Gerenciando dívida técnica: estado atual e novas propostas em métodos de medida

Diogo de Jesus Pina

Dissertação apresentada
ao
Instituto de Matemática e Estatística
da
Universidade de São Paulo
para
obtenção do título
de
Mestre em Ciência da Computação

Programa: Mestrado em Ciência da Computação
Orientador: Prof. Dr. Alfredo Goldman

Durante o desenvolvimento deste trabalho o autor recebeu auxílio financeiro da CAPES

São Paulo, Agosto de 2016

Gerenciando dívida técnica: estado atual e novas propostas em métodos de medida

Esta é a versão original da dissertação elaborada pelo
candidato (Diogo de Jesus Pina), tal como
submetida à Comissão Julgadora.

Agradecimentos

Em primeiro lugar, quero agradecer a Deus por permitir que eu pudesse realizar mais este sonho com muita dedicação e sacrifícios.

Quero agradecer a minha família, que sempre me apoiaram nos estudos e se esforçaram para que eu pudesse estudar, tanto com recursos financeiros e, principalmente, com amor, desde a educação básica até os dias de hoje.

Quero agradecer ao professor Alfredo Goldman que desde a minha graduação vem me apoiando e me orientando em relação aos estudos e vida acadêmica, em especial sobre dívida técnica. Junto com o professor, quero agradecer também a minha amiga Graziela Tonin, que me apresentou o tema de dívida técnica e é minha parceira de discussões de pesquisa.

Resumo

Pina, Diogo de Jesus **Gerenciando Dívida Técnica: Estado atual e novas propostas em métodos de medida**. 2016. 120 f. Dissertação (Mestrado) - Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2016.

A metáfora da dívida técnica foi criada por Cunningham e descreve que as questões técnicas de qualidade de software podem trazer problemas futuros caso não sejam resolvidas imediatamente. Após mais de dez anos da criação da metáfora, foram publicados estudos de classificação, como o de McConnel e Fowler, e também de conceitualização, como o artigo *Mensuring and Monitoring Techical Debt.*

Este trabalho foca em métodos de medida de dívida técnica e uma revisão da literatura mostrou uma divisão dos trabalhos já realizados conforme a abordagem utilizada, dividindo-os em: conceituais, análise de métricas de código, análise de violações de propriedades de qualidade, aprendizagem de máquina, enfase em ferramentas e pesquisa qualitativa.

Este trabalho também apresenta alguns métodos de medida de dívida técnica que visam ser precisos, flexíveis e contextualizáveis para ajudar no processo de tomada de decisão. Estes métodos são: Padrão Sonar Qube que captura o custo calculado pelo Sonar Qube, Regressão Estatística que realiza uma regressão polinomial para predizer o custo de pagar um item de dívida técnica, Linguagem de Remediação de Custo que permite escrever expressões matemáticas para modelar o custo de cada tipo de dívida técnica e Classe de Modelagem que permite escrever uma classe em PHP para modelar o custo de pagar um item de dívida técnica de um dado tipo.

Para viabilizar o desenvolvimento dos quatro métodos de medida, foi criada a ferramenta *Technical Debt Analyser* (TDA) que, integrada ao Sonar Qube, implementa os métodos. Além disso, com a ferramenta é possível visualizar os itens de dívida técnica utilizando filtros e alterá-los e complementá-los, como por exemplo, com informações de tempo gasto com o pagamento, comentários e criticidade.

Um estudo de caso foi realizado para avaliar os métodos de medida desenvolvidos neste trabalho. O objetivo era verificar as situações que cada um deles apresentaria melhor resultado. O estudo de caso falhou, porém várias lições foram aprendidas, principalmente, em relação a divulgação e condução do estudo de caso. Além disso, foi proposta uma nova abordagem para medir o custo de pagar a dívida técnica. Este método analisa diferentes versões do código-fonte, compara as árvores sintáticas abstratas e identifica os itens de dívida técnica que foram pagos. Com os dados, é gerado um modelo para expressar a dívida técnica em termos de *statements* de código, que podem ser associados a horas de desenvolvimento, oferecendo o custo de pagar a dívida técnica em horas de desenvolvimento.

Palavras-chave: dívida técnica, métodos de medida, qualidade de software, sonar qube.

Abstract

Pina, Diogo de Jesus **Managing Technical Debt: Current state and new new proposals on measuring methods**. 2016. 120 f. Dissertação (Mestrado) - Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2016.

The metaphor of technical debt was coined by Cunningham and describes that technical quality issues of software may bring future problems if not resolved immediately. More than ten years after the metaphor creation, there were several published studies about classification, as of McConnel and Fowler, and also about conception, as of the paper *Mensuring and Monitoring Techical Debt.*

This dissertation focus on measuring methods of technical debt and a literature review has showed a division in the works already done according to the approach used, they are: conceptual, source code metrics , analysis of quality property violations, machine learning, focused on tools and qualitative research.

This dissertation also proposes some measuring methods of technical debt that aim to be accurate, flexible and contextualized to support the decision-making process. The methods are: Padrão Sonar Qube that catches the cost computed by Sonar Qube, Regressão Estística which computes a polynomial regression to predict the cost to pay a technical debt item, Linguagem de Remediação de Custo which allows to write mathematical expressions to model the cost of each type of technical debt and Classe de Modelagem which allows to write a PHP class to model the cost to pay a technical debt item of a given type.

To enable the development of the four measuring methods we created the Technical Debt Analyser (TDA) tool, integrated with Sonar Qube. It implements the proposed methods. Besides that, with the tool it is possible to visualize the technical debt items using filters and change and fill them, for instance, filling with minutes spent to pay a technical debt item, comments and severity.

A case study was conducted to assess the measuring methods developed in this dissertation. The main goal was verify the applicability of the methods. The case study failed, but several lessons were learned, mainly, about advertising and conduction of study case. Besides that, it was proposed a new approach to measure the cost to pay the technical debt. This new method analyses different source code versions, compares the abstract syntax trees and identifies the payed technical debt items. With the data, it is computed a model to express the technical debt cost in terms of code statements, that can be related with development hours, giving the cost to pay the technical debt in development hours.

Keywords: technical debt, measuring method, software quality, sonar qube.

Sumário

Capítulo 1

Introdução

Nos últimos anos, muitas pesquisas foram feitas na área de dívida técnica, algumas tentando definir os conceitos que envolvem a metáfora da dívida técnica, outras tentando classificar as diversas formas de dívida técnica, propondo modelos e arcabouços para gerenciá-la, propondo métodos e ferramentas para identificar, medir e/ou monitorar a dívida técnica.

Os métodos de medida de dívida técnica estão divididos em dois grupos: principal e juros. O principal se refere ao esforço gasto para fazer com que um item de dívida técnica deixe de existir e é o foco desta dissertação. Os juros se referem ao esforço extra no desenvolvimento de software causado pela existência de dívida técnica, o juros é um fator de incerteza, pois ele só é cobrado quando algum esforço extra é realizado, portanto, é difícil conseguir estimá-lo previamente.

Em relação à medida principal da dívida técnica várias formas de cálculo foram propostos, porém nenhuma delas tenta ser flexível, precisa e adaptativa ao contexto ao mesmo tempo. Os métodos propostos nesta dissertação usados de forma conjunta procuram resolver este problema, permitindo a criação de uma ferramenta que pode expressar a dívida técnica para equipes de desenvolvimento com diferentes capacidades técnicas, linguagem de programação e metodologia de desenvolvimento.

1.1 Panorama Atual

A dívida técnica foi criada por Cunningham (Cunnigham, 1992) como a metáfora de assumir uma dívida quando uma entrega de software é realizada. Os primeiros estudos na área tinham como objetivo conceituar e classificar a dívida técnica, por exemplo, na parte de conceituação tem-se o artigo *Mensuring and Monitoring Techical Debt* (Seaman e Guo, 2011) e em classificação tem-se o estudo de McConnel (McConnell, 2007) e Fowler (Fowler, 2009). Em seguida, métodos de identificação, medida, monitoramento e gerenciamento da dívida técnica também foram estudados.

Dois métodos principais podem ser utilizados para realizar a identificação da dívida técnica: os indicadores diários de dívida técnica e o auxílio de ferramentas. Atualmente, existe um número muito maior de ferramentas não específicas para lidar com os vários tipos de dívida técnica. Em geral, elas são utilizadas para identificar defeitos como: *code smells*, padrões de *design* e *grime*, violações de modularidade e testes (Zazworka *et al.*, 2013). Cada item de dívida técnica identificado deve ser armazenado para que possa ser medido e monitorado (Seaman e Guo, 2011).

Uma das formas para medir a dívida técnica de forma precisa é calcular, para cada um dos itens identificados, o custo de levá-lo do estado de qualidade atual para um estado desejado (Letouzey, 2012b). A soma destes custos é a dívida técnica total associada ao software.

Já o monitoramento pode ser feito de forma imediata, analisando o estado atual do projeto ou de alguma de suas versões, sendo usado para a tomada imediata de decisão. Outra forma, é monitorar a longo prazo, reunindo listas de dívidas técnicas de diferentes datas, a fim de acompanhar a evolução e detectar pontos críticos, onde a prioridade de pagamento deve ser alta, evitando assim a falência técnica (Seaman e Guo, 2011).

Em relação a medida da dívida técnica, vários trabalhos já foram realizados, alguns mais conceituais com uma abordagem mais abstrata ((Letouzey, 2012b), (Seaman e Guo, 2011)), outros que

utilizam métricas de código para obter a medida ((S. Chin e Gat, 2010), (Nugroho *et al.*, 2011) e (Marinescu, 2012)), análise de violações de propriedades de qualidade ((Curtis *et al.*, 2012a)), outros com uma abordagem de aprendizagem de máquina ((Akbarinasaji *et al.*, 2016)), outros com ênfase em ferramentas ((Tomas *et al.*, 2013), (Falessi e Reichel, 2015), (Ramasubbu e Kemerer, 2013)) e até pesquisa qualitativa ((Ernst *et al.*, 2015)).

1.2 Motivação

O controle da dívida técnica é, muitas vezes, importante para garantir a sobrevivência de um projeto de software. Isso desperta o interesse tanto da comunidade acadêmica, quanto de empresas e grupos de desenvolvimento de software livre.

Gerentes de projeto de software precisam equilibrar os benefícios da dívida técnica com os custos associados à tomada de decisão de adquirir ou pagar a dívida técnica e quando. A incerteza associada à dívida técnica complica esta tomada de decisão. Portanto, possuir métodos de medida que retratem, com o máximo de precisão, o custo de pagar os itens de dívida técnica ajudam as equipes de desenvolvimento a terem informações para a tomada de decisão.

Este trabalho foca em analisar e desenvolver métodos de medida de dívida técnica que possam expressar a realidade da dívida técnica dentro de cada contexto de desenvolvimento de projeto de software.

1.3 Proposta de Métodos de Medida e Ferramenta

A fim de tentar obter valores mais precisos, flexíveis e contextualizáveis para o custo de pagar um item de dívida técnica, serão apresentados quatro métodos de medida: Padrão Sonar Qube, Regressão Estatística, Linguagem de Remediação de Custo e Classe de Modelagem.

- **Padrão Sonar Qube:** é a medida da dívida técnica realizada pela plataforma Sonar Qube para, em caso de impossibilidade de utilizar algum dos outros três métodos, atribuir um custo para dívida técnica.

- **Regressão Estatística:** para cada item de dívida técnica é realizada uma regressão polinomial multivariada. Nesta regressão as variáveis preditores representam o contexto em que o item de dívida técnica está inserido, como por exemplo, a quantidade de linhas de código, complexidade e densidade de comentários do arquivo. Já a variável dependente expressa o custo de pagar a dívida técnica em minutos.

- **Linguagem de Remediação de Custo:** é uma linguagem que permite, de maneira simples, modelar o custo de pagar um item de dívida técnica de um determinado tipo. A linguagem é similar a expressões matemáticas, disponibilizando as quatro operações básicas, precedências com parenteses e funções matemáticas tais como: seno, cosseno e raiz quadrada. Além disso, é possível definir variáveis para armazenar o valor das expressões e posteriormente utilizá-la dentro de outras expressões. Algumas variáveis são reservadas e possuem em seus valores dados de contexto do projeto, tais como: informações de complexidade, quantidade de linhas de código do arquivo e linhas de código duplicadas.

- **Classe de Modelagem:** para cada tipo de dívida técnica a ferramenta verifica se existe uma classe de modelagem, escrita em PHP, associada para calcular o custo de um item de dívida técnica. Caso a classe exista, a ferramenta a preenche com dados sobre o tipo de dívida técnica, o item de dívida técnica e dados de contexto do projeto, tais como: informações de complexidade, quantidade de linhas de código do arquivo e linhas de código duplicadas. Com isso, a classe de modelagem é capaz de calcular o custo de pagar uma dívida técnica de forma mais flexível e precisa.

A ferramenta *Technical Debt Analyser* (TDA) foi desenvolvida para implementar os quatro novos métodos de medida e, em seguida, permitir validá-los. A ferramenta TDA é integrada ao Sonar Qube, onde ela consulta o banco de dados para obter informações sobre os arquivos dos projetos, características SQALE, regras de conformidade, *issues* e métricas de código. A TDA ainda permite a visualização e alteração dos itens de dívida técnica com uma série de filtros, facilitando a escolha dos itens de dívida técnica para pagamento.

Para validar os métodos propostos será apresentado um estudo de caso que tenta responder em quais situações cada um dos métodos devem ser utilizados. Além disso, será apresentada uma nova abordagem baseada em análise de versões de código, árvores sintáticas abstratas e pagamento de itens para medir dívida técnica com uma análise preliminar do Apache Maven utilizando essa abordagem.

1.4 Objetivos

O principal objetivo deste trabalho é aprimorar o suporte ao gerenciamento da dívida técnica através da análise e desenvolvimento de métodos de medida. Para permitir o desenvolvimento dos métodos, foi criada a ferramenta *Technical Debt Analyser* (TDA), esta ferramenta é integrada a plataforma Sonar Qube complementando as funções de medida e avaliação da dívida técnica.

Para alcançar o objetivo principal, este trabalho tenta responder as seguintes questões de pesquisa:

1. Como calcular, de forma efetiva, o custo de pagar a dívida técnica em diferentes contextos?
2. Como utilizar o histórico de pagamentos de itens de dívida técnica para refinar o custo?
3. Como viabilizar o gerenciamento da dívida técnica, baseado no custo e histórico de pagamento?
4. Para os métodos que exigem configuração prévia:
   (a) Em média, quanto tempo demora para configurar a medida para cada tipo de dívida técnica?
   (b) Como é o grau de dificuldade de configurar estes itens?
   (c) A equipe de desenvolvimento consegue configurar estes métodos?

1.5 Contribuições

As contribuições deste trabalho para a área de qualidade de software e de dívida técnica são:

1. Análise descritiva de métodos de medida de dívida técnica.
2. Desenvolvimento de métodos para calcular o custo de pagar um item de dívida técnica, que utilizam contexto e tentam ser mais adaptativos dos que utilizados os métodos já existentes.
3. Ferramenta livre específica para tratar a dívida técnica, que junto com o Sonar Qube permite realizar a identificação, a medida e o monitoramento.

1.6 Organização do Trabalho

Este trabalho está dividido nos seguintes capítulos:

- O capítulo 2 discute os principais conceitos relacionados a dívida técnica, realizando uma classificação da dívida técnica, fornecendo uma visão geral sobre os métodos de identificação, medida e monitoramento.

- O capítulo 3 apresenta os principais métodos para identificar, medir e monitorar a dívida técnica.
- O capítulo 4 apresenta e discute ferramentas que lidam com dívida técnica, tais como, Sonar Qube, Cast AIP, Kiuwan e SQuORE.
- O capítulo 5 faz uma análise descritiva dos métodos de medida de dívida técnica encontrados na literatura.
- O capítulo 6 descreve o desenvolvimento de quatro métodos de medida da dívida técnica e da ferramenta Technical Debt Analyser (TDA).
- O capítulo 7 relata a condução de um estudo de caso para validar os métodos de medida de dívida técnica desenvolvidos neste trabalho e, também, é proposto um novo método de medida de dívida técnica baseado em árvores sintáticas abstratas.
- O capítulo 8 apresenta as conclusões, considerações finais e sugestões para trabalhos futuros.

Capítulo 2

Conceitos Básicos

Neste capítulo serão apresentados os conceitos fundamentais envolvendo dívida técnica. Primeiramente, a dívida técnica será definida e explicada. Em seguida, serão apresentadas formas de classificar a dívida técnica utilizando diversos critérios, tais como intenção, prudência, prazo de pagamento e internalidade, externalidade.

Além disso, será apresentada uma ontologia para tratar dívida técnica. Também serão apresentados alguns cuidados que devem ser tomados em relação à dívida técnica. Por fim, será apresentada a metologia SQALE, esta metodologia será utilizada no restante do trabalho como modelo de referência para identificar, armazenar, medir e monitorar os itens de dívida técnica de um software.

2.1 Dívida Técnica

Em 1992, utilizando o vocabulário de economia, Ward Cunningham introduziu a metáfora da dívida técnica de forma superficial e sutil. Em seu relato, Cunningham afirma que quando a qualidade do código é comprometida, é como se estivesse assumindo uma dívida, e que esta dívida é paga quando o código é reescrito. Além disso, ele faz um alerta dizendo que a cada minuto em que a dívida não é paga, juros vão sendo acrescidos.

Segue, abaixo, uma tradução de um trecho de seu relato:

"Entregar código imaturo é assumir uma dívida. Uma pequena dívida acelera o desenvolvimento contanto que seja paga rapidamente com uma refatoração. A orientação a objetos torna este custo tolerável. O perigo ocorre quando a dívida não é paga. Cada minuto gasto com código-fonte ruim conta como juros da dívida." (Cunnigham, 1992)

Uma definição mais completa para a metáfora da dívida técnica é a criada por Carolyn Seaman e Yuepo Guo em Mensuring and Monitoring Techical Debt (Seaman e Guo, 2011). Neste texto, a dívida técnica é definida como uma metáfora para artefatos imaturos, incompletos ou inadequados no ciclo de vida do desenvolvimento de software.

Adquirir uma dívida técnica pode trazer benefícios a curto prazo (Guo *et al.*, 2016), tais como, redução do tempo e esforço no desenvolvimento das tarefas. Porém, a longo prazo, pode causar impactos negativos na qualidade do software gerando, assim, maiores custos no desenvolvimento de novas funcionalidades e na manutenção do código existente.

