Evolução de software livre baseada em métricas de qualidade: Um estudo de caso

Auri M. R. Vincenzi[1], Cássio L. Rodrigues[1], Igor R. Vieira[1]
Leonardo da Silva Sousa[1], Vinicius R. L. de Mendonça[1]
Jacson R. Barbosa[1], Mário E. P. Diaz[2]

[1]Instituto de Informática – Universidade Federal de Goiás (UFG)
Caixa Postal 74001-970 – Goiânia – GO – Brasil

{auri, cassio, igorvieira, leonardosousa, viniciusmendonca, jacson}@inf.ufg.br

[2]Instituto e Matemática e Estatística – Universidade Federal de Goiás (UFG)
Caixa Postal 74001-970 – Goiânia – GO – Brasil

mpiscoya@mat.ufg.br

Abstract. *Open source is developed in a decentralized and collaborative way, requiring a set of systematic activities in order to ensure quality. Open Source differs significantly from traditional proprietary software development model, especially by the free access to source code. The goal of this paper is to analyze the evolution of open source based on three different composite metrics of software quality: Quality Index, Total Quality and Technical Debt. For this paper we selected three distinct versions of five open source projects. Each version was evaluated with the Sonar platform, which combines static and dynamic analysis tools for evaluation of source code and generation of reports based on the collected metrics. The results are used to identify trends in the evolution of Open Source development, considering this specific universe of products, suggesting that Total Quality is more sensible de detect projects changes.*

Resumo. *O modelo de desenvolvimento de Open Source é descentralizado e colaborativo, exigindo um conjunto de atividades sistemáticas visando garantir a qualidade e evolução de seus produtos. O desenvolvimento de software livre difere significativamente do modelo tradicional de desenvolvimento de software proprietário, especialmente pelo livre acesso ao código fonte. O objetivo deste artigo é analisar a evolução de Open Source com base em três métricas compostas de qualidade de software: Índice de Qualidade (Quality Index), Qualidade Total (Total Quality) e Débito Técnico (Technical Debt). Para o estudo, foram selecionadas três versões distintas de um conjunto de cinco projetos de Software Livre. Cada versão foi avaliada com a plataforma Sonar, que combina ferramentas de análise estática e avaliação do código-fonte gerando resultados das métricas selecionadas para a análise do projeto. Os resultados são utilizados para identificar tendências na evolução desse conjunto, considerando este universo específico de produtos, sugerindo que a métrica Qualidade Total é mais sensível, a detectar alterações nos projetos.*

1. Introdução

Os projetos na comunidade de software livre têm como principal característica o livre acesso ao código fonte. O lançamento de uma nova versão é normalmente acompanhado da disponibilização do seu respectivo código fonte, possibilitando aos desenvolvedores estudar, aprimorar e sugerir alterações no produto. De maneira oposta, em geral, um usuário, ao adquirir a licença de um software proprietário, está obtendo apenas o direito de executá-lo em seu computador, não incluindo o direito de modificar, estudar ou compartilhar seu código fonte. Portanto, a liberdade dos chamados *Open Source Software* (OSS) revolucionou o desenvolvimento de software, pois antes o acesso ao código fonte de programas de terceiros, pelos profissionais da área, era mais restrito [Weber 2004].

O processo de desenvolvimento na comunidade de OSS é uma atividade contínua, caracterizado pela frequente disponibilização e evolução de versões. Essa propriedade foi chamada por Raymond [Raymond 1999] como "*release early, release often*". A evolução e manutenção dos projetos são asseguradas pela comunidade de desenvolvimento, constituída de desenvolvedores voluntários espalhados pelo planeta, que trabalham em módulos específicos do projeto pelo prazer de trabalhar em algo que gostam [Mishra et al. 2002].

Neste trabalho um conjunto de cinco OSS é avaliado utilizando a plataforma Sonar [Sonar 2013], com o objetivo de realizar um estudo exploratório para verificar como cada projeto evolui, bem como tendências comportamentais em relação às métricas selecionadas e quais os efeitos refletidos em sua qualidade. Para cada OSS foram selecionadas três versões distintas: uma inicial, uma intermediária e a última versão no momento da realização desse estudo.

