UNIVERSIDADE FEDERAL DE GOIÁS
INSTITUTO DE INFORMÁTICA

IGOR RODRIGUES VIEIRA

Avaliando a dívida técnica em produtos de código aberto por meio de estudos experimentais

Goiânia
2014

TERMO DE CIÊNCIA E DE AUTORIZAÇÃO PARA DISPONIBILIZAR AS TESES E DISSERTAÇÕES ELETRÔNICAS (TEDE) NA BIBLIOTECA DIGITAL DA UFG

Na qualidade de titular dos direitos de autor, autorizo a Universidade Federal de Goiás (UFG) a disponibilizar, gratuitamente, por meio da Biblioteca Digital de Teses e Dissertações (BDTD/UFG), sem ressarcimento dos direitos autorais, de acordo com a Lei nº 9610/98, o documento conforme permissões assinaladas abaixo, para fins de leitura, impressão e/ou *download*, a título de divulgação da produção científica brasileira, a partir desta data.

1. Identificação do material bibliográfico: **[X] Dissertação** **[ ] Tese**

2. Identificação da Tese ou Dissertação

Autor (a):	**Igor Rodrigues Vieira**					
E-mail:	igorvieira@inf.ufg.br, igorodvieira@gmail.com					
Seu e-mail pode ser disponibilizado na página? [X]Sim [ ] Não						
Vínculo empregatício do autor	Servidor público federal					
Agência de fomento:					Sigla:	
País:		UF:		CNPJ:		
Título:	Avaliando a dívida técnica em produtos de código aberto por meio de estudos experimentais					
Palavras-chave:	Dívida Técnica, qualidade de software, análise estática, produto de código aberto, estudo experimental					
Título em outra língua:	*Assessing the technical debt in open source products through experimental studies*					
Palavras-chave em outra língua:	*Technical Debt, software quality, static analysis, open source product, experimental study*					
Área de concentração:	Ciência da Computação					
Data defesa: (dd/mm/aaaa)	19/11/2014					
Programa de Pós-Graduação:	Ciência da Computação					
Orientador (a):	Prof. Dr. Auri Marcelo Rizzo Vincenzi					
E-mail:	auri@inf.ufg.br					
Co-orientador(a):*						
E-mail:						

*Necessita do CPF quando não constar no SisPG

3. Informações de acesso ao documento:

Concorda com a liberação total do documento [X] SIM [ ] NÃO[1]

Havendo concordância com a disponibilização eletrônica, torna-se imprescindível o envio do(s) arquivo(s) em formato digital PDF ou DOC da tese ou dissertação.

O sistema da Biblioteca Digital de Teses e Dissertações garante aos autores, que os arquivos contendo eletronicamente as teses e ou dissertações, antes de sua disponibilização, receberão procedimentos de segurança, criptografia (para não permitir cópia e extração de conteúdo, permitindo apenas impressão fraca) usando o padrão do Acrobat.

Igor Rodrigues Vieira

Assinatura do (a) autor (a)

Data: 16 / 01 / 2015

[1] Neste caso o documento será embargado por até um ano a partir da data de defesa. A extensão deste prazo suscita justificativa junto à coordenação do curso. Os dados do documento não serão disponibilizados durante o período de embargo.

IGOR RODRIGUES VIEIRA

Avaliando a dívida técnica em produtos de código aberto por meio de estudos experimentais

Dissertação apresentada ao Programa de Pós–Graduação do Instituto de Informática da Universidade Federal de Goiás, como requisito parcial para obtenção do título de Mestre em Ciência da Computação.

Área de concentração: Ciência da Computação.

Orientador: Prof. Dr. Auri Marcelo Rizzo Vincenzi

Goiânia
2014

Ficha catalográfica elaborada automaticamente
com os dados fornecidos pelo(a) autor(a), sob orientação do Sibi/UFG.

Vieira, Igor Rodrigues
Avaliando a dívida técnica em produtos de código aberto por meio de estudos experimentais [manuscrito] / Igor Rodrigues Vieira. - 2014.
100 f.: il.

Orientador: Prof. Dr. Auri Marcelo Rizzo Vincenzi.
Dissertação (Mestrado) - Universidade Federal de Goiás, Instituto de Informática (INF) , Programa de Pós-Graduação em Ciência da Computação, Goiânia, 2014.
Bibliografia. Apêndice.
Inclui algoritmos, lista de figuras, lista de tabelas.

1. Dívida técnica. 2. Qualidade de software. 3. Análise estática. 4. Produto de código aberto. 5. Estudo experimental. I. Vincenzi, Auri Marcelo Rizzo, orient. II. Título.

Igor Rodrigues Vieira

Avaliando a dívida técnica em produtos de código aberto por meio de estudos experimentais

Dissertação defendida no Programa de Pós-Graduação do Instituto de Informática da Universidade Federal de Goiás como requisito parcial para obtenção do título de Mestre em Ciência da Computação, aprovada em 19 de novembro de 2014, pela Banca Examinadora constituída pelos professores:

Prof. Dr. Auri Marcelo Rizzo Vincenzi
Instituto de Informática – UFG
Presidente da Banca

Prof. Dr. Plínio de Sá Leitão Júnior
Instituto de Informática - UFG

Prof. Dr. José Carlos Maldonado
Instituto de Ciências Matemáticas e de Computação - USP

Igor Rodrigues Vieira

Graduado em Sistemas de Informação, pela Universidade Estadual de Goiás – UEG, com pós-graduação *lato sensu* em Desenvolvimento de Aplicações Web com Interfaces Ricas, pela Universidade Federal de Goiás – UFG. Foi Coordenador da Ouvidoria da UFG e, atualmente, é Analista de Tecnologia da Informação do Centro de Recursos Computacionais – CERCOMP/UFG.

À minha querida avó, Madalena *(in memoriam)*, que em sua sabedoria dizia para eu não estudar muito, pois poderia ficar louco.

Agradecimentos

Em primeiro lugar, agradeço a Deus por me olhar com carinho, me conceder tantas graças e me fortalecer nos momentos de dificuldades. A Ele, toda honra e glória.

À minha família, em especial aos meus pais, Antônio e Nair, que souberam me educar com muita sabedoria e amor. Hoje, mais do que nunca, reconheço o valor da família e o amor incondicional dos meus pais por seus filhos. Também, às minhas irmãs, Bruna e Márcia, pela amizade e confiança.

À minha amada Raquel, que por tantos momentos teve que conviver com a minha ausência e, mesmo assim, ainda continuava a demonstrar seu carinho, incentivo e admiração. Se consegui, foi graças ao seu apoio, meu anjo.

Ao meu orientador e a quem posso chamar de amigo, Prof. Auri Vincenzi, pelo apoio, confiança e contribuições na condução do mestrado e na elaboração da dissertação. Acredito que não poderia estar em melhores mãos. Sempre terei admiração pelo seu otimismo e entusiasmo.

Ao Prof. Mário Piscoya, do Instituto de Matemática e Estatística (IME/UFG), pela disponibilidade em nos auxiliar na etapa de análise estatística dos dados coletados. Nossas reuniões, descontraídas, sempre traziam motivação e novas ideias.

Aos colegas de disciplinas e laboratório, dos quais tenho como amigos, Leonardo, Vinícius, André, Cláudio, Marllos, Lucas e Marcos, agradeço pelo apoio e companheirismo. Em especial, ao Leonardo e Vinícius por terem compartilhado tão de perto os estudos, expectativas, alegrias e frustrações. Obrigado, meus amigos.

Aos professores do Programa de Pós-Graduação do Instituto de Informática (INF/UFG), pelo convívio e orientação acadêmica durante este período de curso.

Aos servidores técnico-administrativos do INF/UFG, em especial à Mirian e Patrícia, que com tanto profissionalismo e atenção me auxiliaram em diversos momentos.

Aos meus colegas de trabalho da Reitoria da UFG, cujos nomes não citarei, por serem muitos, mas acredito que cada um deles sabe o quanto pode me ajudar nessa jornada árdua e, ao mesmo tempo, tão gratificante. Obrigado pelo incentivo e companheirismo.

À UFG, instituição que tanto admiro, pelo acolhimento e oportunidades em mais de dez anos de convivência, seja pelo trabalho e/ou pelos estudos, e na qual inicio uma nova jornada profissional da minha vida.

“O homem nasceu para aprender, aprender tanto quanto a vida lhe permita.”

Guimarães Rosa,
citado em "Relembramentos, João Guimarães Rosa, meu pai", de Vilma Guimarães Rosa.

Resumo

Vieira, Igor Rodrigues. **Avaliando a dívida técnica em produtos de código aberto por meio de estudos experimentais**. Goiânia, 2014. 100p. Dissertação de Mestrado. Instituto de Informática, Universidade Federal de Goiás.

A metáfora da dívida técnica (DT) apresenta-se muito útil para Engenharia de Software, estando diretamente relacionada ao contexto de evolução e manutenção existentes no ciclo de vida de um produto. Ela pode ser entendida como uma relação entre custos e efeitos, de curto e longo prazos, associados a decisões de projeto durante o processo de desenvolvimento de software. Atualmente, grandes empresas e alguns setores do governo ainda possuem restrições quanto à adoção de produtos de código aberto por incertezas relacionadas a sua qualidade e confiabilidade. Nesse contexto, o presente trabalho tem por objetivo avaliar a dívida técnica em produtos de código aberto, no intuito de demonstrar a possibilidade de utilização dessa abordagem para avaliação da qualidade de software. Para tanto, foram realizados estudos experimentais, contemplando a coleta automatizada de dados para um conjunto expressivo de produtos de código aberto, tendo como entrada o respectivo código fonte. Esses produtos foram submetidos à avaliação da Plataforma SonarQube, a qual possibilita coletar diversas métricas sobre a qualidade do código fonte – entre elas a dívida técnica *(technical debt)*. A interpretação dos dados coletados possibilitou a análise da evolução da DT desses produtos, a classificação dos projetos e a verificação da representatividade dos eixos de qualidade que compõem a DT. Os resultados sugerem que a maioria dos projetos avaliados demonstrou diminuição da DT, ao longo de suas versões, e apresentou valores pouco elevados para a métrica. Outra contribuição consiste que os eixos de qualidade "Cobertura", "Violações" e "Complexidade" foram identificados como aqueles que mais contribuem para o incremento da DT do conjunto de produtos avaliados. Foi possível, também, verificar a existência de uma correlação entre a implementação da DT estudada e a metodologia SQALE, no que diz respeito à avaliação da qualidade de software.

Palavras–chave

Dívida Técnica, qualidade de software, análise estática, produto de código aberto, estudo experimental.

Abstract

Vieira, Igor Rodrigues. **Assessing the technical debt in open source products through experimental studies**. Goiânia, 2014. 100p. MSc. Dissertation. Instituto de Informática, Universidade Federal de Goiás.

The metaphor of technical debt (TD) is very useful for Software Engineering, it is directly related to the context of evolution and maintenance in the life cycle of a product. It can be understood as a relation between costs and effects, of short and long term, associated with project decisions during the software development process. Currently, large companies and some government sectors still have restrictions in adopting open source products by uncertainties related to its quality and reliability. In this context, this study aims to evaluate the technical debt in open source products in order to demonstrate the feasibility of this approach to evaluate the software quality. For this, were performed experimental studies, contemplating the automated data collection for a significant set of products open source, having as input its source code. These products were evaluated by SonarQube Platform, which enables collect several metrics about the quality of the source code - including the technical debt. The interpretation of the collected data allowed the analysis of the TD evolution for these products, the classification of the projects and the verification of the representativeness of the quality axis that make up the TD. The results suggest that most of the projects evaluated have shown decreased TD along their versions and they showed slightly elevated values of the metric. Another contribution is that the quality axis Coverage, Violations and Complexity is presented as the main contributors to the TD's increase of from the set of product evaluated. It was also possible to verify the existence of a correlation between the TD implementation and the SQALE methodology, with regard assessing software quality evaluating.

Keywords

Technical Debt, software quality, static analysis, open source product, experimental study

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Códigos de Programas

CAPÍTULO 1

Introdução

A metáfora da dívida técnica (DT) no desenvolvimento de software foi introduzida há mais de duas décadas por Ward Cunningham (CUNNINGHAM, 1992) para explicar às partes interessadas do produto, que não pertenciam à área técnica, sobre a necessidade de refatoração (KRUCHTEN; NORD; OZKAYA, 2012). A intenção era associar os custos a longo prazo para um projeto de software de implementação inferior (EISENBERG, 2012), com artefatos considerados imaturos, incompletos ou inadequados no ciclo de desenvolvimento de software (SEAMAN; GUO, 2011), mas que permitiam a entrega do produto em um menor espaço de tempo.

Atualmente, a metáfora da DT vem ganhando força significativa na comunidade de desenvolvimento de software, por permitir compreender e comunicar questões intrínsecas de qualidade, valor e custo do produto (KRUCHTEN et al., 2012), uma vez que essa metáfora associa termos da área financeira às circunstâncias inerentes ao processo de desenvolvimento e manutenção de software.

Para Zazworka et al. (2013), essa abordagem tem facilitado a discussão entre os profissionais e pesquisadores pelo fato de prover um vocabulário familiar, do domínio financeiro, e que tem potencial para tornar-se uma linguagem universal para comunicação de compensações técnicas. Os autores consideram que, embora seja muito útil enquanto metáfora, a DT vai além ao inspirar um conjunto de ferramentas e métodos para apoiar sua identificação, mensuração, monitoramento, gestão e pagamento.

Segundo Kruchten et al. (2012), os desenvolvedores de software e gerentes de empresas frequentemente discordam sobre decisões importantes de como investir os escassos recursos em projetos de desenvolvimento, especialmente quanto aos aspectos internos de qualidade, os quais são cruciais para a sustentabilidade do produto, mas invisíveis para a gestão e os clientes, além de não gerarem receita a curto prazo. Esses aspectos incluem o código, a qualidade do projeto e a documentação.

Para os autores, a situação é agravada em projetos que precisam equilibrar prazos curtos com a sustentabilidade a longo prazo. O conceito de sustentabilidade de uma arquitetura de software se refere à capacidade da mesma em suportar diferentes tipos de mudanças, por meio de manutenção eficiente e evolução ao longo do seu ciclo de vida.

Esse cenário demonstra claramente que o contexto da DT está associado à necessidade de se estabelecer um equilíbrio entre os objetivos do produto, de curto e longo prazo, durante o processo de desenvolvimento de software.

O conceito de DT será definido com mais propriedade no Capítulo 2, contudo, de início, pode-se entendê-la como uma relação entre custos e efeitos, de curto e longo prazos, associados a decisões de projeto tomadas durante o processo de desenvolvimento de software. A título de ilustração, considere um exemplo simples sobre a aplicação dessa metáfora: durante o processo de desenvolvimento de um software, restrições de tempo (atraso no cronograma do projeto) podem dificultar a aplicação de todos os casos de teste inicialmente previstos. Em consequência, a equipe do projeto decide não codificar alguns dos casos de teste, a fim de garantir a entrega do produto ao cliente no prazo estabelecido.

Entende-se, assim, que essa decisão, a curto prazo, é conveniente ao projeto, pois garante que o produto seja entregue no tempo previsto, evitando, deste modo, uma possível multa contratual. No entanto, ao se considerar critérios técnicos de qualidade do produto, verifica-se que, a longo prazo, a cobertura dos casos de teste ficou comprometida, do mesmo modo que a manutenibilidade do produto. Diante disso, aplica-se a metáfora ao considerar a contratação de uma "dívida", em função da sua conveniência momentânea ao projeto, mas com a necessidade de pagá-la, posteriormente, sob o risco dos "juros" crescerem a níveis incontroláveis (VIEIRA et al., 2013) e, então, comprometer a sustentabilidade arquitetural do produto.

Na prática, várias pessoas têm usado a metáfora da dívida técnica para descrever muitos outros tipos de débitos e males do desenvolvimento de software, abrangendo amplamente tudo o que está no caminho da implementação, venda ou evolução de um software, ou qualquer fator que adicione desgaste aos esforços para seu desenvolvimento (KRUCHTEN; NORD; OZKAYA, 2012). Segundo os autores, atualmente, isso torna diluído o conceito da DT, além de estabelecer, erroneamente, uma relação direta entre a DT e os defeitos existentes no código fonte.

1.1 Contexto e Motivação

Um primeiro contato com o conceito e implementação da dívida técnica aconteceu quando da realização de um estudo inicial, no qual utilizou-se a análise estática automatizada para avaliar a qualidade de produtos de código aberto, tendo como entrada o respectivo código fonte. Esse estudo preliminar teve por objetivo investigar a evolução de produtos de código aberto com base em métricas de qualidade (VINCENZI et al., 2013). Para tanto, foram selecionados cinco produtos e três de suas versões disponíveis – uma inicial, uma intermediária e uma final –, os quais foram submetidos à avaliação da Plataforma SonarQube.

Priorizou-se, naquele trabalho, a coleta de dados para três métricas de qualidade, dentre as disponibilizadas pela Plataforma: *Quality Index*, *Total Quality* e *Technical Debt*, com a intenção de avaliar a evolução desses produtos conforme os valores observados para cada projeto/versão, bem como para o conjunto selecionado. Uma das métricas analisadas foi o *Technical Debt*, uma implementação da DT disponibilizada por meio do *Technical Debt Plugin* integrado à Plataforma SonarQube.

Nesse mesmo período, percebeu-se uma ênfase da comunidade acadêmica e industrial na área de governança de software, o que foi confirmado com a edição da revista *IEEE Software*, em novembro/dezembro de 2012 (Volume 29), tendo por tema principal *"Technical Debt"*. Isso motivou a continuidade daquele estudo preliminar, mas, nessa nova fase, com foco na dívida técnica.

Direcionou-se, então, a atenção para a DT, no intuito de observar dados de um conjunto maior de projetos de código aberto. Isso resultou na proposta de avaliação da DT para um conjunto de quarenta projetos, utilizando a versão mais recente do respectivo código fonte, com o objetivo de observar seu estado, propor uma classificação dos projetos e identificar quais dos eixos de qualidade eram mais representativos na composição da DT (VIEIRA et al., 2013). Para aquele trabalho, entendeu-se por eixos mais representativos aqueles que mais contribuem para o incremento da DT. Os resultados desses dois trabalhos preliminares serão apresentados na Seção 3.2.

A opção por avaliar produtos de código aberto considerou dois aspectos principais, não necessariamente nessa ordem de prioridade. Primeiro, a facilidade de acesso ao código fonte, tido como entrada para as ferramentas de análise estática utilizadas nos estudos experimentais, já que são conhecidas as restrições e dificuldades em se obter acesso ao código fonte de produtos proprietários. Segundo, o interesse em propor uma investigação, ainda que preliminar e exploratória, sobre a qualidade desses produtos, tendo por objeto a avaliação do código fonte e como parâmetro os eixos de qualidade da DT implementados pelo *Technical Debt Plugin*.

Considera-se interessante o segundo aspecto, relacionado à perspectiva de avaliação da qualidade de produtos de código aberto, pois, apesar do aumento gradativo da utilização do *software livre* na indústria de software nas últimas décadas, ainda há empresas e governos (ou setores dos governos) relutantes em adotá-lo devido a dúvidas legais, incertezas comerciais, questões culturais ou falta de confiança na qualidade do produto (código fonte produzido) e do suporte a esse tipo de software (MEIRELLES, 2013).

Nesse sentido, entende-se que o estudo proposto pode contribuir com a investigação sobre a qualidade de produtos de código aberto, fornecendo indícios adicionais da sua confiabilidade, além de questões relacionadas à qualidade do código fonte e à identificação e gestão da DT.

1.2 Objetivos

Conforme apresentado anteriormente, o objetivo principal deste trabalho é realizar uma investigação, ainda que preliminar e exploratória, sobre a qualidade de produtos de código aberto, por meio do planejamento e execução de estudos experimentais para avaliar a dívida técnica (DT) nesses produtos. Para tanto, são utilizadas métricas baseadas em código fonte, por meio de ferramentas automatizadas de análise estática, considerando a série histórica de versões dos projetos.

Espera-se, ainda, alcançar os seguintes objetivos específicos:

1. **Analisar o estado e evolução da DT** para o conjunto de produtos de código aberto selecionados. A intenção é observar os valores coletados sobre a DT desses produtos, ao longos de suas versões disponíveis, e o comportamento evolutivo dessa métrica para cada projeto e para o conjunto;
2. **Estabelecer uma classificação para a DT**, com o intuito de propor um agrupamento para os projetos, conforme os valores e/ou características observadas, e sugerir um limiar *(threshold)* para definir níveis aceitáveis e/ou gerenciáveis da DT, possibilitando, ainda, posicionar esses projetos em relação à classificação proposta;
3. **Verificar quais eixos de qualidade são mais representativos na composição da DT**, no sentido de identificar aspectos do código fonte, relacionados aos eixos de qualidade da DT, que necessitam de maior atenção da comunidade de desenvolvimento. Para este trabalho, entende-se como eixos mais representativos aqueles que mais contribuem para o incremento da DT.

1.3 Metodologia

A metodologia empregada neste trabalho contempla o desenvolvimento de algumas atividades relacionadas ao levantamento bibliográfico do tema; o desenvolvimento dos estudos experimentais propostos; a análise e interpretação dos dados coletados e a apresentação dos resultados, conforme ilustrado na Figura 1.1.

Apesar de reconhcer a relavância da revisão sistemática de literatura para trabalhos dessa natureza, as limitações do cronograma da pesquisa não permitiram sua realização. Dessa forma, o levantamento bibliográfico sobre o tema consistiu de uma revisão não sistemática da literatura, contando com uma cuidadosa busca por estudos e pesquisas recentes sobre o tema, apresentadas em conferências e revistas disponíveis nas principais bases de pesquisa da área (*IEEE Software, ACM e Elsevier*). A partir de então, foram elencados alguns trabalhos relacionados para servirem de orientação ao desenvolvimento desta pesquisa, permitindo, assim, sua complementação, expansão e/ou exploração de novas perspectivas.

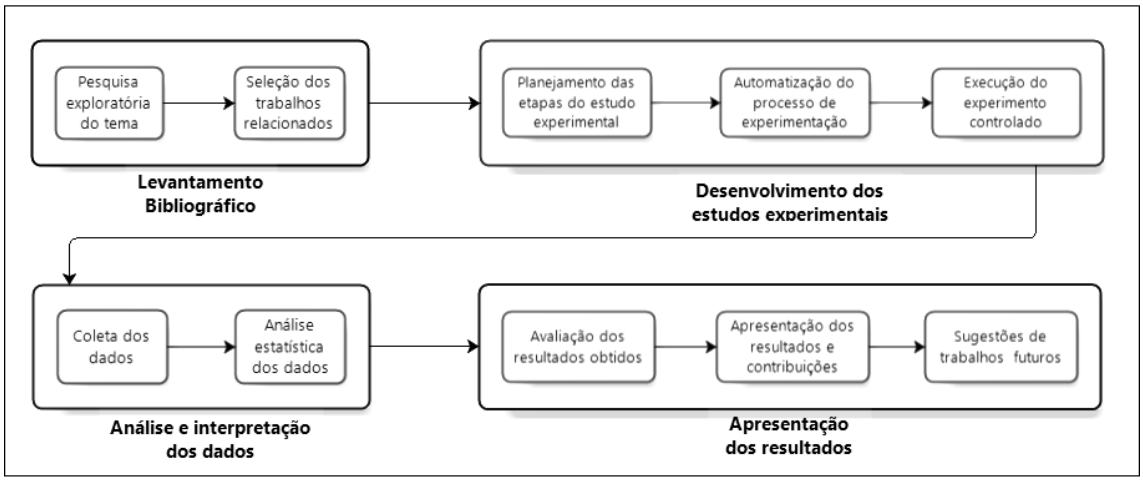

Figura 1.1: *Metodologia empregada no trabalho*

A etapa seguinte consistiu da definição, planejamento e implementação do processo de experimentação. Considerando as características dos estudos experimentais propostos e visando otimizar os recurso disponíveis (especialmente o tempo), optou-se por automatizar o processo de experimentação. Isso possibilitou a execução do experimento controlado e a avaliação de uma quantidade significativa de projetos e suas versões.

O processo de coleta dos dados também foi automatizado. Os dados foram analisados por meio de métodos de estatística descritiva e multivariada, permitindo sua avaliação sob diferentes perspectivas, buscando alcançar os objetivos apresentados na Seção 1.2 e investigar características específicas sobre a qualidade dos produtos de código aberto avaliados, relacionadas a sua dívida técnica.

Por fim, realizou-se a avaliação dos resultados obtidos, de forma a extrair as considerações finais e contribuições deste trabalho, além de sugerir a realização de algumas investigações futuras que permitam a continuidade e/ou complementação do estudo proposto.

1.4 Organização da Dissertação

A presente dissertação está organizada conforme descrito a seguir: este capítulo introduziu a motivação, a metodologia e os objetivos a serem alcançados neste trabalho, considerando a importância que a DT apresenta no contexto do desenvolvimento de software.

No Capítulo 2 são apresentados alguns termos e conceitos importantes para a compreensão do trabalho, incluindo as métricas e ferramentas utilizadas.

O Capítulo 3, por sua vez, introduz o conceito associado à metáfora da dívida técnica e aborda alguns trabalhos que serviram de base para o desenvolvimento da

pesquisa, destacando-se alguns que utilizam ferramentas automatizadas de análise estática para avaliação da DT.

No Capítulo 4 são apresentadas a concepção e as características dos estudos experimentais, com uma descrição detalhada das etapas do processo de experimentação implementado.

O Capítulo 5 traz os resultados obtidos com a execução do experimento, por meio da análise e interpretação dos dados coletados, bem como as ameaças a sua validade.

Por fim, no Capítulo 6 são discutidas as considerações finais, com as contribuições do trabalho, além das perspectivas para realização de estudos futuros.

Compõem, ainda, este trabalho, as referências bibliográficas, o Apêndice A com a relação dos projetos, versões e métricas coletadas, os quais constituem o objeto deste estudo, além do Apêndice B que apresenta o código fonte dos *scripts* implementados para permitir a automatização do processo de experimentação.

CAPÍTULO 2

Terminologia e Conceitos Importantes

Neste capítulo são apresentados alguns termos e conceitos importantes para a compreensão deste trabalho, bem como as ferramentas e métricas utilizadas. A Seção 2.1 oferece uma definição dos principais conceitos utilizados nesse trabalho. A Seção 2.2 traz uma breve descrição das ferramentas utilizadas, mais precisamente sobre a Plataforma SonarQube, com suas ferramentas e *plugins* integrados. Já a Seção 2.3 aborda as métricas de qualidade que foram consideradas e disponibilizadas pelas respectivas ferramentas. Por fim, a Seção 2.4 apresenta as considerações finais do capítulo e introduz o conteúdo do capítulo seguinte.

2.1 Conceitos Importantes

Os conceitos apresentados nesta seção são importantes para a compreensão deste estudo por dois motivos principais: primeiro, oferecem uma definição dos principais conceitos utilizados e, segundo, estabelecem a forma como eles serão empregados ao longo da dissertação.

2.1.1 Dívida Técnica (DT)

Uma vez que a DT é o foco deste estudo, seu conceito, concepção e abordagens serão explorados em praticamente todo o trabalho. A literatura estabelece algumas definições sobre o conceito da DT, sendo que para este trabalho optou-se por adotar o conceito definido por Zazworka et al. (2013), os quais consideram que a DT descreve um equilíbrio entre objetivos a curto e longo prazos no desenvolvimento de software. Desse modo, a DT relaciona-se aos resultados obtidos, a curto prazo, pelo fato de adiar algumas atividades de desenvolvimento e às consequências, a longo prazo, associadas a essa decisão, inclusive os custos e esforços necessários (tempo e trabalho) para realizar ou refazer essas atividades.

Para Seaman e Guo (2011), a dívida técnica é uma metáfora para artefatos de software imaturos, incompletos ou inadequados no ciclo de desenvolvimento de software,

os quais causam maior custo e menor qualidade do produto a longo prazo. Esses artefatos do sistema podem afetar as atividades de desenvolvimento e manutenção posteriores e, por isso, são visto como uma dívida junto ao sistema. Para esses autores, incorrer em DT pode acelerar o desenvolvimento de software, mas tal benefício é conseguido à custa de trabalho extra no futuro, o que pode ser associado ao pagamento de juros sobre uma dívida.