As equipes de desenvolvimento de software precisam equilibrar os benefícios de adquirir uma dívida técnica com o custo e juros de pagá-la (Seaman e Guo, 2011). As incertezas associadas a dívida técnica faz com que esta tomada de decisão seja ainda mais complexa. Portanto, o gerenciamento da dívida técnica (identificação, medida e monitoramento) pode ajudar as equipes com informações relevantes para a tomada de decisão. Isso resulta, em maior visibilidade ao projeto, o que pode melhorar a qualidade do software e a produtividade da equipe para implementar novas funcionalidades e realizar manutenção de forma efetiva.

Como a dívida técnica é uma metáfora da dívida econômica, inclusive no vocabulário utilizado para expressá-la, permite que a diferentes partes envolvidas no projeto de software (Ernst *et al.*, 2015), sejam elas da área técnica ou não, possam entender e acompanhar as questões relacionadas a

dívida técnica. Além disso, o fato de dívida ser um termo que chama atenção de todos é fundamental para mostrar aos envolvidos no projeto a importância de mantê-la sempre sob controle.

A dívida técnica só acontece em softwares que estão sendo implementados ou já estão em produção (Guo *et al.*, 2016), pois de outra forma não seria possível adquirir uma dívida técnica de algo que não existe e, muito menos, seria possível pagá-la. Já projetos de softwares abandonados não acumulam mais dívidas técnicas. Em alguns casos, as dívidas técnicas adquiridas ao decorrer destes projetos são simplesmente anuladas. Porém, em outros casos, a dívida técnica poderá estar atrelada não apenas a um projeto, mas pode se espalhar para vários projetos relacionados ou afetar diretamente o capital das instituições responsáveis pelo seu desenvolvimento. Sendo assim, mesmo com o encerramento do projeto as instituições poderão ter que pagar, pelo menos, parte da dívida e dos juros relacionados a ela.

2.2 Classificação da Dívida Técnica

A partir do momento em que se tem conhecimento da dívida técnica adquirida, McConnell (McConnell, 2007) a classifica em outros dois tipos, dependendo do tempo em que se planeja pagá-las, estes tipos são: dívida técnica de curto prazo e dívida técnica de longo prazo.

Segundo Martin Fowler (Fowler, 2009) a dívida técnica pode ser classificada por um quadrante dividindo as em: não intencional e imprudente, não intencional e prudente, intencional e imprudente. e intencional e prudente.

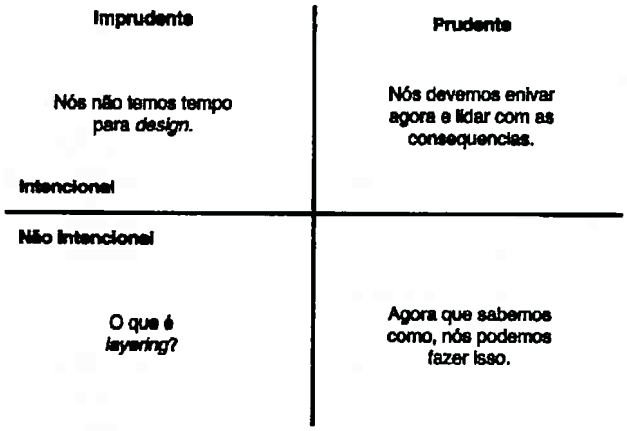

Figura 2.1: *Quadrante de classificação da dívida técnica. (Fowler, 2009)*

A dívida técnica também pode ser divida como externa ou interna (Boegh *et al.*, 1999), dependo da característica ou sub-característica a qual o item está relacionado.

2.2.1 Dívida Técnica não Intencional e Imprudente

Ocorre quando não se conhece a dívida técnica adquirida e ela veio de ações inadvertidas realizadas pela equipe de desenvolvimento, sem fazer parte de uma estratégia. Este é o tipo mais perigoso de dívida técnica, pois a equipe não sabe que a inseriu no código, nem como encontrá-la e pagá-la, ou seja, pode levar um projeto à falência, principalmente, técnica sem que a equipe tome conhecimento das causas.

2.2.2 Dívida Técnica não Intencional e Prudente

Ocorre quando não se conhece a dívida técnica adquirida, mas a equipe de desenvolvimento reconhece que as suas ações vão acumular dívida, mas por algum motivo estratégico é decidido manter essas ações. Neste caso, posteriormente a equipe gastará algum tempo para conseguir aprender a maneira correta de fazer as tarefas e, também, identificar a dívida técnica que adquiriu, sendo este tempo contado como juros.

2.2.3 Dívida Técnica Intencional e Imprudente

Ocorre quando a dívida técnica à ser adquirida já é conhecida e, mesmo sabendo da sua importância, ela é ignorada sem nenhum motivo estratégico. Neste caso, por imprudência a dívida técnica deixa de ser paga e não é feito nenhum planejamento para pagá-la futuramente.

2.2.4 Dívida Técnica Intencional e Prudente

Ocorre quando a dívida técnica a ser adquirida já é conhecida e, por algum motivo importante, geralmente externo a equipe de desenvolvimento, é decidido adquirí-la de maneira consciente. Logo, este tipo de dívida técnica é resultado de uma estratégia, previamente elaborada. Em muitos casos, a necessidade de se assumir a dívida está relacionada, principalmente, com o tempo, ou seja, a necessidade de entregar um conjunto de tarefas num prazo menor do que o realmente necessário para fazê-lo com a qualidade desejada.

2.2.5 Dívida Técnica a Curto Prazo

Bem como uma dívida econômica, a dívida técnica pode ser paga num curto prazo (Brown *et al.*, 2010) e, geralmente, o pagamento deste tipo de dívida técnica é feito para suprir as necessidades imediatas ou quando se tem dinheiro (mão-de-obra) extra.

Outra característica desse tipo de dívida é que ela é paga com certa frequência a fim de reabilitar o crédito para aquisição de novas dívidas técnicas. As dívidas a curto prazo, em geral, não necessitam de um grande planejamento, sendo adquiridas para suprir as necessidades do momento e que não produzem um grande impacto negativo e nem um grande acúmulo de juros.

2.2.6 Dívida Técnica a Longo Prazo

As dívidas técnicas a longo prazo (Brown *et al.*, 2010) são adquiridas, em geral, de forma estratégica, com um planejamento a fim de suprir necessidades de grande impacto. Como por exemplo, o lançamento de um certo produto a fim de ganhar espaço no mercado. Neste caso pode se adquirir dívida de implementação de novas funcionalidades, cobertura de testes, falhas no sistema, falta de um suporte técnico adequado ou qualquer outra necessidade técnica que exijam um grande esforço da equipe. Porém, no futuro, essas dívida técnicas precisarão serem pagas. Caso o gerente e/ou a equipe tenham necessidade, mesmo a dívida técnica sendo a longo prazo, ela pode ser paga rapidamente, ou pode ser gerenciada durante anos.

2.2.7 Dívida Técnica Interna

A dívida técnica interna são os itens que estão relacionados a características ou sub-características de qualidade externa (Boegh *et al.*, 1999), ou seja, a dívida técnica que impacta em como o software funciona em seu ambiente. Exemplos desse tipo de dívida técnica são as relacionadas com usabilidade e confiabilidade.

2.2.8 Dívida Técnica Externa

A dívida técnica externa são os itens que estão relacionados a características ou sub-características de qualidade interna (Boegh *et al.*, 1999), ou seja, a dívida técnica que impacta em como o produto é desenvolvido. Exemplos desse tipo de dívida técnica são as relacionadas com complexidade estrutural, tamanho, cobertura de testes e taxa de falhas.

2.3 Ontologia da Dívida Técnica

A formalização da ontologia da dívida técnica foi desenvolvida por um conjunto de pesquisadores e exposto no artigo *Towards an ontology of terms on technical debt* (Alves *et al.*, 2014). A

identificação dos tipos de dívida técnica foi feita através de uma revisão sistemática da literatura. A partir daí, foi construída uma ontologia que foi avaliada por critérios de qualidade e por especialistas da área.

A ontologia possui as seguintes classes: dívida de arquitetura, dívida de construção, dívida de código, dívida de defeito, dívida de design, dívida de documentação, dívida de processo, dívida de infraestrutura, dívida de automação de teste, dívida de pessoal, dívida de requisito, dívida de serviço, dívida de negócio e dívida de teste. O relacionamento entre essas classes pode ser visto na figura 2.2.

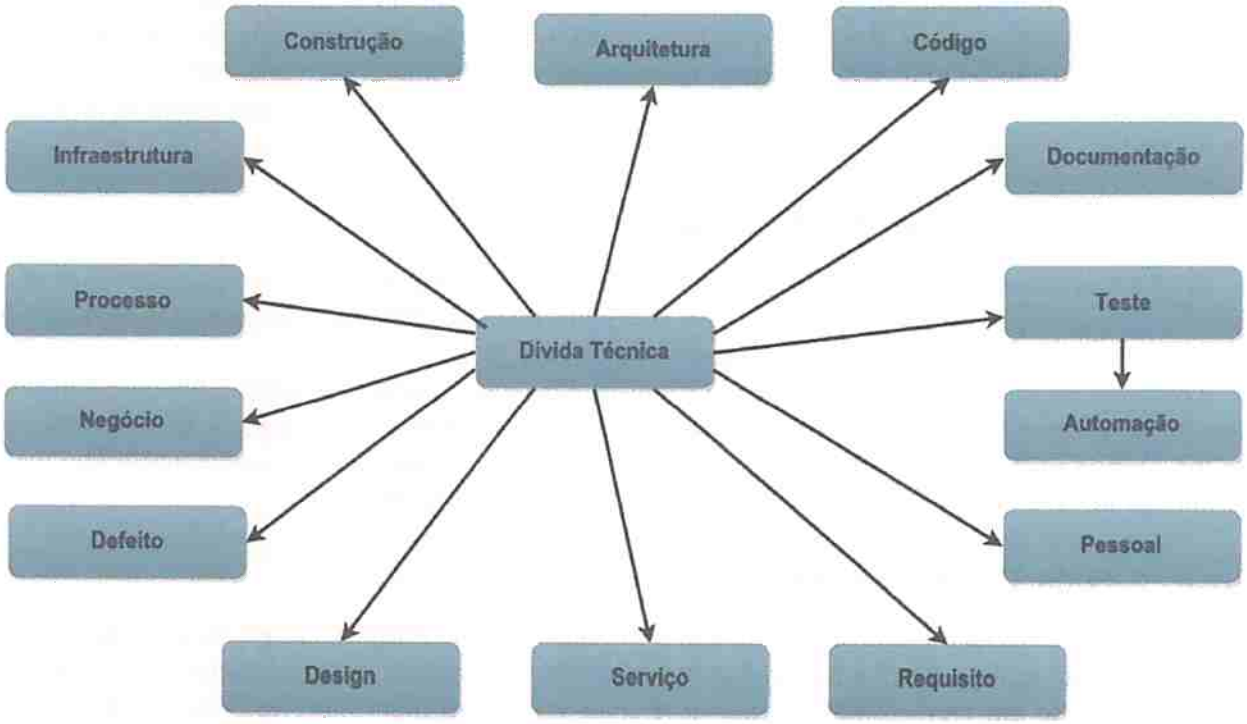

Figura 2.2: *Ontologia da Dívida Técnica.*

Neste trabalho, a classe de dívida técnica mais utilizada é a dívida de código, porém outras classes de dívida técnica também são abordadas como a dívida de arquitetura, dívida de teste e dívida de design.

2.4 Benefícios e Problemas da Dívida Técnica

A metáfora da dívida técnica consiste do problema enfrentado por equipes de desenvolvimento de software de como balançar os valor a curto prazo com a qualidade a longo prazo (Ernst *et al.* , 2015). Uma vez que a dívida técnica se torna visível a equipe pode começar a entender de que maneira ela pode ser benéfica ou prejudicial ao projeto.

Por um lado, a dívida técnica pode ser tomada de forma deliberada, precisando ser monitorada, gerenciada e paga, para atingir valores de negócio. Por outro lado, o acumulo de dívida técnica causa custos extra (juros) na qualidade do sistema em manutenção e evolução.

Para completar o monitoramento e o gerenciamento da dívida técnica é importante se ter métodos que calculem o custo de pagar cada item de dívida técnica de forma precisa e adaptativa a cada contexto de desenvolvimento de software. Caso os valores sejam subestimados a equipe pode achar que leva menos tempo para pagar a dívida técnica do que de fato ela custa e acaber tendo que gastar mais recursos para pagá-la. Caso os valores sejam superestimados a equipe pode deixar de atingir maior valor de negócio achando que o custo de pagar a dívida técnica seria maior do que de fato ele é.

Além disso, revisões sistemáticas da literatura ((Ampatzoglou *et al.*, 2015) e (Tom *et al.*, 2013)) relataram que:

- As pesquisas e ferramentas usam, principalmente, técnicas de análise de qualidade de código para entender a dívida técnica.
- Alguns trabalhos exploram a dívida técnica em diferentes etapas do ciclo de vida de desenvolvimento de software, como por exemplo, dívida de requisitos, dívida de testes, dívida de *design*.

A dívida técnica acumulada pode causar custo contínuos (juros) para a qualidade do sistema em manutenção e evolução. A dívida técnica pode ser

2.5 Alertas sobre a Dívida Técnica

A dívida técnica mais perigosa que se tem a priori é a sem intenção. Se a equipe não utilizar métodos e ferramentas para identificá-la e, em seguida, medi-la e monitorá-la, ela pode fazer um projeto ir à falência técnica ou até mesmo financeira.

A falência de um projeto significa que ele deverá ser reescrito ou até mesmo abandonado. Qualquer um dos casos pode gerar um grande volume de trabalho, uma grande perda de tempo e esforços ou até mesmo criar um impacto financeiro muito grande.

Porém, a dívida técnica intencional também pode ser perigosa caso seja medida de maneira inadequada, não tenha um acompanhamento de sua evolução ou não tenha um plano estratégico para realizar seu pagamento. Esses perigos valem tanto para a dívida técnica de curto ou longo prazo, pois por causa dos juros a dívida técnica de curto prazo pode se tornar uma de longo prazo, e a de longo prazo pode ficar inviável de pagar.

Sempre que possível, adquirir uma dívida técnica deve ser muito bem planejado, para evitar que esta não seja paga e acumule juros excessivos. Portanto, a equipe deve levar em conta o contexto do projeto, sua capacidade técnica e financeira de pagá-la, o tamanho da dívida e sua forma de pagamento na hora de realizar uma aquisição.

O contexto em que o software é utilizado está diretamente ligado com a dívida técnica. Podendo ser no aspecto de qual dívida técnica pode ser assumida, seu tempo de pagamento e seu valor.

Por exemplo, ao adquirir uma dívida técnica na cobertura de testes de um aplicativo web com poucos acessos pode implicar em uma dívida técnica baixa, pois mesmo que o aplicativo tenha algum problema, este problema não afetará de forma significativa. Além disso, pagar a dívida técnica pode ser feito de forma rápida e com um baixo custo.

Por outro lado, no caso de um software de controle de aeronave não possuir uma cobertura de testes ideal, poderá acarretar em acidentes e consequentemente em grandes prejuízos financeiros e, principalmente, civis. Neste caso, o pagamento da dívida técnica pode ser demorada e cara, pois trocar o software de cada avião poderá demorar e ainda ter um custo alto da equipe de manutenção. O fato que o avião necessitará ficar fora de operação, também acarretará em prejuízos que devem ser inclusos no cálculo da dívida. Além disso, o prejuízo com a imagem da empresa, também, deve ser acrescentado.

2.6 Metodologia SQALE

Para que seja viável a identificação, medida e gerenciamento da dívida técnica de forma semi-automática, será utilizado o modelo de qualidade da metodologia SQALE (Letouzey, 2012b). Este modelo, junto com sua metodologia, permite realizar a automação dos métodos de identificação e, principalmente, viabilizar o cálculo de custo do pagamento de cada item da dívida técnica, padronizando e dando interpretações em vários níveis para os custos obtidos. Esta flexibilidade faz com o que a dívida técnica possa ser analisada do ponto de vista de cada uma das partes interessadas

no projeto. Estas partes interessadas podem ser: clientes, investidores, gerentes de projeto, analistas, programadores ou qualquer grupo ou pessoa que tenha interesse no software ou em parte dele.

Além disso, a metodologia SQALE foi desenvolvida para avaliar a qualidade de software em geral, por isso, pode ser aplicado a qualquer representação do software. Essas representações podem tanto ser código-fonte, que é o principal objetivo neste trabalho, quanto documentações e e diagramas, tais como o diagrama UML.

2.6.1 Modelo de Qualidade SQALE

A metodologia SQALE propõe um arcabouço para a criação de modelos de qualidade. A arquitetura deste arcabouço é composta por uma hierarquia flexível de três níveis: Características, Subcaracterísticas e Propriedades de Qualidade; como mostra a figura 2.3.