Na próxima seção serão apresentados alguns trabalhos relacionados sobre evolução de OSS. Na Seção 3 são descritas as métricas utilizadas para realização do estudo. Na Seção 4 é apresentada a metodologia utilizada no trabalho, descrevendo as fases de preparação e execução do estudo de caso. Na Seção 5 é realizado um estudo de caso de cada OSS avaliado, iniciando com a descrição do projeto, a apresentação do valor das métricas computadas e a análise desses resultados. A Seção 6 utiliza os resultados individuais dos projetos com o objetivo de identificar uma tendência na evolução dos OSS. Por fim, na Seção 7 apresentam-se as considerações finais e possibilidades de trabalhos futuros.

2. Trabalhos Relacionados

O Estudo da Evolução de Software tem como um dos seus principais expoentes Meir M. Lehman [Lehman et al. 1997]. Ele é o criador das leis de evolução de software. Em seu trabalho, publicado em 1974, foram propostas três leis, mas posteriormente esta quantidade foi aumentada até que no ano de 1996, essas leis totalizaram oito. Posteriormente, surgiram novas pesquisas restringindo a aplicação dessas leis em outros ambientes de desenvolvimento. Entre esses ambientes está o OSS, objeto de estudo desse trabalho. Ainda não existe um consenso da validade de aplicação dessas leis nessa abordagem de desenvolvimento, mas estudos relevantes são realizados nesse campo de pesquisa [Johari & Kaur 2011].

Lehman et al. [Lehman et al. 1997] realizaram um estudo com três tipos de softwares: kernel do Linux, Kernel de Softwares Livres da família BSB e outros sistemas. Para estudo da evolução do software foram escolhidas versões anteriores e posteriores ao lançamento da versão 1.0, ao contrário dos estudos tradicionais que consideram apenas versões entregues aos clientes, pois os softwares comerciais são apenas disponibilizados quando estão de acordo com os requisitos desejados pelo cliente. Outra diferença está na utilização da medição da evolução do software em razão do tempo, Lehman et al. utilizaram uma abordagem diferente, considerando o tamanho do software em razão do número de lançamentos. Essa mudança de métrica foi justificada pelo processo de liberação semi-contínuo que caracteriza a maioria dos projetos OSS. Os sistemas apresentam uma taxa de crescimento superior ao regular.

Em outro trabalho foi realizado um estudo de caso do Apache Server. Esse sistema foi comparado com cinco softwares comerciais, sendo observado que os principais desenvolvedores da comunidade da Apache Group são mais eficientes quando comparados aos principais desenvolvedores de softwares comerciais estudados, pois esses realizam uma atividade voluntária e em tempo parcial [Mockus et al. 2000]. Além do estudo da evolução do sistema OSS, foi realizada uma pesquisa aprofundada sobre o funcionamento de sua comunidade de desenvolvimento e sua influência na evolução do sistema.

Godfrey e Tu estudaram o kernel Linux e o editor Vim [Godfrey & Tu 2001], referente ao primeiro foi observado que o crescimento do kernel Linux é constante mesmo quando o código fonte ultrapassa milhões de linhas de código e que mais da metade de sua composição total está relacionada a uma grande variedade de controladores (*drivers*). Por outro lado, a arquitetura do Vim passa por um processo de deterioração, pois a sua evolução está se centralizando em dois grandes arquivos fontes misc1 e misc2. A conclusão é que os OSS apresentam um desenvolvimento dinâmico possibilitando um crescimento superlinear por períodos prolongados.

Em relação a métricas, no trabalho de Bogoni e Ruiz [Bogoni 2008] foi desenvolvido um método para extração, organização e apresentação de métricas para Processo de Desenvolvimento de Software (PDS). O objetivo do trabalho era resgatar medições feitas em projetos passados sob diferentes modelos de PDS e programas de métricas, dessa maneira sendo possível construir um base sólida de informações desses projetos, considerando a evolução do próprio PDS e do conjunto de métricas analisadas. A principal contribuição obtida pelos autores foi viabilizar um melhor controles dos projetos de software e a qualidade de seus produtos através de medições presentes e passadas que foram mantidas em um repositório únicos da organização, permitindo que as próprias medições pudessem ser comparadas entre si.

Este trabalho explora parte desse conhecimento na avaliação de OSS, uma vez que também considera diferentes versões de um mesmo produto OSS durante a coleta e análise de dados. Entretanto, emprega-se um conjunto diferente de métricas para a análise visando não apenas tentar identificar uma tendência de evolução e/ou regressão dos OSS investigados, mas também contribuir para avaliar as próprias métricas em relação a sua consistência e sensibilidade na percepção que elas oferecem para detectar as alterações sofridas no projeto.