Conforme será discutido na Seção 3.2, há na literatura considerações divergentes sobre as vantagens e desvantagens da identificação automatizada da DT (KRUCHTEN; NORD; OZKAYA, 2012), (ZAZWORKA et al., 2013) e (VETRO', 2012). A DT tem relação com determinados contextos do processo de desenvolvimento e tomada de decisão da organização. Sendo assim, a identificação manual (por meio da elicitação humana) é capaz de prover informações contextuais adicionais importantes de cada instância da DT, as quais seriam impossíveis de serem recolhidas automaticamente (ZAZWORKA et al., 2013).

Por outro lado, considera-se que as ferramentas automatizadas de análise estática possibilitam que a avaliação do código fonte do produto seja realizada em um curto espaço de tempo, especialmente quando se trata de um grande volume de código e/ou de produtos. Para essas situações, a abordagem automatizada ainda pode servir como um guia para a elicitação humana da DT, uma vez que, sem esse auxílio, a identificação manual tornaria-se praticamente inviável.

Outra questão importante é que essas ferramentas, especialmente a Plataforma SonarQube, possibilitam uma abordagem gerencial da DT, oferecendo informações importantes aos desenvolvedores, gerentes e demais interessados quanto a identificação, monitoramento e gestão da DT.

Esses benefícios constituem-se como algumas das motivações para identificação automatizada da DT empregada neste trabalho, especialmente pelas características dos estudos experimentais propostos.

2.1.2 Análise Estática e Análise Dinâmica

Segundo Sommerville (2011), a análise estática é uma técnica de verificação, na qual não é necessária a execução do sistema para que os defeitos sejam encontrados. Essa característica é sua principal vantagem, pois os defeitos são localizados sem a interrupção do processo de execução. Outra vantagem consiste na possibilidade de sua utilização em qualquer etapa do processo de desenvolvimento de software. A análise estática utiliza como entrada artefatos tipo documento de requisitos, diagramas, código fonte e outros.

À vista disso, não é possível assegurar, apenas por meio da análise estática, que um produto está suficientemente seguro para ser utilizado. Contudo, esse tipo de análise

é uma excelente técnica de verificação. Pressman (2011) considera que as revisões e a verificação formal são exemplos de atividades estáticas e funcionam como um "filtro" para o processo de Engenharia de Software.

A análise dinâmica constitui-se como uma técnica de verificação que envolve a execução do código fonte com a finalidade de detectar defeitos. Um dos principais representantes desse tipo de análise é o teste de software, um método dinâmico executado com a intenção de visualizar o comportamento do software desenvolvido e verificar se o mesmo está de acordo com as especificações estabelecidas (SOMMERVILLE, 2011).

Nesse contexto, torna-se interessante observar as vantagens e desvantagens da utilização da análise estática e dinâmica como técnicas de verificação e validação. A análise estática constitui uma técnica de menor custo e tempo, capaz de identificar algumas anomalias no código fonte, mas que, em muitos casos, possui o inconveniente da geração de falsos positivos, ou seja, defeitos não existentes no código que são reportados como reais. Já a análise dinâmica, apesar de ser uma abordagem mais cara e com maior demanda de tempo, é capaz de revelar alguns aspectos comportamentais do produto, por meio da sua execução, os quais não podem ser identificados por ferramentas de análise estática.

A coleta de dados realizada neste trabalho, com o objetivo de analisar a DT em produtos de código aberto, foi possível pela submissão do código fonte dos projetos avaliados à Plataforma SonarQube. A Plataforma, por sua vez, integra *plugins* e outras ferramentas, tanto de análise estática como de avaliação da cobertura e execução dos casos de teste. A Seção 3.2 abordará alguns trabalhos relacionados à utilização de ferramentas automatizadas de análise estática para investigação da DT.

2.1.3 Qualidade de Software

Segundo Pressman (2011), em termos gerais, a qualidade de software é a satisfação de requisitos funcionais e de desempenho explicitamente declarados, normas de desenvolvimento explicitamente documentadas e características implícitas que são esperadas em todo o software desenvolvido profissionalmente.

A norma ISO 25010/2011, que revisou a ISO:9126/1991, constitui-se como uma das principais diretrizes sobre qualidade de software, uma vez que foi desenvolvida na tentativa de identificar seus atributos. A norma estabelece seis atributos-chave de qualidade, a saber:

- *Funcionalidade* - grau em que o sistema satisfaz as necessidades declaradas, conforme indicado pelos seguintes subatributos: adequabilidade, precisão, interoperabilidade, atendibilidade e segurança;

- *Confiabilidade* - período de tempo em que o software está disponível para uso, conforme indicado pelos seguintes subatributos: maturidade, tolerância a falha e recuperabilidade;
- *Usabilidade* - grau em que o software é fácil de usar, conforme indicado pelos seguintes subatributos: inteligibilidade, facilidade de aprendizado e operabilidade;
- *Eficiência* - grau em que o software faz uso otimizado dos recursos do sistema, conforme indicado pelos seguintes subatributos: comportamento em relação ao tempo e comportamento em relação aos recursos;
- *Manutenibilidade* - facilidade com a qual podem ser feitos reparos no software, conforme indicado pelos seguintes subatributos: analisabilidade, mutabilidade, estabilidade e testabilidade;
- *Portabilidade* - facilidade com a qual o software pode ser transposto de um ambiente para outro, conforme indicado pelos seguintes subatributos: adaptabilidade, instabilidade, conformidade e permutabilidade.

Segundo Pressman (2011), a garantia de qualidade de software pode ser alcançada por meio de quatro fatores: métodos de engenharia de software, técnicas de gerenciamento de projeto, ações de controle de qualidade e garantia da qualidade de software. Para este trabalho, a qualidade do produto está relacionada aos eixos de qualidade implementados pelo *Technical Debt Plugin*, conforme será descrito adiante na Seção 2.3.

2.1.4 Métricas de Software

Pressman (2011) afirma que a medição nos permite obter entendimento do processo e do projeto de software, fornecendo um mecanismo para avaliação objetiva. Segundo o autor, métricas são medidas quantitativas que permitem aos engenheiros de software ter ideia da eficácia do processo de software e dos projetos que são produzidos. Isso permite coletar dados básicos de qualidade e de produtividade e serve como uma ferramenta de gestão. Métricas também são usadas para detectar áreas de problemas, de modo que soluções possam ser desenvolvidas e que o processo de software possa ser melhorado.

Apesar dos termos *métricas*, *medidas* e *medição* serem usados frequentemente quase como sinônimos, existem diferenças sutis entre eles. Pressman (2011) busca esclarecer esses termos, ao afirmar que uma *medida* fornece uma indicação quantitativa da extensão, quantidade, dimensão, capacidade ou tamanho de algum atributo de um produto ou processo; já a *medição* é o ato de determinar uma medida; e, por último, uma *métrica* representa uma medida quantitativa do grau em que um sistema, componente ou processo possui um determinado atributo.

As métricas utilizadas nesse trabalho serão detalhadas na Seção 2.3 e estão diretamente relacionadas às ferramentas utilizadas para o desenvolvimento dos estudos experimentais propostos, conforme será descrito na Seção 2.2.

2.1.5 Produtos de Código Aberto

Segundo Meirelles (2013), o software livre é aquele que permite aos usuários usá-lo, estudá-lo, modificá-lo e distribuí-lo, em geral, sem restrições para tal e prevenindo que não sejam impostas restrições aos futuros usuários. Apesar de descrever modelos similares de desenvolvimento, os termos *software livre* e *software de código aberto* possuem algumas diferenças ideológicas. O *software livre* defende uma ideia fundamental de liberdade do usuário e do uso do software. O termo *software de código aberto*, por sua vez, estabelece um ponto de vista puramente técnico, relacionado ao modelo de desenvolvimento.

O termo software de código livre e aberto, do inglês *FLOSS (free/libre/open source software)*, refere-se a um software que é tanto livre como de código aberto. Esse é um termo mais inclusivo e que cobre ambos significados. Para esse trabalho, será adotado o termo software de código aberto, ou melhor, *produtos de código aberto* (ou a sigla PCA) para fazer referência aos produtos que serão avaliados. Ao longo deste trabalho os termos produto e projeto serão utilizados como o mesmo significado. O termo projeto, por exemplo, será utilizado com mais frequência nos capítulos relacionados ao desenvolvimento dos estudos experimentais.

2.2 Ferramentas utilizadas

Conforme comentado anteriormente na Seção 1.1, o primeiro contato com a Plataforma SonarQube, aconteceu quando da realização de um estudo preliminar, com o objetivo de avaliar a evolução de produtos de código aberto com base em métricas de qualidade (VINCENZI et al., 2013). A princípio, não foi realizada uma investigação criteriosa sobre a existência de outras ferramentas com a mesma finalidade. Contudo, foram avaliadas as características e recursos oferecidos pela Plataforma, concluindo-se que estes atendiam, de forma satisfatória, às necessidades e expectativas para o desenvolvimento dos estudos experimentais propostos.

A Figura 2.1 apresenta a arquitetura empregada nos estudos experimentais, tendo a Plataforma SonarQube como núcleo. A descrição sobre a utilização das respectivas ferramentas, bem como as versões utilizadas, serão descritas na Seção 4.2. Apresenta-se, a seguir, uma síntese das principais características e funcionalidades dessas ferramentas, com informações constantes da documentação disponível em suas respectivas *homepages*.

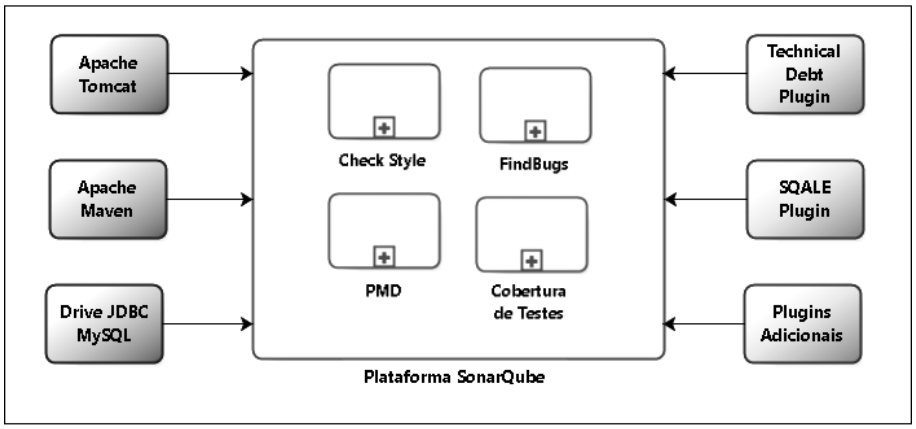

Figura 2.1: *Ferramentas empregadas nos estudos experimentais*

2.2.1 Plataforma SonarQube

O SonarQube[1] é uma plataforma de código aberto para gerenciamento da qualidade do código fonte, dedicada a analisar e medir continuamente a qualidade técnica a partir de um portfólio de projetos.

É uma aplicação WEB robusta, multilinguagem e com uma interface configurável, a qual permite o agrupamento e visualização de medidas e métricas sobre a qualidade do código fonte do projeto, considerando ainda sua ordem histórica de avaliação. Outra vantagem é a possibilidade de utilização de *plugins* que permitem acrescentar regras de avaliação e o cálculo de métricas avançadas, constituindo-se um poderoso mecanismo dessa ferramenta (SONARQUBE, 2014).

A Plataforma avalia a qualidade do código fonte de acordo com sete eixos: Arquitetura e Projeto *(Architecture and Design)*, Comentários *(Comments)*, Regras de Codificação *(Coding Rules)*, Erros Potenciais *(Potential Bugs)*, Complexidade *(Complexity)*, Testes Unitários *(Units Tests)* e Duplicações *(Duplications)*, conforme ilustrado na Figura 2.2.

A Plataforma integra ainda outras ferramentas de análise estática para avaliação do código fonte e posterior extração das métricas de software. Destacam-se abaixo algumas dessas principais ferramentas e uma breve descrição das mesmas:

- *CheckStyle*: é uma ferramenta de desenvolvimento *open source* que busca ajudar programadores a escrever código Java seguindo um padrão de codificação. Ela automatiza o processo de verificação de código Java para poupar os desenvolvedores dessa entediante (mas importante) tarefa. Essa ferramenta é altamente configurável,

[1]No final de 2013, a Plataforma Sonar alterou seu nome para SonarQube. Assim, optou-se por adotar a denominação atual, mesmo considerando que o trabalho foi realizado com uma versão anterior da Plataforma.

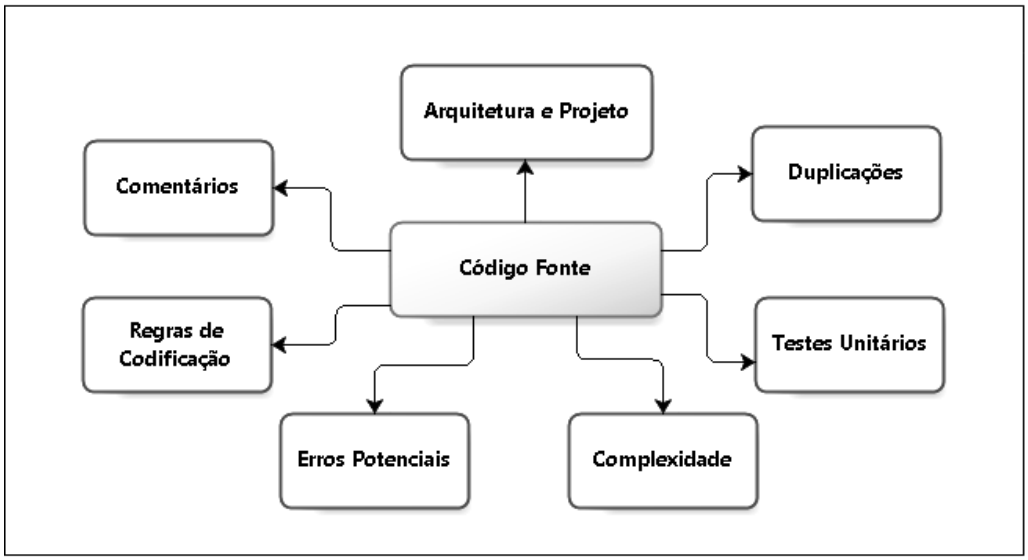

Figura 2.2: *Eixos de qualidade da Plataforma SonarQube, adaptada de (SONARQUBE, 2014)*

além de suportar a maioria dos padrões de codificação. Os avisos são classificados conforme seu nível de prioridade (CHECKSTYLE, 2014);

- *FindBugs*: é uma ferramenta *open source* de análise estática que visa identificar problemas de codificação e erros de programação em código Java. Ela é projetada para evitar a geração de avisos de falsos positivos, ou seja, *warnings*, que não correspondem a verdadeiros defeitos no código (FINDBUGS, 2014);
- *PMD*: é um analisador estático *open source*, compatível com as linguagens *JavaScript*, *Java*, *XML* e *XSL*, que analisa o código fonte buscando identificar falhas comuns de programação, tais como: variáveis não utilizadas, blocos vazios *(try/catch)*, código duplicado, criação de objetos desnecessários e outros. Os avisos relacionados a essas falhas de programação são categorizados em cinco níveis de severidade (PMD, 2014).

2.2.2 Apache Maven

O Apache Maven, ou simplesmente Maven, é uma ferramenta de gerenciamento de projetos de software que automatiza o processo de construção de projetos Java. Ela é similar à ferramenta *Ant*, mas é baseada em conceitos e trabalho diferentes, sendo capaz de gerenciar a construção do projeto, a elaboração de relatórios e a documentação a partir de uma peça central de informações (MAVEN, 2014).

O Maven é baseado no conceito de modelo de objeto do projeto e utiliza um arquivo *XML* (pom.xml) para descrever como o projeto de software deve ser construído, incluindo suas dependências sobre módulos e componentes externos, a ordem de compilação e os diretórios e *plugins* necessários. Durante sua execução, a ferramenta baixa as

bibliotecas e *plugins* dinamicamente, de um ou mais repositórios, e os armazena em uma pasta local (MAVEN, 2014).

Os produtos de código aberto selecionados para a realização dos estudos experimentais tiveram que atender alguns requisitos técnicos, entre eles o de que o produto deveria utilizar o Maven como ferramenta de construção, verificando-se a existência do arquivo pom.xml no diretório do código fonte.

2.2.3 Technical Debt Plugin

A Plataforma SonarQube utiliza o *Technical Debt Plugin* (TDP, 2014) para realizar o cálculo da DT do projeto. Primeiramente, são calculados os valores de cada um dos seis eixos de qualidade: Cobertura *(Coverage)*, Comentários *(Comments)*, Complexidade *(Complexity)*, Duplicações *(Duplications)*, Projeto *(Design)* e Violações *(Violations)*, considerando seus respectivos pesos pré-definidos. Em seguida, essas métricas são somadas para fornecer uma métrica global. A representação gráfica da métrica é ilustrada na Figura 2.3.

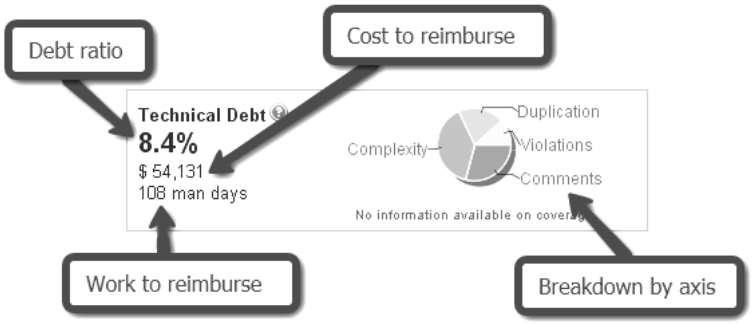

Figura 2.3: *Representação gráfica da DT pelo Technical Debt Plugin (TDP, 2014)*

Os componentes da representação gráfica da métrica são descritos abaixo:

- Proporção da dívida *(debt ratio)*: fornece o percentual da DT atual em relação à dívida total possível para o projeto;
- Custo para reembolsar *(cost to reimburse)*: oferece uma estimativa monetária necessária para corrigir todos os defeitos em cada eixo;
- Trabalho para reembolsar *(work to reimburse):* fornece o custo para reembolsar, expresso no esforço homens/dia, necessário para corrigir os defeitos;
- Repartição por eixos *(breakdown by axis)*: apresenta uma visão sobre a distribuição da dívida entre os seis eixos de qualidade, por meio de um gráfico de pizza.

A versão do *Technical Debt Plugin* (1.2.1) utilizada neste trabalho realiza o cálculo da dívida conforme demonstrado na Equação 2-1 abaixo:

$$DT = ccDUP + ccVIO + ccCMA + ccRTE + ctCPI + ccCNP \quad (2\text{-}1)$$

Onde:
DT = Dívida Técnica (em homens/dia)
$ccDUP$ = custo para corrigir um bloco * número de blocos duplicados
$ccVIO$ = custo para corrigir uma violação * número de violações (desconsiderando as violações informativas)
$ccCMA$ = custo para comentar uma API pública * número de APIs públicas não comentadas
$ccRTE$ = custo para cobrir um requisito de teste * número de requisitos de testes não cobertos (considera-se como objetivo atingir 80% de cobertura)
$ctCPI$ = custo para dividir um método * (distribuição da complexidade de função >= 8) + custo para dividir uma classe * (distribuição da complexidade de classe >= 60)
$ccCNP$ = custo para cortar uma aresta entre dois arquivos * peso das arestas do pacote

A Plataforma SonarQube permite que os custos associados a cada eixo sejam configurados pelo usuário e/ou adaptados segundo o critério da organização/empresa. Para este trabalho, foi utilizada a configuração padrão da Plataforma, com valores/pesos pré-estabelecidos, conforme descritos na Tabela 2.1. Após o cálculo dos custos e a multiplicação pelos respectivos pesos é possível expressar o valor da DT em termos do esforço homens/dias (considerando o dia com 8 horas). O valor resultante é, então, multiplicado por $500 (quinhentos dólares) para se obter o custo monetário da DT.

Tabela 2.1: *Custo padrão para corrigir cada componente da DT, adaptada de (TDP, 2014)*

Atividade	**Custo (em horas)**
Custo para corrigir um bloco (ccDU)	2
Custo para corrigir uma violação (ccVI)	0,1
Custo para comentar uma API (ccCM)	0,2
Custo para cobrir um requisito de teste (ccRTE)	0,2
Custo para dividir um método (ccCPI)	0,5
Custo para dividir uma classe (ccCPI)	8
Custo para cortar uma aresta entre dois arquivos (ccCNP)	4

Como mencionado anteriormente, optou-se por utilizar o cálculo da DT implementado pelo *Technical Debt Plugin*, com a configuração padrão da Plataforma, uma vez que não se constituía como um dos objetivos desse estudo avaliar e/ou questionar o modo como a DT é implementada pelo *plugin*, mas sim utilizar os valores coletados para avaliação e comparação da DT entre os diferentes produtos de código aberto selecionados.

A partir da versão 4.0 da (antiga) Plataforma Sonar, o *Technical Debt Plugin* foi descontinuado, mantendo-se na versão 1.2.1, e o cálculo da DT foi implementado no núcleo da Plataforma. Diante disso, a (nova) Plataforma SonarQube implementou uma versão comercial do método SQALE (que será apresentado logo adiante) para avaliar a dívida técnica. Entretanto, por uma questão técnica de tempo para execução do experimento controlado, optou-se por permanecer com a avaliação oferecida pelo *Technical Debt Plugin*, mantendo uma versão compatível da (antiga) Plataforma Sonar (3.7.2) para sua execução.

2.2.4 SQALE Plugin

A *Inspearit* (anteriormente chamada *DNV ITGS*) desenvolveu o método *SQALE (Software Quality Assessment based on Life-cycle Expectations)* para atender a uma necessidade genérica e permanente de avaliar a qualidade do código fonte (SQALE, 2014). Esse método foi projetado para medir e gerenciar a qualidade do código-fonte dos projetos a serem entregues, sendo aplicável a qualquer tipo de linguagem e a qualquer metodologia de desenvolvimento, além de ser direcionada para implementação automatizada. É publicado sob uma licença de código aberto e é livre de *royalties* (LETOUZEY; ILKIEWICZ, 2012).

O Modelo de Qualidade SQALE é usado para formular e organizar os requisitos não-funcionais que se relacionam com a qualidade do código. Ele é organizado em três níveis hierárquicos: o primeiro nível é composto de características, o segundo de subcaracterísticas e o terceiro é composto de requisitos que se relacionam com os atributos internos do código fonte. Estes requisitos geralmente dependem do contexto do software e da linguagem e qualquer violação destas exigências induz à DT (SQALE, 2014).

O método SQALE normaliza os relatórios resultantes das ferramentas de análise estática do código-fonte, transformando-os em custos de remediação e custos de não-remediação. Posteriormente, ele define as regras para agregar esses custos (TDESQALE, 2014).

O *Technical Debt Evaluation Plugin (SQALE)* é uma extensão comercial da Plataforma SonarQube, constituindo-se como uma implementação completa da metodologia SQALE, com base em regras e questões armazenada pela própria Plataforma SonarQube.

Essa implementação do método SQALE pela Plataforma Sonar, a qual será denominada nesse trabalho como *SQALE Plugin*, mede a dívida técnica de uma aplicação em termos do tempo que será necessário para corrigi-lo. Para cada regra (também conhecida como requisito não-funcional), existe uma estimativa do tempo que será gasto para corrigir o problema gerado a partir dessa regra. Essas regras são agrupados em características e sub-características ou áreas de interesse, tais como a testabilidade,

confiabilidade e eficiência, as quais estão em conformidade com a norma ISO/IEC 9126 (TDESQALE, 2014).

A partir da versão 4.0 da Plataforma SonarQube, os conceitos básicos do método SQALE passaram a fazer parte do núcleo da Plataforma, sendo que a dívida técnica dos produtos é calculada automaticamente em cada análise. O *SQALE Plugin* permite gerir o cálculo e a priorização da dívida técnica de maneira refinada, além de oferecer várias formas sofisticadas para categorizar a dívida, nos níveis macro e micro (TDESQALE, 2014).

2.3 Métricas de qualidade consideradas

As métricas de qualidade descritas abaixo estão entre as disponibilizadas pelo *Technical Debt Plugin* e pelo *SQALE Plugin*, integrados à Plataforma SonarQube. O que segue abaixo é uma breve descrição de cada uma das métricas, dos eixos de qualidade e do cálculo implementado, conforme informações constantes da documentação das ferramentas disponíveis em suas respectivas *homepages*. Também é estabelecida a forma como essas métricas foram utilizadas ao longo do presente estudo.

2.3.1 Technical Debt Ratio e seus eixos de qualidade

O cálculo da proporção da dívida *(Technical Debt Ratio)* corresponde basicamente ao valor da dívida em relação à Dívida Total Possível (DTP) para o projeto, multiplicado por 100 (TDP, 2014). A primeira coisa a ser calculada é o DTP para cada eixo e depois somá-las:

- *Cobertura* (CB): a dívida total possível é o custo de elevar a cobertura para 80%;
- *Comentários* (CM): a dívida total possível é o custo para comentar cada API pública;
- *Complexidade* (CP): a dívida total possível é o custo de dividir cada método e classe;
- *Duplicação* (DU): dados o número de blocos de duplicação e a porcentagem atual de linhas duplicadas, calcular o número de blocos que haveria se fosse 100%. A dívida total possível é o custo para corrigir esses blocos;
- *Projeto* (PR): dadas as arestas existentes entre os arquivos, a dívida total possível é o custo de ter que cortar todas elas (índice emaranhado de pacote = 100%).
- *Violações* (VI): dado o Índice de Observância das Regras (IOC) atual e o número de violações, calcular quantas violações seriam necessárias para trazer o IOC para zero. A dívida total possível é o custo para corrigir esses blocos;

2.3.2 SQALE Rating

O *SQALE Rating* é uma das métricas disponibilizadas pelo *SQALE Plugin*, integrado à Plataforma SonarQube. O método SQALE calcula a DT de um produto baseado nas regras e custos de remediação. Uma vez que a dívida é calculada, cada produto recebe uma classificação da sua "saúde", com base na densidade da sua dívida técnica. Isso representa, basicamente, o esforço necessário para corrigi-lo em relação ao esforço do nível em que ela já se encontra, ou seja, corresponde ao esforço de levar o projeto de um nível ao outro (TDESQALE, 2014).

A fórmula da densidade do SQALE corresponde ao *custo de remediação* dividido pelo *custo de desenvolvimento*. Essa fórmula pode ser reescrita considerando o *custo de remediação* dividido pelo *custo para desenvolver uma linha de código*, multiplicando o resultado pelo número de linhas de código. Outra abordagem da fórmula, dependendo da métrica escolhida, corresponderia ao *custo de remediação* dividido pelo *custo para desenvolver um ponto de complexidade*, sendo o resultado multiplicado pela respectiva complexidade.

Pode-se considerar, como exemplo, um software com 2.500 linhas de código e o custo de remediação total em 50 dias. É dado que o custo padrão para desenvolver uma linha de código é estimado em 0,06 dias. Então temos:

*50 / (0,06 * 2.500) = 0,33*

Os eixos de classificação padrão do SQALE são apresentados na Tabela 2.2. Deste modo, de acordo com a tabela, a classificação SQALE do software citado como exemplo seria "C". Na escala de classificação SQALE, fatores como o tempo para desenvolver uma linha de código fazem parte do Modelo de Governança SQALE. Esses fatores são configuráveis, mas apenas em nível global, uma vez que não é possível ajustar esses valores em nível de projeto.

Tabela 2.2: *Eixo de classificação padrão do SQALE (TDESQALE, 2014)*

Classificação	Intervalo de Valor
A	0,00 a 0,10
B	0,11 a 0,20
C	0,21 a 0,50
D	0,51 a 1,00
E	maior que 1,00

Considerando a "saúde" do projeto, associada à densidade da sua DT, é atribuída sua classificação, com valores escalares de "A" a "E" e seus respectivos níveis de coloração, conforme ilustrado na Figura 2.4. Para este trabalho, considerando a análise

dos resultados apresentada no Capítulo 5, foram utilizados apenas os valores da métrica *SQALE Rating*, por representar um indicador, mais geral, sobre a "saúde" do produto e a densidade da sua DT.