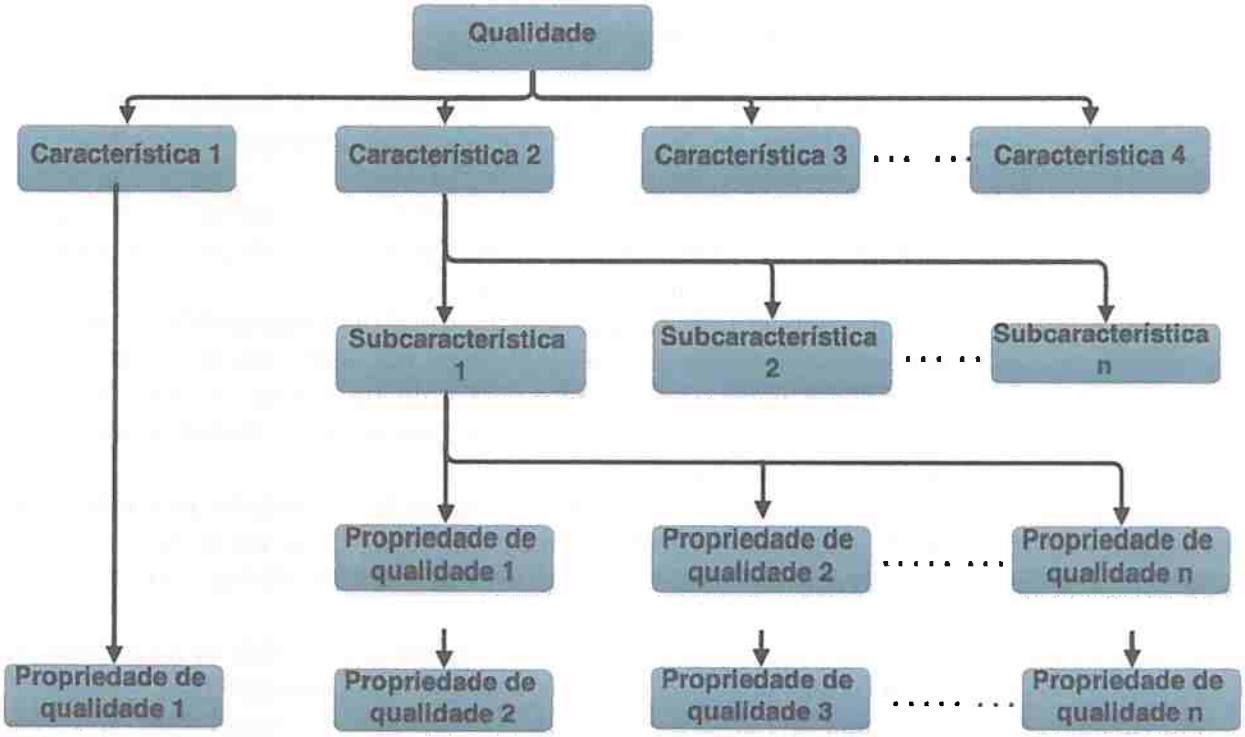

Figura 2.3: *Arquitetura geral do modelo de qualidade SQALE (Letouzey, 2012b).*

Para cada propriedade de qualidade, diversos itens podem ser encontrados. Logo podemos verificar se cada um destes itens está ou não em conformidade com a propriedade. Estes itens podem ser vistos como possíveis itens de dívida técnica. Sendo assim, um item de dívida técnica pode ser definido como a violação de uma propriedade de qualidade.

Caso a propriedade tenha sido violada, é possível calcular o custo, em horas ou dinheiro, que é necessário investir para que este item de dívida técnica respeite a propriedade. Caso a propriedade esteja sendo respeitada, este custo é zero, pois não é preciso fazer nenhum tipo de esforço para pagá-la.

A soma de cada item de dívida técnica relacionado com uma propriedade de qualidade é o custo de tornar tal propriedade válida. Logo, a soma de todos estes custos é o custo de tornar todas propriedades de qualidade válidas e, isso, pode ser visto como a quantidade de dívida técnica do projeto.

2.6.1.1 O Nível de Características

O nível de Características é o primeiro nível da hierarquia e é definido de acordo com as características que se deseja avaliar. Estas características podem ser entendidas como habilidades do sistema, ou seja, os principais pontos pelo qual o software pode ser avaliado.

2.6.1.2 O Nível de Sub-características

Cada Característica pode ser dividida em diversas Sub-características. As Sub-características são especializações de uma Característica ou até mesmo de outra Sub-característica. Assim como as Características, elas são usadas para agrupar as Propriedades de Qualidade de maneira que elas possam ser analisadas e ganhar um significado mais forte.

2.6.1.3 O Nível de Propriedade de Qualidade

Este nível contém todas as propriedades relacionadas com a qualidade do software, ou seja, cada Propriedade de Qualidade é uma regra a qual o software deve seguir. Para que estas propriedades sejam realmente significativas elas devem ser formuladas com um grande rigor de qualidade e o máximo formalismo para evitar dualidades.

Mesmo em modelos pré-definidos, poderão ser inseridas e/ou retiradas Propriedades de Qualidade a fim de que o modelo possa expressar o contexto do software.

2.6.2 Diagrama do Modelo SQALE

A figura 2.4 mostra um exemplo do diagrama do modelo de qualidade SQALE. Neste diagrama estão presente todas as características sugeridas pela metodologia que são: reuso, portabilidade, manutenção, segurança, eficiência, usabilidade, confiabilidade e testabilidade. Abaixo de cada uma das características são colocados alguns exemplos de sub-características correspondentes.

Figura 2.4: *Exemplo do diagrama do modelo SQALE (Letouzey, 2012b).*

2.7 Conclusão do Capítulo

Neste capítulo, foram apresentados os principais conceitos, classificações, ontologia e alertas da dívida técnica. E, por fim, foi mostrada a metodologia SQALE.

Primeiramente, este capítulo traz alguns conceitos sobre a dívida técnica, desde sua primeira definição feita, por Ward Cunningham, até conceitos mais detalhados. Junto com o conceito, também são apresentadas as classificações da dívida técnica que são dívidas em três grupos: o primeiro pela intenção e pela prudência, o segundo pela internalidade e externalidade e o terceito pelo tempo de pagamento. Em seguida, os benefícios e problemas em relação a dívida técnica acompanhados de alguns alertas em relação a dívida técnica foram feitos.

Por fim, foi descrita a metologia SQALE, em especial, seu modelo de qualidade com níveis de características e sub-características e propriedade de qualidade, utilizado para permitir o suporte ao gerenciamento de dívida técnica.

No próximo capítulo serão apresentados métodos de identificação, medida e monitoramento da dívida técnica.

Capítulo 3

Métodos de Identificação, Medida e Monitoramento da Dívida Técnica

Neste capítulo serão apresentados os métodos para viabilizar o gerenciamento da dívida técnica, iniciando pelos diversos métodos de identificação, passando pelos métodos de medidas, em especial os da metodologia SQALE, e, por fim, métodos para monitorar a dívida técnica.

3.1 Métodos de Identificação

Com o objetivo de prevenir a dívida técnica e seu acúmulo, ou decidir quando, onde e quanto pagar da dívida, o primeiro passo é identificá-la.

A dívida técnica externa está relacionada diretamente com as necessidades e vontades das partes interessadas, como por exemplo as de usabilidade e confiabilidade (Boegh *et al.*, 1999), logo, estabelecer um método para sua identificação pode ser difícil e automatizá-lo, geralmente, é inviável.

Já a dívida técnica interna se relaciona diretamente com as descrições do software, em especial ao código-fonte, como por exemplo complexidade estrutural e tamanho (Boegh *et al.*, 1999), sendo assim, fica mais fácil detectá-la, em especial, de forma automática. Além disso, a dívida técnica interna tem uma forte relação com a externa. Sendo assim, quando se paga uma dívida técnica interna, indiretamente, pode se estar pagando uma dívida técnica externa, mesmo que esta não tenha sido explicitamente identificada.

Para identificar a dívida técnica baseada em código-fonte (parte da dívida interna) serão utilizados os métodos apresentados em *Identifying and Managing Technical Debt* (Zazworka *et al.*, 2013) e *Comparing Four Approaches for Technical Debt Identification* (Zazworka *et al.*, 2014). Uma destas técnicas são os indicadores diários de dívida técnica e a outra é o uso de ferramentas.

O objetivo destas ferramentas é encontrar defeitos e ausência de padrões no código-fonte de forma automática. Esses elementos podem gerar dívida técnica ou apenas serem falsos positivos.

As ferramentas de identificação utilizam os seguintes métodos para encontrar os defeitos: análise automática (de código) estática, *code smells*, padrões de *design*, violações de modularidade e cobertura de testes.

Utilizando o modelo de qualidade da metodologia SQALE é possível definir que encontrar uma dívida técnica é encontrar uma violação de uma propriedade de qualidade. Como vamos lidar, em particular, com o modelo de qualidade de produto, definimos que uma dívida técnica é uma violação de um requisito funcional, ou seja, é qualquer código que não esteja em conformidade com o que foi previamente definido.

3.1.1 Indicadores Diários de Dívida Técnica

Os indicadores diários de dívida técnica são situações que ocorrem no cotidiano da equipe de desenvolvimento que indicam que o software possui dívida técnica. Estas situações podem determinar dívida técnica pontual e que, em geral, podem ser encontradas facilmente. Mas, também,

podem determinar dívida técnica genérica, onde pode ser preciso algumas horas ou até mesmo dias para que se possa identificá-la.

Exemplos de identificadores diários retirados de (Zazworka e Seaman, 2013):

- Não se preocupe com a documentação agora.
- O único que pode mudar este código é o João.
- Está bom para o momento, mas precisaremos refatorar mais tarde.
- *ToDo* e *FixMe* no código-fonte.
- Vamos apenas copiar e colar esta parte.
- Quem sabe onde armazenamos a senha de acesso ao banco de dados?
- Eu sei, se eu tocar neste código ele quebrará.
- Vamos terminar os testes na próxima versão.

3.1.2 Análise Estática e Automática de Código

Este tipo de método de análise verifica problemas em nível de linha de código (Zazworka e Seaman , 2013). O código analisado pode ser tanto o código-fonte quanto o código compilado (Zazworka *et al.* , 2014). Esta análise é feita com base em violações de boas práticas de programação ou padrões de projeto para a definição de nomenclatura.

Com este método é possível encontrar partes do código que podem causar falhas ou podem degradar alguns aspectos da qualidade do software, especialmente os relacionados a manutenção do código. Seguem abaixo, alguns exemplos de sub-características de dívida técnica onde as violações encontradas por este método podem ser associadas:

- Práticas ruins de programação;
- Corretude;
- Experimentação;
- Internacionalização;
- Vulnerabilidade a códigos maliciosos;
- Desempenho;
- Segurança;
- Corretude de multi-*thread* e
- Estilo.

Mesmo que as ferramentas encontrem diversos tipos destas violações é importante analisar o contexto do software para que se possa escolher aquelas que realmente importam, ou seja, excluir os falsos positivos. Essa separação pode ser feita com base no modelo de qualidade que foi definido para encontrar a dívida técnica.

É interessante observar que a dívida técnica encontrada com este método pode estar relacionada a defeitos no código, mas também pode estar relacionada a violações de bons padrões, fato que afeta a dívida técnica de manutenção. Com isso, a dívida técnica que é encontrada com este método pode ser de grande interesse de pagamento, pois caso o software necessite receber manutenção de código ou que novas funções sejam implementadas, existe uma grande chance de que juros sejam aplicados, fazendo com que a manutenção ou a implementação seja mais demorada do que seria se não existisse a dívida técnica.

3.1.3 *Code Smells*

O conceito de *code smells* foi definido por Matin Flower (Fowler, 1999), com a ajuda de Kent Beck, como um sinal de que a área afetada, normalmente corresponde a um problema mais profundo. Geralmente, *code smells* descreve problemas relacionados com orientação a objetos e outros problemas comuns que envolvem conjuntos de linhas de código.

Como *code smells* envolve uma variedade de problemas é necessário que várias técnicas sejam utilizadas para conseguir um método que possa identificá-los, podendo ser forma manual ou de forma automática.

Seguem alguns problemas que são definidos:

- Código duplicado.
- Métodos/funções longos.
- Classes/módulos longos.
- Métodos/funções com uma grande lista de parâmetros.
- Aglomeração de dados.
- Tipo incorporado nos nomes.

Em geral, os problemas encontrados exigem uma refatoração para que possam ser corrigidos. Logo, a dívida técnica encontrada com este tipo de problema no código está fortemente ligada a manutenção e reuso. Além disso, muitos critérios utilizados, também, dizem respeito a confiabilidade, a segurança e a eficiência.

Com este método, não é possível encontrar toda a dívida técnica de forma automática, mas é possível encontrar de forma semi-automática. Isso ocorre, pois alguns itens de dívida técnica são intuitivos podendo ser falsos positivos e outros mesmo sendo dívida técnica para alguns contextos, para outros podem ser essenciais e não serem considerados como dívida técnica.

3.1.4 Padrões de Design

Os padrões de *design* têm como objetivo fazer com que o código-fonte seja mais fácil de ser lido e compreendido e, portanto, mais fácil de realizar manutenções e menos propenso a defeitos e falhas (Zazworka e Seaman, 2013). Ele faz isso por meio da padronização e da descrição de como as classes podem trabalhar juntas.

O acumulo dos códigos que estão fora de conformidade com os padrões de *design* formulados para determinados software são chamados de *grimes* (Zazworka *et al.*, 2014), ou seja, no nível de interação entre as classes. Estas inconformidades podem ser geradas na criação do código ou em uma alteração, neste último caso, está quebra de integridade será chamada de *rot*.

Tanto a violação dos padrões de *design*, quanto *grimes* e *rot* são considerados dívidas técnicas, já que quebram propriedades de qualidade definidas para o projeto.

Os conceitos de padrões de *design* estão associados, em geral, a um nível mais alto do software, envolvendo classes e métodos. Portanto, para identificar problemas nesses padrões, muitas vezes é preciso análises mais complexas. Portanto, é mais difícil encontrar ferramentas que verifiquem problemas de padrões de design automaticamente.

3.1.5 Testes

O método de testes tem como objetivo mostrar que partes do código estão tendo o comportamento esperado em algumas situações, portanto, se o teste falha pode ser um indicativo de defeito no software. Estes defeitos podem ser tanto erros, quanto violações de propriedades externas. Quando se trabalha com softwares críticos é necessário se fazer teste por meio de verificação formal, com a

finalidade de garantir que o código-fonte corresponde ao modelo elaborado. Já para softwares menos complexos, o teste por meio de validação, em geral, já é o suficiente (Delamaro *et al.*, 2007).

Os testes podem ser identificadores de dívida técnica, pois eles podem ser escritos para identificar a validade de propriedades de qualidade. Logo, um teste pode ser capaz de identificar se determinado trecho de código viola, ou não, uma dada propriedade de qualidade.

Além disso, os testes são capazes de identificar problemas em abstrações em um nível mais alto. Como por exemplo, se um determinado módulo está respeitando a arquitetura do sistema ou se ele atende as expectativas das partes envolvidas.

A ausência de testes ou o nível de cobertura dos testes podem ser considerados como dívida técnica. Como escrever testes também implica em custo de desenvolvimento e sua ausência ou inadequação pode impactar na qualidade do software a escrita e manutenção de testes pode entrar no cálculo da dívida técnica.

3.1.6 Outros

Conforme o contexto em que o software está inserido, outros métodos de identificação poderão ser necessários para encontrar os itens de dívida técnica. Isso pode acontecer, pois o contexto pode fazer com que um tipo dívida técnica se derive ou até mesmo que novos tipos de dívida técnica apareçam.

3.2 Métodos de Medida

Para que se possa realizar a medida da dívida técnica será utilizado o método descrito em *The SQALE Methods*. Este método consiste em um conjunto de regras que são utilizadas para viabilizar e padronizar as medidas. Além disso, ele padroniza os controles relativos ao código-fonte e às regras de agregações de valores.

Para que se possa fazer esta padronização será associada a cada propriedade de qualidade uma função que calcula o custo de corrigir uma dada violação, esta função será chamada de função de remediação.

A partir das funções de remediação é possível calcular o custo de cada item de dívida técnica e, com estes resultados, é possível calcular a dívida técnica total do software ou, também, agrupá-las seguindo critérios e obter a dívida técnica de parte do software.

3.2.1 Função de Remediação

O objetivo da função de remediação é determinar o custo de sair do estado atual para um estado desejado de qualidade (Letouzey, 2012b). Logo, para cada propriedade de qualidade é associada uma função de remediação, permitindo que o custo de pagamento seja calculado de forma eficaz e padronizado.

A entrada dessa função é uma lista contendo um indicador se a propriedade de qualidade foi violada ou não, e, caso existam, os valores associados a essa violação. Como saída se tem um valor de custo, que pode ser um custo monetário, um custo em horas de trabalho ou um custo simbólico, de acordo com a necessidade de cada projeto.

A função de remediação pode ser dada por uma função simples que atribui 0, caso não ocorra violação, e uma constante, caso ocorra. Outra possibilidade é definir um fator multiplicativo para a violação de acordo com os valores associados a essa violação.

A função de remediação também pode ser definida por funções mais complexas, utilizando outras funções, tais como as exponenciais, logarítmicas, trigonométricas ou quaisquer outras funções que modelem de forma adequada o custo de pagar a dívida técnica. O uso de funções mais complexas podem ajudar na modelagem do custo de pagar a dívida técnica, pois a complexidade, por exemplo, pode fazer com que o tempo para pagamento aumente exponencialmente.

Exemplo: Suponha que a propriedade é: "Todo método deve estar coberto por pelo menos um teste de unidade.". Vamos definir a função de remediação como sendo: $f(x, y) = xln(y)$, onde x é

0, caso o método esteja coberto, e 1, caso não; e y a quantidade de linhas de cada método que não foi coberto. A função $f(x, y)$ devolverá o custo em horas para que o método seja coberto.

É importante observar que toda função de remediação deverá ser 0, caso não exista violação da propriedade de qualidade.

3.2.2 Armazenando um Item de Dívida Técnica

Para ajudar na organização e na automação do cálculo da dívida técnica é necessário utilizar uma estrutura de dados que pode ser similar com a proposta em *Measuring and Monitoring Technical Debt* (Seaman e Guo, 2011). Para cada modelo ou contexto a estrutura pode ser modificada para ficar adequada.

Tabela 3.1: *Exemplo de um registro de dívida técnica.*

ID	42
Data - Hora	01/06/13 11:30:34
Propriedade de Qualidade	Todo método deve estar coberto por pelo menos um teste de unidade.
Local	Classe ABC : Método XYZ
Custo de Remediação	2 horas e 30 minutos
Prioridade de Pagamento	Alto

3.2.3 Cálculo do Custo da Dívida Técnica

Conforme o SQALE Method (Letouzey, 2012a), para que se possa calcular a dívida técnica do software com exatidão é preciso que:

- Todos os tipos de qualidade sejam independentes, ou seja, um item de dívida técnica não pode estar associado a dois tipos de qualidade;
- Todas as características do modelo devem ser independentes;
- Todas as sub-características do modelo devem ser independentes;
- Todas as propriedades de qualidade do modelo devem ser independentes;
- A partir das propriedades de qualidade, todos os custos de dívida técnica devem ser encontrados.

As regras acima, na prática, são difíceis de serem cumpridas. Logo, para que o cálculo seja viável, podemos calcular parte da dívida técnica de uma dada propriedade de qualidade, sub-característica, característica ou tipo de qualidade que atenda aos requisitos.