3. Métricas Utilizadas

A plataforma Sonar proporciona a avaliação da qualidade do código do software por intermédio de várias métricas. Para esse estudo foram selecionadas as seguintes métricas de qualidade: Índice de Qualidade, Qualidade Total e Débito Técnico.

3.1. Índice de Qualidade ($\mathcal{IQ}$)

É uma métrica que calcula a média global de qualidade e o fator de complexidade do método. O índice de qualidade global é baseado em quatro eixos: codificação (*coding*) (*co*), complexidade (*complexity*) (*cc*), cobertura (*coverage*) (*cv*) e estilo (*style*) (*st*). O $\mathcal{IQ}$ mede os 4 eixos ponderados de qualidade e combina-os para dar uma nota global entre 0 e 10 para o projeto. O cálculo dessa métrica segue a seguinte fórmula padrão [QualityIndex 2011]:

$$\mathcal{IQ} = 10 - (4,5 * co - 2,0 * cc - 2,0 * cv - 1,5 * st) \tag{1}$$

3.2. Qualidade Total ($\mathcal{QT}$)

É uma métrica que proporciona uma visão geral das medidas de qualidade do projeto, combinando quadro domínios – arquitetura (*architecture*) (*ar*), projeto (*design*) (*de*), código (*code*) (*co*) e teste (*test*) (*te*) – com o objetivo de calcular a "saúde" global e unificada da qualidade do projeto. Sua fórmula padrão é a seguinte [Total Quality 2013]:

$$\mathcal{QT} = 0,25 * ar + 0,25 * de + 0,25 * co + 0,25 * te \tag{2}$$

3.3. Débito Técnico ($\mathcal{DT}$)

É uma métrica que calcula a dívida técnica do projeto [Technical Debt 2011, Eisenberg 2012, Gat & Ebert 2012]. Primeiramente são calculados os valores dos seus eixos básicos: duplicação (*duplication*) (*du*), violação (*violations*) (*vi*), complexidade (*complexity*) (*cc*), cobertura (*coverage*) (*cv*), documentação (*documentation*) (*do*) e projeto (*design*) (*de*). Em seguida, essas métricas são somadas para fornecer uma medida global.

A versão 1.2.1 do plugin utilizado pela plataforma Sonar, usa a fórmula da equação (3) para calcular o custo para correção de cada componente da métrica e multiplicar esse resultado pelo preço padrão da dívida ($500 por dia), considerando o dia com 8 horas. Para mais informações sobre o cálculo pode ser consultado [Technical Debt 2011]:

$$\mathcal{DT} = (emhomens/dia) = ccd + ccv + cca + ccc + ctc + ccl \tag{3}$$

onde:

- ccd – custo para corrigir duplicatas;
- ccv – custo para corrigir violações;
- cca – custo para comentar API Pública;

- ccc – custo de corrigir complexidade descoberta;
- ctc – custo para trazer a complexidade abaixo do valor limite;
- ccl – custo de cortar ciclos com nível de pacote.

4. Metodologia

A investigação da evolução dos OSS proposta neste trabalho é baseada na análise do código fonte de cinco projetos e suas respectivas versões, utilizando ferramentas de análise estática a fim de obter métricas que possam subsidiar a avaliação da evolução desses sistemas.

Alguns dos trabalhos relacionados, apresentados na Seção 2, baseiam sua análise sobre a evolução de software OSS por meio de métricas quantitativas relacionadas ao crescimento do código fonte (número de linhas), quantidade de correções realizadas a cada versão ou quantidade de módulos adicionados ou alterados. Neste trabalho, busca-se utilizar métricas mais abrangentes, possibilitando, portanto, uma análise menos subjetiva sobre o crescimento e evolução dos sistemas selecionados. Para alcançar tais resultados, o seguinte conjunto de fases foi aplicado no estudo de caso:

- Seleção de um conjunto de projetos OSS, considerando informações inerentes ao projeto, tais como documentação e disponibilidade dos casos de testes, conforme descrito a seguir;
- Obtenção do código fonte de três versões distintas de cada software, baixados de seus respectivos repositórios. As versões obtidas correspondem à versão inicial do software (não necessariamente a primeira), uma versão intermediária e a última versão do software (versão atual no período deste trabalho);
- Cada versão dos OSS estudados foi migrada para o formato do Maven 2.0 [Maven 2013], para viabilizar a coleta das informações de cobertura de teste estrutural pela plataforma Sonar;
- Obtenção e análise das métricas estáticas e dinâmicas por meio da submissão dos projetos à plataforma Sonar, versão 3.2.1.