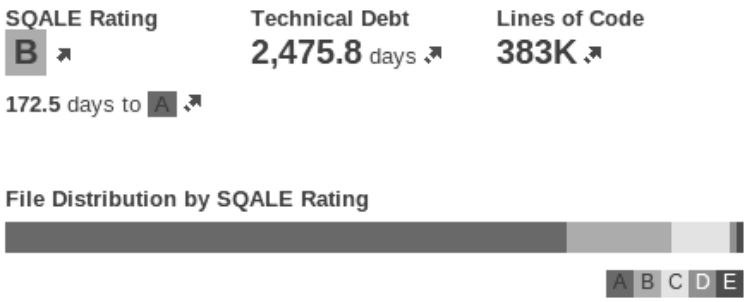

Figura 2.4: *Visualização da métrica SQALE Rating (TDESQALE, 2014)*

Durante o desenvolvimento dos estudos experimentais, descritos no Capítulo 4, considerou-se apropriado comparar a abordagem de avaliação da DT realizada pelo *Technical Debt Plugin* e pelo *SQALE Plugin*. Logo, para cumprir tal objetivo, foi avaliada a classificação oferecida pelo *SQALE Rating*, de forma a verificar a existência de alguma correlação entre os valores observados para essa métrica e o *Technical Debt Ratio*. Essa avaliação será apresentada no Capítulo 5.

2.4 Considerações Finais

Este capítulo abordou os principais conceitos, ferramentas e métricas utilizadas no presente trabalho, demonstrando, ainda, a perspectiva de como eles foram empregados no desenvolvimento dos estudos experimentais propostos. Entende-se que após esse capítulo é possível identificar com mais clareza o interesse em investigar a dívida técnica em produtos de código aberto.

Apesar de não se ter realizado uma ampla investigação sobre a existência de outras ferramentas com a mesma finalidade, buscou-se promover uma avaliação criteriosa sobre as características e recursos disponibilizados pela Plataforma SonarQube, incluindo seus *plugins* e ferramentas integradas, ao ponto de optar por sua utilização para o desenvolvimento dos estudos experimentais propostos.

Torna-se importante esclarecer que a Plataforma SonarQube foi núcleo da arquitetura de ferramentas empregadas neste trabalho, conforme ilustrado na Figura 2.1. Assim, a seleção das demais ferramentas necessárias ao desenvolvimento dos estudos experimentais necessitou observar os requisitos e características da Plataforma, bem como suas atualizações, o que, por um outro lado, pode estabelecer uma dependência tecnológica para a condução dos experimentos.

O capítulo seguinte apresentará a fundamentação teórica e os trabalhos relacionados, os quais servirão de base para o desenvolvimento dessa pesquisa. Além disso, o conteúdo do capítulo buscará reforçar o interesse em investigar a qualidade de produtos de código aberto, bem como de utilizar ferramentas automatizadas de análise estática para avaliar características de um grande volume de projetos.

CAPÍTULO 3

Dívida Técnica: Contexto e Trabalhos Relacionados

Neste capítulo é introduzido o conceito associado à metáfora da dívida técnica e a abordagem de alguns trabalhos relacionados, os quais serviram de base para o desenvolvimento dessa pesquisa. A Seção 3.1 traz informações sobre o surgimento e algumas compreensões da aplicação dessa metáfora no processo de desenvolvimento e manutenção de software. A Seção 3.2, por sua vez, destaca algumas pesquisas e estudos experimentais relacionados à identificação, avaliação e gerenciamento da dívida técnica no desenvolvimento de software. Por fim, a Seção 3.3 apresenta algumas considerações finais sobre o capítulo, além de introduzir o conteúdo do capítulo seguinte.

3.1 Contexto da Dívida Técnica

A metáfora da dívida técnica *(the metaphor of technical debt)* foi introduzida em 1992 por Ward Cunningham, quando a Wyatt Software lançava um sistema de gerenciamento de portfólios (WyCASH+) empregando a tecnologia de orientação a objetos, com o uso da linguagem *Smalltalk* (CUNNINGHAM, 1992). Na época, a empresa considerava que esse novo paradigma de programação orientado a objetos se adaptava melhor à diversidade e dinâmica do mercado e apresentava uma capacidade de resposta mais rápida e eficiente às necessidades de ajuste do produto.

Esse cenário direcionava a empresa a tomar uma consciente decisão de projeto, ao empregar uma nova tecnologia, pouco conhecida pelos desenvolvedores, mas que oferecia vantagem às características do negócio, mesmo considerando que isso poderia exigir reescrita futura de trechos do código. Assim, assumia-se uma "dívida", em função da sua conveniência ao projeto, mas com a necessidade e o compromisso de pagá-la, posteriormente, sob o risco dos "juros" crescerem a níveis incontroláveis (VIEIRA et al., 2013).

Existem algumas definições sobre o conceito da DT, dentre as quais optou-se por adotar a de Zazworka et al. (2013): "a dívida técnica descreve um equilíbrio entre obje-

tivos a curto e longo prazos no desenvolvimento de software, relacionando-se aos resultados obtidos, a curto prazo, pelo fato de adiar algumas atividades de desenvolvimento e às consequências, a longo prazo, relacionadas a essa decisão". Para esses autores, embora seja muito útil enquanto metáfora, a DT vai além ao inspirar um conjunto de ferramentas e métodos para apoiar sua identificação, mensuração, monitoramento, gestão e pagamento.

Segundo Kruchten, Nord e Ozkaya (2012) várias pessoas usam a metáfora da "dívida" para descrever muitos outros tipos de débitos e males do desenvolvimento de software, abrangendo amplamente tudo o que está no caminho da implementação, venda ou evolução de um sistema de software ou qualquer coisa que adicione desgaste aos esforços para seu desenvolvimento. Para eles, a generalização do uso do termo para outras questões do desenvolvimento de software tornam o conceito da DT diluído atualmente.

Já Tom, Aurum e Vidgen (2013) acreditam que, embora a DT seja considerada prejudicial para o sucesso do desenvolvimento de software, a longo prazo, ela parece ser mal compreendida na literatura acadêmica. Para eles, ainda é ausente uma clara definição e modelo para a DT, o que agrava o desafio de se obter sua adequada identificação e gerenciamento, além de restringir a percepção sobre sua utilidade como um instrumento de comunicação conceitual e técnico.

Esses autores conduziram um trabalho utilizando uma técnica de estudo de caso exploratório, que envolve revisão de literatura e entrevistas com profissionais de software e acadêmicos, para estabelecer os limites do fenômeno da DT, além de fazer um exame crítico e consolidar a compreensão sobre sua natureza e implicações para o desenvolvimento de software.

Para alguns autores (KRUCHTEN; NORD; OZKAYA, 2012) e (EISENBERG, 2012) a DT está diretamente relacionada ao contexto de evolução e manutenção existente no ciclo de vida de um produto de software. Eisenberg (2012), por exemplo, considera que a dívida passa a ser uma âncora na produtividade e manutenção do software e impede a capacidade de adicionar novas características eficientemente.

O autor considera que essa dívida pode resultar da tomada de decisões conscientes durante o projeto e a implementação, favorecendo a conveniência, mas que podem afetar negativamente a "saúde" do software a longo prazo. Ele destaca, ainda, que a dívida pode ocorrer também de forma mais presente, por meio de uma lenta decadência causada pelo fato de, repetidamente, remendar um sistema existente sem a devida refatoração.

A metáfora da DT também é empregada na metodologia de desenvolvimento ágil. Kruchten, Nord e Ozkaya (2012) consideram que algumas equipes de desenvolvimento ágil acreditam estar completamente imunes à DT pelo simples fato de usarem um processo de desenvolvimento iterativo. Contudo, o efeito pode ser contrário, uma vez que desenvolver e entregar rapidamente o produto, sem tempo para pensar em aspectos do

projeto, refletir sobre os custos a longo prazo ou adotar análises sistemáticas, incluindo testes automatizados, pode levar alguns projetos a rapidamente aumentarem suas DTs.

É perceptível na literatura que não há pretensão de eliminar a DT, mas sim mantê-la em um nível gerenciável, uma vez que ela é inerente à própria dinâmica do processo de desenvolvimento de software, especialmente à tomada de decisão (KRUCHTEN; NORD; OZKAYA, 2012). A metáfora traz o entendimento de que quando o projeto gerencia adequadamente a DT, ela pode ajudá-lo a alcançar seus objetivos mais cedo do que seria possível de outra forma. A gestão da DT envolve identificar, rastrear, tomar decisões fundamentadas e evitar seus piores efeitos. No entanto, o nível que um projeto ou empresa precisa para gerenciar a sua dívida depende claramente do seu contexto (LIM; TAKSANDE; SEAMAN, 2012).

Eisenberg (2012) entende que a metáfora da dívida técnica auxilia gerentes de desenvolvimento e engenheiros de software da mesma forma. Ela é útil para gestão porque provê uma forma mais concreta de, objetivamente, medir custos reais ou potenciais, além de servir como um indicador importante de custos futuros e riscos de programação, tal como atividades de integração pós-desenvolvimento.

3.2 Trabalhos Relacionados

Kruchten, Nord e Ozkaya (2012) estabeleceram uma forma de concepção, organização e visualização da DT que pode ser comparada ao processo de melhoria de software em um dado estado. O panorama sobre a DT proposto pelos autores faz uma distinção entre elementos visíveis, tais como adição de novas funcionalidades e correção de defeitos, e elementos invisíveis (ou visíveis apenas aos desenvolvedores), relacionados às lacunas tecnológicas e questões de arquitetura e codificação. Ainda sobre os elementos visíveis, os autores destacam questões de evolução e de qualidade, conforme apresentado na Figura 3.1.

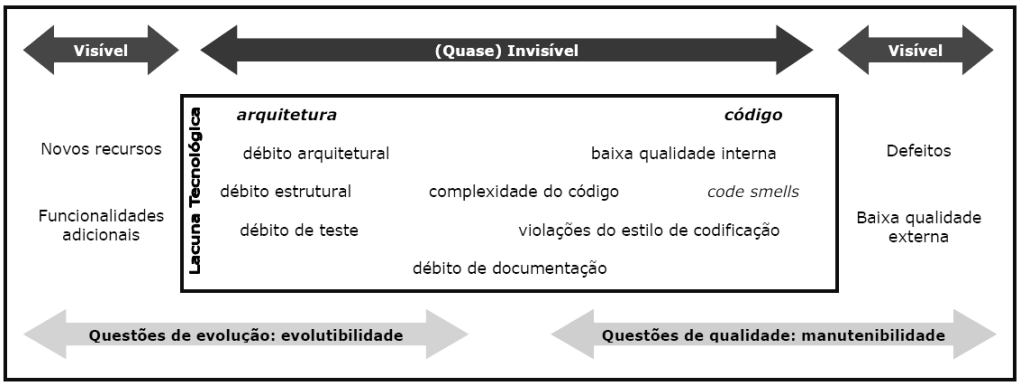

Figura 3.1: *Panorama da Dívida Técnica, adaptada de (KRUCHTEN; NORD; OZKAYA, 2012)*

Esses autores acreditam que a DT não pode ser tratada isoladamente no que se refere à adição de novas funcionalidades ou correções de defeitos, considerando apenas aspectos do código e sua qualidade, mas sim expressar todas as atividades do desenvolvimento de software, em termos de uma sequência de mudanças associadas a custos e valores sobre o tempo.

No estudo conduzido por Eisenberg (2012), a Plataforma SonarQube foi inicialmente selecionada para medir a DT, pelo fato de possuir o *Technical Debt Plugin*. Contudo, o autor considerou difícil aplicar, de maneira prática e significativa, a avaliação da DT oferecida pela Plataforma. Assim, apesar de considerar que ela apresentava uma interface robusta e outras características desejáveis, ele preferiu não empregar o cálculo da DT implementado pelo *plugin*, mas utilizou-se das métricas oferecidas pela Plataforma para calcular componentes da DT.

O trabalho citado acima é significativo, pois estabelece um conjunto de métricas importantes, bem como limites específicos aos projetos para definir níveis gerenciáveis para os eixos da DT. O autor definiu níveis (ou limiares) para cada eixo de qualidade, conferindo um *status* para cada nível – verde/amarelo/vermelho – partindo do gerenciável ao crítico. Para determinar o *status* da dívida foi necessário estabelecer prioridades no seu cálculo. O objetivo era reduzir a dívida ao *status* verde, representando um nível aceitável. Isso permitiu estabelecer parâmetros para avaliação e priorização de eixos da DT no sistema analisado.

Siebra et al. (2012) afirmam que o estado da arte sobre a DT ainda não culminou em rigorosos modelos de análise para projetos em larga escala. Considerando isso, os autores analisaram um projeto industrial sob a perspectiva das decisões de projeto e os eventos relacionados, para então caracterizar a existência da DT e mostrar a evolução dos seus parâmetros.

O trabalho dos autores teve como objeto de estudo o sistema SMB *(Samsung Mobile Business)*, um aplicativo comercial para dispositivos móveis que pode ser executado em diferentes plataformas *Samsung* dessa natureza. Uma característica importante daquele sistema era possuir um elevado número de modificações, sendo conveniente aplicar a abordagem da DT. O método empregado nesse trabalho consistiu da identificação, coleta de dados e análise de três cenários de decisão, que foram tidos como uma abordagem adequada para caracterização da evolução história da DT. A avaliação da DT teve como foco a análise dos cenários e as decisões de projeto associadas a eles.

3.2.1 Utilização da Análise Estática Automatizada

Ferramentas de análise estática oferecem suporte à identificação e análise da DT, uma vez que possibilitam a automatização do processo de verificação do código. Contudo,

Kruchten, Nord e Ozkaya (2012) acreditam que essa abordagem pode levar ao perigo de desconsiderar grande quantidade de DT potencial, não detectadas por essas ferramentas, tal como lacunas tecnológicas existentes. De igual forma, Zazworka et al. (2013) defendem que ferramentas de análise estática automatizada podem apoiar a identificação de alguns tipos de dívida, mas outros tipos necessitam da avaliação humana e manual para serem encontradas. Entende-se, assim, que esses posicionamentos demonstram a necessidade de abordagens complementares para identificação e gerenciamento da DT.

Zazworka et al. (2013) conduziram um estudo interessante com foco na identificação da dívida técnica. Para tanto, realizaram uma avaliação da elicitação humana da DT e a compararam com a identificação obtida por ferramentas de análise estática, com dados coletados por meio de três ferramentas de identificação automática da DT. Já a elicitação humana da dívida foi conduzida por meio de *template* da DT e um questionário entregue à equipe de desenvolvimento.

Como resultado desse trabalho, observou-se que há pouca sobreposição entre a DT relatada pelos diferentes desenvolvedores que colaboraram com a pesquisa. Assim, ao invés de consenso, considerou-se mais apropriado combinar esses múltiplos relatos complementares obtidos. Quanto às ferramentas automatizadas de análise estática, percebeu-se que elas são úteis para identificar a DT de defeito, mas não podem ajudar a identificar outros tipos de dívida. Diante disso, os autores consideraram necessário envolver a percepção humana no processo de identificação, assumindo-a como uma abordagem que favorece o aperfeiçoamento da avaliação realizada por ferramentas automatizadas de análise estática.

Conforme destacado no trabalho de Zazworka et al. (2013), abordagens humanas e manuais para identificação da DT podem ser mais demoradas, mas apresentam duas vantagens (ao menos na teoria) em relação a abordagens automatizadas: a primeira é que elas podem ser mais exatas e propensas a identificar DT que são mais significativas e a segunda é que atores humanos são capazes de prover informações contextuais adicionais importantes a cada instância de DT, as quais seriam praticamente impossíveis de serem recolhidas automaticamente.

Uma das contribuições do estudo, destacada pelos autores, relaciona-se ao fato de terem sido comparados vários tipos de ferramentas automatizadas que apoiam a identificação da DT. Segundo eles, algumas pesquisas recentes têm direcionado a questão de como abordagens automatizadas podem aproximar-se dos resultados da identificação manual da DT. Alguns desses estudos indicam que é possível identificar certas classes potenciais de DT (dívida de *design*, por exemplo) com métodos assistidos por computador.

Vetro' (2012) conduziu um estudo com foco no uso da análise estática automatizada para identificar a DT, a nível de código, levando em consideração diferentes dimensões de qualidade, tais como: funcionalidade, eficiência e manutenção. Para o autor,

avaliar o impacto da análise estática automatizada nas diferentes dimensões de qualidade permite aos desenvolvedores e administradores gerenciar melhor a DT, bem como detectar anomalias no sistema que impactam negativamente em aspectos específicos de qualidade. Para ele, as atividades de manutenção poderiam ser priorizadas de acordo com as dimensões de qualidade mais degradadas ou com aquelas que impactam mais profundamente no propósito do software.

Conforme já comentado no Capítulo 1, foram desenvolvidos, no contexto do Grupo de Teste de Software do INF/UFG, alguns estudos iniciais utilizando a análise estática automatizada para avaliar a qualidade de software. Um desses estudos preliminares buscou investigar a evolução de produtos de código aberto com base em métricas de qualidade, as quais foram obtidas por meio da análise estática automatizada do código fonte (VINCENZI et al., 2013). Para tanto, foram selecionados cinco projetos *open source* e três de suas versões disponíveis - uma inicial, uma intermediária e a última versão na época do estudo.

Esses projetos, e suas respectivas versões, foram submetidos à avaliação da Plataforma SonarQube. Priorizou-se a análise de três métricas de qualidade, dentre as disponibilizadas pela Plataforma: *Quality Index*, *Total Quality* e *Technical Debt*, com a intenção de avaliar a evolução desses produtos conforme os valores observados para cada projeto/versão. Esse estudo possibilitou um primeiro contato com o conceito e implementação da DT.

Por se tratar de um estudo exploratório, composto por uma pequena amostra de produtos de código aberto, observou-se um comportamento diferenciado para cada projeto, além de sugerir que, entre as três métricas, o *Total Quality* era o mais sensível para detectar alterações nos projetos. Diante disso, a pesquisa sugeriu a realização de alguns trabalhos futuros, no sentido de ampliar a base de dados, com um conjunto maior de projetos e, assim, verificar se é possível identificar tendências na evolução dos valores dessas métricas. Outra sugestão foi encontrar uma métrica que melhor indique as alterações e/ou falhas no projeto, no sentido de alertar a equipe de desenvolvimento sobre áreas do código fonte que necessitam de maior atenção.

A partir desse trabalho, direcionou-se a atenção para a DT, com a perspectiva de observar dados de um conjunto maior de projetos de código aberto. Isso resultou em um trabalho para avaliação da DT em um conjunto de quarenta projetos de código aberto, utilizando o código fonte da versão mais recente na época do estudo, no intuito de observar o estado e propor uma classificação, ainda que preliminar, dos projetos em relação à DT, além de identificar quais dos eixos de qualidade são mais representativos na composição da DT (VIEIRA et al., 2013). Esse estudo observou uma concentração significativa da DT nos eixos de qualidade *Complexidade* e *Cobertura*, representando quase 50% (cinquenta por cento) da composição da DT dos projetos avaliados.

Como questões a serem abordadas em trabalhos futuros, para aquele estudo, destacou-se a necessidade de, novamente, ampliar a base de dados, reforçar a análise estatística para avaliação da composição da DT, além de avaliar o histórico de evolução da dívida técnica para versões distintas de produtos de código aberto. Essas proposições para trabalhos futuros constituem algumas das motivações para realização do presente estudo.

Na Tabela 3.1, apresenta-se um quadro resumo com os principais trabalhos relacionados, bem como a proposta de utilização, expansão e/ou complementação dos mesmos no desenvolvimento do presente estudo.

Tabela 3.1: *Quadro Resumo dos principais trabalhos relacionados*

Autor(es)	**Características do estudo**	**Proposta de expansão do estudo**
Kruchten, Nord e Ozkaya (2012)	Estabelece um panorama para concepção, organização e visualização da DT que pode ser comparada ao processo de melhoria de software.	Considerando os resultados obtidos com o desenvolvimento dos estudos experimentais propostos, espera-se estabelecer um panorama inicial sobre o estado e evolução da DT em produtos de código aberto.
Eisenberg (2012)	Estabelece um conjunto de métricas importantes, bem como limites específicos aos projetos, para definir níveis gerenciáveis para os eixos da DT.	Sugerir uma classificação para a DT, considerandos os valores dos próprios projetos avaliados, de forma a estabelecer limiares que auxiliem na definição de patamares aceitáveis e/ou críticos para a DT.
Zazworka et al. (2013)	Estudo com foco na identificação da DT, comparando a elicitação humana e manual com a identificação automatizada, obtida com ferramentas de análise estática. Reconhece a necessidade de envolvimento humano no processo de identificação de determinados tipos de DT.	Utilizar a abordagem de identificação automatizada da DT, para avaliar seu estado, características e evolução sobre o conjunto de produtos de código aberto selecionado.
Siebra et al. (2012)	Analise de um projeto industrial, sob a perspectiva das decisões de projetos e dos eventos relacionados para caracterizar e existência da DT e apresentar a evolução dos seus parâmetros, conforme os cenários associados.	Realizar uma avaliação da DT, em larga escala, para um conjunto significativo de produtos de código aberto, de forma a observar seu comportamento e tendências evolutivas ao longo do tempo.
Vieira et al. (2013)	Avalia a DT para um conjunto de quarenta projetos de código aberto, considerando sua versão mais recente na época, com o objetivo de verificar o estado, sugerir uma classificação e identificar quais eixos de qualidade são mais representativos na composição da DT.	Desenvolver uma proposta de expansão do trabalho, considerando uma quantidade significativa de projetos e algumas de suas versões disponíveis, com o intuito de complementar os resultados obtidos anteriormente.

Os estudos listados acima e a proposta de utilização dos mesmos demonstram a relevância e contribuição do tema para a investigação da qualidade de produtos de código aberto, sob a perspectiva de avaliação da DT e seus eixos de qualidade. Torna-se

importante destacar que, no momento do desenvolvimento deste trabalho, considerando o levantamento bibliográfico realizado, não se observou na literatura trabalhos similares a este, especialmente quando se considera a avaliação de uma quantidade significativa de projetos de código aberto e sua relação com a DT.

3.2.2 Interesse em investigar Produtos de Código Aberto

Os projetos da 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. Portanto, a liberdade dos chamados produtos de código livre e aberto 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 software livre é uma atividade contínua, caracterizada pela frequente disponibilização e evolução de versões. Essa propriedade foi chamada por Raymond (1999) como *"release early, release often"*. A evolução e manutenção dos projetos são assegurados pela comunidade de desenvolvimento, a qual é constituída de desenvolvedores voluntários espalhados pelo planeta, que trabalham em módulos específicos do projeto pelo prazer de trabalhar em algo de que gostam (MISHRA et al., 2002).

Meirelles (2013) destaca, citando outros autores, que no início parte do desenvolvimento de software livre seguia um tipo de abordagem de desenvolvimento denominada, por Raymond (1999), como *"bazar"*, sugerindo a não existência de qualquer processo de desenvolvimento com um nível maior de Engenharia de Software, em oposição ao estilo de mercado mais formal denominado, também por Raymond (1999), como *"catedral"*. Contudo, é notório, especialmente a partir do século XXI, um crescente avanço e movimentação em torno do desenvolvimento de software livre de forma mais estruturada (GOUSIOS et al., 2007), bem como a preocupação com a qualidade e confiabilidade de tais produtos por parte de grandes empresas privadas (PEZUELA et al., 2010).

Conforme comentado no Capítulo 1, considera-se importante o desenvolvimento de novas abordagens para avaliação da qualidade de produtos de código aberto. Nesse sentido, entende-se que o estudo aqui proposto pode contribuir com a investigação sobre a qualidade desses produtos, com foco na qualidade do código fonte produzido, bem como estabelecer um panorama sobre a DT na comunidade de software livre.

3.3 Considerações Finais

Este capítulo procurou sintetizar alguns estudos existentes e contribuir para o entendimento do conceito e aplicação da DT. Apesar de alguns autores (ZAZWORKA et al., 2013) e (KRUCHTEN; NORD; OZKAYA, 2012) demonstrarem sua preocupação quanto à utilização de ferramentas automatizadas de análise estática para identificação da DT, é perceptível na literatura o uso dessa abordagem em várias situações, mesmo considerando algumas limitações e reconhecendo a necessidade de uma investigação complementar para cobrir a identificação da DT não percebida por essas ferramentas.

O resumo dos trabalhos relacionados apresentados na Seção 3.2 permite perceber o diferencial e as contribuições deste trabalho no que se refere à investigação da qualidade de produtos de código aberto, considerando o estado, evolução e características da DT desses produtos, conforme os objetivos já apresentados na Seção 1.2.

No capítulo seguinte, serão apresentados os estudos experimentais conduzidos, com o detalhamento do processo de experimentação, a descrição de cada uma de suas etapas, o seu ambiente operacional e as características dos projetos de código aberto avaliados.

CAPÍTULO 4

Estudo Experimental

Neste capítulo é descrita a concepção e características dos estudos experimentais conduzidos neste trabalho. A Seção 4.1 aborda o processo de experimentação, conforme proposto por Wohlin et al. (2012), com a definição e implementação de algumas de suas etapas. A Seção 4.2 apresenta a descrição do experimento controlado realizado, com informações relacionadas ao ambiente operacional, os mecanismos de automatização utilizados e as características dos produtos de código aberto avaliados. E, por fim, a Seção 4.3 traz algumas considerações finais pertinentes do capítulo, além de introduzir o conteúdo do capítulo seguinte, relacionado à análise e interpretação dos dados coletados.

4.1 Processo de Experimentação

Wohlin et al. (2012) consideram que a experimentação não é uma tarefa simples, pois envolve preparar, conduzir e analisar experimentos corretamente. Os autores destacam como uma das principais vantagens da experimentação o controle dos sujeitos, objetos e instrumentação, o que torna possível extrair conclusões mais gerais sobre o assunto investigado. Outras vantagens incluem a habilidade de realizar análises estatísticas utilizando métodos de teste de hipóteses e oportunidades para replicação.

Sendo assim, é necessário utilizar-se de um processo de suporte para permitir que os experimentos sejam realizados corretamente. O principal objetivo de um experimento é, em grande parte, avaliar uma teoria, hipótese ou relação (causa e efeito). Para os autores, um processo provê o suporte necessário para construir e conduzir um experimento, uma vez que ele pode ser usado como *checklists* e orientações de "o que fazer" e "como fazer".

Wohlin et al. (2012) entendem que, inicialmente, é importante perceber se um experimento é apropriado para a questão que se deseja investigar. Os autores dividem o processo de experimentação em cinco atividades principais: **escopo** (define o experimento em termos de problema, objetivos e metas), **planejamento** (determina o projeto do experimento, considera a instrumentação e avalia as ameaças ao experimento), **operação** (o experimento segue o que foi previsto no projeto), **análise e interpretação** (analisa e avalia os dados mensurados e coletados) e **apresentação e pacote** (apresenta os resultados

em um pacote que permite uma possível replicação). A ordem das atividades no processo, a princípio, indica a sequência de início das atividades, mas não necessariamente sua ordem de execução, uma vez que algumas etapas permitem um processo interativo, com retornos e refinamentos. A única exceção que os autores destacam quanto à impossibilidade de retorno é quando a etapa de operação é executada. As etapas do processo de experimentação são ilustradas na Figura 4.1.

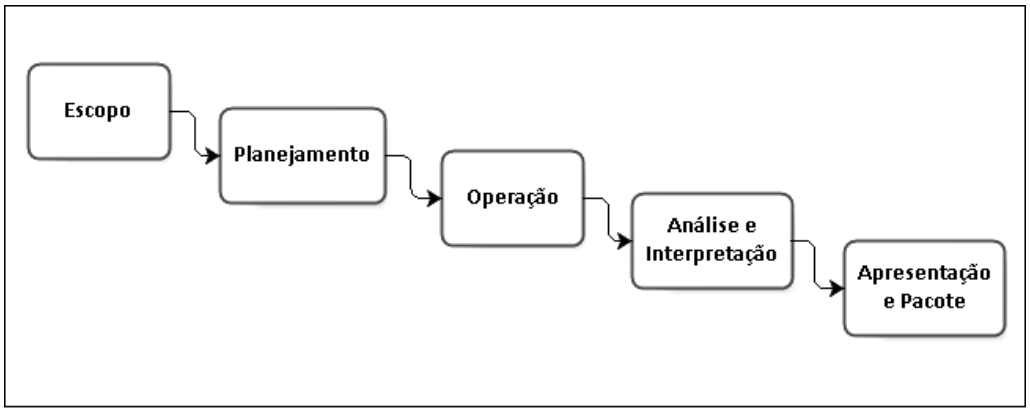

Figura 4.1: *Visão Geral do Processo de Experimentação, adaptada de (WOHLIN et al., 2012)*

A seguir, são descritas as etapas do processo de experimentação, conforme proposto por Wohlin et al. (2012), e como algumas dessas etapas foram utilizadas e/ou adaptadas para os estudos experimentais em questão. É importante destacar que as informações apresentadas abaixo são complementadas em outros capítulos e/ou seções deste estudo: boa parte das questões do escopo estão são descritas no Capítulo 1; o planejamento e a operação são detalhados com mais propriedade na Seção 4.2; e as etapas de análise e interpretação e de apresentação e pacote são analisadas no Capítulo 5.