Caso não seja possível garantir independência e/ou que toda dívida técnica seja encontrada teremos:

- Limitante superior: caso se possa encontrar toda dívida técnica, mas não se possa garantir independência. Isso ocorre devido a intersecção entre os itens de dívida técnica, fazendo com que os custos sejam somados mais de uma vez;
- Limitante inferior: caso se possa garantir a independência, porém não se possa garantir que toda dívida foi encontrada. Neste caso, a dívida técnica será contada apenas uma vez, não podendo ser menor do que o valor calculado. Por outro lado, como alguns itens podem não terem sido encontrados, a dívida técnica pode ser maior do que o valor calculado.

- Aproximação: caso não se possa garantir a independência, nem que toda dívida técnica seja encontrada. Este é o caso menos ideal, porém o mais comum. Neste caso, não pode ser um limitante superior, pois os itens de dívida técnica que não foram encontrados poderiam ultrapassar este limite. Por outro lado, não pode ser um limitante inferior, pois podem existir intersecções entre os itens de dívida técnica, logo o valor encontrado é uma aproximação para o valor real de dívida técnica.

O cálculo da dívida técnica pode ser feita em vários níveis, desde o mais baixo com as propriedades de qualidade até o mais alto, calculando a dívida do software como um todo. Ou seja, podem ser agrupados, em diversos níveis, criando assim índices de dívida técnica.

Os índices de dívida técnica padronizam a soma de diversas medidas em um conjunto que faça sentido e seja importante para determinadas partes de interesse. Por exemplo, pode se agrupar a dívida para cada módulo ou arquivo, ou também por características ou sub-características.

3.2.4 Passo-a-passo para Calcular a Dívida Técnica

A dívida técnica (DT) (Letouzey, 2012a) pode ser calculada pela seguinte fórmula:

$$DT = \sum_{i=1}^{n} f_i$$

sendo f_i o custo de fazer a propriedade i ser válida em todos os itens; calculada da seguinte forma:

$$f_i(P_1, P_2, ..., P_{m_i}) = \sum_{j=1}^{m_i} h_i(P_j)$$

em que P_j é um vetor que descreve cada item que está sendo avaliado e

$$h_i = \begin{cases} 0, & \text{se i não foi violada,} \\ x > 0, & \text{caso contrário.} \end{cases}$$

representa o custo de fazer o item de avaliação (P_j) satisfazer a propriedade i.

3.2.5 Agrupamento da Dívida Técnica

Muitas vezes é interessante saber qual é a dívida técnica para cada características do sistema. Para isso, basta agrupar a dívida técnica em sub-características que são diretamente associadas a uma característica.

Em **The SQALE Methods** (Letouzey, 2012a) é criado um índice para cada característica que representa a soma da dívida técnica nas propriedades relacionadas a ela ou as suas sub-características.

Estes índices foram criados para as seguintes características:

- Testabilidade (STI);
- Confiabilidade (SRI);
- Alterabilidade (SCI);
- Eficiência (SEI);
- Segurança (SSI);
- Manutenção (SMI);
- Portabilidade (SPI) e
- Reuso (SRuI).

Como em um projeto podemos ter diversas partes interessadas, muitas vezes, se faz necessário o agrupamento dos índices para que o valor possa refletir algo de interesse específico para alguma destas partes.

Por exemplo, alguém poderia estar interessado em saber se o sistema está funcionando corretamente. Logo, ele poderia querer ver o agrupamento da dívida técnica de testabilidade e confiabilidade juntos, gerando assim, um novo índice:

$$SCRI = STI + SRI.$$

3.2.6 Dimensionamento da Dívida Técnica

Para dimensionar a dívida em relação ao código-fonte, é possível criar índices de densidade. Estes índices são definidos como a divisão dos índices absolutos pela medida total possível para cada uma das propriedades associadas ao índice.

Por exemplo, dado o índice absoluto de testabilidade $STI = 50$, ou seja, para pagar toda a dívida técnica em relação a testabilidade custaria 50 minutos. Então, é preciso encontrar o total de horas que seria gasto se nenhum teste tivesse sido realizado, suponha que este valor seja 500 minutos. Então temos índice de densidade de testabilidade $STID = \frac{50}{500} = 0,1$, ou seja, a dívida técnica de testabilidade está presente em 10% do código.

Esta técnica pode ser aplicada a qualquer índice absoluto, inclusive os que agrupam mais de um índice. Sendo assim é possível criar um índice de densidade para a medida da dívida técnica.

É importante observar que na prática, códigos pequenos podem ter índices de densidade altos e códigos grandes, mesmo possuindo um grande número de itens de dívida técnica, pode ter índices de densidade baixos.

3.2.7 Representações Gráficas da Dívida Técnica

A medida da dívida técnica pode ser representada em diversas escalas ou utilizando artifícios como gráficos e pirâmides.

3.2.7.1 Escala SQALE

Em SQALE (Letouzey, 2012a) são usadas três unidades de escala: classificação, porcentagem e cor. Em geral, a escala é dividida em cinco ou mais valores, como mostra a figura 3.1. Esta escala é aplicada diretamente nos índices de densidade e permitem verificar facilmente, o grau de dívida técnica nos pontos de interesse.

Nota	Até	Cor
A	1%	
B	2%	
C	4%	
D	8%	
E	∞	

Figura 3.1: *Tabela de notas e cores da escala SQALE.*

Exemplo de uso: suponha que o índice de testabilidade (STI) seja igual a 18 e, que todos os pontos possíveis somam 600. Logo o índice de densidade STID é igual a 0.03, que em porcentagem é igual a 3%. Logo sua classificação é C e sua cor é amarela.

3.2.7.2 Gráfico Kiviat

O gráfico de Kiviat é uma maneira rápida e simples, de visualizar a dívida técnica de várias características em apenas um lugar, como mostra a figura 3.2. Esse gráfico é dividido em setores que são dados pela escala SQALE utilizada.

Para montar este gráfico, basta seguir os seguintes passos:

1. Para cada característica, marque o ponto em que se encontra a dívida técnica;
2. Ligue todos os pontos da dívida técnica, obtendo, assim, um polígono da dívida técnica atual;
3. Para cada característica, marque o ponto com o valor máximo que se deseja ter a dívida técnica e
4. Ligue todos os pontos de valores máximos, obtendo um polígono de valor máximo da dívida técnica.

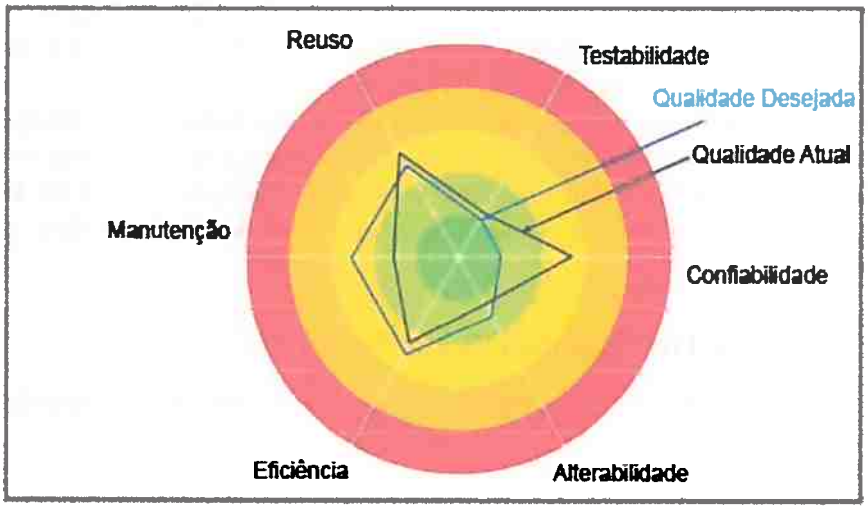

Figura 3.2: *Gráfico Kiviat para representação da dívida técnica por características proposto pelo método SQALE.*

Se o polígono da dívida técnica atual estiver totalmente dentro do polígono da dívida técnica máxima significa que a dívida técnica está sob controle. Caso parte do polígono de dívida técnica esteja fora, significa que a característica relacionada com o ponto que está fora passou dos níveis aceitáveis e precisa ser paga.

3.2.7.3 Pirâmide SQALE

Como as características podem ser colocadas em ordem de importância, então se pode construir uma pirâmide com as seguintes características:

- Cada campo, de cada linha, recebe o valor da dívida técnica da característica associada.
- Na última linha, toda a dívida da coluna deve ser somada.

Esta pirâmide permite visualizar a dívida acumulada de diferentes características, como mostra a figura 3.3.

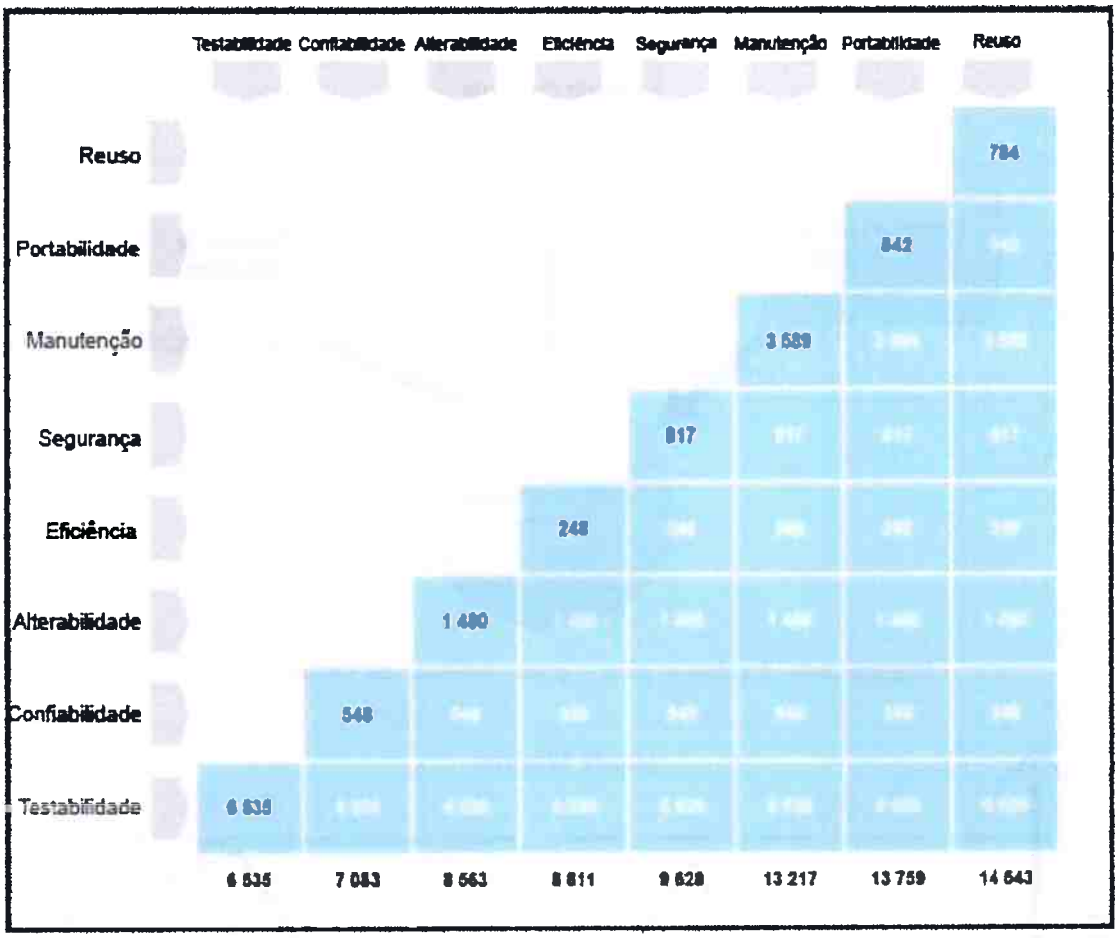

Figura 3.3: *Pirâmide de dívida técnica da metodologia SQALE.*

3.3 Métodos de Monitoramento

O objetivo de identificar e medir a dívida técnica é viabilizar e facilitar as tomadas de decisões (Seaman e Guo, 2011). Este conjunto que consiste de procedimentos para identificar, medir e tomar decisões, nesta ordem, gerando e tendo como base uma lista de itens de dívidas técnicas, será o arcabouço utilizado para o gerenciamento da dívida técnica, como mostra a figura 3.4.

O monitoramento da dívida técnica pode ser feito de duas maneiras. A primeira é de forma imediata e a segunda ao longo do tempo.

Além da dívida técnica explícita na lista de dívida técnica, é possível utilizar o monitoramento a fim de encontrar dívidas técnicas implícitas (Freire).

O que deve ser monitorado pode variar conforme as partes interessadas. Por exemplo: para o comprador do sistema pode interessar apenas a dívida técnica total do projeto atual, já para responsável pelos testes do sistema pode interessar apenas a dívida relacionada a testes e sua evolução ao longo do tempo.

Os índices de dívida técnica podem auxiliar no monitoramento. Com eles é possível analisar, rapidamente, determinadas partes do software, permitindo que cada parte interessada possa analisar a dívida técnica do seu ponto de vista.

3.3.1 Monitoramento Imediato da Dívida Técnica

Após identificar os itens de dívida técnica de interesse e, ao se processar com as funções de remediação para se obterem as medidas de custo para cada um destes itens de dívida técnica, se chega a uma lista de dívida técnica.

O monitoramento imediato da dívida técnica consiste em realizar o processo descrito acima e

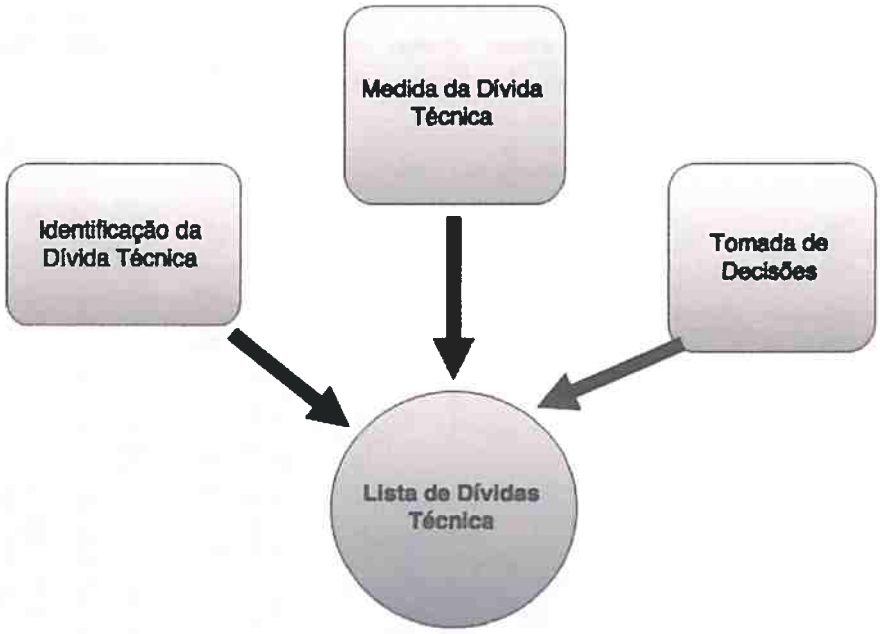

Figura 3.4: *Arcabouço para o gerenciamento da dívida técnica.*

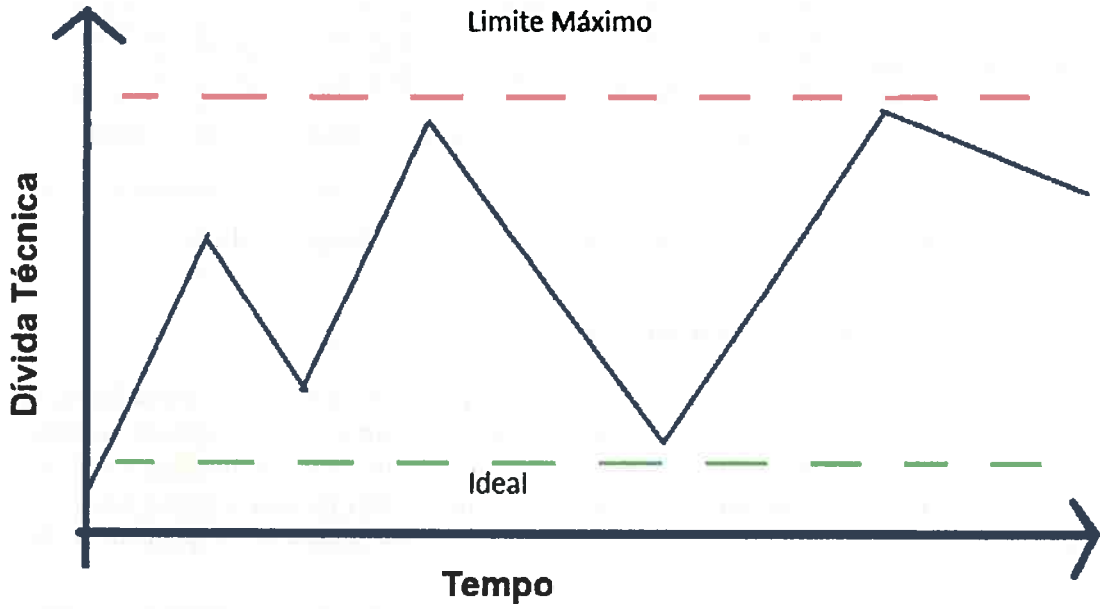

Figura 3.5: *Gráfico de linha para monitorar a dívida técnica ao longo do tempo.*

analisar a dívida técnica a fim de tomar decisões. Este monitoramento permite verificar os pontos com maiores problemas de dívida técnica, ou seja, pontos onde exista uma quantidade considerável de dívida técnica com urgencia de pagamento.

No monitoramento imediato os índices são essenciais para a tomada de decisão, pois eles permitem verificar de forma fácil a quantidade de dívida técnica em determinado ponto.

3.3.2 Monitoramento ao Longo do Tempo

O monitoramento ao longo do tempo consiste em reunir as listas das dívidas técnicas de diferentes datas a fim de analisá-las em diversos aspectos, tendo como objetivo principal o acompanhamento de sua evolução.

Este tipo de monitoramento é importante para se ter um controle da dívida técnica. Isso permite

que não se deixe que ela aumente em pontos críticos onde prioridade de pagamento é alta. Este processo pode evitar com que o projeto entre em uma falência técnica.