Os programas foram selecionados a partir de um grupo de programas OSS em preparaç ao para a condução de estudos experimentais. No momento da escrita deste trabalho cinco programas estavam preparados para a coleta de dados: Cobertura, Commons-IO, FindBugs, HttpUnit e JUnit. A dificuldade na seleção de qualquer programa é a exigência de que cada uma das três versões selecionadas esteja no padrão Maven ou pudesse ser migrada para esse padrão viabilizando a coleta de dados por parte da plataforma Sonar.

Para esse experimento, foram selecionadas três métricas entre as disponibilizadas pelo Sonar: Índice de Qualidade, Qualidade Total e Débito Técnico, direcionadas a área de governança/gerenciamento. Essas métricas foram escolhidas porque são compostas por um conjunto de outras métricas e oferecem uma visão mais abrangente da qualidade do projeto e, consequentemente, sua evolução. A análise individual de cada software é apresentada na seção seguinte.

5. Estudo de Caso

A seção a seguir apresenta os resultados encontrados ao submeter cada projeto escolhido à avaliação da plataforma Sonar. Cada projeto possui uma breve descrição,

uma tabela com as métricas de qualidade coletadas, bem como uma análise evidenciando o seu comportamento evolutivo, baseado nas três métricas de qualidade utilizadas nesse trabalho.

5.1. Cobertura

Cobertura é um OSS que calcula a porcentagem de código executado por meio dos testes, permitindo identificar áreas do software não são cobertas por eles [Cobertura 2006]. As medidas utilizadas para analisar a evolução do projeto Cobertura evidenciaram que ele teve decréscimo de qualidade segundo as métricas utilizadas. Tanto o Índice de Qualidade quanto a Qualidade Total decresceram. Além disso, o Débito Técnico aumentou de 10,0% para 62,5%, entre a primeira e terceira versões, representando um grande esforço necessário, em termos de custos e trabalho homens/dia, para corrigir imperfeições no código fonte (ver Tabela 1).

Tabela 1. As métricas coletadas nas versões do Cobertura

Métricas	Versão 1.0	Versão 1.5	Versão 1.9.4
Release	12.02.2005	05.08.2005	03.03.2010
Quality index (1-10)	8.0	7.3	2.4
Coding (0 - 4.5)	0.3	0.6	4.5
Complexity (0 - 2)	0.1	0.1	1.1
Coverage (0 - 2)	1.6	2.0	1.8
Style(0 -1.5)	0.0	0.0	0.0
Total Quality (%)	74.9	69.7	57.6
Architecture (%)	90.5	89.5	80.8
Design (%)	94.1	89.6	89.7
Test (%)	34.2	19.4	26.3
Code (%)	80.8	79.1	33.4
Technical Debt (ratio %)	10.0	13.1	62.5
Technical Debt ($)	13.710	15.441	526.440
Technical Debt man days	27	31	1.059

Observa-se que nas tabelas, além dos valores das métricas compostas aparecem também o detalhamento das métricas que as compõem. Por exemplo, após o valor de Índice de Qualidade (linha 3) as métricas codificação, complexidade, cobertura e estilo que são usadas no cálculo do $\mathcal{IQ}$. É interessante notar que a escolha da versão intermediária foi baseada na quantidade de versões lançadas entre a primeira e a última versão e não na versão lançada da data intermediária entre as versões.

5.2. Commons-IO

Commons-IO é uma biblioteca de utilitários para ajudar no desenvolvimento de funcionalidades de entrada e saída. Esses utilitários são classificados em 6 áreas: classe de utilitários, entrada, saída, fluxo, comparadores e monitores de arquivo [Commons-IO 2013]. Essa biblioteca ficou praticamente estável em relação ao Índice de Qualidade e à Qualidade Total. Entretanto, considerando as métricas relacionadas à Qualidade Total de forma individualizada, observa-se que a métrica arquitetura alterou sensivelmente. Ela teve uma variação de mais de 50% da primeira para a segunda versão analisada, passando de 100% para 45,5%, sofrendo uma pequena elevação na última versão, passando a 47,8%. O Débito Técnico apresentou crescimento entre as versões analisadas, sendo mais visível entre a segunda e a terceira versões. A Tabela 2 exibe os valores das métricas para o Commons-IO.

