alguns trabalhos importantes da literatura de pesquisa em qualidade de software livre, com destaque em modelos de qualidade de processo e de produto e na atividade de testes. Modelos de qualidade FOSS se baseiam em modelos anteriores propostos com foco no desenvolvimento de software fechado. No entanto, é particularmente importante examinar precisamente as restrições e riscos específicos ao desenvolvimento FOSS. Dado que a quantidade de opções de projetos FOSS é grande, é necessário utilizar métodos de avaliação de qualidade que permitam a diferenciação entre candidatos não só a partir de requisitos técnicos e funcionais, mas também de questões como: a durabilidade do projeto, seu nível de estabilidade, qual o suporte

2.4. CONSIDERAÇÕES FINAIS

disponível pela comunidade e se é possível influenciar futuros desenvolvimentos (Spinellis et al., 2009). Em muitos contextos certificações são necessárias e a avaliação de qualidade utilizando modelos também serve como base para um processo de certificação de qualidade específico para FOSS.

A tendência na literatura é analisar o desenvolvimento FOSS como um fenômeno heterogêneo, mas grande parte das características de qualidade não se aplicam a todos os projetos FOSS e algumas delas se manifestam de formas diferentes dependendo do domínio e do contexto de cada projeto. Por esse motivo, pesquisas futuras devem comparar projetos em diferentes fases de evolução e de diferentes tipos (Kon et al., 2011). Além disso, Crowston et al. (2012) identificam a necessidade de estudos experimentais mais formais; grande parte dos trabalhos da área são estudos de caso caracterizados por pouca ou nenhuma formalidade, sem métodos de pesquisa e coleta de dados bem definidos (apenas 1% dos trabalhos analisados foram classificados como estudos experimentais).

A utilização de testes automatizados e análise de cobertura de testes são consideradas atividades importantes no desenvolvimento FOSS, mas não é comum a definição de políticas específicas nesse sentido (Zaidman et al., 2011). A decisão entre a utilização ou não dessas técnicas fica a cargo de cada desenvolvedor e sua implementação é realizada informalmente. De forma geral, o maior destaque é dado ao papel importante dos usuários de FOSS como testadores *beta* na detecção e eventual correção de bugs, mas poucos trabalhos descrevem a implantação de estratégias de teste bem definidas (Acuna et al., 2012).

Apesar de estar claro que o modelo de desenvolvimento FOSS não é igual ao modelo da Engenharia de Software tradicional, as diferenças exatas entre as duas abordagens ainda não estão claras. Ambas tratam dos desafios do desenvolvimento de sistemas complexos, mas os processos e práticas utilizados diferem significativamente (Ghosh et al., 2002). Os resultados da pesquisa em FOSS estão relacionados à compreensão de diversas áreas como desenvolvimento de software colaborativo e distribuído, evolução de software e o estabelecimento de ecossistemas de software. O modelo FOSS e a Engenharia de Software se complementam em muitos aspectos e diferem em outros; a compreensão desses complementos e diferenças pode auxiliar no avanço de pesquisas futuras em ambas as áreas (Scacchi, 2010).

Visualização de Software

3.1 Considerações Iniciais

Há 10 anos já estimava-se que mais de um milhão de terabytes de informação são gerados por ano, grande parte disponível em formato digital (Keim, 2002). Uma das principais razões é o avanço da tecnologia da informação e das comunicações, além da diminuição dos custos dos dispositivos de armazenamento. Muitas vezes os dados são armazenados automaticamente, por sensores ou dispositivos de monitoração, e até mesmo atividades cotidianas como o uso do cartão de crédito ou do telefone são registradas. Tais dados podem ser recursos valiosos no apoio a atividades como definição de políticas públicas, investigações científicas ou estratégias de negócio; no entanto, para que eles sejam aproveitados ao máximo, é preciso compreendê-los (Heer e Shneiderman, 2012).

A visualização fornece meios poderosos para a compreensão de grandes conjuntos de dados. Em conjunto com sistemas de gerenciamento de dados e algoritmos estatísticos, o mapeamento de atributos em propriedades visuais como posição, tamanho, forma e cor potencializa as habilidades sensoriais do ser humano para o discernimento e interpretação de padrões, agrupamentos, tendências e discrepâncias. A partir de uma representação inicial, o usuário extrai observações e conclusões sobre os dados e interage diretamente com a visualização, moldando-a para atingir os objetivos de sua tarefa (Card et al., 1999; Heer e Shneiderman, 2012; Keim, 2002).

3.1. CONSIDERAÇÕES INICIAIS

O estudo da visualização é dividido em duas grandes vertentes. Quando a visualização é baseada em dados físicos e inherentemente geométricos (como o planeta terra, o corpo humano ou fenômenos da natureza, por exemplo), é chamada de Visualização Científica. Nesse caso o computador é utilizado para exibir graficamente algumas propriedades observadas nesses objetos, utilizando abstrações baseadas no espaço físico. Por outro lado, informações não-físicas – dados financeiros, de negócios ou coleções de documentos, por exemplo – também podem se beneficiar de representações visuais, mas não há nenhuma forma óbvia de se mapear tais dados para imagens. Para isso são pesquisadas técnicas de Visualização de Informação (Card et al., 1999).

Na visualização de informação, dados de entrada consistem em grandes conjuntos de registros, cada um contendo um número de atributos ou dimensões que o identifica (organização similar a uma tabela de dados). A quantidade de dimensões é chamada de dimensionalidade do conjunto de dados. Tipos comuns de dados são os *bidimensionais*, como valores medidos no espaço geográfico, e os *multidimensionais*, como tabelas de bancos de dados relacionais, que podem apresentar centenas de colunas (Oliveira e Levkowitz, 2003). A visualização de dados multidimensionais exige a aplicação de técnicas mais sofisticadas, pois esses tipos de dados não podem ser diretamente mapeados ao espaço 2D ou 3D. Alguns exemplos de técnicas que representam múltiplos atributos visualmente com diferentes abordagens são: as projeções geométricas, como *Coordenadas Paralelas* (Figura 3.1 (a)); as orientadas a pixel, como *Recursive Patterns* e *Circle Segments*; as iconográficas, como *Stick Figures*; e hierárquicas, como *Dimensional Stacking* (Keim, 2002).

Uma alternativa para a visualização de dados multidimensionais é a redução de dimensionalidade dos dados antes da visualização, ou seja, a projeção dos dados de alta dimensionalidade em um espaço de duas ou três dimensões, possibilitando representações visuais como pontos no plano, grafos, superfícies ou volumes. Tipicamente, uma projeção multidimensional tem como objetivo preservar relações de distância entre os dados originais; no entanto, a perda de informação é inevitável e a qualidade dessa preservação depende da precisão da projeção (Paulovich et al., 2007). Um exemplo de visualização criada a partir de uma projeção multidimensional de um conjunto de documentos é ilustrada na Figura 3.1 (b).

O tópico principal abordado neste capítulo – Visualização de Software – é uma sub-área da Visualização de Informação cujos objetivos são auxiliar a compreensão de sistemas complexos de software e melhorar a produtividade do processo de desenvolvimento utilizando visualização. De forma geral, técnicas de Visualização de Software geram representações visuais de diversos aspectos do software e de seu processo de desenvolvimento, sendo

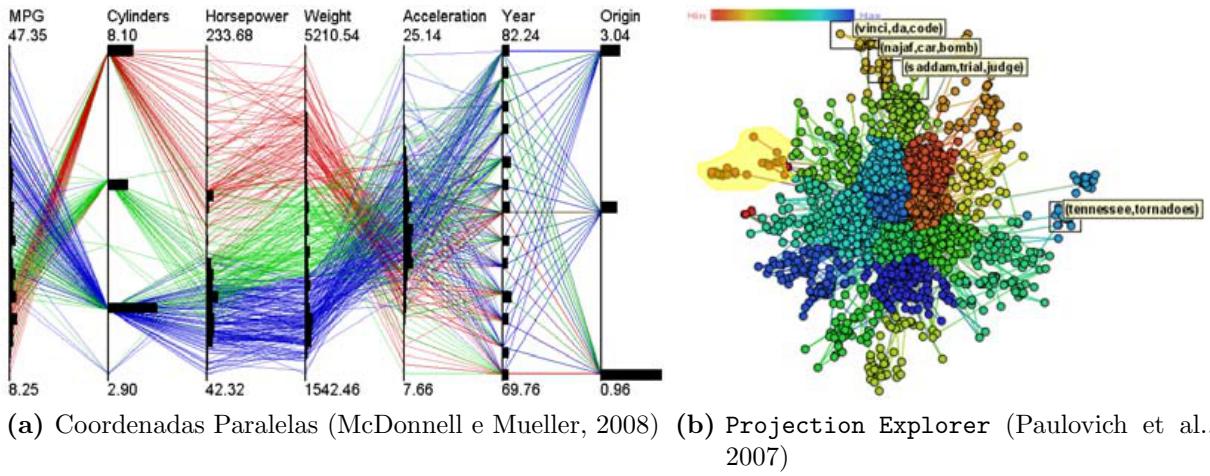


Figura 3.1: Exemplos de visualização de dados multidimensionais. Em ambos as cores representam uma pré-classificação dos dados. (a) Sete dimensões do conjunto de dados (características de carros) são representadas em eixos verticais. (b) Documentos agrupados por similaridade a partir de uma projeção multidimensional, com tópicos frequentes destacados em grupos selecionados.

utilizadas para facilitar tarefas como gerenciamento, projeto, implementação, depuração, análise e manutenção (Diehl, 2007).

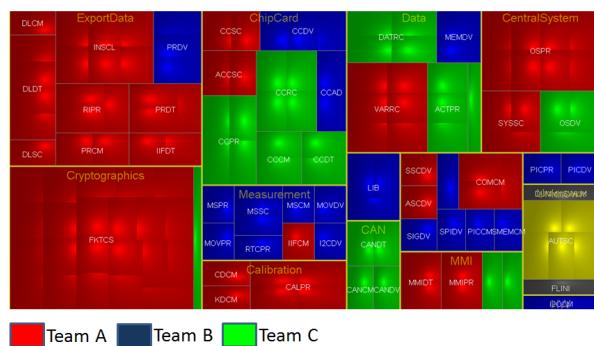
De acordo com Telea (2007), a Visualização de Software é um dos braços da Visualização de Informação que mais cresce atualmente. Alguns grupos de pesquisas se destacam no cenário mundial, como o *Scientific Visualization and Computer Graphics* (SVCG) na Universidade de Groningen (Holanda), o Grupo de Engenharia de Software da Universidade de Trier (Alemanha), o REVEAL (*Reverse Engineering, Visualization, Evolution Analysis Lab*) da Universidade de Lugano (Suíça) e, no Brasil, o Laboratório de Engenharia de Software da Universidade Federal da Bahia (UFBA). O surgimento de diversos eventos e edições especiais de periódicos especificamente sobre Visualização de Software tornam essa tendência ainda mais evidente. Dentre os principais eventos destacam-se o *ACM Symposium on Software Visualization* (SOFTVIS), desde 2003; o *IEEE International Workshop on Visualizing Software for Understanding and Analysis* (VISSOFT), desde 2002; e o *Program Visualization Workshop* (PVW), desde 2000.

O restante do capítulo está organizado da seguinte forma: na Seção 3.2 são apresentados os conceitos básicos da área de Visualização de Software, com diversos exemplos de técnicas e ferramentas que os implementam; na Seção 3.3 é apresentada a motivação por trás da inclusão de aspectos sociais em ferramentas de visualização de software e alguns exemplos de técnicas e ferramentas com esse foco; e na Seção 3.4 são apresentadas as considerações finais sobre o capítulo.