Os índices também são muito importantes para o monitoramento ao longo do tempo. Isso ocorre, pois com os índices de diferentes datas é possível verificar a evolução da dívida técnica em pontos chaves. Logo, por meio dos índices é possível de se tomar algumas providências a fim de inibir ou até mesmo diminuir o crescimento desta dívida técnica em específico.

3.3.3 Exemplos de Métodos de Monitoramento

Existem diversos métodos que podem ser utilizados para monitorar a dívida técnica, alguns deles são mais específicos, monitorando a dívida técnica em si e outros mais subjetivos, monitorando fatores que podem impactar na dívida técnica.

- ***Tracker***: é realizado um acompanhamento específico para cada item dívida técnica, ou seja, para cada item é preciso verificar alguns pontos como: a prioridade para pagar a dívida técnica, quando será paga, por quem será paga e informações adicionais. O *tracker* ajuda a realizar um planejamento para o pagamento da dívida técnica, bem como um histórico de seus pagamentos.

- ***Kanban***: é uma ferramenta para fazer o acompanhamento das tarefas realizadas em um projeto. Com a dívida técnica identificada é possível colocar cada item, ou um grupo de itens, como tarefa a ser realizada. Sendo assim o monitoramento de quais dívidas técnicas foram, estão sendo ou serão pagas pode ser feito de forma simples, utilizando o *kanban*.

- **Monitorar a Felicidade do Cliente**: este é um método subjetivo e não exige que a dívida técnica seja identificada, porém a felicidade do cliente pode ser um indício para saber se o projeto está indo na direção certa, e consequentemente, sem um excesso de dívida técnica. Quando um cliente começa a ficar impaciente, irritado ou desinteressado pelo projeto, pode ser que seu projeto tenha excesso de dívida técnica que afeta diretamente o humor do cliente.

- **Monitorar a Dificuldade de Modificar o Projeto**: quando a equipe de desenvolvimento começa a encontrar dificuldade para realizar modificações, sejam elas corretivas ou implementações de novas funcionalidades, é um indício de que a dívida técnica pode estar alta ou aumentando. Este monitoramento diz respeito, principalmente, a dívida técnica das características de reuso, manutenção e alterabilidade. Para este método é importante verificar se a capacidade dos atuais desenvolvedores estão são equivalentes a de desenvolvedores anteriores, caso não sejam, talvez seja necessário um aprimoramento técnico.

- **Monitorar a Equipe**: equipes que não estão felizes, trabalhando além do horário, são desorganizadas, com problemas técnicos ou qualquer outro problema podem ter seus projetos prejudicados. Em especial, o código-fonte pode ser afetado com dívida técnica devido a cansaço, desatenção, incapacidade técnica ou outros fatores relacionados à equipe.

- **Realizar Retrospectivas**: realizar retrospectivas com os envolvidos no projeto é importante para tentar identificar problemas e verificar se as expectativas de todos foram correspondidas. Os problemas expostos nestes eventos devem ser investigados a fim de verificar seu impacto na qualidade do produto, sendo estes problemas, indícios de dívida técnica.

- ***Commits* sobre o Tempo**: um método para verificar se a equipe está empenhada no projeto e verificar a complexidade de implementar novas funções ou realizar manutenção é acompanhar a quantidade e qualidade dos *commits* sobre o tempo. Caso a curva fique decrescente pode ser um indício de que o projeto possui dívida técnica, pois escrever uma linha de código pode ter ficado mais custoso.

É importante observar que muitos desses métodos são intuitivos e necessitam de uma análise humana para verificar se os problemas detectados estão relacionados, ou não, com dívida técnica.

Por exemplo, a quantidade de *commits* poderia diminuir bruscamente de um mês para outro, pois a demanda de novas funcionalidades pode ter diminuído.

3.4 Conclusão do Capítulo

Primeiramente, os métodos de identificação da dívida técnica que são: análise estática e automática de código, *code smells*, padrões de *design* e *grime* e testes. Estes métodos foram detalhados e relacionados com os principais tipos de dívidas técnicas encontrados com cada um deles.

Já para a medida da dívida técnica, foi apresentado um método para calculá-la, utilizando funções de remediação de custo. Além disso, foram apresentados métodos de agrupá-la e dimensioná-la. E por fim, mostrado três métodos para mostrar a quantificação da dívida técnica que são: a escala SQALE, o gráfico de Kiviat e a pirâmide SQALE.

Para o monitoramento da dívida técnica foram apresentadas uma série de técnicas que envolvem métodos práticos e intuitivos. Os métodos práticos são: *tracker*, *kanban* e *commits* sobre o tempo. Os métodos intuitivos compreendem: monitorar a felicidade do cliente, monitorar a dificuldade de modificar o projeto, monitorar a equipe e realizar retrospectivas.

No próximo capítulo serão apresentadas ferramentas que utilizam pelo menos um desses métodos como base para identificar, medir e monitorar a dívida técnica.

Capítulo 4

Ferramentas

Neste capítulo serão apresentadas ferramentas capazes de medir a dívida técnica de projetos de software. Apresentando o Sonar Qube, a extensão Technical Debt Evaluation (SQALE), o CAST AIP, Kiuwan - System Code, o SQuORE Technical Debt, o Mia-Quality e o Quality Reviewer. Todas estas ferramentas foram construídas utilizando como base a metodologia SQALE, em algumas ferramentas a metodologia fica mais evidente, como é o caso do Sonar e sua extensão e, em outras, nem tanto, como o Mia-Quality. As ferramentas foram escolhidas por serem citadas em artigos e também por uma busca no Google e no Google Scholar.

Este capítulo foca em uma análise descritiva das funcionalidades de cada uma das ferramentas, tentando descrever o que é cada, quais são as funcionalidades principais e como elas são utilizadas em relação a dívida técnica.

4.1 Sonar Qube

O Sonar Qube (Source, 2016) é uma plataforma livre para gerenciar qualidade de código. Ela possui uma grande quantidade de recursos nativos, tais como: métricas padrões, regras de código, unidades de teste, gerenciamento básico de dívida técnica e, além disso, possui mais de 50 *plugins* disponíveis.

Esta plataforma está preparada para cuidar de sete eixos de qualidade de código: Arquitetura e *design*, duplicações, testes de unidade, complexidade, possíveis erros, regras de código e comentários, como mostra a figura 4.1.

Figura 4.1: *Os sete eixos de qualidade do Sonar Qube. Retirada de (Source, 2016).*

O Sonar tem um caminho eficiente de navegação ??, um balanço entre visualização de alto-nível, painel de ferramentas, máquina do tempo e ferramentas para encontrar defeitos. Isso permite descobrir rapidamente projetos e/ou componentes que estão com dívida técnica e estabelecer planos de ações para pagá-las.

Mais de 20 linguagens de programação e de marcação são cobertas através dos *plugins*, entre elas estão: Java, C/C++, C#, PLSQL, .NET, PHP e XML, porém algumas extensões são oferecidas

Figura 4.2: *Modelo de navegação do Sonar Qube. Retirada de (Source, 2016).*

apenas comercialmente. Também o gerenciamento de portfólios de projetos pode ser feito utilizando serviços comerciais.

O Sonar ainda permite integração com Servidores de Integração Contínua tais como: o Jenkins, Hudson, Atlassian Bamboo e Apache Continuum. A plataforma também permite integração com o Eclipse, podendo facilitar e aumentar a produtividade do desenvolvimento.

É possível utilizar o Sonar em diversas versões dos mais variados Sistemas Operacionais, tais como: Linux, Mac OS, Solaris e Windows.

A figura 4.2 mostra o modelo de navegação da plataforma Sonar Qube, onde a partir da visualização de portifólios de qualidade, visualizações de projeto e do código-fonte é possível criar planos de ações para realizar melhorias no projeto.

4.1.1 Sonar Qube e a Dívida Técnica

Nas versões mais recentes do Sonar Qube (5.x) (Source, 2016) a dívida técnica já é tratada, porém de forma simples, permitindo apenas a alterações de alguns parâmetros globais de configuração.

Apesar de simples, a plataforma é capaz de identificar possíveis itens de dívida técnica, calcular o valor de pagamento e permite um monitoramento a nível de código-fonte ou um simples monitoramento de alto-nível.

4.1.1.1 Identificando Dívida Técnica com o Sonar Qube

Para o Sonar Qube um item de dívida técnica é uma violação de regra e recebe o nome de *issue*, podendo ser de diferentes naturezas, tais como: problemas de padronização, comentários, problemas de eficiência, cobertura de testes, entre outros.

As *issues* são criadas ou alteradas enquanto a análise de código está sendo executada. Para cada linguagem de programação a plataforma executa um perfil de qualidade, ou seja, ela analisa um conjunto de regras previamente determinadas.

Além disso, as regras e, consequentemente, as *issues* são classificadas por características de forma similar a proposta pelo modelo de qualidade SQALE. As características principais são as mesmas do modelo: reuso, portabilidade, manutenção, segurança, eficiência, usabilidade, confiabilidade e testabilidade; além da característica de alterabilidade.

O Sonar Qube utiliza um nível de características e um de sub-caraterísticas e todas as regras estão associadas a uma sub-característica. Por exemplo, a regra de conformidade de nomes de métodos está associada a sub-característica legibilidade, que por sua vez está associada a característica de manutenção.

4.1.1.2 Medindo Dívida Técnica com o Sonar Qube

A medida da dívida técnica calculada pelo Sonar Qube é feita utilizando o método SQALE (Letouzey, 2012a). Durante o processo de análise de código, para cada item de dívida técnica o Sonar aplica uma função de remediação de custos definida na regra de conformidade, devolvendo o tempo, em minutos, necessário para pagar aquele item de dívida técnica.

As funções de remediação utilizadas pelo Sonar são todas lineares da forma $y(x) = Ax + B$, sendo A e B parâmetros fixos definidos no banco de dados através do conhecimento de especialistas e x a quantidade de repetições, por exemplo: um bloco de código que está duplicado 15 vezes terá um $x = 15$.

4.1.1.3 Monitorando Dívida Técnica com o Sonar Qube

Para monitorar a dívida técnica o Sonar Qube disponibiliza algumas ferramentas, para visualizar o estado atual da dívida técnica.

Figura 4.3: *Indicador de dívida técnica nativo do Sonar Qube para o Apache Maven.*

A figura 4.3 mostra um indicador geral de dívida técnica, nele é mostrado uma nota na escala do método SQALE (A, B, C, D ou E) e a taxa da quantidade de código que está em dívida técnica.

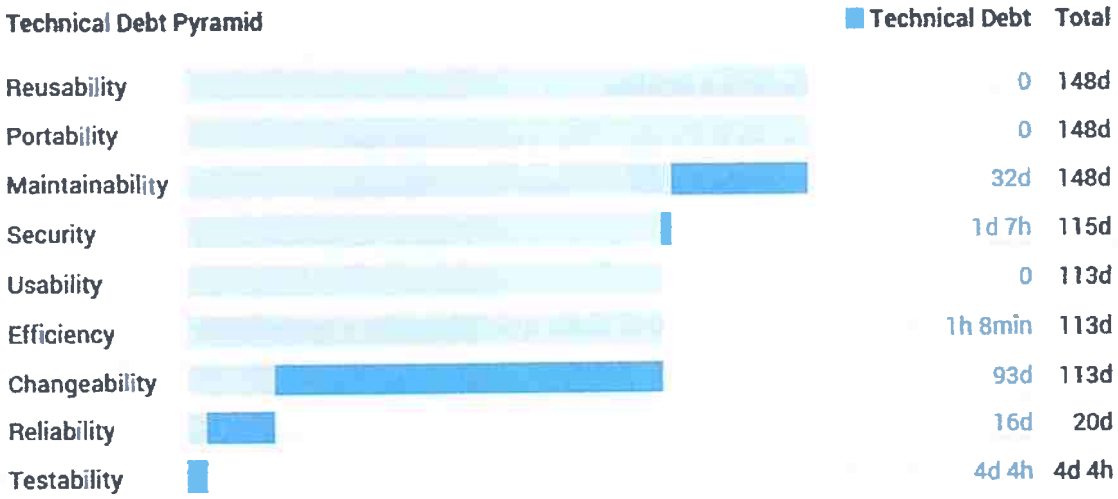

Figura 4.4: *Pirâmide de dívida técnica nativa do Sonar Qube para o Apache Maven.*

A figura 4.4 mostra a pirâmide de dívida técnica. Ela mostra a quantidade de dívida técnica de cada uma das características de qualidade e, também, um acumulado de baixo para cima da quantidade de dívida técnica. É interessante observar que cada barra está dívida em duas partes, o azul mais claro indica o acumulado e o azul mais escuro indica a dívida técnica daquela característica que está sendo adicionado ao acumulado.

Figura 4.5: *Indicadores de dívida técnica nativos do Sonar Qube para o Apache Maven.*

A figura 4.5 possui indicadores que mostram a quantidade total de dívida técnica no projeto analisado, a quantidade de *issues* que geraram a dívida e também a dívida técnica/*issue* em cinco categorias, conforme o risco que ela pode trazer ao software, sendo estes riscos: muito baixo, baixo, médio, alto e muito alto.

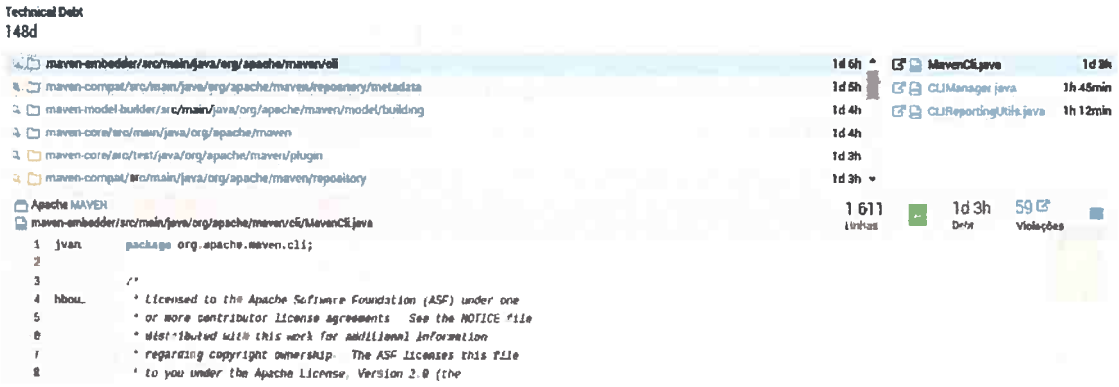

Figura 4.6: *Ferrmenta Drill Down nativa do Sonar Qube para o Apache Maven.*

Na figura 4.6 é mostrada uma ferramenta *drill down* que permite explorar a dívida técnica a nível de código-fonte. Com esta ferramenta é possível filtrar pelas pastas e arquivos do projeto, facilitando a visualização da dívida técnica em partes específicas do software. Além disso, ele apresenta a dívida técnica no código-fonte o que facilita a análise.

Figura 4.7: *Indicadores de métricas de código nativos do Sonar Qube, incluindo dívida técnica.*

A figura 4.7 mostra os indicadores do *drill down* que informam a quantidade de linhas de código do arquivo, a nota da escala SQALE, o custo total para o pagamento da dívida do arquivo e a quantidade de violações com um *link* para visualizá-las.

O Sonar Qube possui uma ferramenta para visualizar a dívida técnica/*issues* em detalhes, exibindo uma lista completa, mostrada na figura 4.8. Esta lista também permite filtrar as informações por risco, situação, data de inserção, regra, marcador, módulo, diretório, arquivo, responsável por pagar, responsável por inserir, responsável por relatar, linguagem de programação e plano de ação.

Na figura 4.9, para cada um dos itens de dívida técnica/*issue* é mostrado qual é o problema, risco, situação, responsável técnico, plano de ação e custo para pagar a dívida. Além disso, também é mostrado a quanto tempo a dívida foi inserida e marcadores para ajudar a classificá-la.

Além disso, também é possível visualizar detalhes da regra que foi violada. Ela mostra a qual característica e sub-característica a regra pertence, uma descrição detalhada e, em muitos casos, possui um guia para mostrar como pagar a dívida técnica, como mostrado na figura 4.10.

A plataforma nativa, também, permite criar planos de ação para o pagamento da dívida técnica. Um plano de ação é o agrupamento de *issues* que devem ser pagas até um determinado prazo, como mostra a figura 4.11.

Para adicionar *issues* ao plano de ação, basta na visualização da *issue* colocá-la em um plano de ação. Para que o plano de ação fique ainda mais completo é interessante que toda *issue* adicionada tenha um responsável técnico, ou seja, tenha o responsável pelo pagamento da dívida técnica, como mostra a figura 4.12.

4.1.1.4 Extensão para Lidar com Dívida Técnica no Sonar Qube

Technical Debt Evaluation (SQALE) (Source) é uma extensão comercial, no valor de 2.700 euros, que amplia a implementação da metodologia SQALE já existente na plataforma Sonar Qube (Source).

Figura 4.8: *Ferramenta Drill Down, nativos do Sonar Qube, para monitorar issues para Apache Maven.*

Figura 4.9: *Issue na ferramenta Drill Down nativa do Sonar Qube para Apache Maven.*

Esta extensão amplia as funcionalidades relacionadas a dívida técnica já existentes na plataforma. Com isso, ela permite alterar alguns parâmetros de configuração, características, subcaracterísticas e funções de remediação de custo.

Apesar da extensão inserir alguns parâmetros, o cálculo da dívida técnica é realizado pela plataforma nativa utilizando as funções de remediação de custo. Outra função que também é estendida da plataforma nativa é o cálculo da escala SQALE, que é feito através do cálculo da densidade da dívida técnica, que é o esforço de pagar toda a dívida técnica sobre o esforço de escrever todo o código já escrito.