Tabela 2. As métricas coletadas nas versões do Commons-IO

Métricas	Versão 1.0	Versão 2.0	Versão 2.4
Release	08.05.2004	18.10.2010	12.06.2012
Quality index (1-10)	7.8	8.5	8.5
Coding (0-4.5)	0.1	0.1	0.1
Complexity (0-2)	0.1	0.1	0.1
Coverage (0-2)	1.4	1.3	1.3
Style (0-1.5)	0.0	0.0	0.0
Total Quality (%)	72.6	70.5	71.1
Architecture (%)	100	45.5	47.8
Design (%)	95.5	94.5	94.5
Test (%)	46.0	47.8	47.6
Code (%)	92.9	94.1	94.5
Technical Debt (ratio %)	2.8	12.7	12.3
Technical Debt ($)	5.422	26.322	28.743
Technical Debt man days	11	53	58

Levando-se em conta a evolução do software considerando a data de lançamento das versões, a versão 1.0 lançada em 08/05/2004, versão 2.0 lançada em 18/10/2010 e versão 5.4 lançada em 12/06/2012, em relação ao aumento da complexidade do código e o crescimento do projeto (em termos de quantidade de linhas de código e novos módulos adicionados), o Commons-IO não evolui rapidamente. Contudo, percebe-se que a equipe de desenvolvimento concentra esforços para aumentar a qualidade do software ou pelo menos garantir sua estabilidade.

5.3. FindBugs

FindBugs é uma ferramenta de análise estática para Java que procura por potenciais defeitos de codificação, avaliando que tipos de defeitos podem ser efetivamente detectados sem a necessidade de execução do código e com uma técnica relativamente simples [Ayewah et al. 2008]. É possível dizer que o Findbugs manteve-se estável em relação às medidas de qualidade. O Índice de Qualidade teve um leve oscilação entre a primeira e última versão analisada. A Qualidade Total apresentou aumento da primeira versão (1.2.1) para a segunda versão (1.3.7), tendo uma queda na última versão (2.0.1). Débito Técnico teve um comportamento semelhante, melhorando da primeira para a segunda versão e piorando na última versão. A Tabela 3 exibe as medidas coletadas, demonstrando a estabilidade do Findbugs [FindBugs 2012].

Analisando mais detalhadamente as medidas que compõem as três métricas globais observadas, o aumento do Qualidade Total, da primeira para a segunda versão, se deve ao número de casos de testes adicionados, em contrapartida, o decremento da segunda para a terceira versão está relacionado a diminuição das medidas Teste, Código e Arquitetura. Aqui percebe-se uma característica interessante em se tratando do aumento do número de casos de testes refletidos no aumento da qualidade do projeto: o número de casos de teste cresceram da primeira para a segunda versão, aumentando também o valor da métrica Teste, porém da segunda para a terceira versão o valor dessa mesma métrica diminuiu, mesmo aumentando o número do caso de testes. Frisando que a métrica Teste é calculada por meio da combinação majoritária da cobertura com o sucesso dos casos de testes. Assim sendo, um aumento no número de casos de teste sem melhoria na porcentagem desta métrica implica que os testes adicionados não executam novos trechos de código e sim aquelas partes já executada anteriormente.

Tabela 3. As métricas coletadas nas versões do FindBugs

Métricas	Versão 1.2.1	Versão 1.3.7	Versão 2.0.1
Release	31.05.2007	30.12.2008	13.07.2012
Quality index (1-10)	6.6	6.9	6.7
Coding (0-4.5)	1.3	1.2	1.2
Complexity (0-2)	0.1	0.1	0.1
Coverage (0-2)	2.0	1.8	2.0
Style (0-1.5)	0.0	0.0	0.0
Total Quality (%)	66.5	70.1	65.5
Architecture (%)	87.7	89.8	87.2
Design (%)	89.3	88.5	88.6
Test (%)	14.2	26.9	12.6
Code (%)	74.6	75.3	73.7
Technical Debt (ratio%)	16.7	14.1	17.0
Technical Debt ($)	420.061	499.061	698.406
Technical Debt man days	856	998	1.397

5.4. HTTPUnit

HTTPUnit é um OSS utilizado para simular o comportamento de um navegador web, incluindo o envio de formulários, JavaScript, redirecionamento automático de páginas e outros recursos [HttpUnit 2008]. Considerando os dados apresentados pela Tabela 4, observamos que o projeto evoluiu satisfatoriamente em relação ao Índice de Qualidade e ao Débito Técnico. O Qualidade Total foi a única métrica, entre as utilizadas, que decrementou em relação à qualidade. Tal comportamento pode ser justificado devido o fato de mesmo com o aumento do número de casos de testes, as outras métricas que compõem o Qualidade Total: Arquitetura, Projeto e Código oscilaram no percurso entre a primeira e última versão analisada.