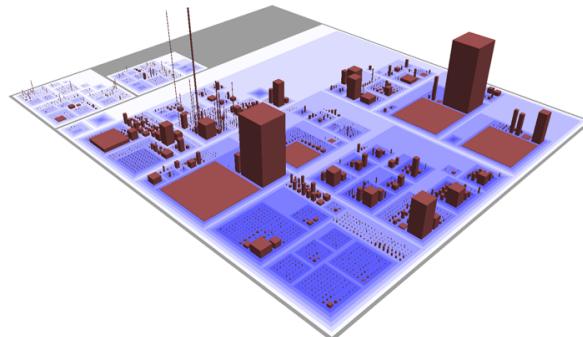
3.2 Conceitos Básicos, Técnicas e Ferramentas

O estudo da Visualização de Software é geralmente dividido em três categorias principais (Diehl, 2007) – *Estrutura, Comportamento e Evolução*.

Estrutura A categoria *Estrutura* se refere às partes e relações estáticas do sistema, ou seja, aquelas que podem ser computadas ou inferidas sem executá-lo. Isso inclui o código-fonte e as estruturas de dados do programa, o grafo de chamadas estático e a organização do programa em módulos. Do ponto de vista da visualização, algumas propriedades de código-fonte são interessantes: (i) exatidão, por ser escrito em linguagens com gramáticas estritamente definidas e sem ambiguidade; (ii) grande escala, já que sistemas modernos possuem milhões de linhas de código; (iii) relações e hierarquias entre entidades do código-fonte; e (iv) diversos atributos que expressam características dessas entidades (Diehl, 2007). Na Figura 3.2 são ilustrados dois exemplos de técnicas para visualização da hierarquia – pacotes, módulos e arquivos – do código fonte de projetos de software orientado a objetos.



(a) *Team Assessment* (Telea e Voinea, 2009)



(b) *CodeCity* (Wettel e Lanza, 2008)

Figura 3.2: Exemplos de visualização de estrutura de software. (a) Hierarquia representada com a técnica *TreeMap* e cores relacionando cada trecho do código com o time responsável pelo seu desenvolvimento (Telea e Voinea, 2009). (b) Abstração similar a uma “cidade de software”, representando classes e interfaces como “prédios” e pacotes como “bairros”.

Grande parte das ferramentas propostas na categoria *Estrutura* compartilham um mesmo modelo conceitual: a geração de um grafo, onde vértices representam entidades do programa, como arquivos ou classes, e arestas representam dependências como usos ou heranças. A técnica *Hierarchical Edge Bundles* (HEB) (Holten, 2006), ilustrada na Figura 3.3, é utilizada para visualizar ao mesmo tempo duas informações do grafo: a estrutura hierárquica dos arquivos do código fonte e suas dependências, ou seja, as chamadas entre funções ou métodos.

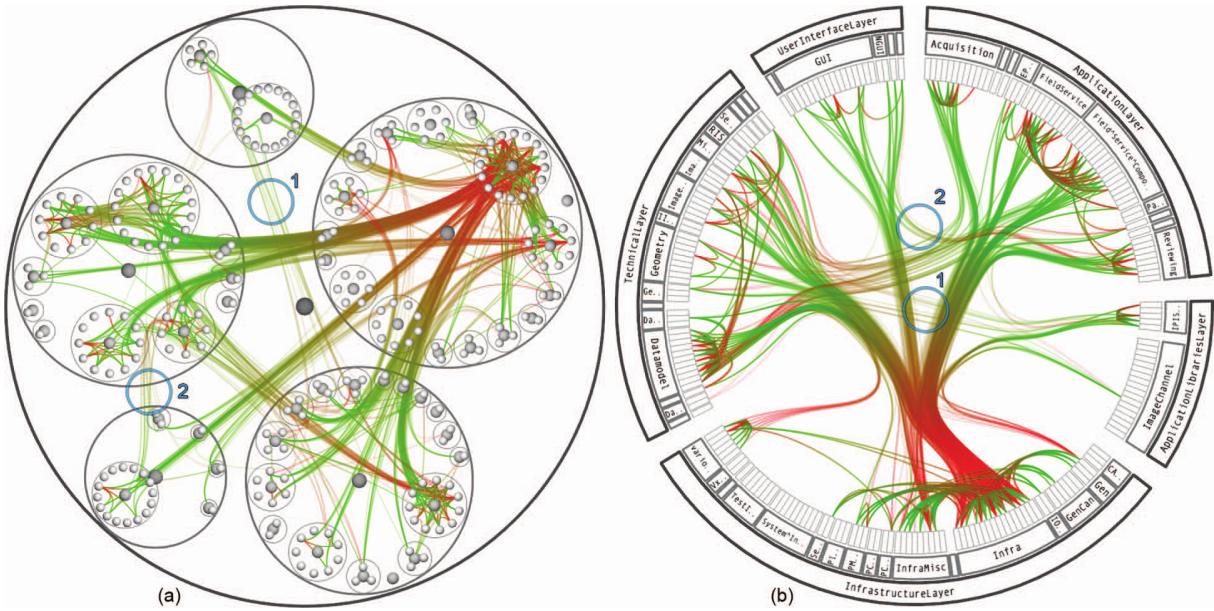


Figura 3.3: Exemplos da técnica HEB (Holten, 2006). A hierarquia de arquivos do código fonte é ilustrada em (a) com layout de balão e em (b) com layout radial. As arestas são agrupadas para minimizar a confusão na interpretação e suas cores indicam a direção da chamada – do verde para o vermelho. As áreas destacadas (1 e 2) mostram as posições das mesmas arestas nos dois exemplos para comparação.

Valores de métricas podem ser incluídos como atributos a serem explorados na visualização da estrutura. A ferramenta **SolidSX** (Reniers et al., 2011) inclui por padrão algumas métricas de código fonte como linhas de código, comentários e complexidade. A exploração visual das métricas é apoiada por três técnicas em conjunto: uma visão *TreeMap* com informações da hierarquia do código; uma visão HEB com informações de hierarquia e dependências; e uma visão *TableLens* (Rao e Card, 1994), que representa cada módulo como uma linha e cada métrica como uma coluna de uma tabela, mapeando os valores das métricas no tamanho e cor de cada célula. A Figura 3.4 apresenta um exemplo da ferramenta.

Comportamento A segunda categoria de técnicas de Visualização de Software – *Comportamento* – se refere à compreensão do que acontece com o programa em tempo de execução, ou seja, qual seu comportamento (quais instruções são executadas e como seus estados mudam) dada uma entrada real ou abstrata. A execução é comumente representada por uma sequência de estados, onde cada estado contém o código atual e as estruturas de dados do programa; nesse caso, a visualização deve combinar dados e código para que seja possível analisar como o programa interage com a memória. Dependendo do paradigma de programação, a execução também pode ser vista – em um nível alto de abstração – como funções chamando outras funções ou como comunicação entre objetos. Possíveis aplicações dessa categoria são: análise de *traces* de execução, visualização dinâmica e

3.2. CONCEITOS BÁSICOS, TÉCNICAS E FERRAMENTAS



Figura 3.4: Exemplo da ferramenta **SolidSX** (Reniers et al., 2011). Múltiplas visões coordenadas são utilizadas para exibir valores de métricas. As áreas predominantemente verdes representam trechos do código com valores de métricas considerados bons; já na área destacada (*hot spot*) são identificados alguns módulos possivelmente problemáticos, com valores ruins para métricas de complexidade e tamanho.

recuperação de arquitetura, animação de algoritmos, depuração visual e apoio visual à atividade de teste (Cornelissen et al., 2009; Diehl, 2007).

Na Figura 3.5 é apresentado um exemplo de técnica de visualização de comportamento. Com o objetivo de apoiar o processo de depuração, a ferramenta **Tarantula** (Jones et al., 2002) utiliza informações da execução de casos de teste para mapear visualmente possíveis localizações de falhas no código. Como pode ser observado no exemplo, cada comando do código fonte do programa é colorido de acordo com sua participação na execução dos testes, em uma escala que vai do vermelho ao verde. Se mais casos de teste bem sucedidos passaram pelo comando, sua cor será mais próxima do verde; do contrário, sua cor se aproximará do vermelho. Analisando os comandos mais próximos do vermelho, o

desenvolvedor tem maiores chances de encontrar as falhas que causaram problemas nos casos de teste executados.

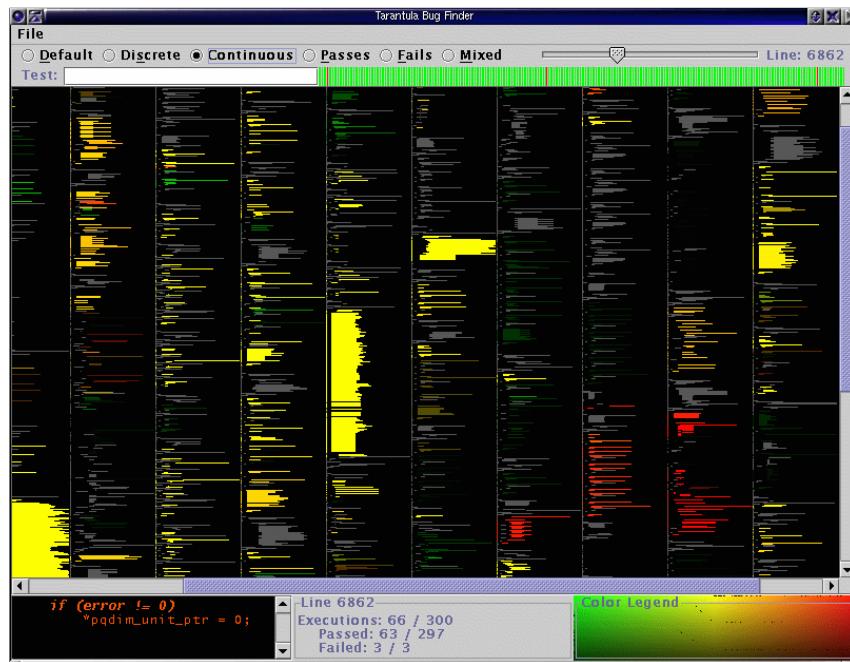


Figura 3.5: Exemplo da ferramenta **Tarantula** (Jones et al., 2002). Para a apoiar a depuração visual, linhas de código são destacadas de acordo com sua participação na execução dos testes, em uma escala que vai do vermelho (falha) ao verde (sucesso).

Evolução Técnicas de visualização da categoria *Evolução* são aplicadas à compreensão das modificações do código ao longo do tempo. A manutenção do código para correção de erros, melhorias, intervenções preventivas e adaptações é uma das fases mais custosas de todo o processo de desenvolvimento, chegando a 80% do custo total do desenvolvimento (Pfleeger et al., 2005). Entender a evolução do projeto sob os pontos de vista administrativo e gerencial pode ser tão ou mais importante do que entender sua estrutura. Nesse contexto, a visualização tem sido utilizada para capturar a dinâmica das modificações ocorridas no software conforme ele evolui, permitindo ao analista elaborar e verificar hipóteses sobre como documentos específicos são modificados, quais as relações entre as modificações de documentos distintos e quais são as tendências dessas modificações (Diehl, 2007; Telea, 2007).

Na Figura 3.6 (a) é ilustrada a evolução do projeto de software aberto VTK (Schroeder et al., 2000) durante o período de 1994 a 2001, utilizando a ferramenta de visualização CVSgrab (Voinea e Telea, 2009). O histórico de modificações dos arquivos, usado como entrada para a ferramenta, é obtido a partir do repositório de controle de versão. A partir da visualização é possível analisar, por exemplo, quais são os arquivos mais antigos e mais modificados, ou grupos de arquivos regularmente modificados em conjunto. Em um outro

exemplo – Figura 3.6 (b) – Zaidman et al. (2011) usam um layout similar ao anterior para responder questões relacionadas à co-evolução entre código de produção e código de testes, como: (i) se cada módulo de produção possui testes associados e se eles são adicionadas e modificados ao mesmo tempo; (ii) se existe relação de sincronia entre o crescimento dos dois tipos de código ou se a atividade de teste é realizada apenas imediatamente antes da distribuição de versões; e (iii) se a cobertura de testes também evolui junto com o código.