4.1.1 Escopo

Nessa etapa inicial, a hipótese deve estar clara e os objetivos e metas do experimento bem definidos. As seguintes perguntas devem ser respondidas, no intuito de definir o escopo do estudo experimental:

1. **O que será estudado?** - A dívida técnica (DT) em produtos de código aberto;
2. **Qual o propósito?** - Avaliar o estado, a evolução e as características da DT para um conjunto de produtos de código aberto;
3. **Qual o foco do estudo?** - Verificar a evolução da DT sobre esses produtos, propor uma classificação da DT (com base nos dados observados) e verificar qual(is) eixo(s) de qualidade é(são) mais representativo(s) na composição da DT;
4. **Qual a perspectiva?** - Verificar se a qualidade desses produtos de código aberto, do ponto de vista dos valores observados para DT, apresenta uma tendência de

evolução (positiva ou negativa) ou, até mesmo, de estabilização, refletindo em sua qualidade e nos aspectos relacionados à gestão da DT;

5. **Onde o estudo será realizado?** - O estudo será realizado em um ambiente controlado (laboratório), utilizando-se de ferramentas automatizadas de análise estática para coleta dos dados e de métodos estatísticos para análise dos resultados.

4.1.2 Planejamento

Constitui a base do experimento, sendo considerado uma etapa crucial do processo de experimentação, uma vez que busca garantir que os resultados obtidos tornem-se úteis. Para essa etapa foram observados os seguintes pontos:

1. **Definir detalhadamente o contexto do experimento:** foi selecionado o conjunto de produtos de código aberto, definido quanto à utilização do *Technical Debt Plugin* da Plataforma SonarQube, para avaliação do código fonte, e verificadas as métricas relacionadas à DT para serem utilizadas;
2. **Formular as hipóteses do experimento, incluindo hipótese nula (HN) e hipótese alternativa (HA):**
   - HN_0 – Os produtos de código aberto selecionados não apresentam tendência de evolução da DT.
   - HA_1 – Os produtos de código aberto apresentam tendência de evolução negativa da DT ao longo do tempo (das versões em ordem histórica).
   - HN_2 – Os eixos de qualidade apresentam a mesma representatividade na composição da DT dos projetos avaliados.
   - HA_3 – Os eixos de qualidade não apresentam a mesma representatividade na composição da DT dos projetos avaliados, ou seja, a dívida pode se concentrar em determinado(s) eixo(s) de qualidade.
3. **Determinar as variáveis independentes *(inputs)* e dependentes *(outputs)*:** métricas relacionadas aos cálculo da DT: proporção, custo (valor monetário e tempo) e eixos de qualidade da DT. Neste caso, as variáveis dependentes são as características de evolução observadas para os produtos e a proposta de níveis de classificação da DT;
4. **Identificar os sujeitos do estudo:** um conjunto de produtos de código aberto e suas respectivas versões disponíveis;
5. **Projetar o experimento:** planejamento do ambiente operacional, com a instalação dos recursos e ferramentas necessários, e preparação da lista com as URLs dos repositórios dos projetos a serem avaliados;
6. **Preparar a instrumentação do experimento:** elaboração dos *scripts* de automação do experimento;

7. **Validar os resultados:** conferência visual dos dados e aplicação de métodos de análise estatística para verificação dos resultados.

4.1.3 Operação

Esta etapa é constituída de três atividades, as quais se observam:

1. **Preparar os sujeitos e os materiais necessários**: conforme planejado na etapa anterior, foram utilizadas as listas com as URLs dos repositórios dos projetos, no intuito de obter diversas versões *(releases)* do respectivo código fonte dos projetos para avaliação;
2. **Executar (garantir que o experimento será conduzido de acordo com o plano e projeto do experimento):** os *scripts* foram executados de forma a obter algumas versões do código fonte, compilá-lo, submetê-lo à avaliação da Plataforma SonarQube e realizar a coleta dos dados obtidos para as métricas desejadas;
3. **Validar os dados (tentar certificar que os dados coletados estão corretos e prover um quadro de validação do experimento):** os dados foram apresentados em tabelas para conferência manual, por amostragem, de forma a verificar se eles correspondiam aos valores apresentados no relatório gráfico disponibilizado pela Plataforma, bem como se apresentavam redundâncias e/ou inconsistências.

4.1.4 Análise e Interpretação

A coleta de dados durante a etapa de operação fornece as entradas necessárias para essa atividade. Uma importante atividade dessa etapa é a interpretação, uma vez que permite a tomada de decisões e a abstração de conclusões sobre como usar os resultados do experimento, incluindo as motivações para estudos posteriores. Algumas atividades são sugeridas para essa etapa, as quais resultam em:

1. **Analisar os dados por meio de estatística descritiva:** tabelas e gráficos foram elaborados e técnicas de análise estatística foram aplicadas, de forma a obter a melhor interpretação para o conjunto de dados disponíveis, como o objetivo de validar as hipóteses formuladas na etapa de planejamento;
2. **Considerar a possibilidade de redução do conjunto de dados:** foram verificadas se existiam inconsistências e/ou redundância dos projetos, métricas ou dados coletados. Os dados foram organizados de acordo com a análise estatística a ser utilizada para cada abordagem de avaliação, conforme descrito na Seção 5.1.

4.1.5 Apresentação e Pacote

Inclui, primeiramente, a documentação dos resultados, a qual pode ser feita por meio de um artigo de pesquisa para publicação, um pacote de laboratório para replicação ou parte de um relato de experiência da indústria. Um experimento nunca fornecerá a resposta final para uma questão, por este motivo é importante facilitar a replicação do experimento e certificar que as lições aprendidas serão atendidas de forma apropriada.

Os resultados obtidos com o experimento controlado proposto foram utilizados como parte deste trabalho, além de serem apresentados em forma de artigo científico, em uma versão resumida.

4.2 Descrição do Experimento Controlado

O experimento controlado conduzido nesse trabalho consistiu no planejamento, preparação, execução, análise e apresentação dos dados relacionados à dívida técnica em produtos de código aberto, segundo avaliação do *Plugin Technical Debt* da Plataforma SonarQube.

Conforme apresentado na Seção 1.1, este trabalho buscou avaliar a DT em produtos de código aberto, por meio de métricas baseadas em código fonte, de forma a observá-la segundo seus eixos de qualidade: Cobertura *(Coverage)*, Comentários *(Comments)*, Complexidade *(Complexity)*, Duplicações *(Duplications)*, Projeto *(Design)* e Violações *(Violations)*.

Os produtos de código aberto, objeto de avaliação, foram obtidos em dois repositórios de controle de versão (SVN e GIT), cujas URLs estavam disponíveis na página da internet do ProjetoNemo (2013) e GitApache (2013), respectivamente. A opção por verificar a situação da DT em projetos de código aberto deu-se, inicialmente, pela facilidade de acesso ao código fonte, bem como pelo interesse em propor, ainda que de forma preliminar e exploratória, uma avaliação do estado, evolução e características da DT para projetos dessa natureza.

Assim, foram selecionados 276 produtos de código aberto (mais precisamente as URLs dos respectivos repositórios) que deveriam atender aos seguintes critérios de inclusão: i) utilizar o Maven como ferramenta de construção, ii) ser escrito em linguagem Java; e iii) possuir casos de testes unitários. Essas restrições técnicas foram impostas por serem essenciais ao cômputo da cobertura dos testes unitários, necessária para o cálculo da DT pela Plataforma SonarQube.

O ambiente operacional projetado para execução do experimento possuía as seguintes características e/ou configurações:

- Sistema Operacional: Linux - Distribuição: Ubuntu 13.04;

- Servidor de Aplicação: Apache Tomcat (versão 7.0.42);
- Banco de Dados (base local): MySQL (versão 5.5.32);
- Java JDK (versão 1.7.0_15);
- SonarQube, na época, Sonar (versão 3.7.2);
- Apache Maven 2 (versões 2.2.1 e 3.1.1);
- Technical Debt Plugin (versão 1.2.1);
- SQALE Plugin (versão 1.10).

Os procedimentos necessários à compilação e execução dos casos de teste dos projetos foram automatizados por meio dos *script-snv* e *script-git*, apresentados no Apêndice B – ScriptSVN B.1 e B.2 e ScriptGIT B.3 e B.4 – seguindo os procedimentos descritos abaixo:

1. Buscava-se acesso ao repositório do projeto, por meio da URL fornecida;
2. Com o devido acesso ao código fonte, verificava-se a partir de qual versão/revisão o projeto passou a utilizar a ferramenta Maven para automação da construção e gerenciamento do projeto;
3. A partir desse ponto (considerando somente as versões que possuíam o arquivo `pom.xml` – característico da ferramenta Maven) os *scripts*, conforme o caso, verificavam, entre as revisões disponíveis, se determinada versão do projeto apresentava construção com sucesso para os procedimentos de compilação do código fonte e execução dos casos de teste. As versões/revisões que apresentavam falhas eram descartadas e iniciava-se o processo para a próxima versão/revisão disponível;
4. Posteriormente, considerando uma construção com sucesso no procedimento anterior, o respectivo *script* submetia a versão do projeto à avaliação da Plataforma SonarQube, verificando, assim, se ele apresentava sucesso para implantação e coleta de dados das métricas. Após, seguia-se para uma nova versão ou um novo projeto a ser avaliado.

Em uma etapa seguinte foi realizada a extração dos dados obtidos, também de forma automatizada, utilizando-se do *script-coleta-dados*, apresentado no Apêndice B – ScriptColeta B.5, B.6, B.7 e B.8 –, tendo por procedimento o acesso à base de dados local. Para tanto, utilizou-se uma API fornecida pela Plataforma SonarQube, que retornava os dados desejados em um arquivo com extensão *.csv*, para serem visualizados em forma de tabelas. A descrição do passo a passo das etapas do processo de automatização do experimento controlado é ilustrada no fluxograma da Figura 4.2.

Considera-se importante destacar que esses estudos experimentais trabalharam com 276 URLs, correspondentes a repositórios de produtos de código aberto. O processo

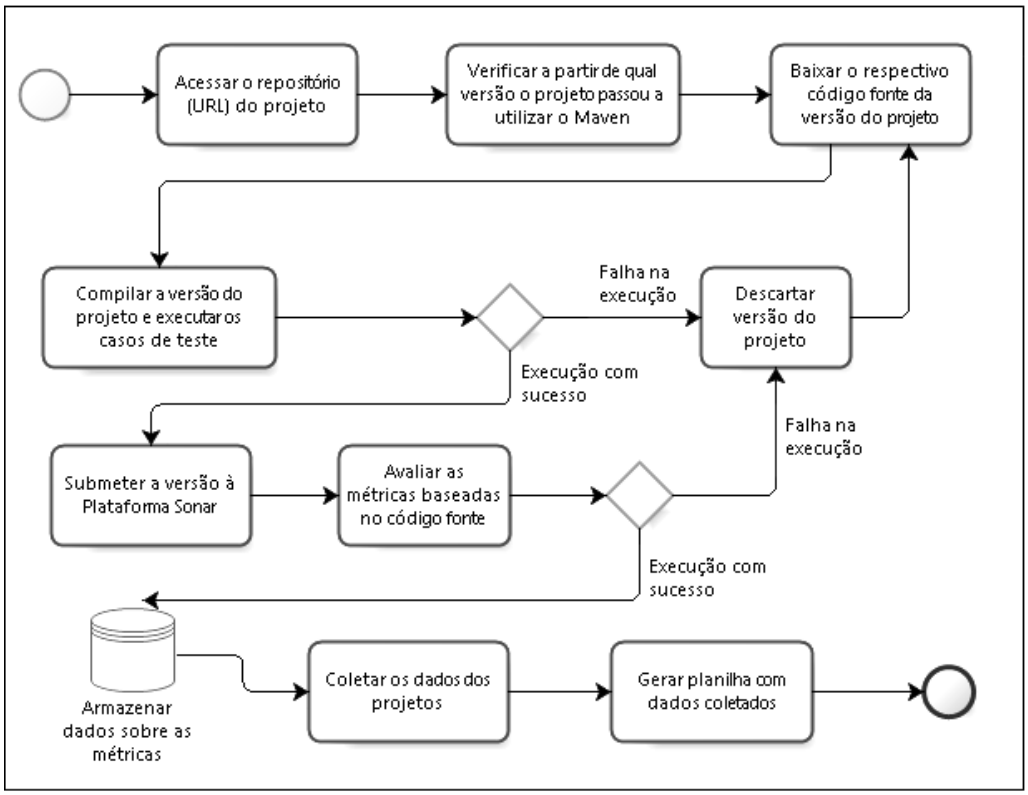

Figura 4.2: *Fluxograma de execução do experimento controlado*

de execução dos *scripts*, incluindo a coleta de dados para os projetos avaliados pela Plataforma SonarQube, exigiu um grande esforço de tempo, necessitando de aproximadamente 5 (cinco) meses do cronograma da pesquisa para sua conclusão.

Assim, ao final da execução do experimento controlado, do total de projetos selecionados, aproximadamente 51% (140 projetos) apresentaram sucesso na execução dos procedimentos implementados nos respectivos *scripts* e os outros 49% apresentaram falhas, seja por erro na URL, de compilação ou de implantação na Plataforma SonarQube. Os projetos que apresentaram falhas foram descartados, para fins deste estudo, e separados para uma investigação futura mais detalhada. A relação dos projetos que apresentaram falha está disponível no Apêndice A – Tabela A.2 – e os resultados da execução do experimento serão detalhados no Capítulo 5.

Com o objetivo de avaliar a evolução histórica da DT para produtos de código aberto, os procedimentos automatizados pelos respectivos *scripts* foram executados para cada uma das versões dos projeto selecionados, a partir da primeira versão do projeto que passou a utilizar a ferramenta Maven. Ao final da execução do experimento, a base de dados armazenou dados de 985 versões, correspondente a 140 projetos avaliados. Isso representa uma massa de dados significativa, sobretudo se for considerado as métricas a serem avaliadas, o que justifica a utilização de métodos de estatística multivariada para respectiva análise dos dados, a qual será descrita no Capítulo 5.

4.3 Considerações Finais

Este capítulo apresentou o processo de experimentação empregado neste trabalho, destacando sua concepção e características. Também foram descritos os estudos experimentais conduzidos, destacando as características do ambiente operacional utilizado, dos produtos de código aberto avaliados e do processo de automatização implementado.

Um aspecto importante a ser destacado consiste em que este trabalho, especialmente os estudos experimentais conduzidos, apoiaram-se nos recursos disponibilizados pela Plataforma SonarQube, com seus *plugins* e ferramentas agregadas. Contudo, conforme já mencionado na Seção 2.2, algumas dessas ferramentas sofreram modificações e atualizações no decorrer do experimento, o que levou à necessidade de fixar tanto uma configuração padrão (descartando as atualizações que surgiam), bem como modificar a perspectiva de coleta, análise e interpretação dos dados.

Essa decisão teve como aspecto fundamental as limitações de tempo para a finalização dos estudos experimentais e, consequentemente, da dissertação. Registra-se o agradecimento ao *SonarSource Team* que gentilmente cedeu uma licença do *SQALE Plugin*, para fins acadêmicos de desenvolvimento dessa pesquisa, pelo período de um ano, possibilitando uma melhor exploração e aproveitamento dos recursos oferecidos pela Plataforma e seus *plugins*.

É importante também esclarecer que a intenção inicial foi automatizar o processo de obtenção e avaliação do código fonte, assim como a coleta dos dados, sem se preocupar, inicialmente, com os motivos das falhas para esse conjunto de projetos, uma vez que essa atividade não fazia parte do escopo do presente trabalho. Para os projetos que apresentaram sucesso, foi obtida uma quantidade variada de versões para avaliação individual e em conjunto. Os dados foram, então, organizados de forma a permitir análises estatísticas específicas para cada conjunto de dados.

O capítulo seguinte abordará a análise e interpretação dos dados obtidos, de forma a verificar as hipóteses de pesquisa apresentadas inicialmente. Também discutirá as limitações e ameaças à validade dos estudos experimentais conduzidos. Para tanto, utilizar-se-á de análise estatística descritiva e multivariada, além de outras abordagens, para observação dos respectivos resultados.

CAPÍTULO 5

Análise dos Resultados

Neste capítulo é apresentado os resultados obtidos com o desenvolvimento dos estudos experimentais, por meio da análise e interpretação dos dados coletados, utilizando-se de estatística descritiva e multivariada, além de outras abordagens, para elaboração e observação dos resultados. A Seção 5.1 aborda as quatro propostas de análise utilizadas para avaliação dos dados relacionados à DT em produtos de código aberto. A Seção 5.2 lista algumas situações que podem constituir-se como limitações e ameaças à validade dos estudos experimentais conduzidos. A Seção 5.3 apresenta algumas considerações gerais sobre os resultados obtidos, assim como outras percepções possíveis pela análise e interpretação desses dados. Por fim, a Seção 5.4 sintetiza o conteúdo deste capítulo.

5.1 Análise e Interpretação dos Dados

Conforme apresentado na Seção 4.2, o experimento proposto envolveu a avaliação de 276 produtos de código aberto, acessados por meio da URL dos respectivos repositórios dos projetos. Após a execução do experimento foi possível obter dados de 140 produtos (50,72%) da lista inicial e descartar outros 136 (49,28%) que apresentaram algum tipo de falha que inviabilizava os procedimentos de avaliação do código fonte e a coleta dos dados de forma automatizada. O panorama sobre os resultados obtidos com a execução dos experimentos é apresentado na Tabela 5.1.

Considerando a massa de dados obtida após a execução do experimento controlado, optou-se pela utilização de métodos de estatística multivariada para prover a respectiva análise dos dados.

Mingoti (2005) afirma que a estatística multivariada consiste em um conjunto de métodos estatísticos utilizado em situações nas quais diversas variáveis são medidas simultaneamente, em cada elemento amostral. A autora destaca, ainda, que a estatística multivariada se divide em dois grupos: um primeiro, consistindo de técnicas exploratórias de sintetização (ou simplificação) da estrutura de variabilidade dos dados e, um segundo, consistindo de técnicas de inferência estatística. Em linhas gerais, esses métodos são

utilizados com o propósito de simplificar ou facilitar a interpretação do fenômeno que está sendo estudado.

Tabela 5.1: *Panorama sobre a execução do experimento*

Resultado do Experimento	**Quantidade**	**Porcentagem**
Projetos com Sucesso - Quantidade de versões obtidas		
Somente 1 versão	46	32,86%
De 2 a 3 versões	28	20,00%
De 4 a 7 versões	22	15,71%
De 8 a 16 versões	25	17,86%
De 17 a 30 versões	17	12,14%
Acima de 30 versões	2	1,43%
Total (projetos com sucesso)	**140**	**100,00%**
Projetos com Falha - Motivo		
URL não localizada	19	13,97%
Falha de compilação e/ou execução dos testes	89	65,44%
Falha de implantação no Sonar (dados não coletados)	15	11,03%
Projetos em duplicidade	13	9,56%
Total (projetos com falha)	**136**	**100,00%**

Diante dos dados obtidos, foram propostas quatro análises distintas para avaliar a DT sobre o conjunto de produtos de código aberto, a saber:

1. **Analisar a evolução da dívida técnica:** nesse caso foi utilizada a métrica *Technical Debt Ratio* (proporção da DT) e os projetos com dados de ao menos 8 versões ao longo do tempo. Assim, foram considerados 44 projetos com o objetivo de avaliar a evolução histórica da DT, baseada em pontos no tempo. Entende-se que a evolução individual de cada projeto pode permitir, de forma exploratória, a análise do comportamento evolutivo do conjunto;
2. **Estabelecer uma classificação para a dívida técnica:** foram considerados tanto o *Technical Debt Ratio*, como seus eixos de qualidade (*comments*, *complexity*, *coverage*, *design*, *duplications* e *violations*). Para essa análise, foram utilizados todos os 140 projetos disponíveis, sendo que, para aqueles que possuíam mais de uma versão, foi utilizada a média aritmética dos valores. Os produtos de código aberto foram organizados em três grupos, construídos conforme os dados observados, no sentido de verificar aqueles que possuíam os menores e maiores valores para DT, bem como para seus eixos de qualidade. Essa classificação permite estabelecer um limiar *(threshold)* para sugerir níveis aceitáveis e/ou gerenciáveis para a DT;
3. **Verificar a representatividade dos eixos de qualidade na composição da DT:** Aproveitando as análises anteriores, foram utilizados o *Technical Debt Ratio* e seus eixos de qualidade para verificar qual(is) dele(s) possuíam maior impacto na composição da DT. Para essa análise, foi considerado o conjunto de 44 projetos com ao menos 8 versões, com o objetivo secundário de identificar aspectos do código

fonte, dentre os eixos de qualidade da DT, que necessitam de maior atenção da comunidade de desenvolvimento;

4. **Verificar a existência de uma correlação entre a dívida técnica *(Technical Debt Ratio)* e o método SQALE *(SQALE Rating)*:** essas métricas possuem maneiras distintas de avaliação da qualidade de software, conforme já descrito na Seção 2.2, por isso, considerou-se apropriado verificar, para o conjunto de projetos avaliados, se existe uma correlação entre suas abordagens de avaliação.

As análises/avaliações apresentadas acima, bem como o método estatístico empregado em cada proposta são detalhados nas subseções seguintes.

5.1.1 Análise da evolução da dívida técnica

Como o processo de desenvolvimento da Comunidade de Software Livre é uma atividade contínua, caracterizada pela frequente disponibilização e evolução de versões (RAYMOND, 1999), considerou-se importante, como primeira análise, verificar se a qualidade desses produtos de código aberto acompanha seu crescimento e desenvolvimento.

Para essa análise, foram selecionados apenas 44 projetos, os quais possuíam ao menos 8 versões com dados disponíveis, o que representa aproximadamente 31,5% dos projetos válidos avaliados. Esse limite, relativo ao número de versões, foi estabelecido por considerar que uma quantidade inferior poderia comprometer a avaliação da evolução histórica da DT, uma vez que disponibilizaria poucos pontos no tempo a serem observados. A tabela com a relação completa dos projetos, versões e valores coletados para as métricas pode ser verificada no Apêndice A – Tabela A.1.

Produziram-se, então, gráficos de tendência temporal, considerando os valores da DT observados para cada versão dos projetos, em ordem histórica. Para facilitar a visualização dos dados e comparação entre os gráficos, foi estabelecido um escore padronizado (TRIOLA, 2011) para cada um dos valores. Esses gráficos são apresentados nas Figuras 5.1 e 5.2 e permitem observar o comportamento evolutivo da DT dos respectivos produtos.

Os gráficos apresentados nas Figuras 5.1 e 5.2 permitem a seguinte leitura: cada projeto recebeu um número identificador, conforme relação da Tabela 5.2; o eixo vertical representa o valor da DT (considerando o escore padronizado) e o eixo horizontal a quantidade de versões de cada projeto, em ordem cronológica. Com a utilização do escore padronizado, observam-se alguns valores negativos, no entanto, isso não significa que os projetos possuem valores negativos para DT, mas sim um valor abaixo da média dos valores da DT do projeto, considerando sua série histórica.

Pela visualização dos gráficos observa-se que o Projeto 07-P (*Apache Diretory Studio*), apesar de não ter um valor alto para o coeficiente estimado, foi um dos projetos

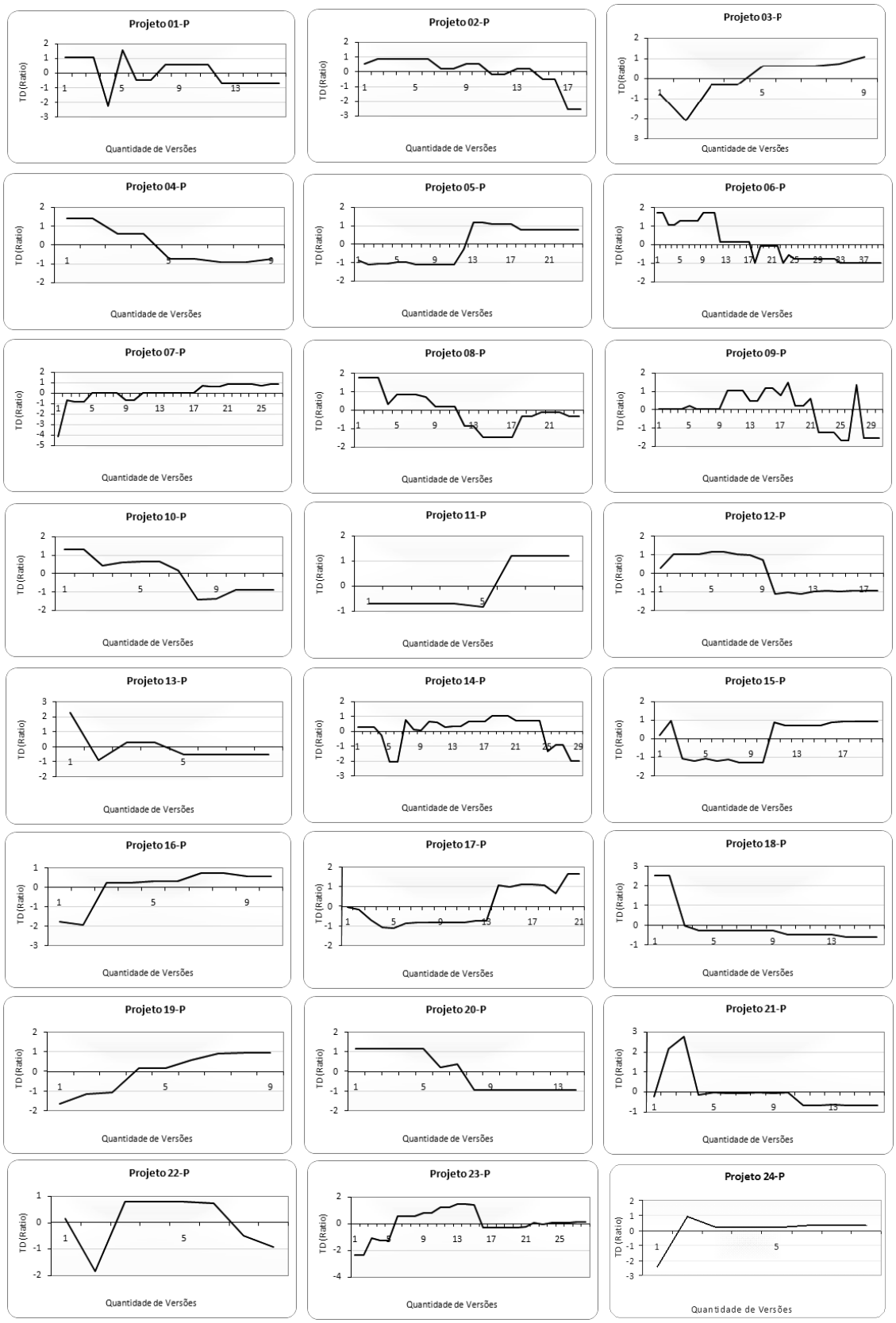

Figura 5.1: *Evolução da DT para os projetos avaliados - Parte 1/2*

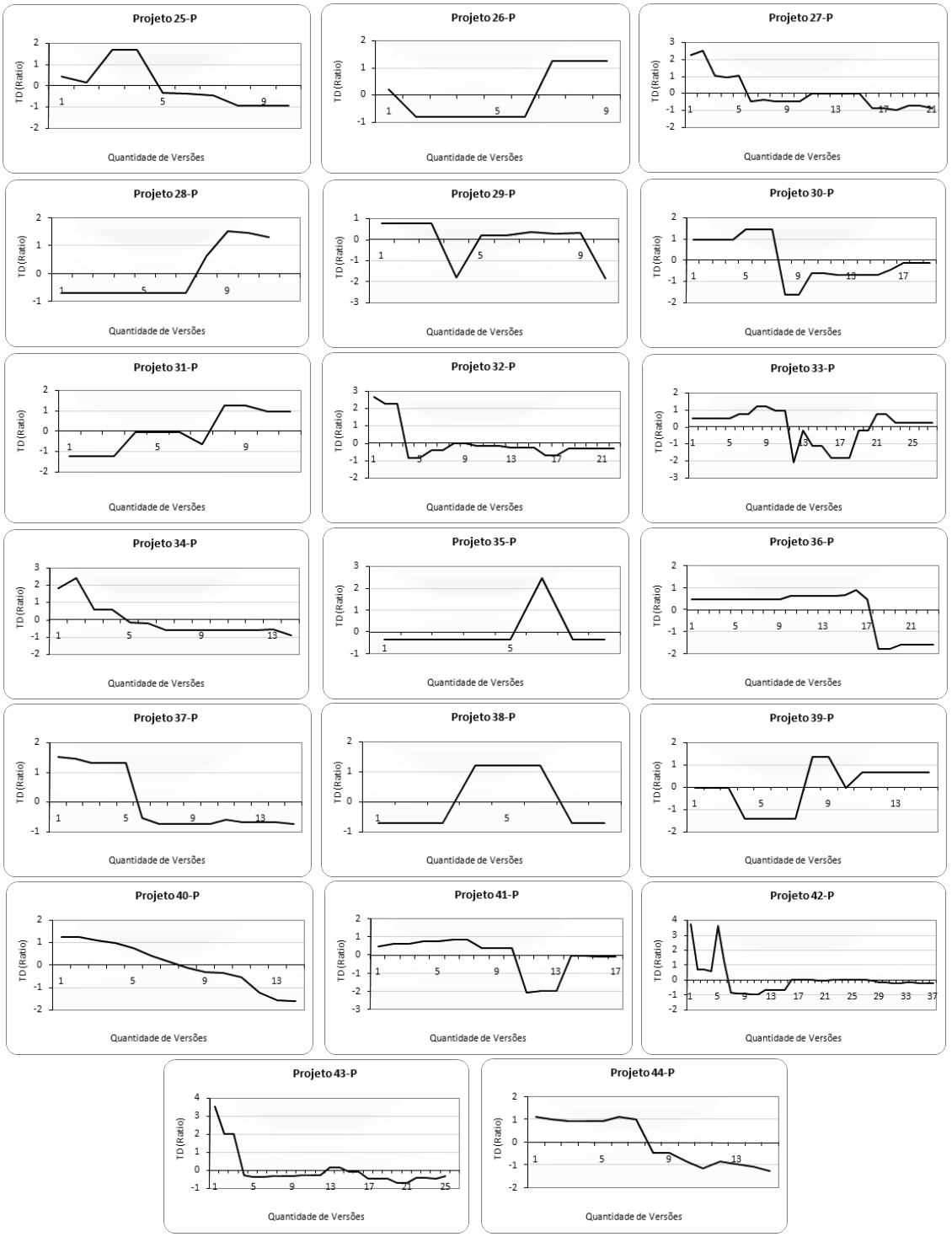

Figura 5.2: *Evolução da DT para os projetos avaliados - Parte 2/2*

que apresentou maior variação dos valores, em forma ascendente, demonstrando considerável aumento da DT ao longo de suas versões. O Projeto 43-P (*Maven SureFire*), apesar de não ter um valor baixo para o coeficiente estimado, foi um dos que apresentou maior variação dos valores, em forma descendente, demonstrando acentuada diminuição da DT ao longo de suas versões. Já os Projetos 32-P e 37-P (*Maven Realease* e *PDF Box Reac-*

tor), respectivamente, apresentaram pouca variação de valores entre suas versões, o que sugere uma tendência de estabilização dos valores da DT.