A fórmula da densidade é:

$$\frac{\text{Custo de Pagar a Dívida Técnica}}{\text{Custo de Desenvolvimento}}.$$

Podendo ser reescrita como:

$$\frac{\text{Custo de Pagar a Dívida Técnica}}{\text{Custo de desenvolver 1 linha de código} * \text{Número de linhas de código}}.$$

Ou utilizando a complexidade:

$$\frac{\text{Custo de Pagar a Dívida Técnica}}{\text{Custo de desenvolver 1 ponto de complexidade} * \text{Complexidade}}.$$

Após calcular a densidade da dívida técnica é possível definir uma nota na escala SQALE, por padrão essa nota é dada pela seguinte regra:

- A $->$ para densidade menor ou igual a 0,1;

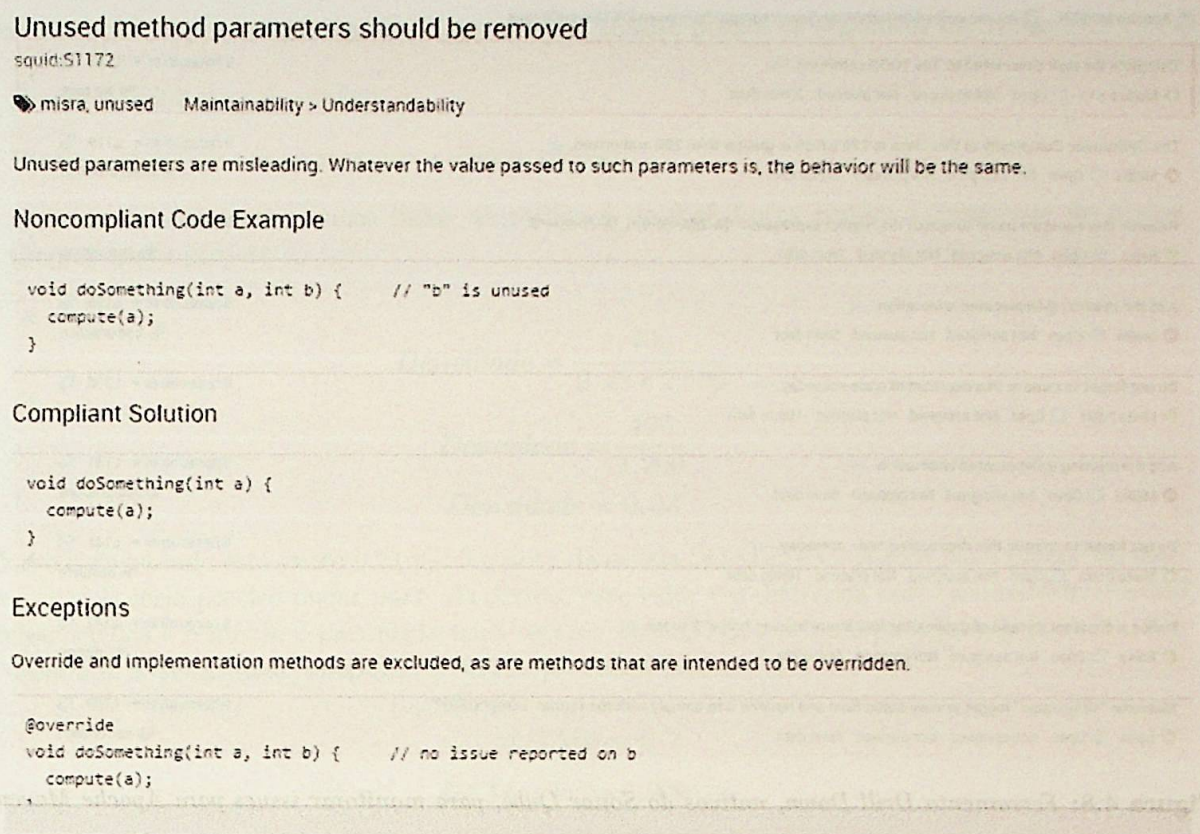

Figura 4.10: *Detalhes de uma issue na ferramenta Drill Down nativa do Sonar Qube para o projeto Apache Maven.*

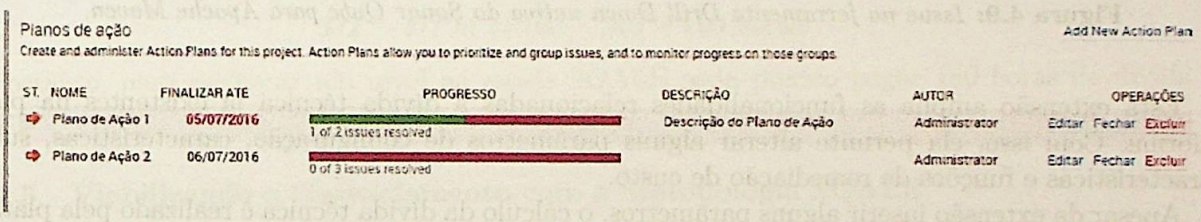

Figura 4.11: *Lista fictícia de planos de ações nativa do Sonar Qube.*

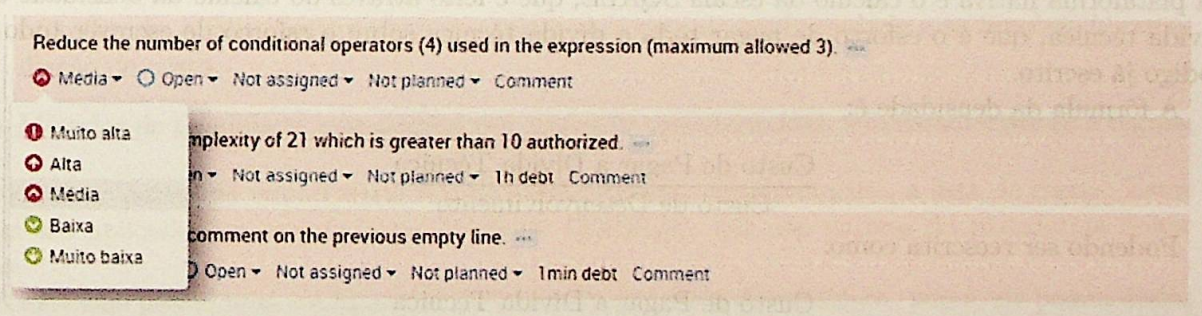

Figura 4.12: *Exemplo de tela do Drill Down nativo do Sonar Qube, que permite alterar informações sobre a dívida técnica/issue.*

- B − > para densidade entre 0,11 e 0,2;
- C − > para densidade entre 0,21 e 0,5;
- D − > para densidade entre 0,51 e 1; e
- E − > para densidade maior que 1.

Por exemplo, suponha que um dado projeto em análise possui os seguintes métricas:

- Possui 2.500 linhas de código
- Precisa de 400 horas para pagar a dívida técnica
- O custo de desenvolver uma linha de código é de 0,48 horas Então a densidade de dívida técnica neste projeto é:

$$Densidade = \frac{400}{0,48 * 2.500}$$

$$Densidade = \frac{400}{1.200}$$

$$Densidade = 0,33$$

Como a densidade está entre 0,21 e 0,5 a nota da escala SQALE é C. Agora para passar de nota C para nota B seria preciso pagar parte da dívida, este valor é a diferença entre a quantidade atual de dívida técnica e o máximo permitido para receber nota B. Como o máximo para a nota B é 0,2, para descobrir a quantidade máxima de dívida podemos fazer:

$$DTM/1.200 = 0,2$$

$$DTM = 0,2 * 1.200$$

$$DTM = 240$$

Logo, pode ter no máximo 240 horas de dívida técnica, como isso temos:

$$DT - DTM = 400 - 240 = 160 \text{ horas.}$$

Portanto, para melhorar um nível na escala SQALE seria preciso pagar 160 horas de dívida técnica.

4.1.1.5 Viabilizando o Gerenciamento com a Metodologia SQALE

A extensão viabiliza o gerenciamento completo das duas partes da metodologia SQALE que são:

- Métodos de Análise: responsável por tratar as regras de conformidade e as funções de remediação de custo.
- Métodos de Qualidade: responsável por tratar as características, sub-características e regras.

Ambas as partes da metodologia SQALE estão ligadas, principalmente, a lista de regras. Estas regras são aplicadas ao projeto pelos Perfis de Qualidade, que são nativos da plataforma Sonar Qube.

A extensão possui uma configuração padrão para o modelo de qualidade, isto é, ele possui dados estruturados de características, sub-características e regras. Para cada linguagem de programação é possível adicionar regras e funções de remediação de custo para realizar a análise e uma categorização padrão das regras em sub-caraterísticas e estas em características.

A figura 4.13 mostra a tela para gerenciar características e sub-caraterísticas onde é possível adicionar, alterar, remover e ordenar, tanto uma quanto a outra. Esta é uma das telas mais importantes para configurar os métodos de qualidade usando o SQALE.

Na figura 4.14 é mostrada a tela para gerenciar as regras onde é possível associar cada regra a uma sub-característica. Além disso é possível definir a função de remediação que deverá ser usada para calcular o custo da dívida técnica. Os três tipos de função são:

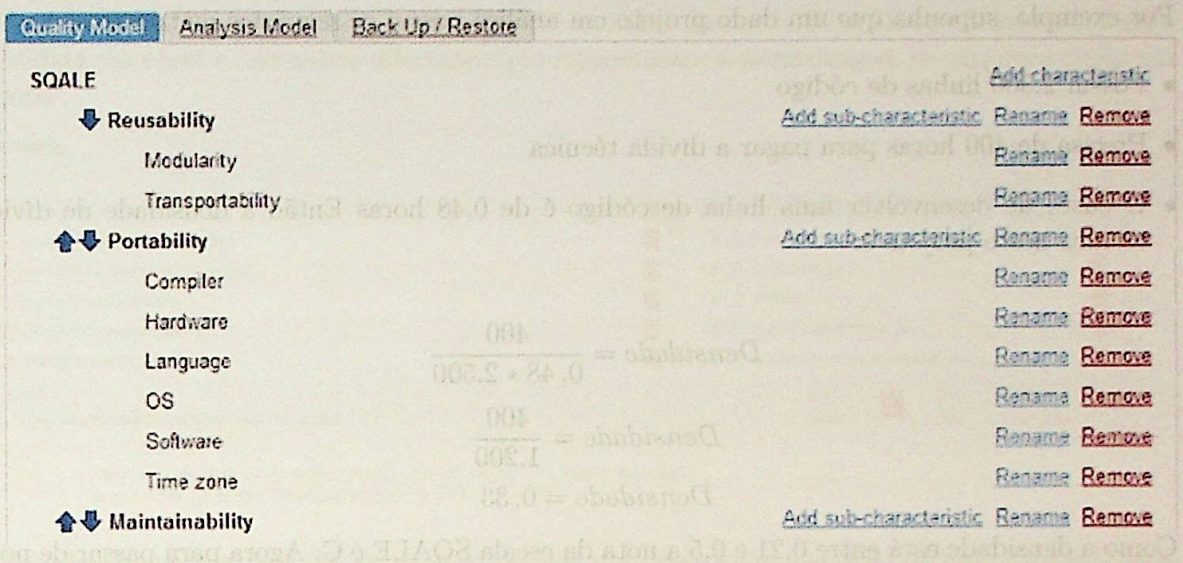

Figura 4.13: *Tela de gerenciamento de características da extensão Technical Debt Evaluation (SQALE). Retirada de http://www.sonarsource.com/products/plugins/governance/sqale/installation-and-usage/.*

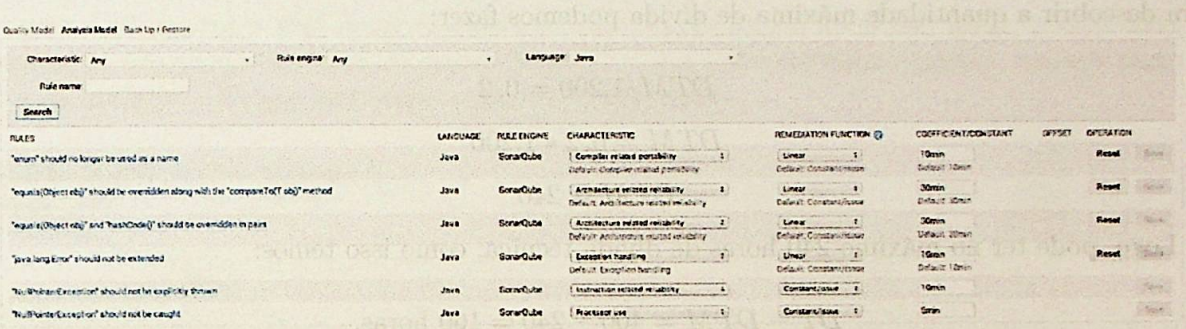

Figura 4.14: *Tela de gerenciamento de regras de conformidade da extensão Technical Debt Evaluation (SQALE). Retirado de http://www.sonarsource.com/products/plugins/governance/sqale/installation-and-usage/.*

- **Constante**: O custo de pagar a dívida técnica de uma dada regra é sempre o mesmo. A fórmula é $DT = constante$.
- **Linear**: O custo de pagar a dívida técnica depende do grau da *issue*. Por exemplo, uma regra relacionada com a quantidade de linhas de código, pode usar este número para definir o grau. A fórmula é $DT =$ grau $*$ tempo para consertar um grau.
- **Linear com parâmetro**: O custo de pagar a dívida técnica depende do grau da *issue* mais um tempo constante para poder analisá-la, por exemplo. A fórmula é $DT =$ (grau $*$ tempo para consertar o grau) $+$ parâmetro.

4.1.1.6 Visualizações da Dívida Técnica

O principal benefício, em relação a plataforma nativa, é inserção de novas formas para visualizar a dívida técnica, principalmente, relacionada ao agrupamento de itens de dívida técnica.

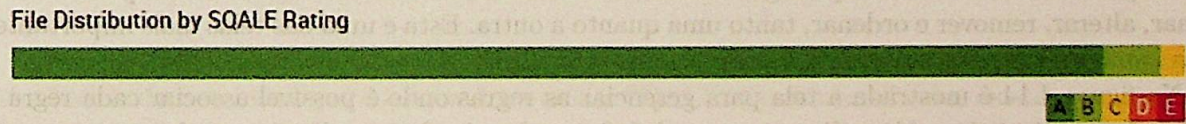

Figura 4.15: *Gráfico de distribuição de nota da escala SQALE da extensão Technical Debt Evaluation (SQALE) para o Sonar Qube 5.2.*

A figura 4.15 mostra um gráfico de distribuição de notas da escala SQALE nos arquivos, a barra está dividida em cores e tamanhos diferentes que representam a porcentagem de arquivos estão em cada nota.

Figura 4.16: *Ferramenta Drill Down com informações adicionais fornecidas pela extensão Technical Debt Evaluation (SQALE) para o JDK 7.*

A figura 4.16 mostra o *drill down* existente na plataforma nativa do Sonar Qube, porém com incremento da informação da nota da escala SQALE tanto para módulos e pastas, quanto para arquivos da listagem e do que está sendo exibido o código-fonte.

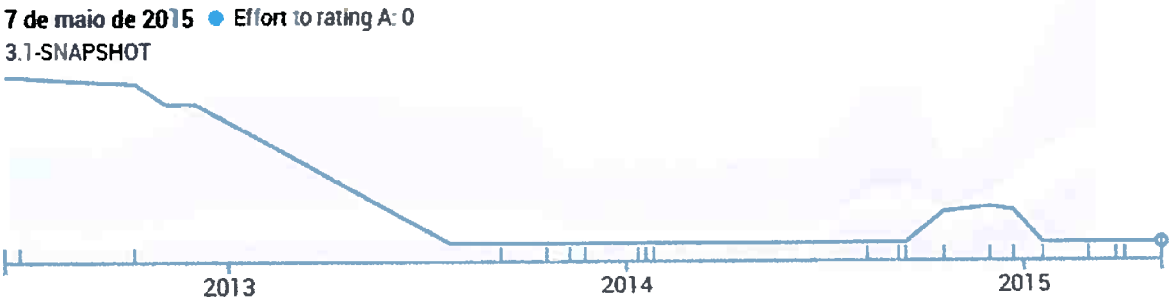

Figura 4.17: *Gráfico de esforço para levar o projeto atual para uma nota A na escala SQALE da extensão Technical Debt Evaluation (SQALE) para o JDK 7.*

Na figura 4.17 é mostrado um gráfico de esforço necessário para levar o projeto atual para uma nota A na escala SQALE pelo tempo. Com este recurso é possível verificar a evolução da dívida técnica com o tempo, ajudando na tomada de decisão de pagar ou assumir dívida técnica. É importante observar que o gráfico pode aumentar ou diminuir, inclusive bruscamente, caso o perfil de qualidade seja alterado, ou seja, em um dado momento no tempo novas regras tenha sido inseridas na análise ou as já existentes tenham sido retiradas.

Figura 4.18: *Visualizador de esforço necessário para conseguir nota A na escala SQALE da extensão Technical Debt Evaluation (SQALE) para o Apache Commons.*

A figura 4.18 mostra um visualizador de quantidade de tempo necessário para pagar a dívida técnica e levá-la da nota atual para a nota A na escala SQALE. Neste visualizador existe, também, um *link* para verificar as *issues* que levam a este valor de tempo.

Na figura 4.19 é mostrado um painel com informações sumarizadas relacionadas com a dívida técnica. Do lado esquerdo, tem-se a nota na escala SQALE atual e também a quantidade necessária

Figura 4.19: *Painel com informações sumarizadas relacionadas com a dívida técnica da extensão Technical Debt Evaluation (SQALE) para o Apache Commons.*

de tempo para pagar a dívida técnica e obter uma nota melhor, até chegar a nota A. Do lado direito, tem-se a densidade de dívida técnica obtida pela técnica da metologia SQALE, em seguida tem-se o valor total, em horas, necessário para pagar toda a dívida técnica do projeto e, por último, tem-se a quantidade de total de linha de código do projeto.