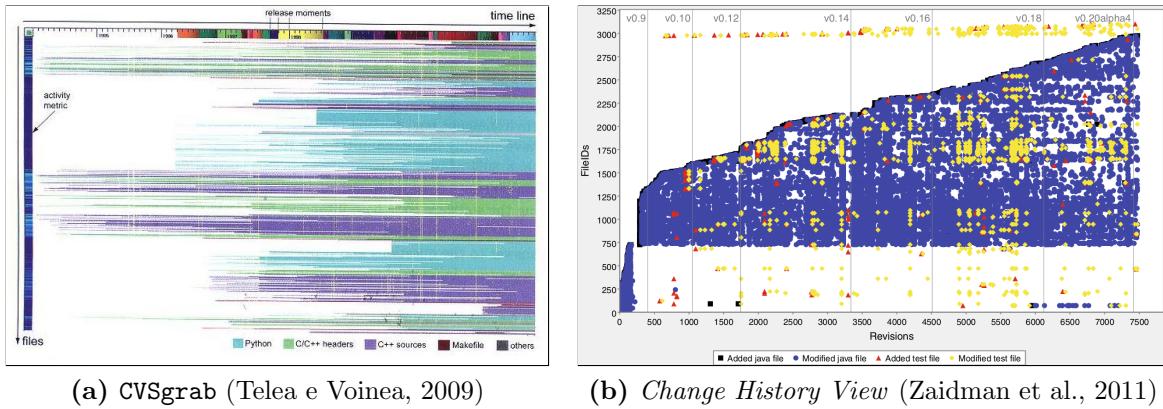


Figura 3.6: Exemplos de visualização de evolução de software. (a) As cores das linhas indicam o tipo dos arquivos (posicionados ao longo do eixo y) e um pixel amarelo separa as diferentes versões de cada arquivo no tempo (eixo x). (b) Um layout bidimensional similar ao anterior é usado para representar a co-evolução entre código de produção (azul) e código de testes (amarelo).

3.3 Aspectos Sociais e Múltiplas Perspectivas

Durante o desenvolvimento de um projeto de software diversos artefatos são gerados, cada um representando diferentes informações importantes para gerentes e pessoal envolvido em tarefas de desenvolvimento e manutenção. A compreensão desses artefatos é a principal motivação das técnicas apresentadas até aqui.

No entanto, recentemente tem crescido também o interesse na compreensão dos relacionamentos entre indivíduos de times de desenvolvimento e seu impacto no software, com pesquisas baseadas principalmente em análise de redes sociais (Sarma et al., 2009; Wasserman e Faust, 1994). Esse aspecto é especialmente importante no contexto de projetos de desenvolvimento distribuído e FOSS. Estudos mostram que desenvolvedores distribuídos em diferentes localidades e fusos horários gastam em torno de 40% do seu tempo se comunicando e tomando consciência do andamento do projeto, gerando uma grande quantidade de comunicação eletrônica (Gutwin et al., 2004; LaToza et al., 2006). Comunidades de software aberto representam espaços sociais vibrantes onde desenvolvedores e usuários, em sua maioria organizados espontânea e voluntariamente, discutem sobre a adição de

funcionalidades, relatórios de bugs, aspectos legais e revisão de código; grande parte dessas discussões estão armazenadas e podem ser utilizadas na compreensão do processo de desenvolvimento (Gilbert e Karahalios, 2007; Ogawa et al., 2007).

Diversas técnicas e ferramentas têm sido propostas para apoiar a análise visual de projetos de desenvolvimento distribuído com foco nas atividades sociais dos desenvolvedores em conjunto com aspectos técnicos do repositório de código fonte. A seguir são apresentados alguns exemplos que utilizam diferentes metáforas para correlacionar essas duas perspectivas.

A técnica ilustrada na Figura 3.7, criada por Ogawa et al. (2007), tem como objetivo representar visualmente a evolução de redes de comunicação em projetos FOSS em conjunto com informações do repositório. Na visão de lista de mensagens (*Mailing List View*) o tempo flui para baixo, dividido em meses; em cada mês as conversações (ovais azuis) entre os participantes são agrupadas em *clusters*, organizados da esquerda para a direita de acordo com o tamanho (maiores primeiro). Arestas representam o fluxo de participantes entre *clusters* de conversações ao longo do tempo. A seleção de um arquivo na visão de repositório (*File Repository View*) destaca na visão de lista de mensagens apenas os desenvolvedores que trabalharam no arquivo. Com o uso da visualização o analista pode acompanhar o andamento das principais discussões, gerais ou relacionadas à determinados arquivos, inferindo se as mesmas se mantêm ao longo do tempo ou se fragmentam em diversas outras pequenas discussões.

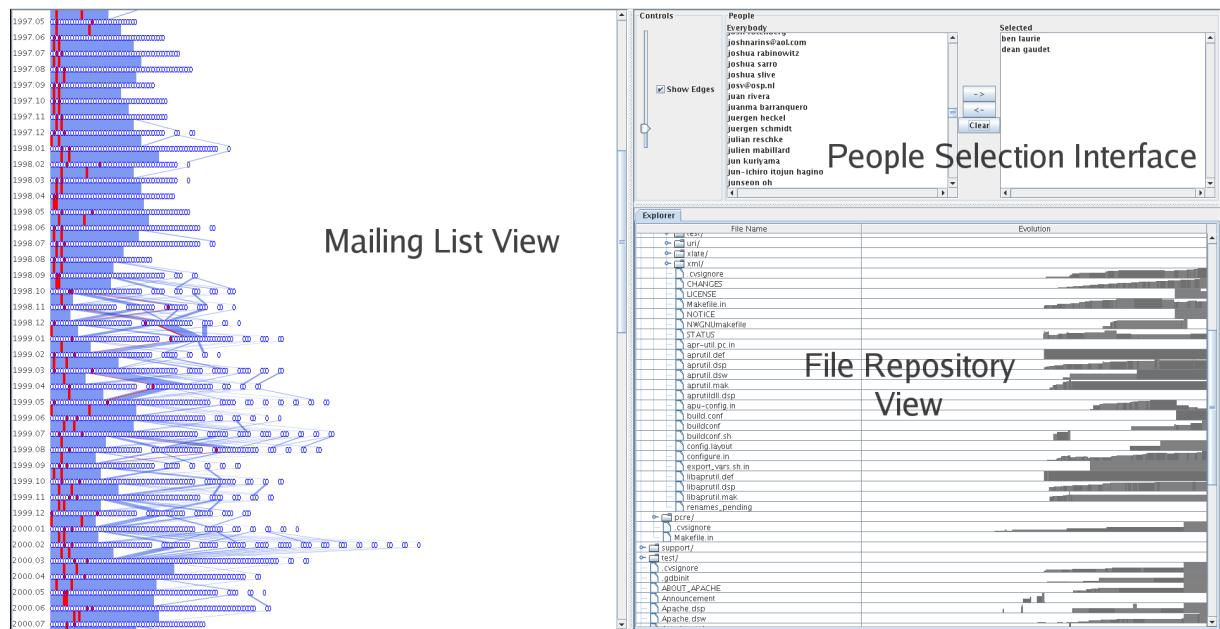


Figura 3.7: Visualização da rede de comunicação do projeto Apache (Ogawa et al., 2007), correlacionando a lista de mensagens, o repositório de código e os desenvolvedores.

3.3. ASPECTOS SOCIAIS E MÚLTIPLAS PERSPECTIVAS

A ferramenta **CodeSaw** (Gilbert e Karahalios, 2007) também foca na perspectiva social para apoiar a análise visual do desenvolvimento de software distribuído mas, ao invés de mostrar medidas complexas do repositório e da lista de discussões, os autores optaram por um projeto rápido e simples, de fácil absorção pelos usuários. No exemplo da Figura 3.8 são apresentadas as atividades de oito desenvolvedores (linhas do tempo na parte inferior); dois deles estão destacados na parte superior, para comparação. Cada linha do tempo exibe a atividade do participante no repositório e na lista de mensagens.



Figura 3.8: Comparação sócio-técnica com a ferramenta **CodeSaw** (Gilbert e Karahalios, 2007). As linhas do tempo representam as atividades no repositório (áreas dos triângulos voltados para cima) e na lista de mensagens (áreas dos triângulos voltados para baixo). A atividade de um dos desenvolvedores foi destacada, mostrando os principais arquivos modificados na faixa de tempo selecionada.

Na ferramenta **StarGate** (Ogawa e Ma, 2008) os autores combinam diversas técnicas para representar, em uma mesma visão integrada, a rede social de participantes (inferida a partir da análise de discussões por e-mail) e a estrutura do repositório de código fonte. Na Figura 3.9 é apresentado um exemplo. O “portal” (*The Gate*) é a base da visualização e mostra a estrutura de diretórios do repositório em uma hierarquia com layout radial. As estrelas (*Stars*) no centro representam os desenvolvedores; o tamanho de cada estrela

indica a quantidade de interações do desenvolvedor com o repositório e elas são posicionadas próximas às áreas do portal nas quais o desenvolvedor mais atuou. Por fim, a “poeira estelar” (*Stardust*) representa o histórico de modificações de cada arquivo; cada ponto indica uma modificação (na mesma cor da estrela do desenvolvedor que a realizou) e pontos mais próximos ao portal representam modificações mais recentes. Os autores afirmam que essa visualização pode beneficiar de formas diferentes tanto novatos no projeto quanto gerentes e pesquisadores em Engenharia de Software (Ogawa e Ma, 2008).

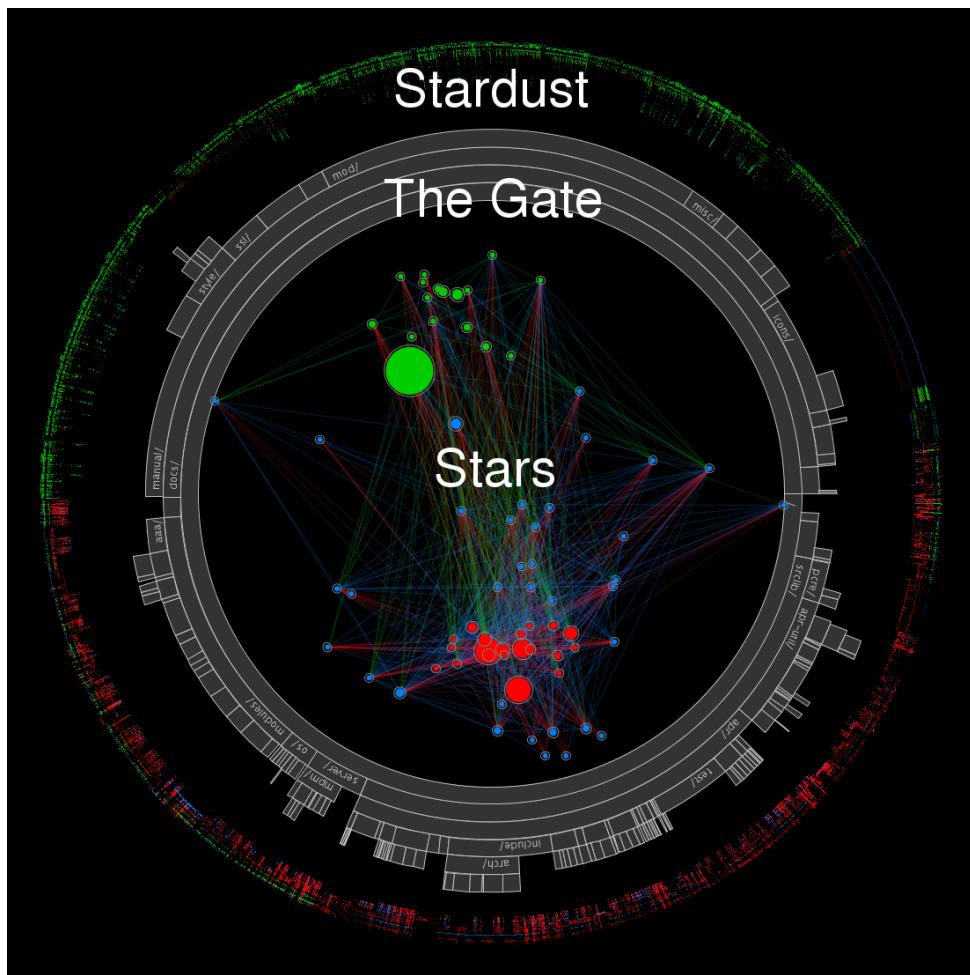


Figura 3.9: Exemplo da ferramenta **StarGate** (Ogawa e Ma, 2008). São representadas em conjunto diferentes informações como a estrutura de diretórios do repositório (*The Gate*), a atividade dos desenvolvedores (*Stars*) e o histórico de modificações dos arquivos (*Stardust*).