Tabela 5.2: *Avaliação da evolução da DT (ordenada pelo valor do coeficiente)*

Identificador	Nome do Projeto	Quant. Versões	Coeficiente	P-Valor
34	Maven Shared Components	14	-0,720	0,000
25	Apache XML Security for Java	10	-0,683	0,018
40	Silverpeas Core	14	-0,610	0,000
4	Apache Cocoon 3: Root	9	-0,526	0,001
10	Apache James IMAP	12	-0,444	0,000
12	Apache MyFaces CODI	18	-0,427	0,000
18	Apache Sanselan	16	0,383	0,003
37	PDFBox reactor	15	-0,285	0,000
43	Maven SureFire	25	-0,255	0,002
29	OSGI Reference Implementation	10	-0,246	0,227*
41	SonarSource :: Language Recognizer	17	-0,232	0,028
36	OPS4J Pax Logging (Build POM)	23	-0,227	0,000
44	XmlSchema	15	-0,206	0,000
30	Doxia Sitetools	19	-0,128	0,012
33	Maven SCM	27	-0,126	0,236*
20	Apache ServiceMix Kernel	14	-0,114	0,000
32	Maven Release	22	-0,111	0,008
27	ApacheDS	21	-0,105	0,000
8	Apache Empire-db	24	-0,092	0,000
42	Struts 2	37	-0,064	0,045
13	Apache Neethi	8	-0,061	0,120*
2	Apache Avro Toplevel	18	-0,045	0,000
9	Apache James :: Protocols	30	-0,037	0,009
1	ActiveMQ-CPP Project	16	-0,036	0,009
6	Apache Directory Shared	40	-0,033	0,000
14	Apache OpenWebBeans	29	-0,027	0,0444*
22	Apache Shiro	8	-0,021	0,947*
35	OPS4J - Pax Logging	8	0,010	0,555*
38	Apache RAT	8	0,010	0,689*
21	Apache Shindig Project	16	0,017	0,863*
7	Apache Directory Studio	27	0,078	0,000
23	Apache Tika	28	0,100	0,106*
24	Apache Wink	8	0,138	0,222*
39	ServiceMix :: Utils	15	0,167	0,046
31	Maven :: Indexer	11	0,181	0,000
15	Apache PDFBox	20	0,200	0,004
17	Apache Rave :: rave-project	21	0,211	0,000
26	Apache Yoko CORBA Server	9	0,233	0,054*
5	Apache Directory Daemon	12	0,246	0,000
11	Apache JAMES jSieve	8	0,266	0,011
16	Apache Pluto	10	0,299	0,007
3	Apache Chemistry OpenCMIS	9	0,355	0,004
19	Apache ServiceMix :: NMR	9	0,688	0,000
28	ApacheDS Installers Parent	11	2,020	0,001

A partir de uma análise visual e preliminar, percebeu-se que o comportamento evolutivo desse conjunto de projetos apresentava algumas similaridades. Para confirmar

essa observação, procedeu-se o cálculo dos coeficientes estimados para os valores da DT de cada projeto, considerando suas respectivas versões. O resultado dessa análise é apresentado na Tabela 5.2.

Para interpretação da Tabela 5.2 algumas situações devem ser consideradas. Primeiro, o valor do coeficiente indica a tendência temporal observada para a DT. Assim, um valor negativo sugere uma diminuição da DT ao longo do tempo, enquanto um valor positivo sugere um aumento. No entanto, essa interpretação somente é possível se o valor da margem de erro *(P-Valor)* for menor que 5%, ou seja, o valor de $(P > |t|)$ deve ser menor que 0,05. Segundo, se a condição anterior for satisfeita, então pode-se afirmar que para o respectivo projeto a DT aumentou ou diminuiu ao longo do tempo, conforme o sinal do coeficiente. Caso contrário, não se tem base para afirmar sobre a evolução da DT do projeto. A Tabela 5.3 auxilia na interpretação do resultado dessa análise.

Dos produtos avaliados, 12 foram considerados inválidos para essa análise, pois apresentaram o valor da margem de erro superior à 0,05 (marcados com * na Tabela 5.2). Assim, a análise dos outros 32 projetos válidos, permite observar uma evolução favorável da DT para aproximadamente 2/3 dos produtos (65,62%) que apresentaram diminuição da DT ao longo do tempo, contra 1/3 dos produtos (34,38%) que demonstraram crescimento da dívida.

Tabela 5.3: *Agrupamento dos projetos conforme comportamento evolutivo da DT*

Situação do Projeto	**Quantidade**	**Porcentagem**
Inválidos para análise $(P > \|t\|)$ maior 0,05)	12	27,28%
Válidos para análise $(P > \|t\|)$ menor 0,05)	32	72,72%
Total	**44**	**100,00%**
Análise dos projetos válidos		
Diminuição da DT (coeficiente negativo)	21	65,62%
Aumento da DT (coeficiente positivo)	11	34,38%
Total	**32**	**100,00%**

Essa análise permite alcançar um dos objetivos deste trabalho e validar uma das hipóteses estabelecidas (HA_1), no sentido de confirmar que a maioria dos produtos de código aberto avaliados apresentam uma tendência de evolução negativa, com diminuição da DT ao longo de suas versões, o que pode sugerir que esses produtos apresentam uma melhoria na qualidade do seu código fonte, segundo os eixos de qualidade da DT.

Outro aspecto a ser destacado consiste em que a análise desenvolvida nesta seção permite observar não somente a evolução, mas também o estado da DT em relação a esses produtos, o que sugere que esses projetos apresentam uma evolução satisfatória e tendem a estabelecer um patamar aceitável de suas DTs. Contudo, considera-se interessante, como proposta para trabalhos futuros, investigar as características e/ou causas dessa diminuição,

aumento ou até mesmo estabilização da DT para produtos dessa natureza. As propostas de trabalhos futuros serão apresentadas no Capítulo 6.

5.1.2 Classificação dos projetos segundo valores da DT

Estendendo o trabalho realizado anteriormente, no qual foi avaliado o estado da DT para um conjunto de 40 produtos de código aberto, considerando somente a versão mais recente de cada um dos projetos, optou-se por aplicar essa mesma abordagem utilizando agora uma base maior de projetos e seu histórico de versões (VIEIRA et al., 2013).

Nesta análise, foram utilizados todos os 140 projetos avaliados, independente da quantidade de versões com dados disponíveis. Para os projetos que apresentavam mais de uma versão, foi considerada a média aritmética dos valores, de forma a obter um único valor que representasse o estado da DT. O agrupamento dos projetos foi realizado segundo sua ordenação, do menor para o maior valor da DT. Em seguida, procedeu-se o cálculo da média aritmética dos valores da DT e seus eixos de qualidade, assim como seu desvio padrão, a fim de estabelecer um limiar inferior (média menos desvio padrão) e um limiar superior (média mais desvio padrão).

Com isso, foi possível classificar os projetos em 3 grupos, conforme os valores observados para a DT: a) aqueles com valores abaixo do limiar inferior (com os menores valores para DT); b) aqueles entre os intervalos limiares (com valores intermediários) e; c) aqueles com valores acima do limiar superior (com os maiores valores para DT).

A Tabela 5.4 (versão resumida) e a Figura 5.3 permitem observar que praticamente 3/4 dos projetos encontram-se no grupo intermediário, com valores do *Technical Debt Ratio* entre 7,55 e 21,80, sugerindo que esses projetos possuem valores pouco elevados para DT. Observa-se, ainda, uma pequena concentração de projetos abaixo do limiar inferior (12%), bem como acima do limiar superior (16%).

Para confirmar isso, observou-se que a Plataforma SonarQube possui um filtro conhecido como *treemap* que classifica os valores das métricas em 5 percentis, com incrementos de 20% cada, e atribui uma cor – do verde ao vermelho, do satisfatório ao crítico – para cada faixa de valor. Nessa situação, considerando os valores observados, praticamente todos esses projetos receberiam a cor verde, nos dois percentis iniciais. A tabela com a versão completa da classificação dos projetos é apresentada no Apêndice A – Tabela A.1.

A leitura da Tabela 5.4 permite observar o cálculo da média, o desvio padrão, o limiar inferior e o limiar superior para a DT e seus eixos de qualidade. O valor da média da DT é 14,33, considerando uma faixa de valores de 1,10 a 43,00. Uma interpretação preliminar desses dados permite observar que os eixos de qualidade "‘Cobertura"’,

"‘Violações"’ e "‘Complexidade"’, nessa ordem, são os que possuem as maiores médias para os valores. Por outro lado, os eixos de qualidade "‘Duplicações"’, "‘Projeto"’ e "‘Comentários"’, nessa ordem, possuem os menores valores para o limiar superior e, consequentemente, representam uma pequena proporção para composição da DT. A análise sobre a representatividade dos eixos de qualidade será analisada com mais propriedade na seção seguinte.

Tabela 5.4: *Classificação dos projetos conforme valores observados para DT (versão resumida)*

Métrica/Eixo	Média Aritmética	Desvio Padrão	Limiar Inferior	Limiar Superior
Dívida Técnica	14,33	6,81	7,52	21,85
Comentários	14,27	11,96	2,68	17,32
Complexidade	15,53	9,40	6,13	21,66
Cobertura	29,55	16,36	13,19	42,74
Duplicações	6,39	7,51	-1,12	5,27
Projeto	10,96	14,75	-3,79	7,17
Violações	22,91	13,85	9,06	31,97

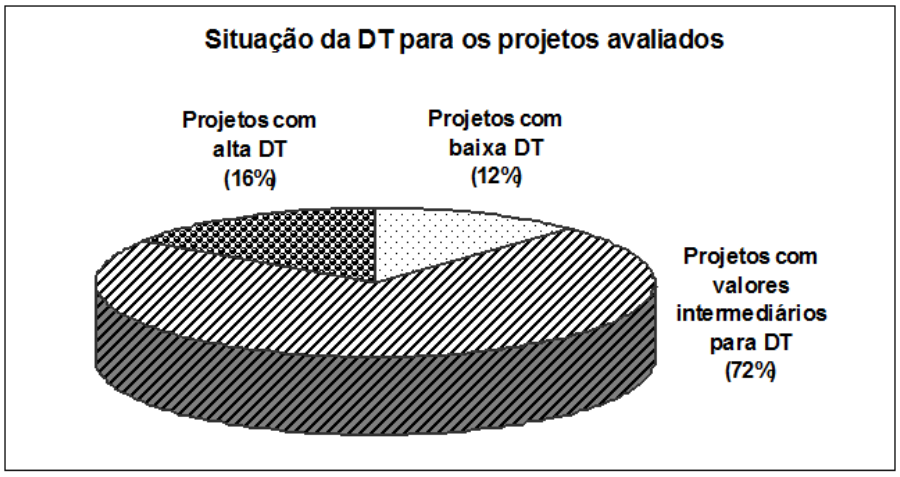

Figura 5.3: *Agrupamento dos projetos conforme classificação da DT*

Além da classificação dos projetos em relação à DT, torna-se interessante analisar seu estado para o conjunto de projetos avaliados. Para tanto, optou-se por elaborar e explorar um diagrama de caixa (*box plot*) do conjunto de dados. Esse diagrama é um gráfico utilizado na estatística descritiva para analisar a distribuição de dados ou a variação de uma variável entre diferentes grupos de dados (E-HANDBOOK, 2013).

Para elaboração do diagrama são calculados a mediana e os quartis. O quartil inferior contém 1/4 dos menores valores e o quartil superior concentra 3/4 de todos os valores. Uma haste inferior conecta a base da caixa ao menor valor observado enquanto outra haste superior conecta o topo ao maior valor observado(E-HANDBOOK, 2013).

O diagrama de caixa é uma importante ferramenta de análise exploratória, uma vez que capta importantes aspectos de um conjunto de dados por meio de cinco

medidas: valor mínimo, primeiro quartil, mediana, terceiro quartil e valor máximo. Com isso, é possível explorar diferentes características como o efeito de valores atípicos ou discrepantes, a assimetria e a dispersão dos dados entre outros aspectos.

Para a interpretação da Figura 5.4, considera-se que o eixo horizontal representa as variáveis a serem analisadas (a dívida técnica e seus eixos de qualidade) e o eixo vertical os valores observados para o conjunto de projetos avaliados (em um intervalo de 0 a 100).

Abstrai-se que quando as duas caixas do diagrama possuem alturas (tamanhos) semelhantes, significa que a distribuição é simétrica e há pouca dispersão dos dados, como se pode observar nos diagramas da dívída técnica (TDR) e dos eixos "Comentários" (TdCm) e "Cobertura" (TdCv). Já os demais eixos – "Complexidade" (TdCp), "Projeto" (TdDs), "Duplicações" (TdDu) e "Violações" (TdVi) – possuem uma distribuição assimétrica, com dispersão dos dados, conforme a altura observada para cada caixa do diagrama.

Outra observação importante refere-se aos valores atípicos ou discrepantes (*outliers*) dos diagramas, sejam inferiores (valores abaixo do quadrante inferior (Qi), na seguinte condição (Qi - 1,5 * (Qs - Qi)) ou superior (valores acima do quadrante superior (Qs), na seguinte condição (Qs + 1,5 * (Qs - Qi)). Assim, observa-se que tanto a dívida técnica quanto seus eixos de qualidade, com exceção do eixo "Cobertura" (TdCv), possuem *outliers* superiores, sendo que há uma concentração dessa situação no eixo "Duplicações" (TdDu).

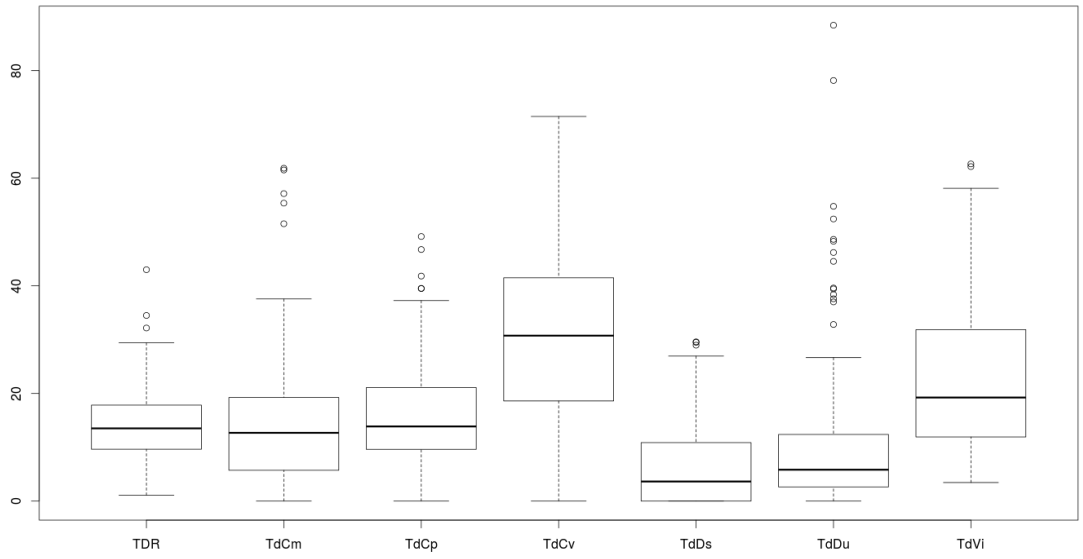

Figura 5.4: *Diagrama de caixa da DT e seus eixos de qualidade*

A análise a ser apresentada na última subseção deste capítulo (Subseção 5.1.4) abordará a classificação dos projetos conforme a avaliação na métrica *SQALE Rating* (que vai de uma faixa de A a E, mas que para esse estudo, conforme os valores observados para o conjunto de projetos avaliados, foi considerada apenas a variação de A a C). Nessa análise, conforme demonstra a tabela apresentada no Apêndice A, é possível perceber que uma grande quantidade desse projetos recebeu índices A e B, sugerindo uma proximidade entre as avaliações de qualidade apresentadas por cada abordagem.

Conforme apresentado no Capítulo 3, o trabalho de Eisenberg (2012) estabeleceu um conjunto de métricas importantes, bem como limites específicos aos projetos para definir níveis gerenciáveis dos eixos de qualidade da DT, o que permite estabelecer parâmetros para avaliação e priorização da DT. Contudo, o autor não estabeleceu um limiar único que representasse a dívida técnica (ou a proporção da DT) do projeto.

O presente trabalho propõe uma abordagem de classificação da DT que considera o próprio conjunto de valores dos produtos para essa métrica. Contudo, essa classificação ainda não permite estabelecer, de forma precisa, um patamar aceitável para a DT, pela ausência de valores de referência para se estabelecer uma comparação e classificação entre esses projetos.

5.1.3 Análise da representatividade dos eixos de qualidade na composição da DT

Essa análise também é uma extensão do trabalho de Vieira et al. (2013). O diferencial consiste em que, para esse estudo, aplicou-se uma análise estatística mais robusta, a fim de verificar a representatividade dos eixos de qualidade e sua relação com a composição da DT. Para tanto, procurou-se responder inicialmente a seguinte pergunta: “é possível expressar a dívida técnica (métrica *Technical Debt Ratio*) como uma expressão linear dos eixos?”. A partir disso, passou-se a realizar algumas tentativas de análise de regressão.

Com uma abordagem exploratória, buscando verificar a existência de uma correlação linear entre a métrica *Technical Debt Ratio* e seus eixos de qualidade, observou-se uma relação linear pobre, uma vez que os diagramas de dispersão obtidos possuíam pontos dispersos e que não se aproximavam do padrão de uma reta (TRIOLA, 2011).

Diante disso, optou-se por aplicar uma transformação logarítmica natural para a métrica *Technical Debt Ratio* em função dos eixos de qualidade, utilizando o método “mínimos quadrados”, o qual consiste de um método de estimação para regressão linear. Essa técnica de computação matemática é capaz de encontrar o melhor ajuste para um conjunto de dados tentando minimizar a soma dos quadrados das diferenças entre o valor estimado e os dados observados (TRIOLA, 2011). Assim, a equação de regressão

representa a reta que "melhor" se ajusta aos pontos do gráfico. A aplicação desse método permitiu expressar a DT por meio da Equação 5-1:

$$LgTD = \beta_0 + \beta_1 TdCp + \beta_2 TdCb + \beta_3 TdCm + \beta_4 TdPT + \beta_5 TdDu + \beta_6 TdVi + E_i \quad (5\text{-}1)$$

Onde: $\beta_0..\beta_6$ = coeficientes obtidos por meio do método "mínimos quadrados"
$LgTD$ = dívida técnica (*Technical Debt Ratio*)
$TdCp$ = eixo de qualidade complexidade
$TdCb$ = eixo de qualidade cobertura
$TdCm$ = eixo de qualidade comentários
$TdPr$ = eixo de qualidade projeto
$TdDu$ = eixo de qualidade duplicações
$TdVi$ = eixo de qualidade violações
E_i = termo de erro aleatório

Essa análise resultou nos dados apresentados na Tabela 5.5, os quais nos permitem observar que os eixos de qualidade "Cobertura", "Violações" e "Complexidade", nessa ordem, possuem os maiores valores para o coeficiente (desconsiderando o sinal) e são os mais representativos na composição da DT, ou seja, os responsáveis pelos maiores débitos nos projetos avaliados. Os eixos "Cobertura" e "Violações" apresentam uma relação positiva com a DT *(Technical Debt Ratio)*, o que indica que se a "Cobertura" aumenta em uma unidade, a DT média deve aumentar em aproximadamente 0,15 unidades.

Similarmente, o aumento de uma unidade do eixo "Violações" originaria um aumento da DT média em 0,13 unidades. Por outro lado, o eixo "Complexidade" apresenta uma relação negativa, sendo que o aumento em uma unidade do eixo "Complexidade" provoca uma diminuição da DT média em aproximadamente 0,09 unidades. Os dados ainda permitem observar (por meio do valor *R-squared = 0,39*) que aproximadamente 39% da variabilidade do *LgTD* é explicado pelas variáveis ajustadas, correspondentes aos seis eixos de qualidade.

Pela análise da Tabela 5.5 interpreta-se que β_0 representa a média do logaritmo da DT em um instante quando todos os outros eixos têm valor zero, o que pode ser interpretado como a dívida média inicial de todos os projetos. Já (Prob > F) é um teste de hipótese que procura confirmar se pelo menos um dos eixos é relevante para explicar a TD. Um valor inferior a 0,05 indica que pelo menos um dos eixos é relevante para essa finalidade, a exemplo do que é apresentado na respectiva tabela.

Diante desses resultados torna-se interessante verificar a relação entre os valores observados para a DT e a proporção dos eixos mais representativos em sua composição.

A Tabela 2.1, Capítulo 2, demonstra que os maiores custos associados ao cálculo da DT estão relacionados ao eixo "Complexidade" (custo para dividir métodos e classes), estabelecendo um maior peso para este eixo na composição da dívida. Realizando algumas simulações com a tabela dos projetos avaliados, disponível no Apêndice A – Tabela A.1 –, verifica-se que os resultados demonstram que o eixo "Complexidade" possui uma relação inversa com a DT, ou seja, para a amostra utilizada, em geral, projetos que apresentam altos valores para a DT, possuem baixos valores para esse eixo. Por outro lado, projetos com altos valores para os eixos "Cobertura" e "Violações" apresentam valores baixos para o eixo "Complexidade", o que demonstra a compensação e/ou distribuição da DT entre seus eixos de qualidade.

Tabela 5.5: *Análise da representatividade dos eixos de qualidade na composição da DT*

$LgTD$ **(Logaritmo da DT)**	**Coeficiente** (β)	**Erro Padrão**	**P-Valor**
$TdCb$ (Cobertura)	0,148	0,053	0,009
$TdVi$ (Violações)	0,128	0,062	0,048
$TdCp$ (Complexidade)	-0,096	0,030	0,003
$TdCm$ (Comentários)	-0,066	0,065	0,316
$TdDu$ (Duplicações)	-0,036	0,070	0,606
$TdPr$ (Projeto)	0,004	0,047	0,926
Cons (β_0) = 1,858 Num. Objetos = 44 Prob > F = 0,00 R-squared = 0,39			

Os resultados dessa análise permitem reforçar o trabalho previamente desenvolvido (VIEIRA et al., 2013), bem como confirmar a hipótese inicialmente estabelecida (HA_3) de que os eixos de qualidade "Cobertura", "Violações" e "Complexidade" estão entre os mais representativos na composição da DT. Essa avaliação, sobre a representatividade dos eixos de qualidade que compõem a DT, considerando uma massa significativa de projetos, demonstra-se pioneira, uma vez que não foram encontrados na literatura, até o presente momento, trabalhos similares.

5.1.4 Correlação entre o Technical Debt Ratio e o SQALE Rating

Conforme apresentou-se anteriormente, a análise sobre a classificação da DT apontou uma grande quantidade de produtos que estão abaixo do limiar superior (aproximadamente 84%), sugerindo então que esses produtos possuem valores pouco elevados para a DT. Diante disso, optou-se por analisar o *SQALE Rating*, definido na Seção 2.3, e compará-lo com os valores observados para o *Technical Debt Ratio*, a fim de verificar a existência de alguma proximidade na avaliação da qualidade do código fonte (e do produto), proporcionada por essas abordagens/ferramentas.

É importante destacar que, no momento da finalização da coleta de dados do experimento controlado, constatou-se que o *Technical Debt Plugin* havia sido integrado à implementação do método SQALE na Plataforma SonarQube. Outro registro importante consiste em que o *SQALE Plugin* é uma extensão comercial da Plataforma SonarQube. Sendo assim, foi solicitado à equipe de desenvolvimento do *plugin* a disponibilização de uma licença de uso, para fins acadêmicos, a qual foi gentilmente disponibilizada, pelo prazo de um ano, subsidiando, dessa forma, o desenvolvimento dos estudos experimentais propostos.

Para essa análise, os 140 projetos avaliados foram organizados em três grupos (G1, G2 e G3), usando percentis, e os valores da DT foram classificados em ordem crescente. Optou-se por criar três grupos para facilitar a comparação com o *(SQALE Rating)*, que apesar de variar de "A" a "E" somente apresentou índices de "A" a "C" para os projetos em questão. Cabe informar que quanto mais próximo de "A" melhor a qualidade do projeto. Feito isso, os grupos foram separados na proporção de 33,33% dos projetos com os menores valores da DT, 33,33% com os valores intermediários da DT e 33,33% com os maiores valores para DT.

Posteriormente, verificou-se a seguinte condição: *se os produtos fossem classificados similarmente segundo ambos os agrupamentos, então, espera-se que uma medida de associação forneça uma concordância quase perfeita*. A partir dos dados coletados e do agrupamento proposto, realizou-se uma análise de associação entre as métricas e os resultados demonstraram uma correlação moderada positiva entre o *Technical Debt Ratio* e o *SQALE Rating*, com valor de *Cramér's V = 0,29*, em uma faixa de valores de 0 a 1.