Tabela 4. As métricas coletadas nas versões do HTTPUnit

Métricas	Versão 1.0	Versão 1.3	Versão 1.7
Release	10.09.2000	28.11.2011	20.05.2008
Quality index (1-10)	8.4	8.4	8.7
Coding (0-4.5)	1.0	1.1	0.8
Complexity (0-2)	0.1	0.1	0.1
Coverage (0-2)	0.5	0.4	0.4
Style (0-1.5)	0.0	0.0	0.0
Total Quality (%)	88.3	86.1	84.3
Architecture (%)	100	100	88.3
Design (%)	98.4	87.5	89.0
Test (%)	81.2	80.5	85.0
Code (%)	79.4	77.6	76.0
Technical Debt (ratio %)	13.4	12.3	8.0
Technical Debt ($)	2.684	11.316	51.500
Technical Debt man days	5	23	103

A métrica Débito Técnico apresentou um aumento no custo e na quantidade de dias necessários para uma possível correção de imperfeições no código. Contudo, a proporção do valor do projeto em relação à dívida técnica decresceu entre cada versão, obtendo sua menor porcentagem na última versão analisada. É importante ressaltar que quanto menor o índice, menor será o esforço e o gasto em relação às correções necessárias ao código do projeto. Considerando que o Débito Técnico decresceu e que a métrica Índice de Qualidade combina medidas globais de qualidade e de complexidade do projeto, percebe-se que o ele continua mantendo a qualidade no decorrer da sua evolução, mesmo que o Qualidade Total tenha caído.

5.5. JUnit

O JUnit [JUnit 2013] é um arcabouço de criação de testes unitários automatizados para a linguagem de programação Java. O arcabouço evoluiu satisfatoriamente em relação às três métricas observadas. O Índice de Qualidade teve um leve aumento em relação à primeira versão (2.0) e à segunda versão (3.8.1) analisadas, porém esse índice teve um aumento maior na última versão (4.0). A Tabela 5 mostra os dados das três métricas.

Tabela 5. As métricas coletadas nas versões do JUnit

Métricas	Versão 2.0	Versão 3.8.1	Versão 4.1.0
Quality index (1-10)	7.2	7.4	8.5
Coding (0-4.5)	1.1	1.0	1.0
Complexity (0-2)	0.1	0.1	0.1
Coverage (0- 2)	1.6	1.5	0.4
Style (0-1.5)	0.0	0.0	0.0
Total Quality (%)	76.9	77.1	86.6
Architecture (%)	100	100	85.0
Design (%)	95.7	92.0	95.5
Test (%)	34.9	39.3	85.7
Code (%)	77.0	77.1	80.0
Technical Debt (ratio %)	20.3	13.4	7.5
Technical Debt ($)	5.231	15.062	17.838
Technical Debt man days	11	30	36

A medida Teste foi a principal responsável por garantir que as métricas obtivessem melhores valores em relação à qualidade do projeto, da segunda para a terceira versão houve um aumento expressivo no número de casos de testes agregados, refletindo no aumento de mais de 46% da cobertura do código.

6. Resultados

Considerando uma análise estatística mais detalhada dos dados coletados na Seção 5, é possível quantificar a correção das métricas Índice de Qualidade, Qualidade Total e Dívida Técnica utilizando coeficientes de correlação de Spearman [Triola 2008]. Um resultado obtido de forma exploratória – a partir de dados de cinco OSS – é a existência de uma associação direta entre o Índice de Qualidade e a Qualidade Total e uma associação inversa dessas duas métricas com a Dívida Técnica.

Além disso, usando a mesma ideia, construiu-se um Indicador de Desempenho baseado nas três métricas, por meio da análise dos componentes principais. Esse indicador permitiu observar a evolução da qualidade dos projetos que tinham melhor "qualidade" considerando todas as três métricas correlacionadas e não somente uma delas, de forma a estabelecer uma classificação entre os projetos em cada versão.