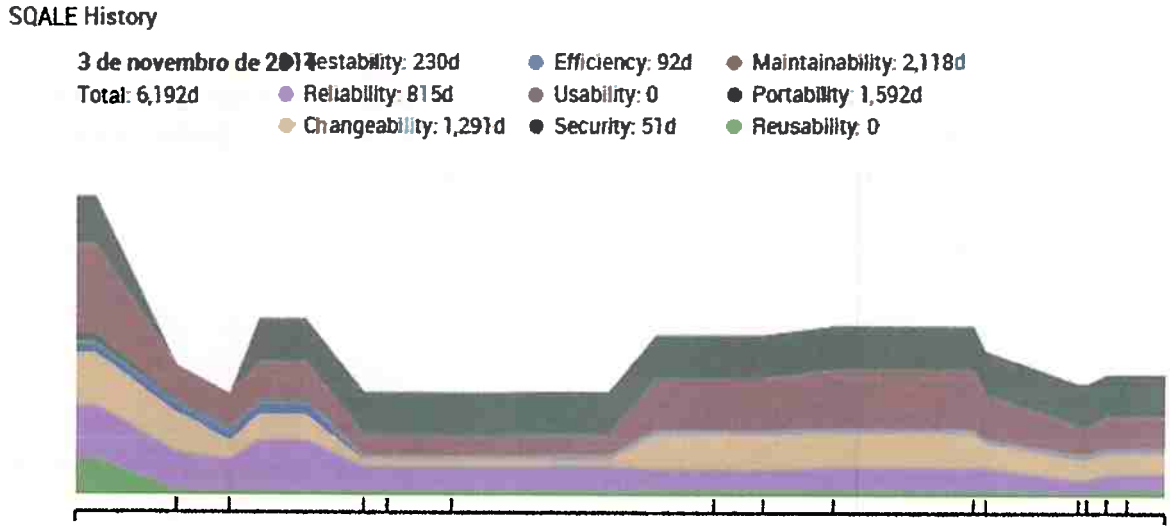

Figura 4.20: *Gráfico de evolução da dívida técnica pelo tempo da extensão Technical Debt Evaluation (SQALE) para o JDK 7.*

A figura 4.20 mostra um gráfico da evolução da dívida técnica pelo tempo. Além disso, para cada intervalo de tempo, é mostrada também a quantidade de dívida técnica em cada uma das características. Este gráfico é útil para monitorar a evolução da dívida técnica dentro de suas respectivas características e, também, ajudar na tomada de decisão de qual tipo de dívida técnica pode ser paga ou adquirida.

Figura 4.21: *Pirâmide de dívida técnica da extensão Technical Debt Evaluation (SQALE) para o JDK 8.*

A pirâmide de dívida técnica agrupada pelo risco, é mostrada na figura 4.21. Nela é mostrada a quantidade de dívida técnica acumulada de cima para baixo. É interessante observar que cada

barra está dívida em duas partes, o azul mais claro indica o acumulado e o azul mais escuro indica a dívida técnica daquele risco que está sendo adicionado ao acumulado.

Figura 4.22: *Lista de arquivos com a maior quantidade de dívida técnica da extensão Technical Debt Evaluation (SQALE) para o JDK 7.*

A figura 4.22 mostra um painel com os cinco arquivos que mais possuem dívida técnica. Além disso, é possível filtrar a dívida técnica pela sua característica, exibindo assim os cinco arquivos com a maior dívida técnica em uma dada característica. Este painel pode ser interessante para escolher quais arquivos serão os primeiros a receberem pagamento da dívida técnica.

Figura 4.23: *Gráfico Sunbirst da extensão Technical Debt Evaluation (SQALE) para o JDK 7.*

O gráfico *Sunburst*, mostrado na figura 4.23, é parecido com um gráfico de pizza, porém ele é

dividido em camadas. Da parte mais interna para a mais externa, a primeira camada representa a porcentagem de dívida técnica em cada uma das características; a segunda camada é limitada ao espaço da primeira e representa a distribuição da dívida técnica dentro das sub-características; e a terceira camada é limita ao espaço da segunda camada e representa a distribuição da dívida técnica em regras de conformidade.

Com este gráfico é fácil observar quais características que estão impactando fortemente para o acu mulo da dívida técnica e, também, é possível ver detalhes, como por exemplo as sub-características e as regras de conformidade, facilitando a decisão de quais tipos de de dívida técnica devem ser priorizados para pagamento.

4.2 CAST

O CAST Application Intelligence Platform (Cast AIP) (Software, 2016) é uma ferramenta para uso empresarial para medição de software e análise de qualidade feita para analisar desde de níveis abstratos ao código-fonte, diferentes aplicações tecnológicas para vulnerabilidades técnicas, aderência à arquitetura, padrões de código e, com isso, fornecer informações as equipes de desenvolvimento.

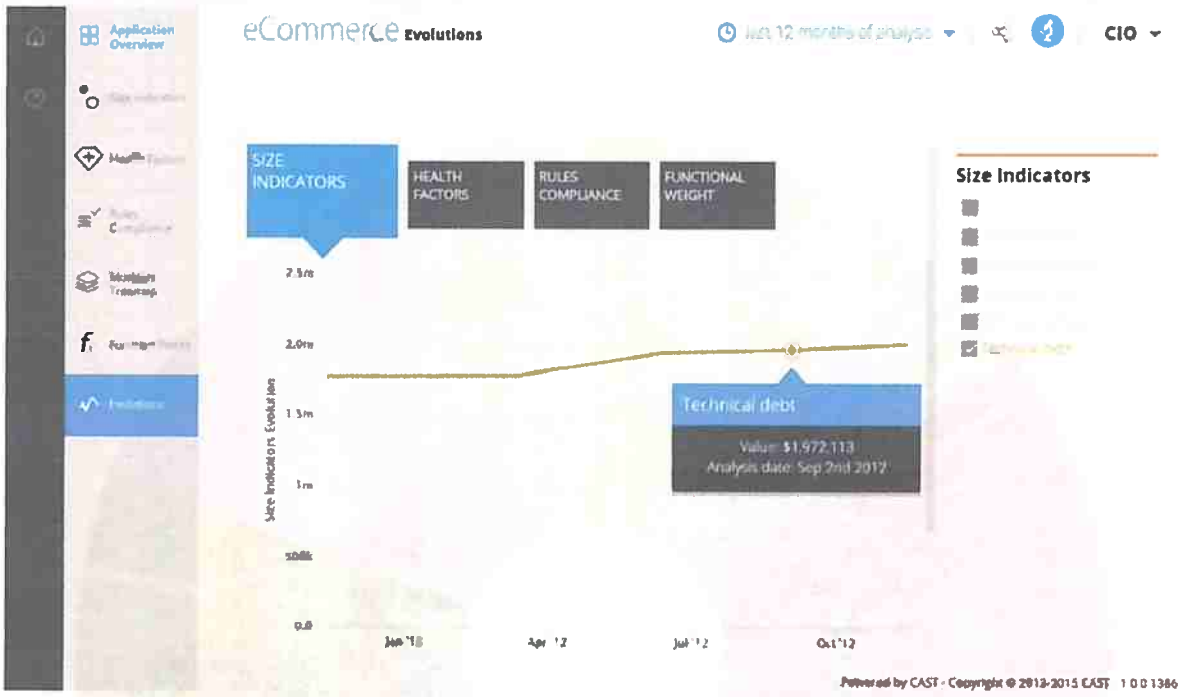

Figura 4.24: *Tela do Cast AIP para analisar dívida técnica. Retirado de http://www.castsoftware.com/solutions/control-your-technical-debt.*

O Cast AIP é constituído por quatro painéis principais que são:

- Application Analytics Dashboard (CAST AAD): fornece informações analíticas para gerentes de tecnologia conduzirem o negócio.
- Application Engineering Dashboard (CAST AED): fornece falhas de código e sistema a nível estrutural para percepção e orientação de remediação para equipes de desenvolvimento de software e de garantia de qualidade.
- Enlighten: fornece um entendimento da estrutura do software para os desenvolvedores.
- Architecture Checker: fornece, para arquitetos, uma solução automática a fim de garantir arquiteturas provendo estabilidade e desempenho das aplicações críticas.

Em relação a dívida técnica, Cast AIP permite medir regularmente a dívida com uma abordagem baseada em fatos. Ele possui uma funcionalidade de diagnóstico que permite a equipe de desenvolvimento identificar as falhas estruturais mais graves adicionando a dívida técnica à aplicação. A figura 4.24 é um exemplo de tela do Cast AIP utilizada para analisar dívida técnica.

O Cast Research Labs (CRL) define a dívida técnica em uma aplicação como o custo de consertar problemas de qualidade estrutural que, se deixada de lado, coloca o negócio em sérios riscos. Para eles, dívida técnica inclui apenas aqueles problemas que são altamente prováveis de causar danos severos ao negócio, ou seja, isto não incluí todos os problemas, mas apenas os mais sérios.

Com base nesta definição e um grande repositório de projetos, a Cast Research Labs realizou uma análise em 1.400 aplicações contendo 550 milhões de linhas de código de mais de 160 organizações. Ela estimou que o custo de pagar a dívida técnica de uma aplicação de médio porte, com cerca de 300 mil linhas de código, é de $ 1.083.000, ou seja, uma média de $ 3,61 por linha de código. A figura 4.25 mostra um gráfico do custo de pagar a dívida técnica para cada 1.000 linhas de código, o gráfico também está dividido em linguagens de programação.

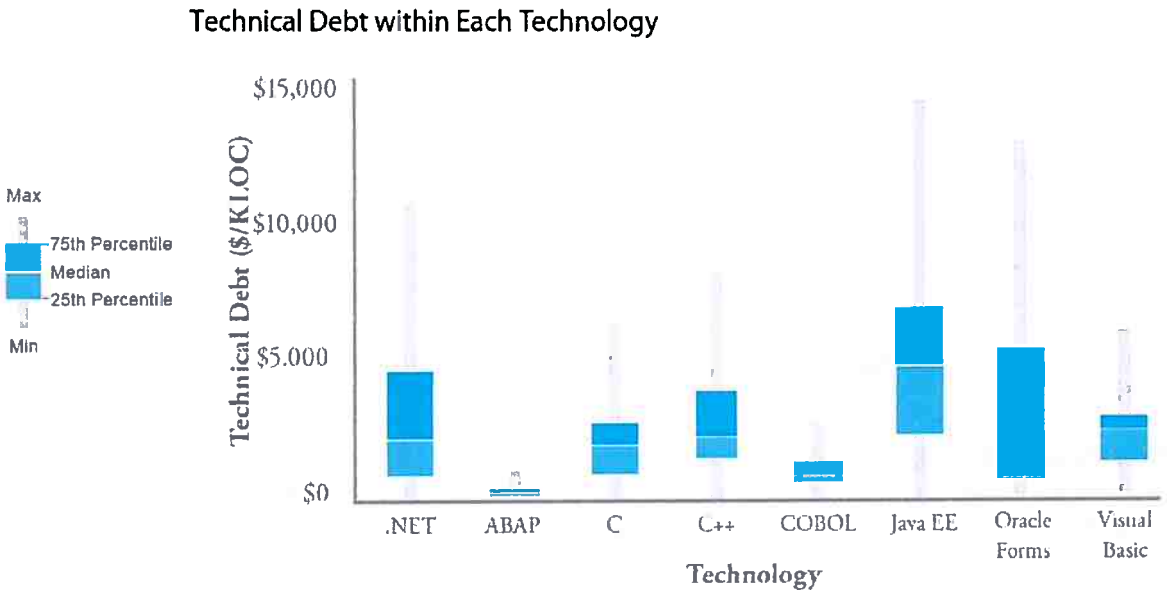

Figura 4.25: *Custo de pagar a dívida técnica para cada linguagem de programação. Retirado de http://www.castsoftware.com/research-labs/technical-debt-estimation.*

Além de medir a dívida técnica, o CAST AIP também fornece outras ferramentas para gerenciar a dívida técnica. Este gerenciamento é feito pelos seguintes passos:

- Identificar problemas de qualidade estrutural de um software.
- Calcular a dívida técnica causada por estes problemas.
- Priorizar a remediação dos problemas com um *benchmark* de boas práticas.
- Medir melhorias na qualidade estrutural e a redução da dívida técnica.

4.3 Kiuwan - System Code

O Kiuwan (Metrixware, 2016) é desenvolvido pela Matrixware e é o sucessor do Insite SaaS. Ele é um software analítico construído com o objetivo de encontrar evidências no código-fonte de aplicações. Conhecer estas evidências, permite evitar, antecipadamente, defeitos no ciclo de vida do desenvolvimento da aplicação, gerenciar os riscos a serem enfrentados no desenvolvimento e avaliar a dívida técnica para tomar decisões e melhorar a qualidade do código-fonte.

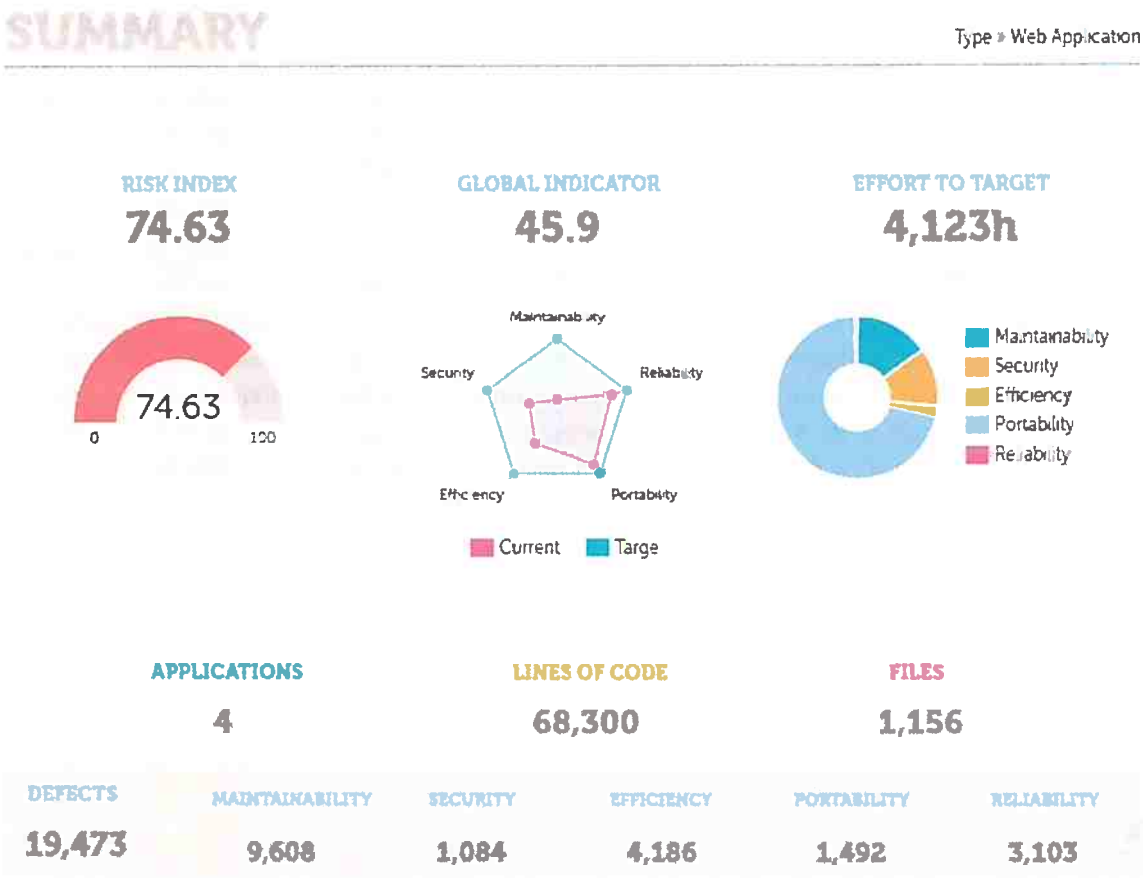

Figura 4.26: *Painel de indicadores gerais do Kiuwan.*

O software conta com Indicadores Gerais: índice de risco, um indicador global, quantidade de dívida técnica a ser paga, aplicações analisadas, quantidade de linhas de código. Como mostra a figura 4.26.

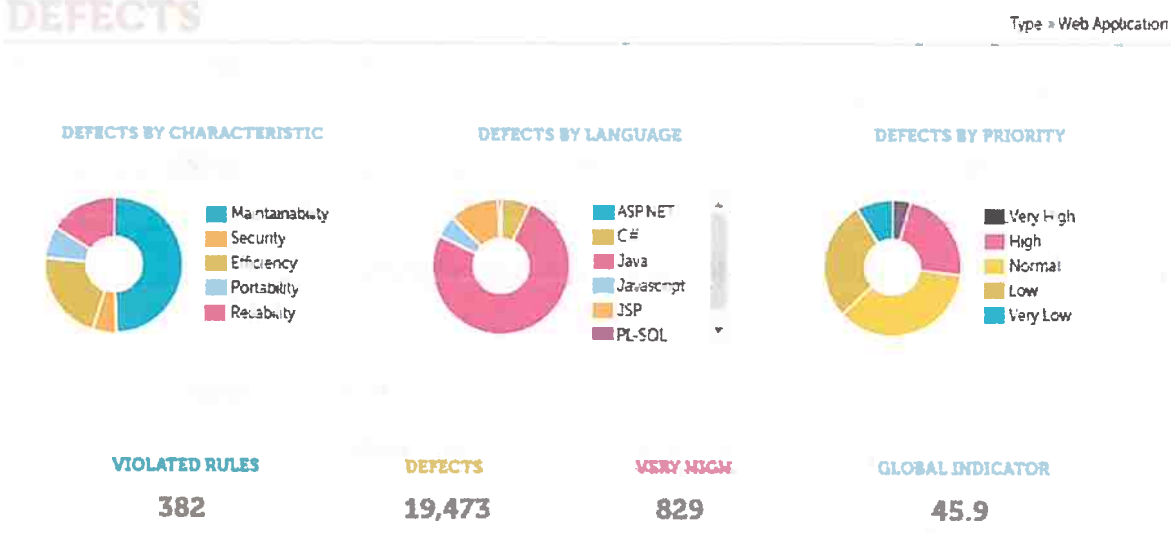

Figura 4.27: *Painel de defeitos do Kiuwan.*

A ferramenta também conta com um painel para visualizar os defeitos, como mostra a figura 4.27, com quantidade de regras violadas, quantidade de defeitos e defeitos de alto impacto e gráficos de defeitos divididos por característica, linguagem de programação e prioridade.