A Tabela 5.6 apresenta uma classificação cruzada dos projetos segundo as formas de agrupamento consideradas (*SQALE Rating* e percentis). Observa-se que dentre os produtos de código aberto classificados com o índice "A" (segundo o *SQALE Rating*), 26 deles (44%) foram classificados no grupo G1; 25 (42%) no grupo intermediário G2 e, finalmente, 8 (14%) no grupo G3. Uma interpretação similar pode ser realizada para os projetos com índice "B" e "C". Esses resultados permitem observar uma concordância relevante entre essas duas abordagens de avaliação.

Tabela 5.6: *Correlação entre Technical Debt Ratio e SQALE Rating*

SQALE Rating	**Grupos da DT**			**Total**
	G1	G2	G3	
A	26	25	8	59
B	14	19	23	56
C	6	3	16	25
Total	45	46	46	**140**
Pearson Chi2(4) = 22,84 Pr = 0,00 Cramér's V = 0,29				

5.2 Limitações e Ameaças à Validade

Como parte integrante dos estudos experimentais conduzidos, apresentam-se nesta seção as ameaças a sua validade, bem como algumas limitações percebidas. Conforme comentado na Seção 4.1, a experimentação não é uma tarefa simples, pois envolve preparar, conduzir e analisar experimentos corretamente (WOHLIN et al., 2012).

De fato, todos os procedimentos relacionados às etapas do processo de experimentação demandaram grande esforço, consumindo praticamente seis meses do cronograma da pesquisa, especialmente pela quantidade de projetos, versões e dados a serem analisados. Assim, tendo ciência de que algumas situações podem constituir limitações e ameaças à validade do experimento, são apresentadas abaixo algumas considerações:

1. Por se tratar de um estudo preliminar e exploratório, inicialmente, não houve a pretensão de estabelecer critérios para a seleção dos projetos de código aberto avaliados, mas sim de definir alguns requisitos técnicos necessários para desenvolvimento dos estudos experimentais, conforme apontado na Seção 4.2. Mesmo considerando a grande quantidade de produtos avaliados, não se pode chegar ao entendimento de que eles representam a Comunidade de Software Livre. Na verdade, eles correspondem a um conjunto expressivo de produtos de código aberto, os quais podem sugerir, com base no estudo desenvolvido, tendências de comportamento para DT dos produtos dessa natureza;
2. A identificação de projetos que atendiam às restrições técnicas não foi uma tarefa trivial. A lista de projetos disponível na página da internet do ProjetoNemo (2013) e GitApache (2013) viabilizou de forma significativa a etapa de seleção dos projetos. Contudo, isso levou a uma concentração de produtos da Comunidade Apache em nosso estudo, sugerindo, por consequência, que o comportamento dos produtos de código aberto observado neste trabalho pode estar mais relacionado às características dos produtos da Comunidade Apache;
3. Embora tenha sido mantida a proposta inicial do trabalho, relacionada à utilização das ferramentas e métricas inicialmente previstas, a descontinuidade de desenvolvimento do *Technical Debt Plugin* e a implementação do método SQALE pela Plataforma SonarQube, comprometeram a execução do experimento, no sentido de não possibilitar, devido a restrições de tempo da pesquisa, que essas atualizações na abordagem da avaliação da DT fossem incorporadas ao experimento. Outra questão consiste em que o novo *Technical Debt Evaluation (SQALE)*, apesar de apresentar caraterísticas e recursos interessantes e mais completos, é uma versão comercial, impedindo seu estudo segundo propósitos do software livre sem a compra da respectiva licença cujo valor é de € 2.700,00;

4. Especificamente, no que se refere à execução do experimento, algumas observações técnicas devem ser consideradas:
    - Mesmo sendo um dos requisitos técnicos, nem todos os projetos possuíam casos de teste implementados e, por esse motivo, apresentaram valores "zerados" para a métrica de "casos de teste com sucesso" e "cobertura dos casos de teste", as quais são subcomponentes do eixo "Cobertura". Esse fato pode ter contribuído para concentração da DT nesse eixo para alguns projetos;
    - A avaliação do código fonte pela Plataforma SonarQube é realizada com base em uma chave *(key)* que identifica o projeto, localizada no arquivo *pom.xml*. No entanto, projetos grandes e/ou compostos por subprojetos, podem ser fragmentados pela Plataforma durante o processo de submissão/implantação e, então, serem considerados como projetos independentes. Assim, a lista de 140 projetos utilizados para análise dos resultados possui alguns subprojetos nessa situação;

5.3 Resultados Obtidos

Conforme apresentado na Seção 1.2, o objetivo principal desse trabalho foi realizar uma investigação, ainda que preliminar e exploratória, sobre a qualidade de produtos de código aberto, por meio do planejamento e desenvolvimento de estudos experimentais para avaliar a DT nesses produtos. Entende-se que este trabalho cumpriu seu objetivo ao permitir o estabelecimento de um panorama sobre o estado, a evolução e as características da DT para um conjunto de 140 produtos de código aberto.

Para demonstrar o alcance dos objetivos específicos, apresenta-se abaixo uma síntese dos principais resultados alcançados com o presente trabalho:

1. **Análise da evolução da DT:** a análise e interpretação dos dados apresentados na Seção 5.1 demonstra que, dos projetos válidos para essa análise, aproximadamente 2/3 deles (65,62%) apresentaram redução da DT ao longo do tempo, o que sugere uma melhoria na qualidade do código fonte produzido, especialmente no que se refere aos eixos de qualidade que compõem a DT. Esse resultado permite atingir o primeiro objetivo específico proposto;
    - **Validação da hipótese:** os dados apresentados na Tabela 5.2 permitem refutar a hipótese nula HN_0 e, ao mesmo tempo, validar a hipótese alternativa HA_1, uma vez que a maioria dos projetos apresentou valores negativos para o coeficiente estimado, demonstrando uma diminuição das suas respectivas DTs ao longo das versões analisadas. Também não foi observado qualquer projeto

com coeficiente igual a 0 (zero), o que demonstraria um comportamento totalmente estável (sem evolução).

2. **Classificação para DT:** utilizando-se dos próprios valores observados para a DT dos projetos avaliados, estabeleceram-se três grupos de classificação: os projetos com menores valores da DT (abaixo do limiar inferior); os projetos com valores intermediários (entre os intervalos limiares) e os projetos com maiores valores (acima do limiar superior). O gráfico apresentado na Figura 5.3, p. 59, permite observar que mais de 85% dos projetos avaliados apresentaram valores para a métrica *Technical Debt Ratio*, que estão abaixo do limiar superior, sugerindo, deste modo, que eles possuem valores pouco elevados para DT. Contudo, considera-se que essa abordagem de classificação ainda não permite estabelecer, de forma precisa, um patamar aceitável para a DT, pela ausência de valores de referência para se estabelecer uma comparação entre os projetos. Assim, entende-se que o segundo objetivo específico foi parcialmente alcançado;
3. **Análise da representatividade dos eixos de qualidade da DT:** com a aplicação de técnicas estatísticas mais robustas e utilização de métodos de regressão linear, foi possível observar, para o conjunto de projetos avaliados, quais eixos de qualidade são mais representativos na composição da DT. Os resultados permitiram alcançar o último objetivo específico, além de validar o trabalho anteriormente realizado (VIEIRA et al., 2013), constatando que os eixos de qualidade "Cobertura", "Violações" e "Complexidade", nessa ordem, são os que mais contribuem para o incremento da DT;
    - **Validação da hipótese:** os dados apresentados na Tabela 5.5, permitem refutar a hipótese nula HN_2 e, ao mesmo tempo, validar a hipótese alternativa HA_3, ao demonstrarem que os eixos de qualidade da DT apresentam valores distintos para os respectivos coeficientes estimados e que os eixos "Cobertura", "Violações" e "Complexidade" são os que apresentam maior representatividade na composição da DT.
4. **Correlação entre as implementações da DT e do método SQALE:** apesar de não ter sido uma abordagem inicialmente prevista nos objetivos deste trabalho, essa avaliação demonstrou-se promissora ao verificar a existência de uma correlação moderada positiva entre a métrica *Technical Debt Ratio* e o *SQALE Rating*, conforme observado na Tabela 5.6. Esses resultados sugerem uma concordância relevante entre essas duas abordagens de avaliação da DT, o que possivelmente pode explicar a decisão do *SonarSource Team* em integrá-las a partir da versão 4.0 da Plataforma SonarQube. Entretanto, com a integração, o cálculo da dívida técnica pela Plataforma SonarQube passou a ser realizado apenas pelo *plugin* comercial (SQALE).

5.4 Considerações Finais

Este capítulo apresentou a análise e interpretação dos dados obtidos com a realização dos estudos experimentais conduzidos nesse trabalho. Por meio de quatro abordagens específicas foi possível analisar a evolução da DT para um conjunto de produtos de código aberto e observar que aproximadamente 2/3 dos projetos apresentaram evolução favorável da DT, com redução dos seus valores ao longo do tempo.

Também foi sugerida uma classificação da DT para os projetos, considerando seus próprios valores para métrica, de forma a estabelecer um limiar inferior e um limiar superior como abordagem de classificação. Utilizando métodos de regressão linear verificou-se que os eixos de qualidade "Cobertura", "Violações" e "Complexidade", nessa ordem, são os mais representativos na composição da DT.

Ainda foi possível analisar o estado da DT e seus eixos de qualidade para o conjunto de projetos avaliados, por meio da interpretação dos respectivos diagramas de caixa, os quais demonstraram um dispersão dos dados para a maioria dos eixos de qualidade e, por outro lado, uma simetria entre os valores observados para DT. Por fim, foi realizada uma análise de associação e verificou-se uma correlação moderada positiva entre o *Technical Debt Ratio* e o *SQALE Rating*.

Torna-se importante reforçar que, até o momento da escrita desta dissertação, não se tinha conhecimento de trabalhos similares, com vistas a evidenciar quais eixos de qualidade são mais representativos na composição da DT de projetos de código aberto, o que demonstra uma das principais contribuições obtidas. Além disso, como não existem trabalhos de referência sobre aquilo que pode ser considerado um valor aceitável para a DT, os estudos aqui apresentados contribuem com um ponto de partida para que a coleta de dados sobre essa métrica seja constante e, a medida que novos dados sejam coletados, o modelo estatístico possa ser atualizado e aperfeiçoado.

O capítulo seguinte abordará as considerações finais deste trabalho, apresentando suas contribuições e as perspectivas para realização de trabalhos futuros, os quais podem considerar tanto a abordagem aqui proposta como incorporar novas perspectivas sobre a temática.

CAPÍTULO 6

Conclusões

A metáfora da dívida técnica apresenta-se útil para produção, desenvolvimento e manutenção de software. Pelo fato de prover um vocabulário familiar, associando termos da área financeira, ela vem ganhando significativa força na comunidade de desenvolvimento de software ao tempo que permite a compreensão e comunicação de questões relacionadas à qualidade, valor e custo do produto. Acredita-se que, além de um conceito, ela inspira um conjunto útil de métodos e ferramentas que buscam apoiar sua concepção, identificação e gestão.

Atualmente, há uma constante ênfase da comunidade acadêmica e industrial na área de governança de software, em certa parte confirmada pela edição especial da revista *IEEE Software*, em novembro/dezembro de 2012 (Volume 29), tendo como tema principal *"Technical Debt"*. Conforme abordado no Capítulo 3, alguns autores acreditam que a DT está diretamente relacionada ao contexto de evolução e manutenção existente no ciclo de vida de um produto de software, influenciando em sua produtividade e sustentabilidade.

Também é importante ressaltar que há diferentes maneiras de se computar a DT – seja pela abordagem manual ou automatizada –, bem como diferentes níveis de identificação – com foco no código fonte, documentação, testes ou pessoas, entre outros. As abordagens manual e automatizada podem ser consideradas complementares, uma vez que apresentam vantagens e desvantagens e devem ser utilizadas conforme o contexto e a finalidade desejada. Por exemplo, uma abordagem automatizada pode ser considerada adequada quando se tem uma grande quantidade de projetos ou versões a serem analisadas, contudo, pode deixar algumas lacunas no processo de identificação, especialmente quando se têm alguns aspectos resultantes do próprio contexto de desenvolvimento e/ou manutenção do produto, os quais podem ser identificadas com mais facilidade por uma abordagem manual.

Para este trabalho, considerando o grande volume de dados obtidos para avaliação – 140 projetos, 985 versões e aproximadamente 35 milhões de linhas de código –, bem como a intenção de propor uma investigação preliminar e exploratória sobre a qualidade dos produtos de código aberto em relação à DT, optou-se por utilizar a abordagem auto-

matizada para identificação da DT, no intuito de formular um panorama inicial sobre o comportamento evolutivo e as características do respectivo conjunto de projetos avaliado.

Este capítulo final tem por finalidade apresentar as conclusões do presente trabalho. A Seção 6.1 apresenta as considerações finais e as principais contribuições observadas sobre o tema estudado. A Seção 6.2 sugere algumas perspectivas para trabalhos futuros, no sentido de estender este estudo e/ou estabelecer novas abordagens de investigação da DT. Por fim, a Seção 6.3 apresenta a produção científica proporcionada pelos estudos desenvolvidos.

6.1 Principais Contribuições

Conforme apresentado na Seção 5.3, entende-se que o objetivo geral e os objetivos específicos deste trabalho foram alcançados, uma vez que foi possível confrontá-los com os resultados obtidos e validar as hipóteses inicialmente previstas. Considera-se que esses resultados possibilitaram o estabelecimento de um panorama sobre o estado, a evolução e as características da DT para um conjunto expressivo de produtos de código aberto. A análise dos resultados da respectiva seção permite extrair algumas considerações:

1. No que se refere à análise da evolução da DT, observou-se que a maioria dos produtos de código aberto avaliados apresentaram uma tendência de evolução negativa, com diminuição da DT ao longo de suas versões, o que pode sugerir uma melhoria na qualidade do código fonte do produto, segundo os eixos de qualidade da DT. Esses resultados reforçam aspectos de qualidade dos produtos de código aberto, demonstrando sua confiabilidade para utilização, além de apresentar um cenário positivo para esses produtos em relação à DT;
2. Sobre a classificação proposta para DT, apesar das limitações, já apresentadas, sobre a abordagem empregada, foi possível observar que aproximadamente 84% dos produtos apresentaram valores abaixo do limiar superior da DT e que apenas 16% dos produtos ficaram acima desse limiar (considerado crítico). Isso sugere que uma quantidade significativa dos produtos avaliados possui valores pouco elevados para a DT, os quais poderiam estar em nível aceitável, do ponto de vista da gestão da DT;
3. Considera-se que a análise da representatividade dos eixos de qualidade na composição da DT apresentou avanços, uma vez que foi possível desenvolver uma análise estatística mais robusta, possibilitando verificar que os eixos “Cobertura”, “Violações” e “Complexidade” possuem maior peso na composição da DT. Considerando essa situação, reforça-se o entendimento sobre a necessidade de maior atenção, por

parte da comunidade de desenvolvimento, para melhorias nas estratégias de teste de software, regras de conformidade e projeto de software. Entende-se que essas ações podem contribuir para o aprimoramento da qualidade de produtos e processos livres, além de indicar uma possível relação entre a gestão da DT e a sustentabilidade desses produtos;

4. O *SQALE Plugin* também se mostrou eficiente para avaliação da qualidade desses produtos, apresentando correlação com a avaliação realizada pelo *Technical Debt Plugin*. Entende-se, ainda, que a implementação do método SQALE pela Plataforma SonarQube *(Technical Debt Evaluation SQALE)*, apesar de ser uma ferramenta comercial, indica um aperfeiçoamento do processo de identificação, priorização e gestão da DT.

De forma ampla, a principal contribuição desse trabalho é a apresentação de um panorama da DT em produtos de código aberto. A análise do estado, evolução, classificação e composição da DT nesses produtos demonstrou um cenário positivo, sendo que boa parte dos projetos avaliados apresentaram melhorias na qualidade do código fonte, ao longo do seu histórico de versões, conforme os valores observados para DT.

No entanto, por ser uma abordagem exploratória, entendemos a necessidade de extensão e continuidade da temática desenvolvida neste trabalho, inclusive com outras abordagens e perspectivas, conforme será detalhado na seção 6.2.

6.2 Trabalhos Futuros

Em diversos momentos desta dissertação foi destacado que o estudo apresentado tratava-se de uma abordagem preliminar e exploratória sobre a investigação da DT em produtos de código aberto. Reforça-se isso porque, apesar dos trabalhos relacionados e dos estudos anteriores, não foram encontrados na literatura, no levantamento bibliográfico realizado, trabalhos com abordagem e/ou objetivos similares a esse.

Outro aspecto importante a ser ressaltado são as questões e limitações existentes no desenvolvimento de estudos experimentais, os quais permitem, de início, apenas estabelecer expectativas e hipóteses sobre o tema, mas, obviamente, não garantem os resultados sobre o que se deseja encontrar. Diante disso, considerando o levantamento bibliográfico, os estudos experimentais conduzidos e os resultados obtidos, apresentados nesta dissertação, sugerem-se os seguintes trabalhos futuros:

1. Com base na análise da evolução da DT, apresentada na Seção 5.1, entende-se por oportuno considerar grupos menores e específicos de produtos, a fim de investigar as características e/ou causas do comportamento evolutivo observado, seja pela diminuição ou pelo aumento da DT ao longo do tempo;

2. A investigação sugerida no item anterior pode ser complementada com a associação de características específicas de um determinado grupo de produtos de código aberto, ou até mesmo da comunidade responsável pelo seu desenvolvimento, e sua relação com o comportamento evolutivo da DT;
3. Torna-se, também, interessante propor uma abordagem comparativa sobre o estado, a evolução e as características da DT em produtos proprietários e compará-los com os resultados obtidos neste trabalho, relacionados aos produtos de código aberto;
4. No que se refere à classificação da DT, proposta na Seção 5.1, considera-se importante estender o presente trabalho, no sentido de estabelecer níveis de referência para serem utilizados na comparação e classificação de diferentes projetos, tanto em relação à DT quanto aos seus eixos de qualidade, possibilitando assim definir, de forma mais adequada, níveis aceitáveis e/ou críticos para DT ao longo do tempo;
5. Estendendo a análise sobre a correlação existente entre a implementação da DT e do método SQALE da Plataforma SonarQube, comparar a avaliação da qualidade do código fonte realizada por essas duas abordagens;
6. Investigar a correlação existente entre a qualidade do projeto, relacionada a sua DT, e outras perspectivas de avaliação, tal como a sustentabilidade da arquitetural desses produtos. Essa abordagem pode ainda verificar a relação existente entre a gestão da DT e a diminuição da erosão arquitetural do produto, a qual pode ser observada, preliminarmente, pelo eixo de qualidade “Projeto”;
7. Investigar outras ferramentas automatizadas de análise estática para avaliação da DT, comparando o cálculo implementado, a fim de indicar a ferramenta que reflete com mais propriedade a qualidade do código fonte produzido;
8. Implementar um serviço que permita que novos projetos possam ser submetidos a uma instância da Plataforma SonarQube, com acesso à base de dados do ProjetoNemo (2013). Com isso, seria possível posicionar determinado projeto (de código aberto ou proprietário) dentro do contexto da base histórica já criada, de modo a comparar os valores do projeto submetido em relação aos projetos já cadastrados;
9. Usar o *WebService* da Plataforma SonarQube para a extração de dados do ProjetoNemo (2013), visando monitorar constantemente a evolução da DT do grupo de projetos cadastrados na Plataforma. Com essa abordagem, não é possível manter a flexibilidade do conjunto de projetos em avaliação, uma vez que a Plataforma não permite a submissão de outros projetos para serem avaliados, entretanto, torna-se importante considerar que essa base de dados possui uma lista atual de 271 projetos, alguns deles com histórico de versões desde dezembro de 2009.

6.3 Produção Científica

A pesquisa desenvolvida neste trabalho, contemplando o levantamento bibliográfico, a condução do experimento controlado e a análise e interpretação dos dados, permitiu ainda a produção dos seguintes artigos científicos:

- **Trabalhos aceitos e/ou apresentados em eventos regionais e nacionais:**
  1. **Título – Evolução de software livre baseada em métricas de qualidade: um estudo de caso**.
     Evento: IX Simpósio Brasileiro de Sistemas de Informação – SBSI. João Pessoa, Paraíba, Brasil. Maio de 2013.
     Modalidade: Evento nacional. Apresentação de artigo completo.
     Situação: Aceito e apresentado.
  2. **Título – Dívida Técnica: um estudo de caso com produtos de código aberto**.
     Evento: XII Simpósio Brasileiro de Qualidade de Software – SBQS. Salvador, Bahia, Brasil. Julho de 2013.
     Modalidade: Evento nacional. Apresentação de artigo completo.
     Situação: Aceito e apresentado.
  3. **Título – Avaliação da Dívida Técnica em produtos de código aberto: um estudo de caso**.
     Evento: X Congresso de Ensino, Pesquisa e Extensão da Universidade Federal de Goiás – CONPEEX. Goiânia, Goiás, Brasil. Outubro de 2013.
     Modalidade: Evento regional. Apresentação oral (pôster).
     Situação: Aceito e apresentado.
  4. **Título – Um estudo experimental para avaliar características da dívida técnica em produtos de código aberto**.
     Evento: II Escola Regional de Informática de Goiás – ERI/GO, SBC. Goiânia, Goiás, Brasil. Novembro de 2014.
     Modalidade: Evento regional. Apresentação de artigo completo.
     Situação: Aceito e apresentado.

Conforme pode-se observar, parte dos resultados obtidos neste estudo já foram publicados em conferências da área e o restante dará origem a outras publicações, em congressos e periódicos, com artigos já submetidos para apreciação, visando divulgar o conhecimento adquirido com o desenvolvimento da pesquisa e a escrita desta dissertação.

Referências Bibliográficas

CHECKSTYLE: Homepage. 2014. Disponível em: <http://checkstyle.sourceforge.net/>. Acesso em: 17 de julho de 2014.

CUNNINGHAM, W. The wycash portfolio management system. In: *Addendum to the Proceedings on Object-oriented Programming Systems, Languages, and Applications (Addendum)*. New York, NY, USA: ACM, 1992. (OOPSLA '92), p. 29–30. ISBN 0-89791-610-7. Disponível em: <http://doi.acm.org/10.1145/157709.157715>.

E-HANDBOOK: Nist/sematech e-handbook of statistical methods. 2013. Disponível em: <http://www.itl.nist.gov/div898/handbook/eda/section3/boxplot.htm>. Acesso em: 02 de dezembro de 2013.

EISENBERG, R. J. A threshold based approach to technical debt. *SIGSOFT Softw. Eng. Notes*, ACM, New York, NY, USA, v. 37, n. 2, p. 1–6, abr. 2012. ISSN 0163-5948. Disponível em: <http://doi.acm.org/10.1145/2108144.2108151>.

FINDBUGS: Homepage. 2014. Disponível em: <http://findbugs.sourceforge.net/>. Acesso em: 17 de julho de 2014.

GITAPACHE: Homepage - git at apache. 2013. Disponível em: <http://git.apache.org/>. Acesso em: 02 de agosto de 2013.

GOUSIOS, G. et al. Software quality assessment of open source software. In: EPY. *Proceedings of the 11th Panhellenic Conference on Informatics (PCI 2007)*. [S.l.], 2007.

KRUCHTEN, P.; NORD, R.; OZKAYA, I. Technical debt: From metaphor to theory and practice. *Software, IEEE*, v. 29, n. 6, p. 18–21, Nov 2012. ISSN 0740-7459.

KRUCHTEN, P. et al. Technical debt in software development: From metaphor to theory report on the third international workshop on managing technical debt. *SIGSOFT Softw. Eng. Notes*, ACM, New York, NY, USA, v. 37, n. 5, p. 36–38, set. 2012. ISSN 0163-5948. Disponível em: <http://doi.acm.org/10.1145/2347696.2347698>.

LETOUZEY, J.; ILKIEWICZ, M. Managing technical debt with the sqale method. *Software, IEEE*, v. 29, n. 6, p. 44–51, Nov 2012. ISSN 0740-7459.

LIM, E.; TAKSANDE, N.; SEAMAN, C. A balancing act: What software practitioners have to say about technical debt. *Software, IEEE*, v. 29, n. 6, p. 22–27, Nov 2012. ISSN 0740-7459.

MAVEN: Homepage. 2014. Disponível em: <http://maven.apache.org/>. Acesso em: 17 de julho de 2014.

MEIRELLES, P. R. M. *Monitoramento de métricas de código fonte em projetos de software livre*. Tese (Doutorado) — Instituto de Matemática e Estatística, USP, 2013.

MINGOTI, S. *Análise de dados através de métodos de estatística multivariada: uma abordagem aplicada*. Editora UFMG, 2005. ISBN 9788570414519. Disponível em: <http://books.google.com.br/books?id=W7sZllHmmGIC>.

MISHRA, B. et al. Quality and profits under open source versus closed source. In: *Twenty-Third International Conference on Information Systems – ICIS'02*. [S.l.: s.n.], 2002. p. 349–363.

PEZUELA, C. et al. *Quality Platform for Open Source Software*. 2010. Position Paper. Disponível em: `https://www.projet-plume.org/files/Position_Paper_Qualipso_Final_0.pdf`. Acesso em: 25/07/2014.

PMD: Homepage. 2014. Disponível em: <http://pmd.sourceforge.net/>. Acesso em: 17 de julho de 2014.

PRESSMAN, R. *Engenharia de Software*. 7 ed.. ed. [S.l.]: McGraw-Hill, 2011. ISBN 8563308335.

PROJETONEMO: Homepage - nemo sonarqube. 2013. Disponível em: <http://nemo-.sonarqube.org/>. Acesso em: 02 de agosto de 2013.

RAYMOND, E. S. *The Cathedral and the Bazaar*. 1st. ed. Sebastopol, CA, USA: O'Reilly & Associates, Inc., 1999. ISBN 1565927249.

SEAMAN, C. B.; GUO, Y. Measuring and monitoring technical debt. *Advances in Computers*, v. 82, p. 25–46, 2011.

SIEBRA, C. S. A. et al. Managing technical debt in practice: An industrial report. In: *Proceedings of the ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*. New York, NY, USA: ACM, 2012. (ESEM '12), p. 247–250. ISBN 978-1-4503-1056-7. Disponível em: <http://doi.acm.org/10.1145/2372251-.2372297>.

SOMMERVILLE, I. *Engenharia de Software*. 9 ed.. ed. [S.l.]: Pearson Addison-Wesley, 2011. ISBN 9788579361081.

SONARQUBE: Homepage. 2014. Disponível em: <http://www.sonarqube.org/>. Acesso em: 17 de julho de 2014.

SQALE: Homepage - software quality assessment based on lifecycle expectations. 2014. Disponível em: <http://www.sqale.org/>. Acesso em: 17 de julho de 2014.

TDESQALE: Homepage - technical debt evaluation (sqale). 2014. Disponível em: <http://www.sqale.org/>. Acesso em: 17 de julho de 2014.

TDP: Homepage - technical debt plugin. 2014. Disponível em: <http://docs.codehaus-.org/display/SONAR/Technical+Debt+Plugin>. Acesso em: 17 de julho de 2014.

TOM, E.; AURUM, A.; VIDGEN, R. An exploration of technical debt. *Journal of Systems and Software*, v. 86, n. 6, p. 1498 – 1516, 2013. ISSN 0164-1212. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0164121213000022>.

TRIOLA, M. F. *Introdução à Estatística*. 10 ed.. ed. [S.l.]: Editora LTC, 2011. ISBN 9878521615866.

VETRO', A. Using automatic static analysis to identify technical debt. In: *Proceedings of the 2012 International Conference on Software Engineering*. Piscataway, NJ, USA: IEEE Press, 2012. (ICSE 2012), p. 1613–1615. ISBN 978-1-4673-1067-3. Disponível em: <http://dl.acm.org/citation.cfm?id=2337223.2337499>.

VIEIRA, I. R. et al. Dívida técnica: um estudo de caso com produtos de código aberto. In: *XII Simpósio Brasileiro de Qualidade de Software (SBQS'2013)*. [S.l.]: Salvador, BA, BRA, 2013.

VINCENZI, A. M. R. et al. Evolução de software livre baseada em métricas de qualidade: um estudo de caso. In: *XI Simpósio Brasileiro de Sistemas de Informação (SBSI'2013)*. [S.l.]: João Pessoa, PB, BRA, 2013.

WEBER, S. *The Success of Open Source*. [S.l.]: Harvard University Press, 2004. Hardcover. ISBN 0674012925.