Entre os resultados, se observa que pelo Indicador de Desempenho os projetos Cobertura, FindBugs e Commons IO apresentam o pior desempenho ao longo das suas três versões. Em contrapartida, o projeto JUnit foi o que melhor evoluiu, uma vez que apresentou melhoria em todas as métricas de qualidade selecionadas ao longo das versões.

O comportamento desses sistemas, no que se refere às métricas de qualidade avaliadas, foi diferenciado, apresentando crescimento e/ou diminuição em algumas das métricas de qualidade observadas. Os projetos como HttpUnit e JUnit apresentaram aumento em praticamente todas as métricas avaliadas, considerando suas

três versões. Já o FindBugs foi o sistema que apresentou menor variação dos índices de qualidade, ficando praticamente estável. Outros projetos como Commons-IO e Cobertura, decresceram suas métricas de qualidade entre a primeira e a terceira versões avaliadas. Cobertura, por exemplo, apresentou uma discrepância expressiva de valores entre a segunda e terceira versões. Como pode-se observar na análise dos dados realizadas na seção anterior, o projeto apresentou um comportamento diferenciado em relação aos demais projetos, sendo interessante estudos mais detalhados que explorassem melhor esse comportamento

Quanto à análise das métricas, observa-se que o Índice de Qualidade foi a métrica que apresentou menor variação em cada projeto e suas versões, obtendo média aritmética de 7,4 (calculada a partir dos valores apresentados nas tabelas da Seção 5) considerando todos os projetos avaliados. Os projetos Cobertura e FindBugs tiveram média abaixo da média geral. A maioria dos projetos apresentou pequena variação nos valores dessa métrica, sendo que as versões intermediárias apresentaram a melhor média.

A Qualidade Total apresentou maior variação de valores entre as métricas coletadas, obtendo média aritmética 74,5 entre os projetos avaliados. A maioria dos projetos avaliados – Cobertura, Commons-IO, FindBugs e HttpUnit – apresentaram queda quanto aos valores dessa métrica. O único que teve os valores da métrica aumentado foi o JUnit. A apesar de ter apresentado redução nos valores dessa métrica, o HttpUnit foi o projeto que apresentou maior média.

O Débito Técnico também apresentou variação entre os valores comparados entre os projetos e suas versões, obtendo média aritmética 15,2 entre os projetos avaliados. Com uma análise mais detalhada dessa métrica, em relação ao Débito Técnico ($) e Débito Técnico em dias (*Technical Debt in days*) métricas que compõem proporção de Débito Técnico (*Technical Debt ratio*), percebe-se que esses valores aumentaram significativamente, acompanhando quase que proporcionalmente o crescimento dos projetos.

Observa-se que as métricas coletadas apresentam pequenas variações entre os projetos e suas versões, demonstrando uma tendência de melhoria e evolução para a maioria dos projetos, uma vez que dos cinco, pelo menos três apresentam índices iguais ou maiores (no caso do Débito Técnico, menores) que a média para cada uma dessas métricas.

7. Conclusão

A evolução é uma resposta às mudanças ocorridas no ambiente, com o objetivo de prolongar a vida útil do software. Portanto, evoluir não se restringe apenas em aumentar a quantidade de linhas de código, classes, métodos etc, é crescer mantendo ou aumentando seus índices de qualidade. Reforça-se que o conceito de evolução abordado nesse trabalho refere-se ao comportamento das métricas de qualidade.

Na avaliação realizada pelo Sonar fica evidente em quais pontos o sistema melhorou/piorou e quais os componentes responsáveis por aumentar/diminuir a qualidade do mesmo. Esta análise, agregada à evolução do sistema, é um ponto chave para garantir a qualidade prática, podendo apoiar o no processo de desenvolvimento,

uma vez que possibilita uma avaliação clara e abrangente do estado do sistema, auxiliando na sua melhoria tanto na fase de concepção, quanto de manutenção e, consequentemente, na sua evolução.

Os dados apresentados no estudo de caso demonstram um comportamento diferenciado para cada projeto. Os resultados evidenciam que a evolução de cada projeto acontece de forma particular, seguindo características próprias. Sendo assim, não se observou um comportamento padrão sobre os dados, mas sim uma evolução, com crescimento ou diminuição das suas métricas de qualidade, conforme o processo específico de desenvolvimento de cada projeto OSS.