Outros dois exemplos recentes de técnicas de visualização e interação que coordenam informações da comunicação entre desenvolvedores e modificações no código são: (i) a ferramenta **Tesseract** – chamada de “um navegador de dependências sócio-técnicas” – que mostra simultaneamente os relacionamentos sociais e técnicos entre diferentes entidades do projeto (desenvolvedores, comunicação, código e bugs); e (ii) a ferramenta **XFlow** (Santana et al., 2010), que utiliza algumas métricas simples como *número de commits por*

3.4. CONSIDERAÇÕES FINAIS

semana e número de artefatos modificados por semana para representar a atividade dos desenvolvedores no histórico do projeto. Ambas apresentam um aspecto importante que as diferencia das anteriores: a representação de dependências lógicas, ou seja, dependências não-explicícitas entre artefatos que são frequentemente modificados em conjunto (Gall et al., 1998).

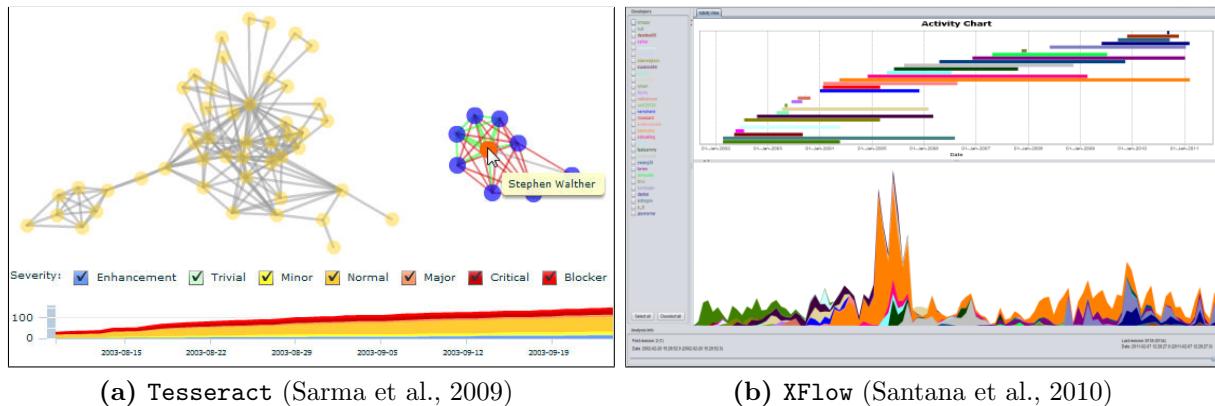


Figura 3.10: Exemplos de visualização sócio-técnica de software. (a) Coordenação entre a rede de comunicação dos desenvolvedores, dependências lógicas entre arquivos e linha do tempo de relatórios de defeitos. (b) Contribuições de cada desenvolvedor (cor) por período (barras na parte superior) e nível de atuação (áreas na parte inferior).

Uma alternativa que também tem sido explorada neste contexto é o uso de animação. A ferramenta `code_swarm` (Ogawa e Ma, 2009) apresenta, de acordo com os autores, uma abordagem diferente da maioria das aplicações de Visualização de Software; ao invés de fornecer uma visão quantitativa e estruturada dos dados, a abordagem qualitativa do trabalho – chamada de “Visualização de Informação Orgânica” – permite complementar visões mais aplicadas e alcançar um público mais casual. Outro exemplo similar é a ferramenta `Gource` (Caudwell, 2010). O histórico de desenvolvimento do projeto é ilustrado como uma árvore animada, com a raiz no centro, diretórios como galhos e arquivos como folhas. Os desenvolvedores são destacados em conjunto com os arquivos conforme sua contribuição no projeto. Exemplos destas duas ferramentas são apresentados na Figura 3.11.

3.4 Considerações Finais

Neste capítulo foi apresentada uma visão geral da área de Visualização de Software, desde conceitos básicos até aplicações mais recentes relacionadas ao presente trabalho.

Cada uma das categorias de Visualização de Software (*Estrutura, Comportamento e Evolução*) é utilizada em contextos diferentes, dependendo das necessidades do analista,

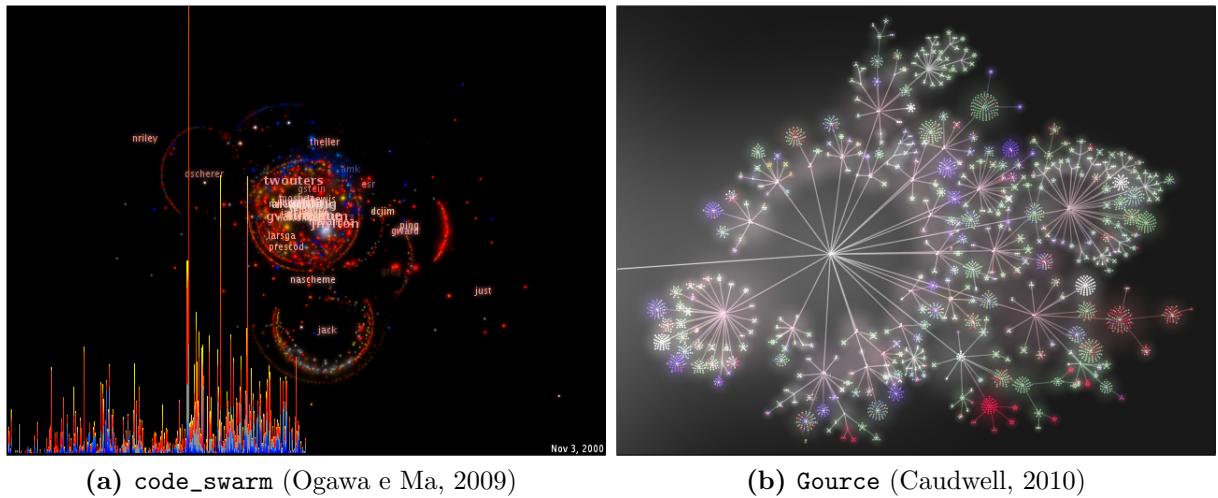


Figura 3.11: Exemplos de visualização de software com animação. (a) Instante da evolução do projeto *Python* mostrando atividades dos desenvolvedores. (b) Uma trecho da hierarquia do projeto *Linux*.

e possui vantagens e desvantagens. Diehl (2007), no entanto, observa que a combinação de visões de categorias diferentes – agregando informações temporais e dinâmicas à visualização estrutural, por exemplo – é um campo importante de pesquisas e deve levar à geração de representações visuais mais poderosas e completas.

A combinação entre a perspectiva clássica da Visualização de Software – com foco em aspectos técnicos, como código fonte – e a perspectiva social, incluindo os diversos tipos de interações entre desenvolvedores e usuários, pode levar à obtenção de informações mais sólidas sobre o processo de desenvolvimento de software. Os padrões de comunicação entre desenvolvedores, por exemplo, podem ser analisados com base nas dependências entre as tarefas realizadas pelos mesmos. Nesse ponto, a análise visual com múltiplas visões coordenadas se torna importante, visto que esses diferentes aspectos constituem uma grande quantidade de informações a serem analisadas ao mesmo tempo (Carneiro et al., 2010; Heer e Shneiderman, 2012).

No contexto do desenvolvimento de FOSS, a visualização de aspectos sociais e técnicos pode beneficiar tanto membros da comunidade quanto interessados externos. Para desenvolvedores e participantes ativos, a análise visual oferece a chance de refletir sobre a comunidade, seus membros e suas contribuições. Como grande parte dos desenvolvedores são voluntários, a percepção das repercussões de suas atividades pode ser uma ferramenta poderosa de motivação. Para potenciais usuários e integradores, a visualização de software fornece a oportunidade de comparar projetos FOSS e analisar de forma mais aprofundada aqueles considerados mais relevantes (Gilbert e Karahalios, 2007).

Um exemplo da percepção atual da comunidade FOSS sobre a utilidade de ferramentas de visualização de software é apresentado por Sarma et al. (2009). Os autores validaram

3.4. CONSIDERAÇÕES FINAIS

sua ferramenta (**Tesseract**) a partir de entrevistas com membros ativos e experientes da comunidade de software aberto, levantando questões sobre suas capacidades e sua real utilidade nesse contexto. De acordo com os autores, as principais funcionalidades identificadas pelo entrevistados foram: (i) a visualização de dependências lógicas entre módulos, possibilitando a análise de quais arquivos são modificados em conjunto e os efeitos de cada modificação; e (ii) as ligações entre desenvolvedores e código, possibilitando explorar questões como o interesse de cada desenvolvedor no projeto, os desenvolvedores mais aptos a responder dúvidas sobre determinados problemas no código ou os responsáveis por mudanças de alto impacto. Algumas opiniões divergiram entre desenvolvedores e gerentes; gerentes consideraram muito útil a representação de ligações entre desenvolvedores, indicando grupos de cooperação implícitos no projeto, enquanto desenvolvedores ativos não a consideraram tão útil por ainda confiarem no conhecimento gerado a partir do acompanhamento constante das listas de discussão.

Por fim, vale a pena reforçar que, não só em Visualização de Software mas em Visualização de Informação no geral, uma única imagem estática apenas responde a algumas poucas questões. Para uma análise visual efetiva de conjuntos de dados complexos é necessário que o analista tenha suporte para realizar um processo iterativo e interativo de criação de visões, exploração e refinamento. A repetição desses passos é o que leva à detecção de relacionamentos importantes, influências contextuais e padrões de causas entre eventos de um domínio. Portanto, ferramentas de análise visual devem apoiar o uso fluente e flexível de visualizações, se adaptando dinamicamente às necessidades do analista, mas ao mesmo tempo evitando a utilização de visões excessivamente complexas e confusas, operações escondidas ou tempos longos de resposta (Heer e Shneiderman, 2012).

Proposta de Trabalho

4.1 Considerações Iniciais

Uma forma comum de análise de qualidade de software é pela aplicação de métricas calculadas por ferramentas de análise de código. Nesses casos, as métricas obtidas são indicadores de qualidade específicos de determinados aspectos do software, como a identificação de padrões em relatórios de defeitos, detecção de clones ou análise de dependências (Deissenboeck et al., 2011).

No entanto, uma única medida não pode ser vista como sinônimo de qualidade ou sucesso de um sistema. Diferentes medidas oferecem diferentes perspectivas sobre o processo de desenvolvimento e o produto de software. Uma análise efetiva deve ser multidimensional, envolvendo um conjunto abrangente de medidas, complementares entre si, obtidas não só do código fonte mas a partir artefatos como documentação, testes e repositório (Jung et al., 2004; Lee et al., 2009; Mens, 2012). Nesse caso os resultados de diversas ferramentas devem ser considerados em conjunto; porém, devido à heterogeneidade dos resultados, essa é uma tarefa complexa (Deissenboeck et al., 2011). Apesar da importância da análise multidimensional de qualidade, grande parte dos estudos experimentais em FOSS apenas considera poucas dimensões (Crowston et al., 2012).