WOHLIN, C. et al. *Experimentation in Software Engineering*. [S.l.]: Springer, 2012. I-XXIII, 1-236 p. ISBN 978-3-642-29043-5.

ZAZWORKA, N. et al. A case study on effectively identifying technical debt. In: *Proceedings of the 17th International Conference on Evaluation and Assessment in Software Engineering*. New York, NY, USA: ACM, 2013. (EASE '13), p. 42–47. ISBN 978-1-4503-1848-8. Disponível em: <http://doi.acm.org/10.1145/2460999.2461005>.

APÊNDICE A

Tabelas com os dados coletados nos estudos experimentais

Este apêndice apresenta os dados brutos coletados dos projetos e que foram utilizados no desenvolvimento dos estudos experimentais conduzidos. A Tabela A.1 relaciona os projetos, a quantidade de versões e os dados relacionados à DT dos produtos de código aberto avaliados. Essa tabela possui a versão completa da classificação dos projetos, ordenada conforme os valores médios da DT. Já a Tabela A.2 relaciona todos os projetos selecionados para desenvolvimento dos estudos experimentais, com a respectiva URL do repositório do projeto e sua situação em relação à execução do experimento controlado.

Tabela A.1: *Classificação dos projetos conforme valores médios da DT (versão completa - ordenada pelo valor da DT)*

Seq.	Produto	Versões	Dívida Técnica	Comentários	Complexidade	Cobertura	Projeto	Duplicações	Violações
1	Autotags	3	1,1	0	29,7	0	0	39,6	30,69
2	Tiles 3	2	1,6	2,21	39,49	3,81	11,08	19,4	23,97
3	Tiles Request Framework	1	1,7	2,39	35,04	0	6,83	37,6	18,11
4	Apache ServiceMix Documentation	1	2	57,14	0	0	0	0	42,85
5	Apache James JSieve	8	3,25	11,7	8,19	10,69	12,32	24,77	32,3
6	ApacheDS 1.5 Build with Dependencies	1	4,8	0,67	3,24	6,44	1,26	78,16	10,2
7	A Release Audit Tool Library	1	5,1	28,38	6,02	1,16	0	54,78	9,64
8	RAT	8	5,11	20,39	11,51	6,47	1,07	44,55	15,98
9	Apache Rave	21	5,27	24,59	15,23	28,39	5,3	12,35	14,12
10	Apache James Hupa Parent	1	5,5	61,85	4,73	0	0	1,99	31,42
11	Lang Math	1	6,2	0,71	31,4	0	11,72	11,72	44,44
12	Apache Directory Shared Parent	40	6,24	0,29	1,65	3,54	1,11	88,43	4,95
13	Any23 - Anything to Triples	2	6,35	28,44	34,95	0	0	3,75	32,84
14	Ambari	2	6,7	29,1	0	46,11	0	0	24,79
15	Gshell	2	6,85	51,54	10,16	0	0	10,35	27,94
16	Camel Parent	1	6,9	35,84	1,57	46,55	12,57	0	3,45
17	Maven Meeper	3	7,2	31,43	21,85	17,2	0	0	29,51
18	Apache Avro Toplevel	18	7,55	14,01	28,59	15,85	3,18	11,52	26,83
19	Apache Giraph Parent	1	7,6	0,72	17,78	64,61	1,41	5,66	9,79
20	Apache Tika Reactor	2	7,6	15,66	19,72	29,37	7,28	5,46	22,48
21	Apache James JSPF	6	7,83	4,6	39,54	19,39	3,37	4,66	28,43
22	Doxia Sitetools	19	8,18	4,68	49,17	27,74	0	8,65	9,74
23	Whirr	3	8,2	18,87	7,38	24,94	26,33	10,72	11,72
24	Apache Incubator Giraph	4	8,25	5,26	23,34	19,34	26,42	6,35	19,25
25	Apache James JSPF Main	1	8,8	6,3	37,26	19,43	0	7,64	29,33
26	Maven Indexer	11	8,84	25,37	26,48	10,06	21,61	2,16	14,29
27	Apache JSecurity	1	9	12,9	15,65	45,59	14,07	0,46	11,29
28	Apache Vysper Parent	5	9,02	12,98	11,33	5,09	25,94	15	29,64
29	Apache Tika	28	9,11	12,95	21,84	29,86	6,54	7,88	20,89
30	Apache HttpAsyncClient	1	9,2	37,56	22,57	32,15	0	2,4	5,29

Continua na próxima página

Tabela A.1 – Continuação da página anterior

Seq.	Produto	Versões	Dívida Técnica	Comentários	Complexidade	Cobertura	Projeto	Duplicações	Violações
31	Apache Mina SSHD	1	9,2	28,28	18,43	12,74	12,08	6,04	22,39
32	Maven SCM	27	9,2	10,81	8,77	37,34	17,41	13,88	11,78
33	Lang	1	9,3	0	46,75	0	5,99	26,67	20,57
34	Apache Shiro	8	9,33	16,67	17,61	35,16	15,23	1,4	13,9
35	Struts 2	37	9,6	24,04	19,4	21,31	4,82	15	15,4
36	Apache James Protocols	30	9,67	8,32	9,16	36,29	29,6	4,75	11,85
37	Apache ACE POM	1	9,8	35,71	0	28,57	0	0	35,71
38	Apache MyFaces CODI Parent	18	9,8	16,37	14,04	60,73	3,53	0,71	4,59
39	Doxia Aggregator	1	10	0,24	22,29	6,69	2,53	5,58	62,65
40	Apache Flume	3	10,23	18,36	22,33	30,12	5,11	10,12	13,93
41	Distributed OSGI Reference Implementation	10	10,43	24,6	16,94	32,47	5,26	4,49	16,22
42	Apache Streams Project	1	10,5	55,37	0	28,09	0	0	16,52
43	Maven JXR Parent	7	10,51	4,09	41,82	26,38	0	0	27,7
44	Apache Shindig Project	16	10,65	13,4	18,85	50,02	8,3	0,88	8,52
45	Geronimo Specifications	3	10,67	30,27	8,76	38,63	6,25	3,91	12,15
46	Maven Integration Testing	1	10,9	17,22	5,38	71,47	0	0	5,92
47	Apache ServiceMix Kernel	14	10,99	10,53	12,46	49,06	0	10,08	17,85
48	Apache ServiceMix NMR	9	11,11	22,19	21,29	39,86	2,13	3,04	11,46
49	Apache James Mailets Aggregator	2	11,15	4,35	21,81	52,13	0	5,33	16,38
50	Apache James IMAP	12	11,16	15,73	17,29	37,73	4,58	12,65	12
51	Apache Wink	8	11,18	17,34	22,89	27,07	7,65	9,15	15,86
52	Apache Cocoon 3 Root	9	11,26	23,94	9,64	41,86	13,08	3,56	7,9
53	Apache Archiva Redback	1	11,3	23,63	14,92	0	0	5,87	55,56
54	Hadoop BSP	1	11,5	61,53	0	0	0	0	38,46
55	Apache Axis2 Transport Root	3	11,57	31,53	26,05	17,09	0,27	5,79	19,24
56	Apache ServiceMix Runtime	3	11,7	9,49	10,72	46,19	0	10,72	22,85
57	Redback Components Aggregator Project	1	11,8	18,56	10,42	21,45	0	1,63	47,92
58	ServiceMix Utils	15	11,91	18,43	16	53	0,75	5,08	6,72
59	Apache Component Companion for Log4j	4	11,93	1,66	12,89	61,58	13,84	4,36	5,65
60	SureFire	25	12,18	28,07	13,76	45,56	1,29	3,42	7,87
61	OpenNLP UIMA Annotators	2	12,4	10,3	13,91	55,87	0	8,24	11,67
62	Apache MyFaces Html5 Parent Project	1	12,5	9,68	14,64	51,11	1,8	9,01	13,74

Continua na próxima página

Tabela A.1 – Continuação da página anterior

Seq.	Produto	Versões	Dívida Técnica	Comentários	Complexidade	Cobertura	Projeto	Duplicações	Violações
63	Apache Sling Builder	3	12,63	8,54	17,5	45,79	3,7	12,41	12,03
64	Apache PDFBox	20	12,84	0,39	13,58	18,72	5,88	7,88	53,53
65	Commons Collections	5	12,96	14,8	21,32	12,29	28,98	13,41	9,19
66	Wicket Parent	5	13,02	1,29	11,38	20,41	20,94	2,94	43,02
67	Apache Tuscany DAS Implementation Project	1	13,1	24,31	24,47	7,15	7,36	5,26	31,42
68	Apache XML Security for Java	10	13,17	3,7	23,04	28,59	15,13	13,44	16,08
69	Tapestry	2	13,25	12,71	8,17	23,75	13,76	1,5	40,1
70	Gnuprologjava	2	13,4	5,46	14,29	0	0	18,09	62,16
71	ApacheDS	21	13,63	5,88	13,18	37,33	21,66	7,31	14,6
72	Apache Tomcat Maven Plugin	7	13,65	5,72	22,28	7,58	0	6,29	58,12
73	XmlSchema	15	13,67	20,26	16,55	34,85	2,33	6,72	19,28
74	Apache Shindig Project Parent	1	13,8	12,66	28,79	48,62	0	0	9,91
75	Maven Release	22	13,85	13,14	14,78	7,64	4,2	4,68	55,53
76	Apache Neethi	8	14,13	22,81	18,79	18,5	29,51	0	10,37
77	Apache Archiva Redback Components Aggregator	1	14,2	17,63	13,46	16,92	0	2,26	49,7
78	Tobago	1	14,4	17,75	12,55	47,92	0,52	9,35	11,88
79	Jakarta BCEL	2	14,5	10,19	14,23	46,51	5,93	5,64	17,49
80	Jsecurity	4	14,78	19,63	21,68	35,99	10,66	0	12,02
81	Giraph	1	14,8	5,58	33,64	15,38	21,33	0	24,04
82	Bookkeeper	2	14,85	16	21,35	35,77	5,51	8,92	12,44
83	OPS4J Pax Logging Build POM	23	14,91	14,19	10,54	35,25	11,2	2,58	26,21
84	Apache Pluto	10	14,94	17,32	8,86	41,88	1,89	8,26	21,76
85	Apache CXF	2	15,15	19,99	21,23	45,12	0,89	5,44	7,32
86	Axiom	5	15,18	17,29	15,92	37,37	9,67	9,06	10,66
87	Apache WSS4J	5	15,4	5,14	23,83	38,01	11,69	10,69	10,62
88	Apache OODT	4	15,48	11,66	12,89	31,57	1,25	22,37	20,24
89	Apache Abdera	1	15,5	25,67	15,35	37,66	2,78	5,63	12,89
90	PDFBox Reactor	15	15,58	0,46	11,01	35,77	5,55	5,77	41,42
91	Mercury	6	15,67	20,85	23,72	28,51	1,46	4,8	20,65
92	Apache Karaf	6	15,85	11,57	18,43	44,96	0,73	6,42	17,87
93	Apache PhotArk Common Services	1	16,3	20,69	0	31,37	0	21,78	26,14
94	Redback	1	16,4	15,77	9,79	27,12	1,66	4,76	40,88

Continua na próxima página

Tabela A.1 – Continuação da página anterior

Seq.	Produto	Versões	Dívida Técnica	Comentários	Complexidade	Cobertura	Projeto	Duplicações	Violações
95	Apache Chemistry OpenCMIS	9	16,4	12,66	15,06	47,18	0,81	14,82	9,45
96	Silverpeas Core	14	16,59	10,25	12,95	47,36	1,93	11,63	15,85
97	Struts Action Framework	1	16,8	10,88	20,97	34,37	15,49	2,15	16,11
98	Apache Directory Build	2	17,05	11,99	13,84	22,76	6,25	7,03	38,11
99	Apache MyFaces CODI	2	17,1	21,98	1,19	41,68	0	0	35,13
100	Roller Project	1	17,3	9,37	10,81	41,64	11,68	14,23	12,23
101	Apache Chukwa	2	17,35	11,89	9,38	37,65	6,13	13,54	21,41
102	Apache Turbine	3	17,5	0,53	11,03	38,65	5,26	7,77	36,73
103	Maven Shared Components	14	17,65	15,87	13,82	23,94	4,02	2,07	40,25
104	Maven Shared Components Aggregator	1	17,7	10,09	17,06	25,07	3,23	2,69	41,83
105	SonarSource Language Recognizer	17	17,81	21,48	6,65	48,49	19,25	0,08	4,02
106	Genesis	7	17,87	9,06	4,53	64,52	0	2,5	19,36
107	Maven Plugins Aggregator	1	18,2	3,75	15,99	27,66	5,8	4,45	42,32
108	PDFBox - Java PDF Library	1	18,4	0,13	8,93	41,34	3,58	6,75	39,23
109	Apache OpenWebBeans	29	18,71	9,29	11,19	17,05	26,95	1,39	34,1
110	Neethi	1	18,8	11,38	18,48	29,51	16,61	10,79	13,2
111	Oozie Main	1	18,9	4,4	11,9	36,3	17,03	17,33	13,02
112	Apache Directory Shared Build	6	18,93	2,8	12,09	23,78	9,3	19,14	32,87
113	Axiom Parent POM	5	19,26	15,15	15,23	51,73	3,92	3,69	10,26
114	Apache Aries	1	19,5	14,71	13,45	49,38	0,22	5,12	17,11
115	Apache Karaf Cave	2	19,5	8,9	21,67	41,88	0	0	27,54
116	OGNL - Object Graph Navigation Library	2	20,2	8,91	15,85	11,58	7,09	21,42	35,11
117	Commons Sanselan	2	20,85	15,64	13,92	12,17	12,5	5,21	40,54
118	ActiveMQ-CPP Project	16	21,09	6,74	12,27	34,17	0	39,43	7,36
119	Apache QPID Proton Project	1	21,3	17,31	8,52	34,44	8,92	3,53	27,24
120	OPS4J - Pax Logging	8	21,34	12,19	5,93	37,04	15,94	0	28,88
121	Apache Directory Studio Plugin	1	21,8	5,73	30,24	48,42	0	0	15,6
122	Maven Shared APIs	6	21,85	23,71	9,59	35,46	12,47	3,14	15,61
123	OpenJPA Aggregate Jar	1	21,9	15,31	18,8	40,99	10,35	6,73	7,79
124	Struts Action	1	22,2	3,86	8,84	25,74	2,6	46,2	12,72
125	Apache Empire-DB	24	22,22	8,79	12,23	31,48	6,32	13,28	27,86
126	Continuum Project Parent	6	23,02	12,6	9,19	33,04	0,26	5,71	39,17

Continua na próxima página

Tabela A.1 – Continuação da página anterior

Seq.	Produto	Versões	Dívida Técnica	Comentários	Complexidade	Cobertura	Projeto	Duplicações	Violações
127	Apache Sanselan	16	23,13	15,21	12,26	15,85	12,26	3,94	40,45
128	Continuum Project	7	23,47	14,12	11,83	29,05	0,4	4,98	39,59
129	Apache Felix	1	24,3	8,97	9,97	37,43	6,06	6,35	31,19
130	Struts	6	24,37	3,57	9,33	24,15	2,68	48,28	11,96
131	Tomcat Maven Plugin Project	2	25,2	0,19	13,31	45,45	0	0	41,03
132	OpenEJB Parent POM	1	25,8	14,2	6,09	24,54	4,11	38,39	12,65
133	Apache Yoko CORBA Server	9	26,58	10,48	5,82	22,03	2,72	37,02	21,89
134	Struts Action Parent	1	26,9	3,77	8,06	21,53	2,83	52,43	11,36
135	ApacheDS Installers Parent	11	27,17	13,24	0,21	38,38	0	0	48,16
136	Apache Directory Studio	27	27,29	4,45	9,2	31,4	8,31	18,97	27,65
137	Assemblies	5	29,42	7,14	11,34	25,35	2,34	48,64	5,17
138	Apache Directory Daemon	24	32,17	5,14	11,2	28,62	11,83	6,3	36,88
139	ActiveMQ-CPP Openwire Generator	1	34,5	3,42	16,76	38,13	0	32,82	8,84
140	Commons DBCP	1	43	10,72	15,44	44,82	0	2,86	26,14
Media dos Projetos/Eixos (MP)			14,33	14,64	15,53	29,55	6,39	10,96	22,91
Desvio Padrao (DP)			6,81	11,96	9,4	16,36	7,51	14,75	13,85
Limiar Inferior (LI = MP - DP)			7,52	2,68	6,13	13,18	-1,12	-3,8	9,07
Limiar Superior (LS = MP + DP)			21,84	17,33	21,66	42,73	5,27	7,16	31,98

Tabela A.2: *Relação dos projetos avaliados (URLs dos repositórios)*

Seq.	Projeto	Endereço/Repositório	Situação da Execução
1	Apache Accumulo	git://git.apache.org/accumulo.git	Falha de execução
2	Apache ACE	git://git.apache.org/ace.git	Falha de execução
3	Apache ActiveMQ	git://git.apache.org/activemq.git	Execução com Sucesso
4	Apache ActiveMQ ActiveIO	git://git.apache.org/activemq-activeio.git	Execução com Sucesso
5	Apache ActiveMQ Apollo	git://git.apache.org/activemq-apollo.git	Execução com Sucesso
6	Apache ActiveMQ CPP	git://git.apache.org/activemq-cpp.git	Execução com Sucesso
7	Apache ActiveMQ Protobuf	git://git.apache.org/activemq-protobuf.git	Execução com Sucesso
8	Apache ActiveMQ Stomp	git://git.apache.org/activemq-stomp.git	Erro na URL
9	Apache Aires	git://git.apache.org/aries.git	Falha de execução
10	Apache Ambari	git://git.apache.org/ambari.git	Erro na URL
11	Apache Amber	git://git.apache.org/amber.git	Erro na URL
12	Apache Any 23	git://git.apache.org/any23.git	Execução com Sucesso
13	Apache Archiva	http://svn.apache.org/repos/asf/archiva/trunk	Falha de execução
14	Apache Archiva	git://git.apache.org/archiva.git	Execução com Sucesso
15	Apache Avro	git://git.apache.org/avro.git	Falha de execução
16	Apache Axis 2	http://svn.apache.org/repos/asf/axis/axis2/java/core/trunk/	Falha de execução
17	Apache Axis2/Java	git://git.apache.org/axis2-java.git	Execução com Sucesso
18	Apache Axis2/Transports	git://git.apache.org/axis2-transports.git	Execução com Sucesso
19	Apache BigTop	git://git.apache.org/bigtop.git	Falha de execução
20	Apache Bookkeeper	git://git.apache.org/bookkeeper.git	Falha de execução
21	Apache BVal	git://git.apache.org/bval.git	Execução com Sucesso
22	Apache Camel	git://git.apache.org/camel.git	Execução com Sucesso
23	Apache Cayenne	http://svn.apache.org/repos/asf/cayenne/main/trunk/	Falha de execução
24	Apache Chainaw	git://git.apache.org/chainsaw.git	Execução com Sucesso
25	Apache Chemistry	git://git.apache.org/chemistry-opencmis.git	Falha de execução
26	Apache Chukwa	git://git.apache.org/chukwa.git	Execução com Sucesso
27	Apache Commons Lang	http://svn.apache.org/repos/asf/commons/proper/lang/trunk	Execução com Sucesso
28	Apache Creadur Rat	git://git.apache.org/creadur-rat.git	Execução com Sucesso
29	Apache Directory (DS)	https://svn.apache.org/repos/asf/directory/apacheds/trunk-with-dependencies	Falha de execução
30	Apache Jackrabbit	http://svn.apache.org/repos/asf/jackrabbit/trunk	Falha de execução

Continua na próxima página

Tabela A.2 – Continuação da página anterior

Seq.	Projeto	Endereço/Repositório	Situação da Execução
31	Apache James	http://svn.apache.org/repos/asf/james/current/	Falha de execução
32	Apache Jena	https://svn.apache.org/repos/asf/jena/trunk/	Falha de execução
33	Apache Mahout	http://svn.apache.org/repos/asf/lucene/mahout/trunk	Falha de execução
34	Apache My Faces CODI	http://svn.apache.org/repos/asf/myfaces/extensions/cdi/trunk/	Execução com Sucesso
35	Apache Open Web Beans	https://svn.apache.org/repos/asf/openwebbeans/trunk	Falha de execução
36	Apache Rave	http://svn.apache.org/repos/asf/rave/trunk	Execução com Sucesso
37	Apache Shiding	http://svn.apache.org/repos/asf/shindig/trunk/	Execução com Sucesso
38	Apache Shiro	http://svn.apache.org/repos/asf/shiro/trunk	Execução com Sucesso
39	Apache Sling	http://svn.apache.org/repos/asf/sling/trunk	Execução com Sucesso
40	Apache Synapse	http://svn.apache.org/repos/asf/synapse/trunk/java	Falha de execução
41	Apache Tika	http://svn.apache.org/repos/asf/tika/trunk	Execução com Sucesso
42	Apache Vysper	http://svn.apache.org/repos/asf/mina/vysper/trunk	Execução com Sucesso
43	BCEL (Jakarta Bcel)	http://svn.apache.org/repos/asf/commons/proper/bcel/trunk	Execução com Sucesso
44	Clerezza	git://git.apache.org/clerezza.git	Falha de execução
45	Cocoon	git://git.apache.org/cocoon.git	Falha de execução
46	Cocoon 3	http://svn.apache.org/repos/asf/cocoon/cocoon3/trunk	Execução com Sucesso
47	Commnos Transaction	git://git.apache.org/commons-transaction.git	Execução com Sucesso
48	Commons CLI	git://git.apache.org/commons-cli.git	Execução com Sucesso
49	Commons Codec	git://git.apache.org/commons-codec.git	Execução com Sucesso
50	Commons Collection	git://git.apache.org/commons-collections.git	Execução com Sucesso
51	Commons Collections	http://svn.apache.org/repos/asf/commons/proper/collections/trunk	Execução com Sucesso
52	Commons Compress	git://git.apache.org/commons-compress.git	Execução com Sucesso
53	Commons Configuration	http://svn.apache.org/repos/asf/commons/proper/configuration/trunk	Falha de execução
54	Commons Configuration	git://git.apache.org/commons-configuration.git	Execução com Sucesso
55	Commons DBCP	http://svn.apache.org/repos/asf/commons/proper/dbcp/trunk	Execução com Sucesso
56	Commons IO	git://git.apache.org/commons-io.git	Execução com Sucesso
57	Commons Lang	git://git.apache.org/commons-lang.git	Execução com Sucesso
58	Commons Logging	git://git.apache.org/commons-logging.git	Execução com Sucesso
59	Commons Math	git://git.apache.org/commons-math.git	Execução com Sucesso
60	Commons Net	git://git.apache.org/commons-net.git	Execução com Sucesso
61	Commons Validator	git://git.apache.org/commons-validator.git	Execução com Sucesso
62	Continuum	http://svn.apache.org/repos/asf/continuum/trunk	Execução com Sucesso

Continua na próxima página

Tabela A.2 – Continuação da página anterior

Seq.	Projeto	Endereço/Repositório	Situação da Execução
63	Continuum	git://git.apache.org/continuum.git	Falha de execução
64	CXF	git://git.apache.org/cxf.git	Falha de execução
65	CXF DOSGI	git://git.apache.org/cxf-dosgi.git	Execução com Sucesso
66	CXF Fediz	git://git.apache.org/cxf-fediz.git	Falha de execução
67	CXF Fediz		Erro na URL
68	Direct Memory	git://git.apache.org/directmemory.git	Falha de execução
69	Directory Clients	git://git.apache.org/directory-clients.git	Erro na URL
70	Directory Daemon	git://git.apache.org/directory-daemon.git	Falha de execução
71	Directory Installers	git://git.apache.org/directory-installers.git	Falha de execução
72	Directory Project	git://git.apache.org/directory-project.git	Falha de execução
73	Directory Server	git://git.apache.org/directory-server.git	Execução com Sucesso
74	Directory Shared	git://git.apache.org/directory-shared.git	Falha de execução
75	Directory Skins	git://git.apache.org/directory-skins.git	Falha de execução
76	Directory Studio	git://git.apache.org/directory-studio.git	Falha de execução
77	Directory Studio Plugin	git://git.apache.org/directory-studio-plugin.git	Execução com Sucesso
78	Empire DB	http://svn.apache.org/repos/asf/empire-db/trunk/	Execução com Sucesso
79	Empire DB	git://git.apache.org/empire-db.git	Falha de execução
80	Felix	git://git.apache.org/felix.git	Falha de execução
81	Flex	git://git.apache.org/flex-falcon.git	Erro na URL
82	Flume	git://git.apache.org/flume.git	Falha de execução
83	FTPServer	git://git.apache.org/ftpserver.git	Execução com Sucesso
84	Geronimo	git://git.apache.org/geronimo.git	Erro na URL
85	Geronimo DevTools	git://git.apache.org/geronimo-devtools.git	Execução com Sucesso
86	Geronimo Genesis	git://git.apache.org/geronimo-genesis.git	Falha de execução
87	Geronimo Gshell	git://git.apache.org/geronimo-gshell.git	Execução com Sucesso
88	Geronimo Jaspi	git://git.apache.org/geronimo-jaspi.git	Erro na URL
89	Geronimo Javamail	git://git.apache.org/geronimo-javamail.git	Erro na URL
90	Geronimo Schema 1.4	git://git.apache.org/geronimo-schema-14.git	Falha de execução
91	Geronimo Schema 5	git://git.apache.org/geronimo-schema-5.git	Falha de execução
92	Geronimo Schema 6	git://git.apache.org/geronimo-schema-6.git	Falha de execução
93	Geronimo Specifications	git://git.apache.org/geronimo-specs.git	Falha de execução
94	Geronimo Xbeans	git://git.apache.org/geronimo-xbean.git	Execução com Sucesso

Continua na próxima página

Tabela A.2 – Continuação da página anterior

Seq.	Projeto	Endereço/Repositório	Situação da Execução
95	Geronimo Yoko	git://git.apache.org/geronimo-yoko.git	Falha de execução
96	Geronomo TXManager	git://git.apache.org/geronimo-txmanager.git	Execução com Sucesso
97	Giraph	git://git.apache.org/giraph.git	Falha de execução
98	Gora	git://git.apache.org/gora.git	Falha de execução
99	Hadoop Commons	git://git.apache.org/hadoop-common.git	Execução com Sucesso
100	Hadoop Mapreduce	git://git.apache.org/hadoop-mapreduce.git	Erro na URL
101	Hama	git://git.apache.org/hama.git	Falha de execução
102	Hbase	git://git.apache.org/hbase.git	Falha de execução
103	Hcatalog	git://git.apache.org/hcatalog.git	Falha de execução
104	Hive	git://git.apache.org/hive.git	Erro na URL
105	Http Client	git://git.apache.org/httpclient.git	Execução com Sucesso
106	Http Components Core	git://git.apache.org/httpcore.git	Execução com Sucesso
107	HttpPasyncClient	git://git.apache.org/httpasyncclient.git	Execução com Sucesso
108	Incubator Cloudstack	git://git.apache.org/incubator-cloudstack.git	Erro na URL
109	Incubator Crunch	git://git.apache.org/incubator-crunch.git	Erro na URL
110	Incubator HDT	git://git.apache.org/incubator-hdt.git	Falha de execução
111	Incubator Helix	git://git.apache.org/incubator-helix.git	Execução com Sucesso
112	Incubator Wink	http://svn.apache.org/repos/asf/incubator/wink/trunk	Erro na URL
113	Isis	git://git.apache.org/isis.git	Falha de execução
114	Jackrabbit	git://git.apache.org/jackrabbit.git	Falha de execução
115	Jackrabbit Oak	git://git.apache.org/jackrabbit-oak.git	Execução com Sucesso
116	James	git://git.apache.org/james.git	Execução com Sucesso
117	James Hupa	git://git.apache.org/james-hupa.git	Execução com Sucesso
118	James IMAP	git://git.apache.org/james-imap.git	Execução com Sucesso
119	James JDKIM	git://git.apache.org/james-jdkim.git	Execução com Sucesso
120	James Jsieve	git://git.apache.org/james-jsieve.git	Execução com Sucesso
121	James JSPF	git://git.apache.org/james-jspf.git	Execução com Sucesso
122	James MailBox	git://git.apache.org/james-mailbox.git	Execução com Sucesso
123	James Mailet	git://git.apache.org/james-mailet.git	Execução com Sucesso
124	James Mime4	git://git.apache.org/james-mime4j.git	Execução com Sucesso
125	James Postage	git://git.apache.org/james-postage.git	Falha de execução
126	James Postage	git://git.apache.org/james-postage.git	Erro na URL