É importante ressaltar que mesmo não identificando um padrão evolutivo no desenvolvimento dos OSS, tal avaliação, proporcionada pela ferramenta de gerenciamento dos projetos como a plataforma Sonar, é importante para verificar a situação global da qualidade do projeto. Acompanhar a evolução do produto por meio de métricas de qualidade possibilita que os desenvolvedores e a comunidade OSS tenham controle sobre a qualidade do projeto, de forma a melhor gerencia-lo, principalmente devido ao fato de identificar quais áreas dos sistemas precisam receber maior atenção da equipe de desenvolvimento numa futura reestruturação.

Por se tratar de um estudo exploratório, mesmo com uma amostra pequena de produtos OSS, algumas tendências foram observadas ao analisar de métricas selecionadas e os valores coletadas. Como proposta para trabalhos futuros, pretende-se avaliar mais profundamente cada uma das métricas de qualidades escolhidas (Índice de Qualidade, Qualidade Total e Débito Técnico), observando a correlação entre elas e o comportamento dos seus componentes. Por serem métricas compostas elas acabam por encobrir algumas características mais particulares do projeto e que possibilitariam uma avaliação mais abrangente e prática sobre a qualidade. Outra questão é ampliar a base de dados, coletando dados de um conjunto maior de projetos, de forma a verificar se é possível identificar tendências de evolução no processo de desenvolvimento de projetos OSS. Além disso pretende-se estabelecer a métrica que melhor identifica as alterações dos projetos visando alertar a equipe de desenvolvimento para atacar determinadas áreas visando a manutenção e/ou melhoria da qualidade dos produtos de software.

Referências

Ayewah, N., Hovemeyer, D., Morgenthaler, J., Penix, J., and Pugh, W. (2008). Using static analysis to find bugs.

Bogoni, L. P. & Ruiz, D. D. A. (2008). Um Método para Suporte à Evolução de Métricas de PDS

Cobertura. Home page Cobertura (2006). `http://cobertura.sourceforge.net/`. Acesso em 15/04/2013.

COMMONS-IO. Home page Commons-IO (2013). `http://commons.apache.org/io/`. Acesso em 15/04/2013.

Crowston, K., Annabi, H., & Howison, J. (2003). Defining open source software project success.

Eisenberg, R. J. (2012). A threshold based approach to technical debt.

FindBugs. Home page FindBugs (2012). http://findbugs.sourceforge.net/. Acesso em 15/04/2013.

Gat, I. & Ebert, C. (2012). Point counterpoint: Technical debt as a meaningful metaphor for code quality.

Godfrey, M. & Tu, Q. (2001). Growth, evolution, and structural change in open source software.

HttpUnit. Home page HttpUnit (2008). http://httpunit.sourceforge.net/. Acesso em 15/04/2013.

Johari, K. & Kaur, A. (2011). Effect of software evolution on software metrics: an open source case study.

JUnit. Home page JUnit (2013). http://www.junit.org/. Acesso em 15/04/2013.

Lehman, M. M., Ramil, J. F., Wernick, P. D., Perry, D. E., & Turski, W. M. (1997). Metrics and laws of software evolution - the nineties view.

Maven. Home page Maven (2013). http://maven.apache.org/. Acesso em 15/04/2013.

Mishra, B., Prasad, A., Raghunathan, S., Mishra, B., Prasad, A., & Raghunathan, S. (2002). Quality and profits under open source versus closed source.

Mockus, A., Fielding, R., & Herbsleb, J. (2000). A case study of open source software development: the apache server.

QualityIndex. Home page Quality Index Plugin (2011). http://docs.codehaus.org/display/SONAR/Quality+Index+Plugin. Acesso em 15/04/2013.

Raymond, E. S. (1999). *The Cathedral and the Bazaar*.

Sonar. Home page Sonar (2013). http://www.sonarsource.org/. Acesso em 15/04/2013.

Spinellis, D. & Szyperski, C. (2004). How is open source affecting software development?

Technical Debt Plugin. Home page Technical Debt Plugin (2011). http://docs.codehaus.org/display/SONAR/Technical+Debt+Plugin. Acesso em 15/04/2013.

Total Quality. Home page Total Quality Plugin (2013). http://docs.codehaus.org/display/SONAR/Total+Quality+Plugin. Acesso em 15/04/2013.

Triola, M.F. (2008) Introdução à estatística, LTC.

Weber, S. (2004). *The Success of Open Source*. Harvard University Press.