4.2. MOTIVAÇÃO

Algumas iniciativas como **Ohloh**, **Nemo** (baseado na plataforma **Sonar**) e **conQAT**¹ coletam, agregam e exibem dados de diferentes ferramentas de métricas especializadas, construindo um “perfil” para projetos de software que permite o acompanhamento de atividades de desenvolvimento. No entanto, nesses casos o problema principal é que não existe uma conexão direta entre os muitos indicadores (valores de métricas) obtidos e exibidos e características ou atributos específicos de qualidade, o que dificulta a interpretação de qual o real problema por trás de valores anormais medidos (Deissenboeck et al., 2011; Marinescu et al., 2010).

Modelos de qualidade de software atacam esse problema definindo características e atributos de qualidade que são decompostos em métricas; os valores das métricas, depois de agregados, definem a pontuação de cada característica de qualidade do sistema. A pontuação geral obtida a partir da agregação de cada característica de qualidade pode ser útil para uma comparação de alto nível entre diferentes projetos, caso o avaliador não esteja interessado em características específicas de cada candidato. No entanto, para os desenvolvedores e gerentes de projeto, ou mesmo para avaliadores que desejam informações mais aprofundadas, esse valor não agrupa informações sobre exatamente onde estão as vantagens e desvantagens de cada produto. Nesse caso, uma análise dos fatores específicos que determinaram a pontuação final de qualidade é necessário. No entanto, esses modelos são grandes, complexos e fornecem uma grande quantidade de dados que devem ser analisados ao mesmo tempo. Pode ser encontrado ferramental para a extração semi-automática dos dados necessários para realizar a avaliação, mas na maioria das vezes o analista não tem apoio para analisar e interpretar toda essa informação (Deissenboeck et al., 2011).

4.2 Motivação

Para que modelos de qualidade sejam adotados na prática é importante que eles sejam validados experimentalmente e mostrem evidências sólidas da real importância da aplicação de suas diretrizes. No entanto, grande parte dos estudos em FOSS, incluindo modelos de qualidade, não apresentam esse tipo de validação (Crowston et al., 2012). Outro ponto importante da aplicação de modelos de qualidade é a customização. Gerentes de projeto precisam de mecanismos para determinar, a partir da análise do histórico do projeto, quais são os atributos de qualidade mais importantes para o projeto em questão.

Conforme apresentado nos Capítulos 2 e 3, a análise da comunidade em torno de projetos FOSS é um fator importante tanto na avaliação de qualidade quanto na definição de

¹<http://ohloh.net/>; <http://nemo.sonarsource.org/>; <https://www.conqat.org/> – Último acesso: 11/10/2012.

técnicas de visualização de software. A intersecção entre essas duas realidades, no entanto, ainda não foi explorada satisfatoriamente; técnicas atuais de visualização de software que consideram aspectos sociais e técnicos em conjunto (apresentadas na Seção 3.3) apresentam algumas limitações quanto à sua utilização na avaliação de qualidade: (i) no geral são ferramentas de exploração, especializadas na análise de um conjunto limitado de características de software; (ii) métricas, quando apresentadas, são consideradas individualmente ou em pequenos grupos, devido à dificuldade na visualização de diversos indicadores ao mesmo tempo e na interpretação de seus valores; e (iii) nenhuma das técnicas de visualização estudadas está acoplada a um processo de desenvolvimento ou a um modelo de qualidade, o que dificulta sua aplicação na prática e a interpretação dos dados visualizados especificamente para a avaliação e monitoramento de qualidade.

Em um trabalho recente (Martins et al., 2012) foram propostas técnicas de análise visual de redes sociais baseadas nos múltiplos atributos numéricos, temporais ou textuais comumente associados aos vértices (atores) e arestas (relacionamentos) das redes. A partir do uso de redes heterogêneas – onde os vértices representam diferentes elementos – e de técnicas de *layout* baseadas em similaridade de atributos e conexões (ao invés de abordagens mais comuns que utilizam algoritmos baseados em força), as técnicas apresentadas apoiam a identificação de grupos de elementos com propriedades similares. Também é possível coordenar a visualização baseada em redes com outras visualizações que destaquem diferentes características do conjunto de dados, como similaridade de conteúdo textual.

Algumas características da avaliação de qualidade FOSS tornam interessante a possibilidade de aplicação de técnicas de visualização multidimensional de redes: (i) tanto aspectos sociais – como comunicação entre desenvolvedores – quanto aspectos técnicos – como relações de dependência e hierarquia de código fonte – do desenvolvimento FOSS podem ser modelados como redes, formando conjuntos de vértices e arestas; (ii) modelos de qualidade FOSS definem múltiplas métricas que podem ser mapeadas em atributos de vértices e arestas; (iii) a visualização multidimensional auxilia na resolução do problema da análise de grandes quantidades de atributos; e (iv) aspectos sociais e técnicos do desenvolvimento FOSS podem ser visualizados em conjunto tanto como redes heterogêneas quanto com diferentes visões coordenadas.

É importante também ressaltar que um modelo de qualidade por si só não é o bastante para a realização da avaliação de qualidade de um produto de software. Também é necessário um método ou processo que descreva a utilização do modelo e guie sua aplicação efetiva. A norma ISO/IEC 14598 (ISO/IEC, 1998), por exemplo, foi projetada para ser utilizada em conjunto com o modelo definido na norma ISO 9126 (ISO/IEC, 2001), e descreve todo o processo de planejamento, execução e documentação da avaliação de

qualidade de produtos de software, considerando pontos de vista como desenvolvedores, integradores e avaliadores externos (da Rocha et al., 2001). Os modelos de qualidade FOSS também são acompanhados de métodos que guiam sua aplicação. O método QSOS por exemplo, descrito na Seção 2.2, envolve um elaborado processo iterativo de *feedback* para garantir que o conhecimento da comunidade sobre as avaliações realizadas seja constantemente atualizado; por outro lado, o modelo QualOSS, por exemplo, foca mais na automatização da extração das métricas e fornece apenas uma sequencia genérica de passos para sua aplicação.

4.3 Detalhes da Proposta e Objetivos

O principal objetivo deste trabalho é a definição de um processo de avaliação, monitoramento e melhoria de qualidade de projetos FOSS baseado em técnicas de visualização. Para isso o projeto foi dividido em três etapas, detalhadas a seguir.

Etapa 1: Avaliação de Modelos de Qualidade

Nesta etapa do trabalho serão realizados estudos experimentais com os modelos de qualidade disponíveis na literatura para responder às seguintes questões de pesquisa:

- *Como os modelos de qualidade FOSS se comparam a modelos tradicionais?*

Para responder a essa questão, avaliações de projetos de ambos os contextos (aberto e fechado) serão realizadas tanto com modelos tradicionais quanto com modelos específicos para FOSS. Os resultados deverão mostrar se: (i) os modelos tradicionais realmente não são suficientes para a análise de projetos FOSS; (ii) se os modelos de qualidade FOSS são capazes de detectar características técnicas consideradas importantes, como confiabilidade e segurança; e (iii) se os modelos de qualidade FOSS podem identificar características de projetos fechados – como a utilização de um processo de desenvolvimento distribuído, por exemplo – que os tornam candidatos a serem abertos pela organização,

- *Os modelos de qualidade FOSS consideram corretamente a importância da atividade de teste para a qualidade?*

O principal objetivo deste estudo será verificar experimentalmente qual o impacto da atividade de testes na qualidade FOSS, com o objetivo de determinar se os modelos de qualidade FOSS consideram corretamente essa informação na avaliação geral da qualidade. Para isso serão analisados os processos de teste e a qualidade dos conjuntos de teste de

projetos FOSS importantes e esses dados serão comparados com a qualidade avaliada com base nos modelos existentes. Os dados obtidos serão importantes na consideração das medidas que refletem com mais precisão a qualidade de projetos FOSS.

- *As métricas definidas pelos modelos estão corretamente relacionadas com as características agregadas?*

De acordo com Jung et al. (2004), se as métricas utilizadas em um modelo de qualidade não se relacionam corretamente com outras medidas que definem uma característica de qualidade, o valor agregado deixa de representar corretamente essa característica e a avaliação de qualidade retornará valores incorretos. Nesse caso, as métricas devem ser reagrupadas corretamente em características de acordo com suas relações. Um estudo das correlações entre as métricas dos modelos de qualidade FOSS deverá indicar quais métricas são mais importantes (ou não são importantes) para definir cada característica de qualidade e se é necessário reagrupar métricas em diferentes características para melhor refletir suas correlações.

Etapa 2: Aplicação e Implementação de Técnicas de Visualização

O objetivo principal desta etapa é identificar técnicas de visualização que sejam apropriadas para a análise de dados provenientes de medidas de qualidade FOSS. Os principais requisitos são: (i) análise conjunta de diferentes fatores como código, testes e comunidade; e (ii) suporte à visualização de múltiplos indicadores de qualidade. As técnicas identificadas no Capítulo 3 e o trabalho de Martins et al. (2012) (considerados candidatos) deverão ser avaliados quanto à aplicabilidade neste contexto e os resultados desta avaliação deverão ser utilizados na melhoria das técnicas para a definição final de um processo de análise visual de qualidade FOSS.

A seguir são relacionadas algumas questões de pesquisa que deverão guiar o trabalho nesta etapa:

1. Quais técnicas de visualização são mais adequadas para os dados obtidos a partir de medidas de qualidade de FOSS?
2. A utilização dessas técnicas torna o processo de avaliação de qualidade de FOSS mais eficaz e/ou eficiente? Se sim, quais são os passos necessários para realizar essa análise visual da melhor forma possível?
3. É possível melhorar essas técnicas para que gerem melhores resultados quando aplicadas especificamente no contexto da avaliação de qualidade FOSS? Se sim, como?

Etapa 3: Melhoria Colaborativa de Qualidade FOSS

Considerando os resultados das etapas anteriores, ou seja, a análise dos atributos de qualidade FOSS e as técnicas de visualização específicas para sua avaliação, a etapa final do trabalho consistirá na definição de um processo para guiar não só a avaliação mas também a melhoria colaborativa de qualidade FOSS. Dessa forma, os resultados do trabalho beneficiarão não só avaliadores externos, interessados em comparar projetos FOSS para aquisição e integração, mas também os times de desenvolvimento de organizações que apoiam FOSS e a comunidade em geral, que poderão utilizar o método para monitorar constantemente a qualidade de seus produtos e tomar decisões informadas quanto às melhorias necessárias.

O processo deverá ser baseado na ColabSPI (Malheiros, 2010), uma estratégia colaborativa e distribuída para a melhoria de processos de software em equipes heterogêneas e geograficamente dispersas. A ColabSPI fornece um referencial com foco nos fatores principais de influência em atividades de melhoria de processo em cenários de desenvolvimento distribuído (como o desenvolvimento FOSS): coordenação, comunicação, grau de participação e motivação. Entre os princípios que norteiam a aplicação da ColabSPI estão a evolução iterativa do processo, a participação dos usuários do processo nessa evolução e a priorização à comunicação aberta e transparente. Também são identificados alguns requisitos principais, como a disponibilidade de ferramentas de comunicação cooperativas, acesso à informação por meio de um ponto de acesso único e definição de diretrizes sobre como contribuir.

Com base nessas observações, os principais objetivos desta etapa são:

- Estabelecer um processo colaborativo de avaliação, monitoramento e melhoria de qualidade FOSS com base em: (i) técnicas de visualização implementadas na etapa anterior; (ii) as melhores práticas identificadas na Etapa 1 a partir dos métodos de avaliação existentes; e (iii) a abordagem ColabSPI.
- Implementar uma plataforma colaborativa de avaliação de qualidade de FOSS que dê suporte completo ao processo estabelecido para incentivar a participação da comunidade em sua validação, utilização e evolução contínua.