Continua na próxima página

Tabela A.2 – Continuação da página anterior

Seq.	Projeto	Endereço/Repositório	Situação da Execução
127	James Protocols	git://git.apache.org/james-protocols.git	Execução com Sucesso
128	Jena	git://git.apache.org/jena.git	Erro na URL
129	JSPWiki	git://git.apache.org/jspwiki.git	Falha de execução
130	Kalumet	git://git.apache.org/kalumet.git	Falha de execução
131	Karaf	https://svn.apache.org/repos/asf/karaf/trunk	Falha de execução
132	Karaf	git://git.apache.org/karaf.git	Execução com Sucesso
133	Karaf Cave	git://git.apache.org/karaf-cave.git	Execução com Sucesso
134	Karaf Cellar	git://git.apache.org/karaf-cellar.git	Execução com Sucesso
135	Karaf Site	git://git.apache.org/karaf-site.git	Falha de execução
136	Karaf Web Console	git://git.apache.org/karaf-webconsole.git	Falha de execução
137	Lenya	git://git.apache.org/lenya.git	Falha de execução
138	Log4 CXX	git://git.apache.org/log4cxx.git	Falha de execução
139	Log4j	git://git.apache.org/log4j.git	Execução com Sucesso
140	Log4j Component	git://git.apache.org/log4j-component.git	Execução com Sucesso
141	Log4j Extras	git://git.apache.org/log4j-extras.git	Execução com Sucesso
142	Log4j Recievers	git://git.apache.org/log4j-receivers.git	Execução com Sucesso
143	Log4j Zeroconf	git://git.apache.org/log4j-zeroconf.git	Execução com Sucesso
144	Log4net	git://git.apache.org/log4net.git	Falha de execução
145	Logback	http://svn.qos.ch/repos/logback/trunk/	Erro na URL
146	Logging Log4php	git://git.apache.org/logging-log4php.git	Falha de execução
147	Mahout	git://git.apache.org/mahout.git	Execução com Sucesso
148	Manifoldcf	git://git.apache.org/manifoldcf.git	Falha de execução
149	Maven Ant Tasks	git://git.apache.org/maven-ant-tasks.git	Execução com Sucesso
150	Maven App Engine	git://git.apache.org/maven-app-engine.git	Falha de execução
151	Maven Archetype	git://git.apache.org/maven-archetype.git	Execução com Sucesso
152	Maven Doxia	http://svn.apache.org/repos/asf/maven/doxia/trunks/	Execução com Sucesso
153	Maven Doxia	git://git.apache.org/maven-doxia.git	Execução com Sucesso
154	Maven Doxia SiteTools	git://git.apache.org/maven-doxia-sitetools.git	Execução com Sucesso
155	Maven Doxia Tools	git://git.apache.org/maven-doxia-tools.git	Execução com Sucesso
156	Maven Enforcer	git://git.apache.org/maven-enforcer.git	Execução com Sucesso
157	Maven Indexer	git://git.apache.org/maven-indexer.git	Execução com Sucesso
158	Maven Integration Testing	git://git.apache.org/maven-integration-testing.git	Execução com Sucesso

Continua na próxima página

Tabela A.2 – Continuação da página anterior

Seq.	Projeto	Endereço/Repositório	Situação da Execução
159	Maven JKR	git://git.apache.org/maven-jxr.git	Execução com Sucesso
160	Maven Mercury	git://git.apache.org/maven-mercury.git	Execução com Sucesso
161	Maven Plugin Testing	git://git.apache.org/maven-plugin-testing.git	Execução com Sucesso
162	Maven Plugin Tools	git://git.apache.org/maven-plugin-tools.git	Execução com Sucesso
163	Maven Plugins	https://svn.apache.org/repos/asf/maven/plugins/trunk/	Falha de execução
164	Maven Plugins	git://git.apache.org/maven-plugins.git	Falha de execução
165	Maven Pom	git://git.apache.org/maven-pom.git	Erro na URL
166	Maven Release	https://svn.apache.org/repos/asf/maven/release/trunk/	Execução com Sucesso
167	Maven Release	git://git.apache.org/maven-release.git	Execução com Sucesso
168	Maven Repository Tools	git://git.apache.org/maven-repository-tools.git	Execução com Sucesso
169	Maven Resources	git://git.apache.org/maven-resources.git	Falha de execução
170	Maven SCM	git://git.apache.org/maven-scm.git	Execução com Sucesso
171	Maven Shared	git://git.apache.org/maven-shared.git	Execução com Sucesso
172	Maven Shared (Componentes)	http://svn.apache.org/repos/asf/maven/shared/trunk/	Execução com Sucesso
173	Maven Site	git://git.apache.org/maven-site.git	Falha de execução
174	Maven Skins	git://git.apache.org/maven-skins.git	Falha de execução
175	Maven Surefire	git://git.apache.org/maven-surefire.git	Execução com Sucesso
176	Maven Wagon	git://git.apache.org/maven-wagon.git	Execução com Sucesso
177	Maven2	git://git.apache.org/maven-2.git	Execução com Sucesso
178	Mina SSHD	git://git.apache.org/mina-sshd.git	Execução com Sucesso
179	Mrunit	git://git.apache.org/mrunit.git	Falha de execução
180	MyFaces	git://git.apache.org/myfaces.git	Execução com Sucesso
181	MyFaces Extcdi	git://git.apache.org/myfaces-extcdi.git	Execução com Sucesso
182	MyFaces Extval	git://git.apache.org/myfaces-extval.git	Falha de execução
183	MyFaces HTML5	git://git.apache.org/myfaces-html5.git	Execução com Sucesso
184	MyFaces Portlet Bridge	git://git.apache.org/myfaces-portlet-bridge.git	Falha de execução
185	MyFaces Scripting	git://git.apache.org/myfaces-scripting.git	Falha de execução
186	MyFaces Tobago	http://svn.apache.org/repos/asf/myfaces/tobago/trunk	Falha de execução
187	Npanday	git://git.apache.org/npanday.git	Falha de execução
188	Npanday-Its	git://git.apache.org/npanday-its.git	Falha de execução
189	Nutch	git://git.apache.org/nutch.git	Execução com Sucesso
190	Nuvem	git://git.apache.org/nuvem.git	Falha de execução

Continua na próxima página

Tabela A.2 – Continuação da página anterior

Seq.	Projeto	Endereço/Repositório	Situação da Execução
191	OGNL - Object Graph Navigation Library	http://svn.apache.org/repos/asf/commons/proper/ognl/trunk/	Execução com Sucesso
192	Open EJB	https://svn.apache.org/repos/asf/openejb/trunk/openejb3	Erro na URL
193	Open NLP	https://svn.apache.org/repos/asf/opennlp/trunk/opennlp/	Falha de execução
194	Open NLP Maxent	https://svn.apache.org/repos/asf/opennlp/trunk/opennlp-maxent/	Erro na URL
195	Open NLP Tools	https://svn.apache.org/repos/asf/opennlp/trunk/opennlp-tools/	Falha de execução
196	Open NLP UIMA	https://svn.apache.org/repos/asf/opennlp/trunk/opennlp-uima/	Execução com Sucesso
197	PDF Box	http://svn.apache.org/repos/asf/pdfbox/trunk/	Execução com Sucesso
198	Pluto	https://svn.apache.org/repos/asf/portals/pluto/trunk/	Falha de execução
199	Sand Box (Apache Commons)	http://svn.apache.org/repos/asf/struts/sandbox/trunk	Falha de execução
200	Spamassassin	http://svn.apache.org/repos/asf/spamassassin/trunk	Falha de execução
201	Struts 1	http://svn.apache.org/repos/asf/struts/struts1/trunk	Falha de execução
202	Struts 2	http://svn.apache.org/repos/asf/struts/struts2/trunk	Falha de execução
203	Turbine	http://svn.apache.org/repos/asf/turbine/core/trunk	Execução com Sucesso
204	Xwork	https://svn.opensymphony.com/svn/xwork/trunk	Erro na URL
205	Projeto com falha	git://git.apache.org/ode.git	Falha de execução
206	Projeto com falha	git://git.apache.org/odftoolkit.git	Falha de execução
207	Projeto com falha	git://git.apache.org/oodt.git	Falha de execução
208	Projeto com falha	git://git.apache.org/oozie.git	Falha de execução
209	Projeto com falha	git://git.apache.org/openjpa.git	Falha de execução
210	Projeto com falha	git://git.apache.org/openwebbeans.git	Falha de execução
211	Projeto com falha	git://git.apache.org/pdfbox.git	Falha de execução
212	Projeto com falha	git://git.apache.org/photark-mobile.git	Falha de execução
213	Projeto com falha	git://git.apache.org/photark.git	Falha de execução
214	Projeto com falha	git://git.apache.org/pivot.git	Falha de execução
215	Projeto com falha	git://git.apache.org/pluto.git	Falha de execução
216	Projeto com falha	git://git.apache.org/qpid-proton.git	Falha de execução
217	Projeto com falha	git://git.apache.org/rampart.git	Falha de execução
218	Projeto com falha	git://git.apache.org/rat.git	Falha de execução
219	Projeto com falha	git://git.apache.org/rave.git	Falha de execução
220	Projeto com falha	git://git.apache.org/redback-components.git	Falha de execução
221	Projeto com falha	git://git.apache.org/redback-core.git	Falha de execução
222	Projeto com falha	git://git.apache.org/roller.git	Falha de execução

Continua na próxima página

Tabela A.2 – Continuação da página anterior

Seq.	Projeto	Endereço/Repositório	Situação da Execução
223	Projeto com falha	git://git.apache.org/sandesha.git	Falha de execução
224	Projeto com falha	git://git.apache.org/sanselan.git	Falha de execução
225	Projeto com falha	git://git.apache.org/santuario-java.git	Falha de execução
226	Projeto com falha	git://git.apache.org/servicemix-archetypes.git	Falha de execução
227	Projeto com falha	git://git.apache.org/servicemix-components.git	Falha de execução
228	Projeto com falha	git://git.apache.org/servicemix-documentation.git	Falha de execução
229	Projeto com falha	git://git.apache.org/servicemix-maven-plugins.git	Falha de execução
230	Projeto com falha	git://git.apache.org/servicemix-utils.git	Falha de execução
231	Projeto com falha	git://git.apache.org/servicemix-website.git	Falha de execução
232	Projeto com falha	git://git.apache.org/servicemix3.git	Falha de execução
233	Projeto com falha	git://git.apache.org/servicemix4-bundles.git	Falha de execução
234	Projeto com falha	git://git.apache.org/servicemix4-features.git	Falha de execução
235	Projeto com falha	git://git.apache.org/servicemix4-kernel.git	Falha de execução
236	Projeto com falha	git://git.apache.org/servicemix4-nmr.git	Falha de execução
237	Projeto com falha	git://git.apache.org/servicemix5.git	Falha de execução
238	Projeto com falha	git://git.apache.org/shindig.git	Falha de execução
239	Projeto com falha	git://git.apache.org/shiro.git	Falha de execução
240	Projeto com falha	git://git.apache.org/sling.git	Falha de execução
241	Projeto com falha	git://git.apache.org/stanbol.git	Falha de execução
242	Projeto com falha	git://git.apache.org/streams.git	Falha de execução
243	Projeto com falha	git://git.apache.org/struts-sandbox.git	Falha de execução
244	Projeto com falha	git://git.apache.org/struts-site.git	Falha de execução
245	Projeto com falha	git://git.apache.org/struts1.git	Falha de execução
246	Projeto com falha	git://git.apache.org/struts2.git	Falha de execução
247	Projeto com falha	git://git.apache.org/synapse.git	Falha de execução
248	Projeto com falha	git://git.apache.org/tapestry3.git	Falha de execução
249	Projeto com falha	git://git.apache.org/tapestry4.git	Falha de execução
250	Projeto com falha	git://git.apache.org/tika.git	Falha de execução
251	Projeto com falha	git://git.apache.org/tiles-autotag.git	Falha de execução
252	Projeto com falha	git://git.apache.org/tiles-request.git	Falha de execução
253	Projeto com falha	git://git.apache.org/tiles.git	Falha de execução
254	Projeto com falha	git://git.apache.org/tomcat-maven-plugin.git	Falha de execução

Continua na próxima página

Tabela A.2 – Continuação da página anterior

Seq.	Projeto	Endereço/Repositório	Situação da Execução
255	Projeto com falha	git://git.apache.org/tomee.git	Falha de execução
256	Projeto com falha	git://git.apache.org/tuscany-das.git	Falha de execução
257	Projeto com falha	git://git.apache.org/tuscany-sca-1.x.git	Falha de execução
258	Projeto com falha	git://git.apache.org/tuscany-sca-2.x.git	Falha de execução
259	Projeto com falha	git://git.apache.org/tuscany-sdo.git	Falha de execução
260	Projeto com falha	git://git.apache.org/vysper.git	Falha de execução
261	Projeto com falha	git://git.apache.org/webservices-axiom.git	Falha de execução
262	Projeto com falha	git://git.apache.org/webservices-commons-xmlschema.git	Falha de execução
263	Projeto com falha	git://git.apache.org/webservices-neethi.git	Falha de execução
264	Projeto com falha	git://git.apache.org/webservices-xmlschema.git	Falha de execução
265	Projeto com falha	git://git.apache.org/whirr.git	Falha de execução
266	Projeto com falha	git://git.apache.org/wicket.git	Falha de execução
267	Projeto com falha	git://git.apache.org/wink.git	Falha de execução
268	Projeto com falha	git://git.apache.org/wss4j.git	Falha de execução
269	Projeto com falha	git://git.savannah.gnu.org/gnuprologjava.git	Falha de execução
270	Projeto com falha	https://github.com/Silverpeas/Silverpeas-Core.git	Falha de execução
271	Projeto com falha	https://github.com/SonarSource/sonar.git	Falha de execução
272	Projeto com falha	https://github.com/ops4j/org.ops4j.pax.logging.git	Falha de execução
273	Projeto com falha	https://github.com/SonarSource/sslr.git	Falha de execução
274	Projeto com falha	https://github.com/xwiki/xwiki-rendering.git	Falha de execução
275	Projeto com falha	git://git.apache.org/abdera.git	Falha de execução
276	Projeto com falha	git://git.apache.org/cayenne.git	Falha de execução

APÊNDICE B

Automatização dos estudos experimentais

Este apêndice apresenta os *scripts* elaborados para o processo de automatização do experimento controlado, incluindo a coleta de dados. Os ScriptSVN B.1 e B.2 relacionam-se ao código fonte utilizado para execução do experimento nos projetos do repositório SVN. Os ScriptGIT B.3 e B.4 relacionam-se ao código fonte utilizado para execução do experimento nos projetos do repositório GIT. Já os ScriptPython B.5, B.6, B.7 e B.8 apresentam o código fonte utilizado para a coleta de dados dos projetos, após a execução do experimento controlado, possibilitando a visualização dos dados em tabelas.

Script SVN B.1 Script para experimentação de projetos do repositório Subversion

```
#!/bin/bash
################################################################################
# Percorre uma lista de repositorios, realiza o download dos mesmos em uma pasta
# numerada incrementalmente, compila e tenta carregar os projetos no sonar.
#
# A lista de repositorios deve ser fornecida de parametro na chamada do script.
# Além disso, o numero inicial que será usado para numerar os projetos tambem
# deve ser fornecido.
#
# Por exemplo:
#
# script-svn-new.sh lista-svn 1
#
# Inicia lendo as linhas do arquivo lista-svn e cria diretorios PROJETO_XX
# onde XX e iniciado de 1. Para cada projeto, executa o build para cada versão.
################################################################################

FILE=$1
COUNTER=$2
for REPO in $(cat $FILE)
do
   PRJ=SVNPRJ_$COUNTER
   LOG=SVNPRJ_LOG_$COUNTER

   echo "### $REPO - $PRJ" &> $LOG


   # Identificando lista de revisões do repositorio
   revisions-svn.sh $REPO > $PRJ-revisions.txt

   OLD_VERSION="\"<version>\""

   for REV in $(cat $PRJ-revisions.txt)
   do
      # Realizando o download do projeto
      svn co -r $REV $REPO $PRJ &>> $LOG

      MVN=/programs/apache-maven-2.2.1/bin/mvn
```

Script SVN B.2 Script para experimentação de projetos do repositório Subversion (cont.)

```
40          # Verifica se foi feito checkout
41          if [ -d "$PRJ" ]; then
42             # Entrando no diretorio do projeto
43             cd $PRJ
44
45             # Realizando o download do projeto
46             CUR_VERSION=$($MVN org.apache.maven.plugins:maven-help-plugin:2.1.1:
                   evaluate -Dexpression=project.version | awk '{if(index($1,"[INFO]"
                   )==0) print $0;}')
47
48             echo "------------------------------------------------" &>> ../$LOG
49             echo "DADOS ATUAIS - Revisão: $REV - Versão Antiga: $OLD_VERSION -
                   Versão Atual: $CUR_VERSION" &>> ../$LOG
50             echo "------------------------------------------------" &>> ../$LOG
51
52             diff  <(echo $OLD_VERSION ) <(echo $CUR_VERSION)
53             RETVAL=$?
54
55             if [ $RETVAL -eq 1 ]; then
56                # Rodando o maven
57                $MVN clean install test-compile -DskipTests=true -Dgpg.skip=true &>>
                        ../$LOG
58                RETVAL=$?
59
60                if [ $RETVAL -eq 0 ]; then
61                   $MVN sonar:sonar -Dmaven.test.failure.ignore=true -Dsurefire.
                         timeout=30000 -Dgpg.skip=true -Dsonar.jdbc.url="jdbc:mysql
                         ://localhost:3306/sonar?useUnicode=true&characterEncoding=
                         utf8&rewriteBatchedStatements=true" -Dsonar.jdbc.
                         driverClassName="com.mysql.jdbc.Driver" &>> ../$LOG
62                else
63                   MVN=/programs/apache-maven-3.1.1/bin/mvn
64                   $MVN clean install test-compile -DskipTests=true -Dgpg.skip=true
                         &>> ../$LOG
65                   $MVN sonar:sonar -Dmaven.test.failure.ignore=true -Dsurefire.
                         timeout=30000 -Dgpg.skip=true -Dsonar.jdbc.url="jdbc:mysql
                         ://localhost:3306/sonar?useUnicode=true&characterEncoding=
                         utf8&rewriteBatchedStatements=true" -Dsonar.jdbc.
                         driverClassName="com.mysql.jdbc.Driver" &>> ../$LOG
66                fi
67                RETVAL=$?
68             fi
69             #Saindo do diretorio
70             cd ..
71             if [ $RETVAL -eq 0 ]; then
72                OLD_VERSION=$CUR_VERSION
73             fi
74          fi
75       done
76
77       #Incrementando o contador de projetos
78       let COUNTER=COUNTER+1
79    done
```

Script Git B.3 Script para experimentação de projetos do repositório GitHub

```
#!/bin/bash
################################################################################
# Percorre uma lista de repositórios, realiza o download dos mesmos em uma pasta
# numerada incrementalmente. Após, compila e tenta carregar os projetos no Sonar
#
# A lista de repositórios deve ser fornecida como parâmetro do script.
# Além disso, o número inicial que será usado para enumerar os projetos também
# deve ser fornecido.
#
# Por exemplo:
#
# script-git-new.sh lista-git 1
#
# Inicia lendo as linhas do arquivo lista-git-new e cria diretorios GITPRJ_XX
# onde XX e iniciado com o número fornecido. Para cada projeto, executa o build
# para cada versão.
################################################################################

FILE=$1
COUNTER=$2
for REPO in $(cat $FILE)
do
   PRJ=GITPRJ_$COUNTER
   LOG=GITPRJ_LOG_$COUNTER

   echo "### $REPO - $PRJ" &> $LOG

   # Chekout do repositório
   git clone $REPO $PRJ &>> $LOG

   # Identificando lista de revisões do repositorio
   /home/inf/Documentos/Igor/EXPERIMENTO/revisions-git.sh $PRJ > $PRJ-revisions.
       txt

   OLD_VERSION="\"<version>\""

   cd $PRJ

   for REV in $(cat ../$PRJ-revisions.txt)
   do
      # Restaurando determinada revisão
      git reset --hard $REV &>> ../$LOG

      MVN=/programs/apache-maven-2.2.1/bin/mvn

      # Realizando o download do projeto
      CUR_VERSION=$($MVN org.apache.maven.plugins:maven-help-plugin:2.1.1:
          evaluate -Dexpression=project.version | awk '{if(index($1,"[INFO]")
          ==0) print $0;}') &>> $LOG

   echo "-------------------------------------------------" &>> ../$LOG
      echo "DADOS ATUAIS - Revisão: $REV - Versão Antiga: $OLD_VERSION - Versão
          Atual: $CUR_VERSION" &>> ../$LOG
      echo "-------------------------------------------------" &>> ../$LOG

      diff  <(echo $OLD_VERSION ) <(echo $CUR_VERSION)
      RETVAL=$?
```

Script Git B.4 Script para experimentação de projetos do repositório GitHub (cont.)

```
55        if [ $RETVAL -eq 1 ]; then
56           # Rodando o maven
57           $MVN clean install test-compile -DskipTests=true -Dgpg.skip=true &>>
                 ../$LOG
58           RETVAL=$?
59
60           if [ $RETVAL -eq 0 ]; then
61              $MVN sonar:sonar -Dmaven.test.failure.ignore=true -Dsurefire.timeout
                    =30000 -Dgpg.skip=true -Dsonar.jdbc.url="jdbc:mysql://localhost
                    :3306/sonar?useUnicode=true&characterEncoding=utf8&
                    rewriteBatchedStatements=true" -Dsonar.jdbc.driverClassName="
                    com.mysql.jdbc.Driver" &>> ../$LOG
62           else
63              MVN=/programs/apache-maven-3.1.1/bin/mvn
64              $MVN clean install test-compile -DskipTests=true -Dgpg.skip=true &>>
                     ../$LOG
65              $MVN sonar:sonar -Dmaven.test.failure.ignore=true -Dsurefire.timeout
                    =30000 -Dgpg.skip=true -Dsonar.jdbc.url="jdbc:mysql://localhost
                    :3306/sonar?useUnicode=true&characterEncoding=utf8&
                    rewriteBatchedStatements=true" -Dsonar.jdbc.driverClassName="
                    com.mysql.jdbc.Driver" &>> ../$LOG
66           fi
67           RETVAL=$?
68        fi
69
70        if [ $RETVAL -eq 0 ]; then
71           OLD_VERSION=$CUR_VERSION
72        fi
73     done
74     #Saindo do diretório
75     cd ..
76
77     #Incrementando o contador de projetos
78     let COUNTER=COUNTER+1
79  done
```

Script Python B.5 Script para coleta de dados dos projetos

```
#!/usr/bin/python
# coding: iso-8859-1
import json
import urllib2
import csv

keyProj = {}         #dicionário contendo a chave e o nome de cada projeto (chave, nome)
projVersion = {}     #dicionário contendo todas as versões de cada projeto (chave, versões[(version,date)])
projMetrics = {}     #dicionário contendo as métricas de cada versão do projeto (chave, [date,metricas])
metricas = "ncloc,coverage,total-quality,qi-quality-index,sqale_index,sqale_rating,technical_debt,technical_debt_days,technical_debt_ratio,technical_debt_repart"


#-------------------------------------------------------#
#   Obter as chaves                                     #
#-------------------------------------------------------#
def getKeys():
   global keyProj
   key = ""
   name = ""

   url = "http://localhost:9000/api/resources?format=json"
   data = json.load(urllib2.urlopen(url))

   #Percorrer cada projeto
   for proj in data:
      #recupera a chave o nome de cada projeto
      key = proj["key"]
      name = proj["name"]

      #salva no dicionario
      keyProj[key] = name

#-------------------------------------------------------#
#   Consulta Events                                     #
#-------------------------------------------------------#
def getEvent():
   global keyProj
   global projVersion

   url = "http://localhost:9000/api/events?resource=[key]&categories=Version&format=json"

   for item in keyProj.items():
      key = item[0]
      novaUrl = url.replace("[key]", key) #cria url com a chave do projeto

      #Executa a consulta
      data = json.load(urllib2.urlopen(novaUrl))

      extraiVersao(key, data)

#-------------------------------------------------------#
# Extrai as versões e datas de cada projeto             #
#-------------------------------------------------------#
def extraiVersao(key, data):
   global projVersion
   vers = []   #lista com todas as versões e datas do projeto

   #Recupera todas as versões do projeto e adiciona na lista
   for v in data:
      vers.append((v["n"], v["dt"]))      #(version, date)

   #adiciona no dicionário a chave do projeto e suas versões
   vers.reverse()
   projVersion[key] = vers
```

Script Python B.6 Script para coleta de dados dos projetos (cont.)

```
#-------------------------------------------------------#
#   Consulta Time Machine                               #
#-------------------------------------------------------#
def getTimeMachine():
   global keyProj
   global metricas

   urlTM = 'http://localhost:9000/api/timemachine?resource=[key]&format=json&
       metrics=[metricas]'
   url = urlTM.replace("[metricas]", metricas)

   #Percorre cada chave do dicionário
   for item in keyProj.items():
      key = item[0]
      novaUrl = url.replace("[key]", key) #cria url com a chave do projeto

      #Executa a consulta
      data = json.load(urllib2.urlopen(novaUrl))

      data = data[0]['cells']

      listaDados = []            # lista (data, metricas)

      for d in data:
         listaVersao = [d['d']]

         #adiciona as metricas na lista
         lista = d['v']

         #adiciona as metricas na lista, percorrer ate o penultimo elemento
         for m in lista[:-1]:
            listaVersao.append(m)


         listaVersao += normalizeTD(lista[-1])

         #lista de listas[date,metricas]
         listaDados.append(listaVersao)

      #listaDados.reverse() -> sonar traz em ordem diferento da requisão eventos
      projMetrics[key] = listaDados
```

Script Python B.7 Script para coleta de dados dos projetos (cont.)

```
108 #-------------------------------------------------------#
109 #   Normaliza os campos do TD                           #
110 #-------------------------------------------------------#
111 def normalizeTD(string):
112     campos = ["Comments=","Complexity=","Coverage=","Design=","Duplication=", "
            Violations="]
113 
114     for c in campos:
115         if c in string:
116             valor = string.split(c)[1]
117             valor = valor.split(";")[0]
118             campos[campos.index(c)] = valor
119         else:
120             campos[campos.index(c)] = 0.0
121 
122     return campos
123 
124 #-------------------------------------------------------#
125 #   Cria um arquivo csv                                 #
126 #-------------------------------------------------------#
127 def saveCsv():
128     global metricas
129 
130     #Cria o arquivo csv com os dados
131     f = open("ColetaExperimento1.csv", "wb")
132 
133     #Cria o cabeçalho
134     m = ["Chave", "Nome", "Versao", "Data"]
135     m += metricas.split(",")
136 
137     #remove o ultimo campo, td
138     m.pop()
139 
140     m += ["TD-Comments", "TD-Complexity", "TD-Coverage", "TD-Design","TD-
            Duplication","TD-Violations"]
141 
142     out = csv.writer(f, delimiter=',',quoting=csv.QUOTE_ALL)
143 
144     #Salva o cabeçalho no arquivo
145     out.writerow(m)
146 
147     saveData(out)
148 
149     f.close()
```

Script Python B.8 Script para coleta de dados dos projetos (cont.)

```
#-------------------------------------------------------#
#   Salva os dados em um arquivo csv                    #
#-------------------------------------------------------#
def saveData(out):
   global keyProj
   global projVersion
   global projMetrics

   listaFile = []

   #Recupera todas as chaves
   for key in keyProj:

      versions = projVersion[key]   #versions[(version,date)]
      metricas = projMetrics[key]   #lista[date,metricas]


      #percorre cada versao
      for version in versions:
         listaFile = [key, keyProj[key]]  #[chave, nome]

         listaFile.append(version[0])

         #extrair a data
         date = version[1]

         #percorrer cada elemento em metricas em busca da data
         for elem in metricas:
            if date == elem[0]:
               #percorre cada elemento da lista e adiciona na lista final
               for e in elem:
                  listaFile.append(e)

               #salva no arquivo
               out.writerow(listaFile)

#-------------------------------------------------------#
#   Main                                                #
#-------------------------------------------------------#
getKeys()
getEvent()
getTimeMachine()
saveCsv()
```