4.4 Metodologia

Esta pesquisa pode ser classificada como aplicada, pois objetiva gerar conhecimentos para aplicação prática, sendo utilizadas abordagens de pesquisa qualitativa e quantitativa de acordo com a disciplina de Engenharia de Software Experimental (ESE) para responder

às questões de pesquisa e alcançar os objetivos finais definidos. Três tipos principais de estudos devem ser realizados em um trabalho de pesquisa em ESE (Sjoberg et al., 2007; Wohlin et al., 2000):

- **Survey:** um *survey* ou pesquisa de opinião é um estudo realizado em retrospectiva que investiga o entendimento sobre a forma que a população reage a um determinado método, ferramenta ou técnica. Neste trabalho, *surveys* serão aplicados para: (i) avaliar quais fatores de projetos FOSS são percebidos pela comunidade como indicadores de qualidade; e (ii) analisar a impressão dos usuários sobre o impacto do uso das técnicas de visualização no processo de avaliação de qualidade FOSS.
- **Estudo de Caso:** um estudo de caso consiste em uma pesquisa de um fenômeno contemporâneo em um contexto real, a partir de observação. A aplicação das técnicas de visualização e o processo de avaliação de qualidade proposto serão aplicados como estudo de caso em projetos de desenvolvimento reais nos quais não seja possível o controle rígido das variáveis.
- **Experimento Controlado:** o experimento controlado, segundo (Wohlin et al., 2000), caracteriza-se por um controle rígido das variáveis do ambiente para o estudo de uma relação causa-feito, devendo seguir um processo bem definido. Durante o trabalho serão projetados e realizados experimentos controlados para testar as hipóteses sobre a utilização de técnicas de visualização e o processo de avaliação de qualidade proposto.

Nos três casos, recursos e técnicas estatísticas serão utilizados para a avaliação dos resultados, que serão obtidos a partir de experiências de usuários com as visualizações geradas (qualitativo) e extração automática de indicativos dos resultados a partir das ferramentas (quantitativo).

Para a condução da pesquisa será adotada a metodologia proposta por (Mafra et al., 2006), uma extensão do trabalho de (Shull et al., 2001), onde é definida uma metodologia com objetivo de avaliar experimentalmente o grau de maturidade (limitações, pontos fortes) de uma tecnologia antes de sua transferência para a indústria. A extensão proposta por (Mafra et al., 2006) é dividida em duas fases: Definição Inicial da Tecnologia (Extensão – em destaque na Figura 1) e Refinamento da Tecnologia (Shull et al., 2001).

O foco da fase de Definição Inicial da Tecnologia é identificar evidências disponíveis na literatura sobre a tecnologia e desta maneira minimizar dificuldades e incertezas no processo de definição de uma nova tecnologia. Já a fase de Refinamento da Tecnologia é composta por quatro etapas: estudo de viabilidade, estudo de observação, estudo de caso (ciclo de vida) e estudo de caso (indústria).

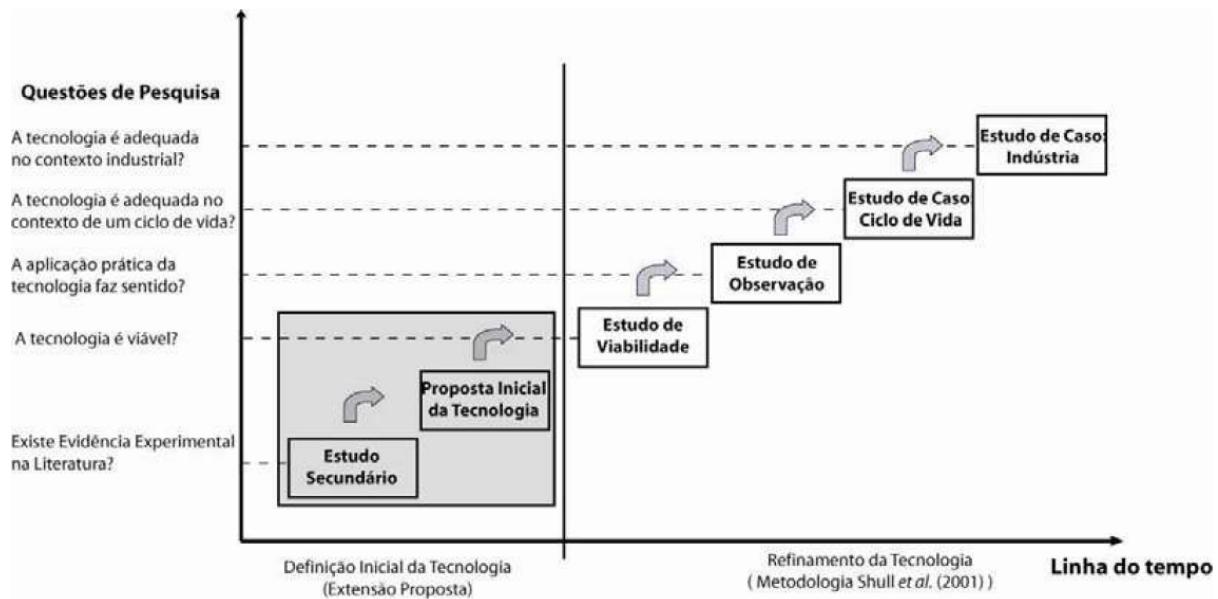


Figura 4.1: Metodologia experimental proposta por Mafra et al. (2006)

4.5 Ferramentas e Dados

As seguintes ferramentas e dados serão utilizados durante o desenvolvimento do trabalho:

Plataformas e Ferramentas de Extração de Métricas: para avaliação do modelo OMM, a plataforma Qualipso oferece diversas ferramentas de medição e avaliação de qualidade, sendo possível integrar novas ferramentas à plataforma através de extratores, caso necessário. As seguintes ferramentas já estão integradas à plataforma²: MacXim e Kali-bro (métricas de código), JaBUTi Service (métricas de teste) e StatCVS/SVN (métricas de repositório). Outras plataformas livres de extração de métricas também associadas a modelos de qualidade são: Melquiades, construída no contexto do projeto FLOSSMetrics para dar apoio ao modelo QualOSS; e Alitheia Core, construída com o objetivo de apoiar o modelo SQO-OSS³.

Dados de Projetos Reais: como o trabalho trata da avaliação de projetos FOSS, não são previstos problemas no acesso a dados de projetos reais para os estudos experimentais propostos. Em alguns casos, como nos projetos FLOSSmole e FLOSSmetrics⁴, esses dados já estão disponíveis em formatos adequados para análise.

Plataforma de Visualização: inicialmente será utilizada a plataforma VisPipeline para o desenvolvimento das técnicas de visualização; com uma arquitetura modular, ela permite a implementação de novas técnicas com a reutilização de módulos já existentes

²http://www.qualipso.org/trustworthy_results (último acesso em 11/10/2012)

³<http://melquiades.flossmetrics.org/>; <http://www.sqo-oss.org/> (último acesso em 11/10/2012)

⁴<http://flossmole.org>; <http://flossmetrics.org> (último acesso em 14/10/2012)

com a criação de aplicações baseadas em *pipelines*. Um *pipeline* é composto de componentes que realizam operações em objetos de entrada e geram objetos de saída que, por sua vez, são repassados a outros componentes. Essa interação gera, no final, a visualização desejada. A vantagem dessa plataforma é a possibilidade de reutilizar componentes existentes, que já realizam tarefas comuns e necessárias a um grande conjunto de *pipelines*, e implementar novos componentes específicos para cada contexto de aplicação. Na Etapa 3 será então estudada a integração das visualizações geradas à uma plataforma colaborativa de avaliação de qualidade.

4.6 Atividades e Cronograma da Pesquisa

Para a concretização do objetivo deste trabalho e o cumprimento das exigências para obtenção do título de Doutor em Ciências de Computação e Matemática Computacional do ICMC/USP devem ser realizadas várias atividades, agrupadas em oito semestres e divididas em *Atividades Requeridas pelo Programa* (P) – exigências do Programa de Doutorado em Ciências de Computação e Matemática Computacional do ICMC/USP para obtenção do título de Doutor – e *Atividades Específicas ao Trabalho Proposto* (T) – atividades fundamentais relacionadas à execução dos objetivos deste trabalho. Na Tabela 4.1 é mostrado o cronograma dessas atividades.

- (a) *Créditos (P)*: obtenção de 36 créditos em disciplinas de pós-graduação, sendo aproximadamente três disciplinas; duas delas devem constar do núcleo básico de disciplinas da Computação.
- (b) *Revisão Bibliográfica (T)*: levantamento de dados existentes sobre modelos de qualidade de software livre e uso de técnicas de visualização de informação para avaliação dessa qualidade, identificando iniciativas existentes e problemas importantes ainda em aberto na área.
- (c) *Preparação de Dados (T)*: identificação das ferramentas e plataformas de extração de métricas de qualidade a serem utilizadas no trabalho e obtenção de dados de projetos reais para experimentação.
- (d) *Qualificação (P)*: elaboração da presente monografia de qualificação.
- (e) *Exame de proficiência na Língua Inglesa (P)*.
- (f) *Programa Sanduíche (T)*: está programado o intercâmbio do aluno com a Universidade de Groningen (Holanda), no grupo de pesquisa *Scientific Visualization and Computer Graphics* (SVCG) sob a supervisão do Prof. Alexandru Telea.

4.6. ATIVIDADES E CRONOGRAMA DA PESQUISA

- (g) *Desenvolvimento 1 (T)*: avaliação experimental dos modelos e métricas de qualidade e maturidade encontrados na literatura. Como resultado desse passo deverão ser identificadas as métricas mais importantes na avaliação de qualidade FOSS.
- (h) *Desenvolvimento 2 (T)*: implementação e avaliação de novas técnicas de visualização específicas para apoiar a avaliação de qualidade de FOSS com base nas métricas identificadas na etapa anterior. Como resultado desse passo deverão ser identificados os pontos fortes e fracos das técnicas implementadas.
- (i) *Desenvolvimento 3 (T)*: implementação de melhorias nas técnicas de visualização, de acordo com resultados do passo anterior, e proposta e avaliação de um método colaborativo de melhoria de qualidade FOSS com base nas novas técnicas de visualização implementadas. Como resultado desse passo deverão ser identificados os pontos fortes e fracos do método proposto.
- (j) *Desenvolvimento 4 (T)*: revisão do método com base nos resultados do passo anterior e inclusão em um ambiente de desenvolvimento colaborativo, incentivando a criação de comunidades de melhoria de qualidade FOSS.
- (k) *Escrita da Tese (P)*.
- (l) *Escrita de artigos (P)*.

Tabela 4.1: Cronograma de atividades

	2011		2012		2013		2014	
	1º Semestre	2º Semestre	3º Semestre	4º Semestre	5º Semestre	6º Semestre	7º Semestre	8º Semestre
(a)								
(b)								
(c)								
(d)								
(e)								
(f)								
(g)								
(h)								
(i)								
(j)								
(k)								
(l)								

Bibliografia

Acuna, S. T.; Castro, J. W.; Dieste, O.; Juristo, N. A systematic mapping study on the open source software development process. In: *16th International Conference on Evaluation Assessment in Software Engineering (EASE)*, 2012, p. 42–46.

Atos Origin Method for qualification and selection of open source software (QSOS). Version 1.6, 2006.

Disponível em http://www.qsos.org/?page_id=3

Benkler, Y. *The wealth of networks: How social production transforms markets and freedom*. Yale University Press, 528 p., 2006.

del Bianco, V.; Chinosi, M.; Lavazza, L.; Morasca, S.; Taibi, D. *How european software industry perceives OSS trustworthiness and what are the specific criteria to establish trust in OSS*. Deliverable D5.1.1, Qualipso, 2008.

Disponível em <http://www.qualipso.org/node/45>

Boehm, B. W.; Brown, J. R.; Lipow, M. Quantitative evaluation of software quality. In: *Proceedings of the 2nd International Conference on Software Engineering*, ICSE '76, Los Alamitos, CA, USA: IEEE Computer Society Press, 1976, p. 592–605.

Burkhard, R.; Spescha, G.; Meier, M. Aha!: How to visualize strategies with complementary visualizations. In: *Proceedings of the International Conference on “Visualising and Presenting Indicator Systems”*, Federal Statistical Office, Neuchâtel, Swiss, 2005.

Burnstein, I. *Practical software testing: a process-oriented approach*. Springer Professional Computing, 732 p., 2002.

Card, S. K.; Mackinlay, J. D.; Shneiderman, B., eds. *Readings in information visualization: Using vision to think*. Morgan Kaufmann Publishers Inc., 712 p., 1999.

- Carneiro, G. d. F.; Silva, M.; Mara, L.; Figueiredo, E.; Sant'Anna, C.; Garcia, A.; Mendonça, M. Identifying code smells with multiple concern views. In: *Proceedings of the 24th Brazilian Symposium on Software Engineering*, SBES '10, Washington, DC, USA: IEEE Computer Society, 2010, p. 128–137.
- Caudwell, A. H. Gource: visualizing software version control history. In: *Proceedings of the ACM International Conference Companion on Object Oriented Programming Systems, Languages and Applications*, SPLASH '10, New York, NY, USA: ACM, 2010, p. 73–74.
- Chrissis, M. B.; Konrad, M.; Shrum, S. *CMMI: Guidelines for process integration and product improvement*. SEI Series in Software Engineering, 2 ed. Addison-Wesley Longman, 704 p., 2006.
- Cornelissen, B.; Zaidman, A.; van Deursen, A.; Moonen, L.; Koschke, R. A systematic survey of program comprehension through dynamic analysis. *IEEE Transactions on Software Engineering*, v. 35, n. 5, p. 684–702, 2009.
- Crespo, A. N.; Jino, M.; Argollo, M.; Bueno, P. M. S.; Barros, C. P. *Modelo de processo genérico de teste de software*. Relatório Técnico, Centro de Tecnologia da Informação Renato Archer – CTI / Ministério da Ciência e Tecnologia – MCT, 2010.
- Crowston, K.; Howison, J.; Annabi, H. Information systems success in free and open source software development: Theory and measures. *Software Process: Improvement and Practice*, v. 11, n. 2, p. 123–148, 2006.
- Crowston, K.; Wei, K.; Howison, J.; Wiggins, A. Free/libre open source software development: What we know and what we do not know. *ACM Computing Surveys*, v. 44, n. 2, p. 7:1–7:35, 2012.
- D'Ambros, M.; Lanza, M.; Pinzger, M. “A Bug’s Life”: Visualizing a bug database. In: *Proceedings of the 4th IEEE International Workshop on Visualizing Software for Understanding and Analysis*, VISSOFT, Banff, Alberta, Canada: IEEE Computer Society, 2007, p. 113–120.
- Deissenboeck, F.; Heinemann, L.; Herrmannsdoerfer, M.; Lochmann, K.; Wagner, S. The Quamoco tool chain for quality modeling and assessment. In: *Proceedings of the 33rd International Conference on Software Engineering*, ICSE '11, New York, NY, USA: ACM, 2011, p. 1007–1009.
- Delamaro, M.; Maldonado, J. C.; Jino, M. *Introdução ao teste de software*. Elsevier, 408 p., 2007.

- Deprez, J.-C.; Alexandre, S. Comparing assessment methodologies for free/open source software: OpenBRR and QSOS. In: *Proceedings of the 9th international conference on Product-Focused Software Process Improvement*, PROFES '08, Berlin, Heidelberg: Springer-Verlag, 2008, p. 189–203.
- Deprez, J.-C.; Haaland, K.; Kamseu, F. *QualOSS methodology and QualOSS assessment methods*. Deliverable D4.1, CETIC, 2008.
- Diehl, S. *Software visualization: Visualizing the structure, behavior and evolution of software*. Springer-Verlag, 187 p., 2007.
- Dromey, R. G. A model for software product quality. *IEEE Transactions on Software Engineering*, v. 21, n. 2, p. 146–162, 1995.
- Duijnhouwer, F.-W.; Widdows, C. *Open source maturity model*. Expert letter, Capgemini, 2003.
- Erdogmus, H. A process that is not. *IEEE Software*, v. 26, n. 6, p. 4–7, 2009.
- Forrester Consulting *Open source paves the way for the next generation of enterprise IT*. Relatório Técnico, Forrester Research, 2008.
- Fuggetta, A. Software process: a roadmap. In: *Proceedings of the Conference on The Future of Software Engineering*, ICSE '00, New York, NY, USA: ACM, 2000, p. 25–34.
- Gall, H.; Hajek, K.; Jazayeri, M. Detection of logical coupling based on product release history. In: *Proceedings of the International Conference on Software Maintenance*, ICSM '98, Washington, DC, USA: IEEE Computer Society, 1998, p. 190.
- Ghosh, R. A.; Glott, R.; Krieger, B.; Robles, G. *Free/libre and open source software: Survey and study*. Deliverable D18, International Institute of Infonomics – University of Maastricht, 2002.
- Gilbert, E.; Karahalios, K. CodeSaw: A social visualization of distributed software development. In: Baranauskas, C.; Palanque, P.; Abascal, J.; Barbosa, S., eds. *Human-Computer Interaction – INTERACT 2007*, v. 4663 de *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, p. 303–316, 2007.
- Grady, R. B.; Caswell, D. L. *Software metrics: establishing a company-wide program*. Prentice-Hall, 275 p., 1987.
- Groven, A.-K.; Haaland, K.; Glott, R.; Tannenberg, A. Security measurements within the framework of quality assessment models for free/libre open source software. In:

- Proceedings of the Fourth European Conference on Software Architecture: Companion Volume*, ECSA '10, New York, NY, USA: ACM, 2010, p. 229–235.
- Gutwin, C.; Penner, R.; Schneider, K. Group awareness in distributed software development. In: *Proceedings of the 2004 ACM conference on Computer Supported Cooperative Work*, CSCW '04, New York, NY, USA: ACM, 2004, p. 72–81.
- Heer, J.; Shneiderman, B. Interactive dynamics for visual analysis. *Communications of the ACM*, v. 55, n. 4, p. 45–54, 2012.
- Hohn, E. *KITest: Um arcabouço de conhecimento e melhoria de processo de teste*. Tese de Doutoramento, ICMC-USP, 2011.
- Holten, D. Hierarchical Edge Bundles: Visualization of adjacency relations in hierarchical data. *IEEE Transactions on Visualization and Computer Graphics*, v. 12, n. 5, p. 741–748, 2006.
- IEEE *Software and system test documentation*. Standard 829, IEEE, 2008.
- ISO/IEC *Information technology – software product evaluation*. Standard 14598, ISO/IEC, 1998.
- ISO/IEC *Software engineering – product quality*. Standard 9126, ISO/IEC, 2001.
- ISO/IEC *Systems and software engineering – software life cycle processes*. Standard 12207, ISO/IEC, 2008.
- ISO/IEC *Information technology – process assessment*. Standard 15504, ISO/IEC, 2012.
- Izquierdo-Cortazar, D.; Gonzalez-Barahona, J. M.; Duenas, S.; Robles, G. Towards automated quality models for software development communities: The QualOSS and FLOSSMetrics case. In: *Proceedings of the Seventh International Conference on the Quality of Information and Communications Technology*, QUATIC '10, Washington, DC, USA: IEEE Computer Society, 2010, p. 364–369.
- Jones, J. A.; Harrold, M. J.; Stasko, J. Visualization of test information to assist fault localization. In: *Proceedings of the 24th International Conference on Software Engineering*, ICSE '02, New York, NY, USA: ACM, 2002, p. 467–477.
- Jung, H.-W.; Kim, S.-G.; Chung, C.-S. Measuring software product quality: a survey of ISO/IEC 9126. *IEEE Software*, v. 21, n. 5, p. 88–92, 2004.

- Keim, D. A. Information visualization and visual data mining. *IEEE Transactions on Visualization and Computer Graphics*, v. 8, n. 1, p. 1–8, 2002.
- Kon, F.; Meirelles, P.; Lago, N.; Terceiro, A.; Chavez, C.; Mendonça, M. Free and open source software development and research: Opportunities for software engineering. In: *Proceedings of the 25th Brazilian Symposium on Software Engineering*, SBES '11, Washington, DC, USA: IEEE Computer Society, 2011, p. 82–91.
- Lanza, M.; Ducasse., S. Understanding software evolution using a combination of software visualization and software metrics. In: *Proceedings of Languages et Modeles a Objets*, LMO, Hermes Publications, 2002, p. 135–149.
- LaToza, T. D.; Venolia, G.; DeLine, R. Maintaining mental models: a study of developer work habits. In: *Proceedings of the 28th International Conference on Software Engineering*, ICSE '06, New York, NY, USA: ACM, 2006, p. 492–501.
- Lee, S.-Y. T.; Kim, H.-W.; Gupta, S. Measuring open source software success. *Omega*, v. 37, n. 2, p. 426 – 438, 2009.
- Lemos, O. A. L.; Vincenzi, A. M. R.; Maldonado, J. C.; Masiero, P. C. Control and data flow structural testing criteria for aspect-oriented programs. *Journal of Systems and Software*, v. 80, n. 6, p. 862–882, 2007.
- Lewis, W. E. *Software testing and continuous quality improvement*. 3 ed. Auerbach Publications, 688 p., 2008.
- Mafra, S. N.; Barcelos, R. F.; Travassos, G. H. Aplicando uma metodologia baseada em evidência na definição de novas tecnologias de software. In: *Anais do XX Simpósio Brasileiro de Engenharia de Software*, SBES, Florianópolis: SBC, 2006, p. 239–254.
- Malheiros, V. *Uma contribuição para a melhoria colaborativa e distribuída de processos de software*. Tese de Doutoramento, ICMC-USP, 2010.
- Marinescu, R.; Ganea, G.; Verebi, I. InCode: Continuous quality assessment and improvement. In: *Proceedings of the 14th European Conference on Software Maintenance and Reengineering*, CSMR '10, Washington, DC, USA: IEEE Computer Society, 2010, p. 274–275.
- Martin, K.; Hoffman, B. An open source approach to developing software in a small organization. *IEEE Software*, v. 24, n. 1, p. 46–53, 2007.

BIBLIOGRAFIA

- Martins, R. M.; Andery, G. F.; Heberle, H.; Paulovich, F. V.; Lopes, A. A.; Pedrini, H.; Minghim, R. Multidimensional projections for visual analysis of social networks. *Journal of Computer Science and Technology*, v. 27, n. 4, p. 791–810, 2012.
- McCall, J. A.; Richards, P. K.; Walters, G. F. *Factors in software quality*. Relatório Técnico ADA049014, General Electric CO, 1977.
- McDonnell, K. T.; Mueller, K. Illustrative parallel coordinates. *Computer Graphics Forum*, v. 27, n. 3, p. 1031–1038, 2008.
- Meirelles, P.; Santos Jr., C.; Miranda, J.; Kon, F.; Terceiro, A.; Chavez, C. A study of the relationships between source code metrics and attractiveness in free software projects. In: *Proceedings of the 24th Brazilian Symposium on Software Engineering (SBES)*, Washington, DC, USA: IEEE Computer Society, 2010, p. 11–20.
- Mens, T. On the complexity of software systems. *IEEE Computer*, v. 45, n. 8, p. 79–81, 2012.
- Midha, V.; Palvia, P. Factors affecting the success of open source software. *Journal of Systems and Software*, v. 85, n. 4, p. 895–905, 2012.
- Mills, E. E. *Software metrics*. Curriculum Module CMU/SEI-88-CM-012, Software Engineering Institute (SEI), 1988.
- Modak, S.; Singh, B. Building a robust linux kernel piggybacking the Linux Test Project. In: *Proceedings of the Linux Symposium*, OLS, Ottawa, Canada, 2008, p. 91–100.
- Modak, S.; Singh, B. Putting LTP to test: Validating both the linux kernel and test-cases. In: *Proceedings of the Linux Symposium*, OLS, Ottawa, Canada, 2009, p. 209–220.
- Morasca, S.; Taibi, D.; Tosi, D. Towards certifying the testing process of open-source software: New challenges or old methodologies? In: *Proceedings of the 2009 ICSE Workshop on Emerging Trends in Free/Libre/Open Source Software Research and Development*, FLOSS ’09, Washington, DC, USA: IEEE Computer Society, 2009, p. 25–30.
- Morasca, S.; Taibi, D.; Tosi, D. OSS-TMM: Guidelines for improving the testing process of open source software. *International Journal of Open Source Software and Processes*, v. 3, n. 2, p. 1–22, 2011.
- Morell, L. J. A theory of fault-based testing. *IEEE Transactions on Software Engineering*, v. 16, n. 8, p. 844–857, 1990.

- Ogawa, M.; Ma, K.-L. StarGate: A unified, interactive visualization of software projects. In: *Proceedings of the IEEE VGTC Pacific Visualization Symposium*, PacificVis, Kyoto, Japan: IEEE, 2008, p. 191–198.
- Ogawa, M.; Ma, K.-L. code_swarm: A design study in organic software visualization. *IEEE Transactions on Visualization and Computer Graphics*, v. 15, n. 6, p. 1097–1104, 2009.
- Ogawa, M.; Ma, K.-L.; Bird, C.; Devanbu, P.; Gourley, A. Visualizing social interaction in open source software projects. In: *Proceedings of the 6th International Asia-Pacific Symposium on Visualization*, APVIS '07, Sydney, Australia: IEEE, 2007, p. 25–32.
- Oliveira, M. C. F.; Levkowitz, H. From visual data exploration to visual data mining: A survey. *IEEE Transactions on Visualization and Computer Graphics*, v. 9, n. 3, p. 378–394, 2003.
- OpenBRR *Business readiness rating for open source: A proposed open standard to facilitate assessment and adoption of open source software*. Relatório Técnico BRR 2005 – RFC 1, openbrr.org, 2005.
- Paulovich, F. V.; Oliveira, M. C. F.; Minghim, R. The projection explorer: A flexible tool for projection-based multidimensional visualization. In: *Proceedings of the Brazilian Symposium on Computer Graphics and Image Processing (SIBGRAPI)*, Washington, DC, USA: IEEE Computer Society, 2007, p. 27–36.
- Petrinja, E.; Nambakam, R.; Sillitti, A. Introducing the OpenSource Maturity Model. In: *Proceedings of the 2009 ICSE Workshop on Emerging Trends in Free/Libre/Open Source Software Research and Development*, FLOSS '09, Washington, DC, USA: IEEE Computer Society, 2009, p. 37–41.
- Petrinja, E.; Sillitti, A.; Succi, G. Comparing OpenBRR, QSOS, and OMM assessment models. In: Ågerfalk, P.; Boldyreff, C.; González-Barahona, J. M.; Madey, G. R.; Noll, J., eds. *Open Source Software: New Horizons*, v. 319 de *IFIP Advances in Information and Communication Technology*, Springer Berlin Heidelberg, p. 224–238, 2010.
- Pfleeger, S. L.; Verhoef, C.; van Vliet, H. Analyzing the evolution of large-scale software. *Journal of Software Maintenance and Evolution: Research and Practice*, v. 17, n. 1, p. 1–2, 2005.
- Potdar, V.; Chang, E. Open source and closed source software development methodologies. In: *Collaboration, Conflict and Control: The 4th Workshop on Open Source*

- Software Engineering, ICSE '04, Washington, DC, USA: IEEE Computer Society, 2004, p. 105–109.
- Pressman, R. *Software engineering: A practitioner's approach.* 6 ed. McGraw-Hill, 2004.
- Rao, R.; Card, S. K. The table lens: merging graphical and symbolic representations in an interactive focus+context visualization for tabular information. In: *Conference companion on Human factors in computing systems*, CHI '94, New York, NY, USA: ACM, 1994, p. 222.
- Reniers, D.; Voinea, L.; Telea, A. Visual exploration of program structure, dependencies and metrics with SolidSX. In: *Proceedings of the 6th IEEE International Workshop on Visualizing Software for Understanding and Analysis*, VISSOFT 2011, Williamsburg, VA, USA: IEEE, 2011, p. 1–4.
- Riehle, D. The economic motivation of open source software: Stakeholder perspectives. *IEEE Computer*, v. 40, n. 4, p. 25–32, 2007.
- Rincon, A. M.; Vincenzi, A. M. R.; Chaim, M. L.; Delamaro, M. E.; Maldonado, J. C. Avaliação empírica de conjuntos de testes de projetos de software livre. In: *Proceedings of the 5th Brazilian Workshop on Systematic and Automated Software Testing*, SAST '11, 2011.
- Rocha, A.; Rincon, A. M.; Delamaro, M. E.; Maldonado, J. C.; Vincenzi, A. M. R. Avaliando a qualidade de conjuntos de teste de software de código aberto por meio de critérios de teste estruturais. In: *Anais do Workshop de Software Livre / Fórum Internacional de Software Livre*, WSL / FISL, 2010.
- da Rocha, A. R. C.; Maldonado, J. C.; Weber, K. C. *Qualidade de software: Teoria e prática.* Prentice Hall, 303 p., 2001.
- Samoladas, I.; Gousios, G.; Spinellis, D.; Stamelos, I. The SQO-OSS quality model: Measurement based open source software evaluation. In: Russo, B.; Damiani, E.; Hissam, S.; Lundell, B.; Succi, G., eds. *Open Source Development, Communities and Quality*, v. 275 de *IFIP International Federation for Information Processing*, Springer Boston, p. 237–248, 2008.
- Santana, F. W.; Oliva, G. A.; de Souza, C. R. B.; Gerosa, M. XFlow: An extensible tool for empirical analysis of software systems evolution. In: *Proceedings of the VIII Experimental Software Engineering Latin American Workshop*, ESELAW, 2010.

- Sarma, A.; Maccherone, L.; Wagstrom, P.; Herbsleb, J. Tesseract: Interactive visual exploration of socio-technical relationships in software development. In: *Proceedings of the 31st International Conference on Software Engineering*, ICSE '09, Washington, DC, USA: IEEE Computer Society, 2009, p. 23–33.
- Sato, D.; Goldman, A.; Kon, F. Tracking the evolution of object-oriented quality metrics on agile projects. In: *Proceedings of the 8th International Conference on Agile Processes in Software Engineering and Extreme Programming*, XP'07, Berlin, Heidelberg: Springer-Verlag, 2007, p. 84–92.
- Scacchi, W. The future of research in free/open source software development. In: *Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research*, FoSER '10, New York, NY, USA: ACM, 2010, p. 315–320.
- Schroeder, W. J.; Avila, L. S.; Hoffman, W. Visualizing with VTK: A tutorial. *IEEE Computer Graphics and Applications*, v. 20, n. 5, p. 20–27, 2000.
- Shollo, A.; Pandazo, K. *Improving presentations of software metrics indicators using visualization techniques*. Relatório Técnico 2008:020, University of Göteborg, Sweden, 2008.
- Shull, F.; Carver, J.; Travassos, G. H. An empirical methodology for introducing software processes. In: *Proceedings of the 8th European Software Engineering Conference held jointly with 9th ACM SIGSOFT international symposium on Foundations of Software Engineering*, ESEC/FSE-9, New York, NY, USA: ACM, 2001, p. 288–296.
- Sjoberg, D. I. K.; Dyba, T.; Jorgensen, M. The future of empirical methods in software engineering research. In: *Future of Software Engineering*, FOSE '07, Washington, DC, USA: IEEE Computer Society, 2007, p. 358–378.
- Soto, M.; Ciolkowski, M. The QualOSS open source assessment model measuring the performance of open source communities. In: *Proceedings of the 3rd International Symposium on Empirical Software Engineering and Measurement*, ESEM '09, Washington, DC, USA: IEEE Computer Society, 2009, p. 498–501.
- Spinellis, D.; Gousios, G.; Karakoidas, V.; Louridas, P.; Adams, P. J.; Samoladas, I.; Stamelos, I. Evaluating the quality of open source software. *Electronic Notes in Theoretical Computer Science*, v. 233, p. 5–28, 2009.
- Stallman, R. Viewpoint: Why “open source” misses the point of free software. *Communications of the ACM*, v. 52, n. 6, p. 31–33, 2009.

BIBLIOGRAFIA

- Tahir, A.; Chaim, M. L. *Definition of standard test approaches, test suites, and benchmarks of open source software.* Deliverable A5.D1.5.4, Qualipso, 2008.
Disponível em <http://www.qualipso.org/node/129>
- Taibi, D.; Lavazza, L.; Morasca, S. OpenBQR: a framework for the assessment of OSS. In: Feller, J.; Fitzgerald, B.; Scacchi, W.; Sillitti, A., eds. *Open Source Development, Adoption and Innovation*, v. 234 de *IFIP International Federation for Information Processing*, Springer Boston, p. 173–186, 2007.
- Telea, A. *Data visualization: Principles and practice.* A. K. Peters, Ltd., 460 p., 2007.
- Telea, A.; Voinea, L. Case study: Visual analytics in software product assessments. In: *Proceedings of the 5th IEEE International Workshop on Visualizing Software for Understanding and Analysis*, VISSOFT 2009, IEEE, 2009, p. 65–72.
- Thomas, J.; Cook, K., eds. *Illuminating the path: The research and developments agenda for visual analytics.* IEEE Computer Society Press, 2005.
- Tian, J. Quality-evaluation models and measurements. *IEEE Software*, v. 21, n. 3, p. 84–91, 2004.
- Tosi, D.; Tahir, A. How developers test their open source software products - a survey of well-known oss projects. In: *ICSOFT (2)*, 2010, p. 22–31.
- van Veenendaal, E., ed. *Test Maturity Model Integration (TMMi).* TMMi Foundation, 219 p., version 1.0, 2009.
- Voinea, L.; Telea, A. Visual querying and analysis of large software repositories. *Journal of Empirical Software Engineering*, v. 14, n. 3, p. 316–340, 2009.
- Wasserman, A. I.; Capra, E. Evaluating software engineering processes in commercial and community open source projects. In: *Proceedings of the 1st International Workshop on Emerging Trends in FLOSS Research and Development*, FLOSS ’07, Washington, DC, USA: IEEE Computer Society, 2007.
- Wasserman, S.; Faust, K. *Social network analysis: Methods and applications.* Structural Analysis in the Social Sciences. Cambridge University Press, 857 p., 1994.
- Wettel, R.; Lanza, M. CodeCity: 3D visualization of large-scale software. In: *Companion of the 30th International Conference on Software Engineering*, ICSE Companion ’08, New York, NY, USA: ACM, 2008, p. 921–922.

Wittmann, M.; Nambakam, R.; Ruffati, G.; Oltolina, S.; Petrinja, E.; Ortega, F. *CMM-like model for OSS.* Deliverable A6.D1.6.3, Qualipso, 2008.

Wohlin, C.; Runeson, P.; Höst, M.; Ohlsson, M. C.; Regnell, B.; Wesslén, A. *Experimentation in software engineering: an introduction.* International Series in Software Engineering. Norwell, MA, USA: Kluwer Academic Publishers, 228 p., 2000.

Zaidman, A.; Rompaey, B.; Deursen, A.; Demeyer, S. Studying the co-evolution of production and test code in open source and industrial developer test processes through repository mining. *Empirical Software Engineering*, v. 16, n. 3, p. 325–364, 2011.