Já em relação a métricas, existe um painel que exibe a quantidade de linhas de código, índice

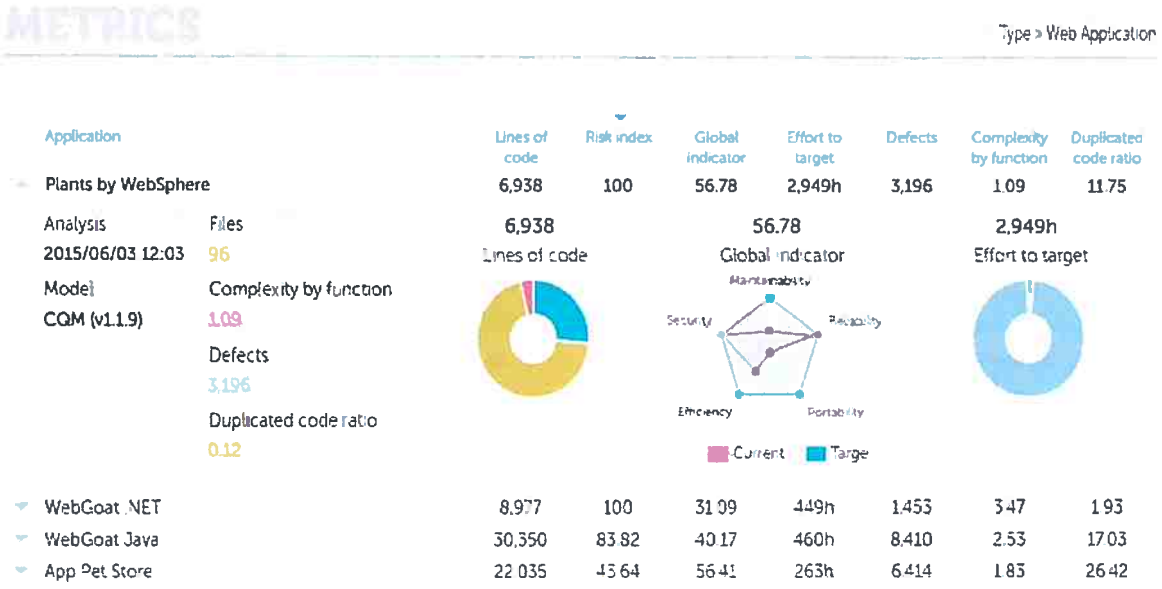

Figura 4.28: *Painel de métricas do Kiuwan.*

de risco, indicador global, quantidade de defeitos, complexidade por função, quantidade de código duplicado e gráficos, como mostra a figura 4.28.

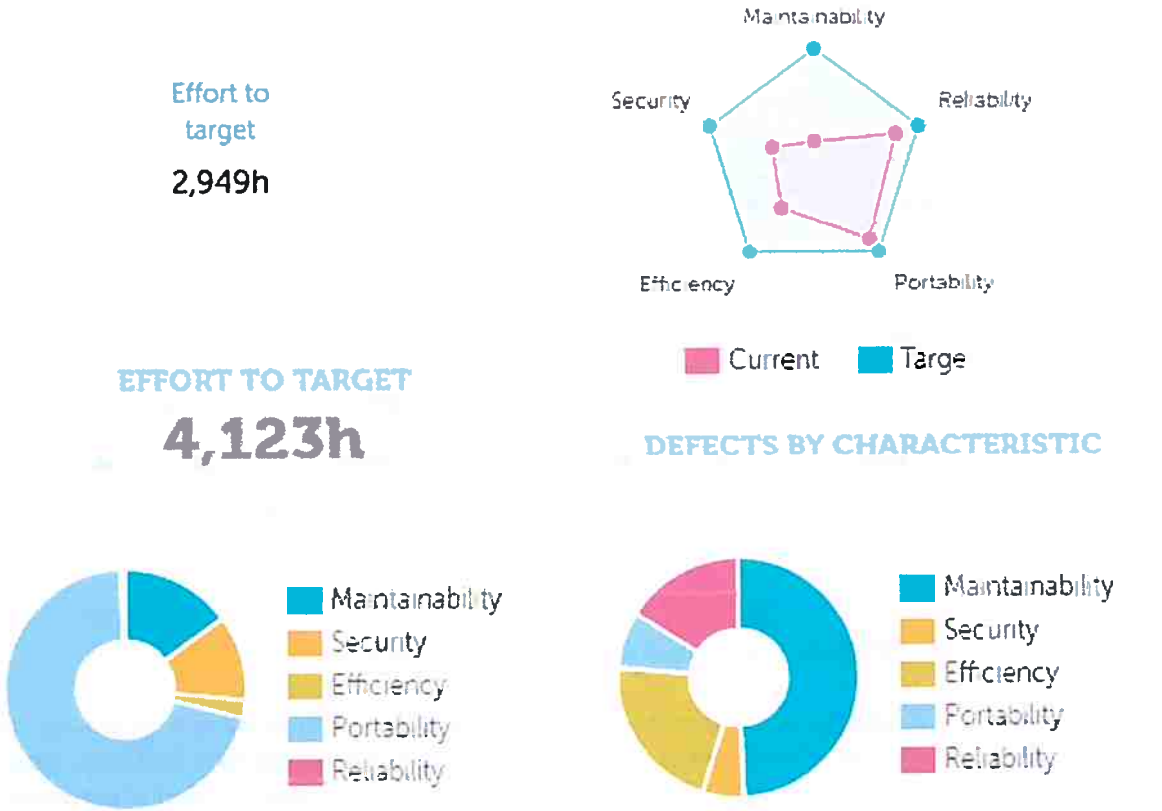

Figura 4.29: *Visualizações da dívida técnica do Kiuwan.*

Para explorar a metáfora da dívida técnica, o Kiuwan analisa os defeitos encontrados no código-fonte. Ao invés de dar o custo da dívida técnica em dinheiro, a ferramenta usa esforço para atingir um objetivo e o esforço para os indicadores globais como *proxies* para dívida técnica geral. Para transformar este valor em dinheiro, basta multiplicar pelo custo de desenvolvimento em horas. A

figura 4.29 mostra algumas visualizações da dívida técnica no Kiuwan.

O Kiuwan pode ser utilizado tanto localmente quanto em nuvem, com foco em segurança e privacidade. Ele possui um analisador local, que é uma aplicação onde é possível escolher o local do código-fonte, a aplicação que será analisada e um rótulos para a análise. Este aplicativo permite se conectar ao servidor e enviar os dados da análise para serem visualizados por todos. Além disso, é possível utilizar o próprio servidor na nuvem para realizar a análise para os aplicativos, apenas enviando um arquivo do tipo zip contendo o código-fonte.

O software permite criar diferentes modelos para a análise de diferentes aplicações, ajustando o modelo para medir o que se deseja medir. É possível, também, integrar automaticamente o Kiuwan para fazer análises continuamente durante todo o ciclo de vida de desenvolvimento do software.

A criação de portfólios permite agrupar as aplicações de uma forma a refletir o negócio e as estruturas do setor de tecnologia da informação. Os portfólios permitem criar diferentes perspectivas do software que podem ser utilizadas para a tomada de decisão por diferentes grupos ou pessoas.

A plataforma ainda facilita a colaboração entre todas as pessoas envolvidas no desenvolvimento do software. Ela cria um ambiente virtual colaborativo, fornecendo informação e ajudando a cada parte interessada a tomar decisões.

Além de toda informação analítica que o Kiuwan disponibiliza aos seus usuários, ele fornece meios de continuamente melhorar as aplicações analisadas. Com o software é possível criar planos de ação manualmente ou automaticamente baseados em objetivos ou disponibilidade de recursos, acompanhar o progresso de cada plano para atingir os objetivos de negócio. Este painel, como mostra a figura 4.30, permite escolher os itens de dívida técnica que serão pagos e realizar uma análise dos efeitos do plano de ação, como por exemplo, o gráfico de Kiviat de antes e depois de executar o plano de ações.

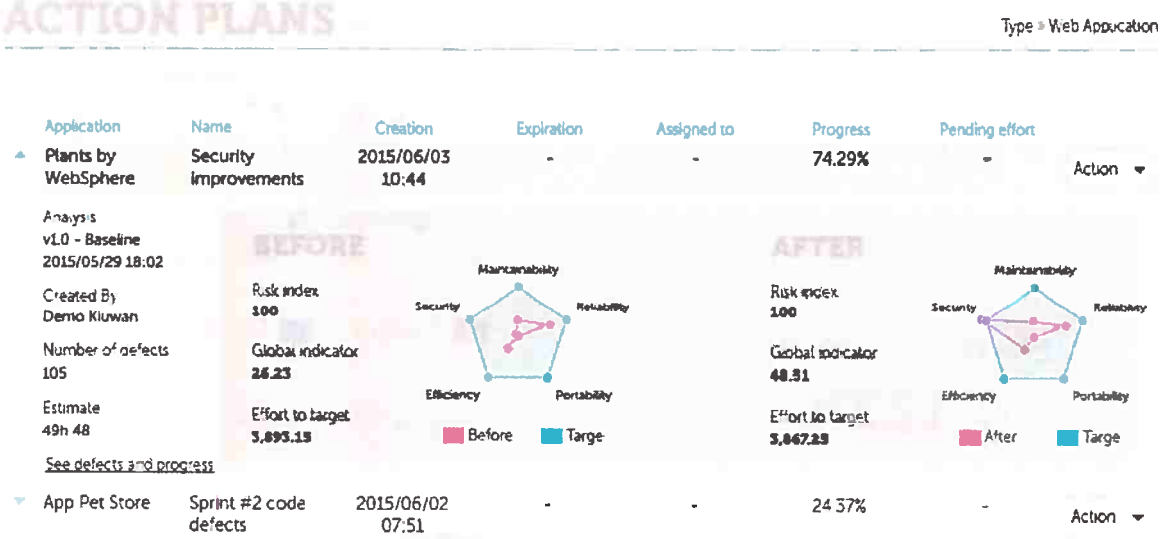

Figura 4.30: *Painel de plano de ações do Kiuwan.*

É possível testar o Kiuwan gratuitamente, porém ele é vendido em duas versões principais: profissional e empresarial. A versão profissional conta com suporte técnico, inspeção contínua, suporte a várias linguagens de programação, suporte a vários modelos de software, permite personalizar as regras e muitas outras funções. Já a versão empresarial, além das funcionalidades da versão profissional, ainda conta com suporte a multiusuário, gerenciamento de portfólio, governança e gerenciamento de entregas. Além dessas duas versões principais, o software vende a versão *One Shot*, válida por 1 mês, e a versão para desenvolvedores, que disponibiliza plugin para o eclipse e integra com outras ferramentas de desenvolvimento.

O software permite analisar diversas linguagens de programação, tais como: Java, Java Sript, PHP, Objective-C, C, C++, C#, Visual Basic, Cobol e Oracle PL/SQL. Ela ainda permite analisar

arcabouços como: Hibernate, Android, PHP Symphony e Angular JS. A ferramenta é utilizada por empresas como a Siemens, BBVA, Zurich e DHL.

4.4 SQuORE Technical Debt

O SQuORE é um software que possui um painel para tomada de decisões e para gerenciar a dívida técnica em produtos de software. Baseado na dívida técnica como principal indicador de desempenho, esta ferramenta é especialmente dedicada a melhorar a qualidade dos projetos de software analisados.

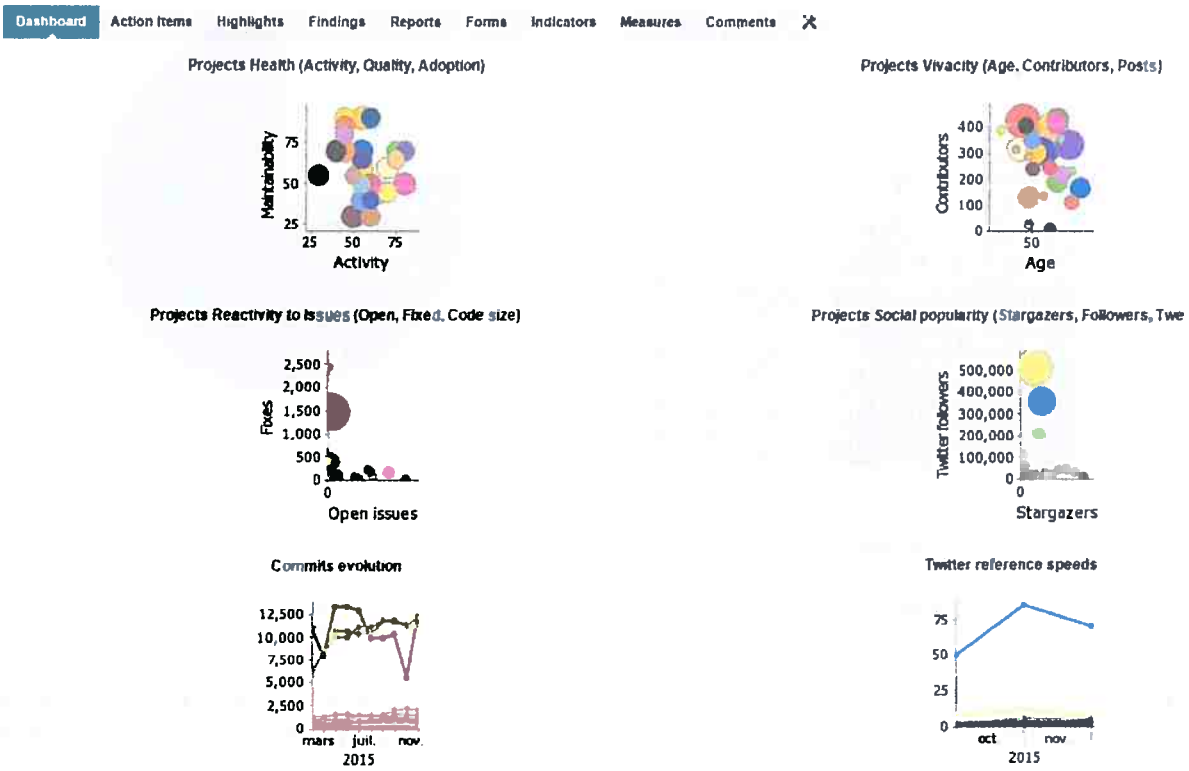

Figura 4.31: *Painel inicial do SQuORE.*

O software realiza monitoramento e controle da dívida técnica, ajudando a reduzir custos de manutenção e melhorando o potencial de desenvolvimento de novas funcionalidades pelos membros da equipe de desenvolvimento. Para gerenciar todo o processo em relação de dívida técnica o SQuORE utiliza a metodologia SQALE (Letouzey, 2012a). A figura 4.32 mostra algumas visualizações da dívida técnica fornecidas pelo SQuORE.

O SQuORE possui as seguintes funcionalidades:

- Pontos de controle padronizados que permitem personalizar os padrões para adaptar aos softwares analisados.
- Um modelo predefinido para analisar dívida técnica baseado em SQALE e outros métodos e integração com critérios de prioridade: risco, criticidade e valor de negócio.
- Visões gerais do progresso de desenvolvimento através de indicadores-chave de desempenho e análises de tendência: detecção imediata de regressões, desvios de planos.
- Análises profundas de componentes de risco podem ser visualizadas a níveis elementares, como funções e métodos.
- Comparação com outros projetos similares para um gerenciamento objetivo e efetivo do portfólio do projeto.

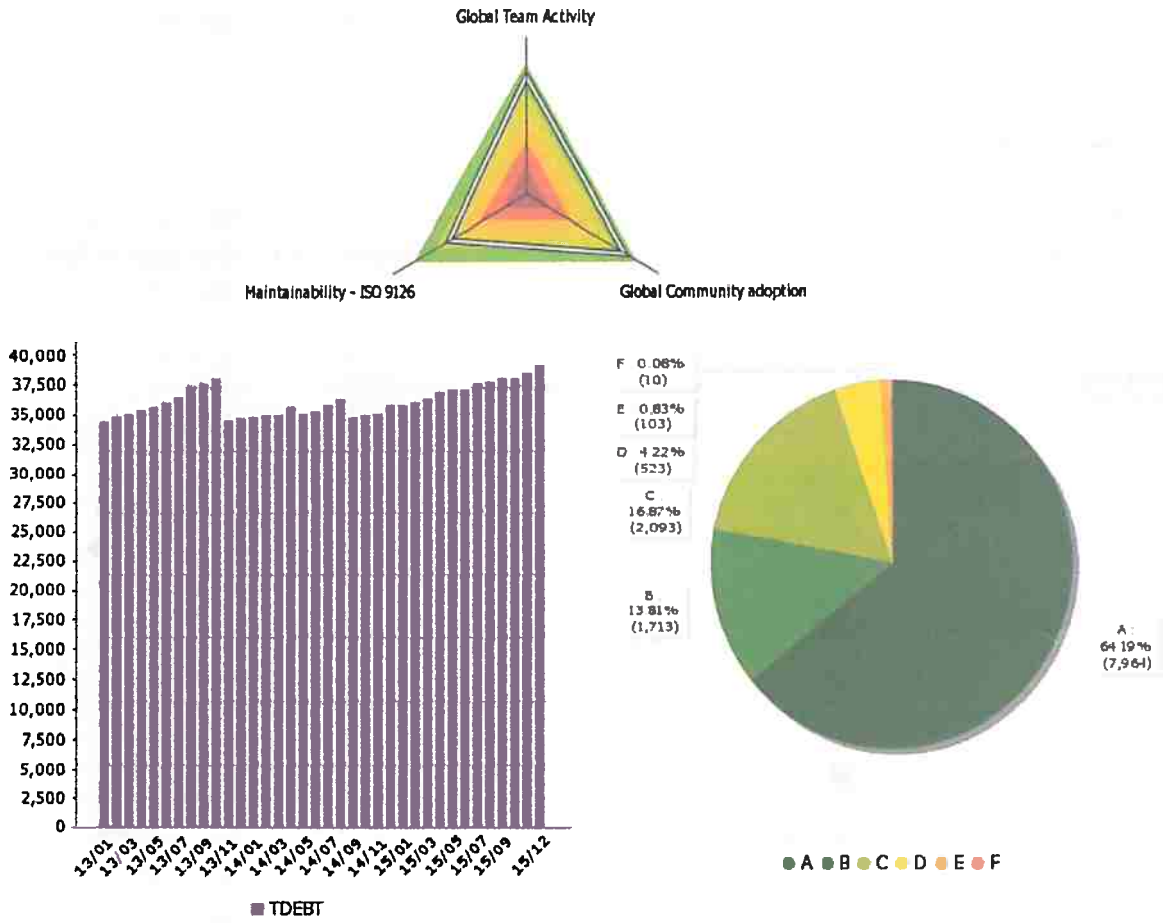

Figura 4.32: *Visualizações da dívida técnica no SQuORE.*

- Objetivos de colaboração aumentados pela centralização de todos os dados que não estão sob conformidade, alerta de notificação automatizado e compartilhamento de listas de afazeres.
- Geração automática e contínua de planos de ações para gerenciar e melhorar a dívida técnica.

A ferramenta pode ser usada em ambiente Windows e Linux, conseguindo realizar análise em diversas linguagens de programação como: Ada, C, C++, C#, Java, Cobol, PL/SQL, Python, Abap e PHP. Além disso, ela permite a importação de dados de outras ferramentas, tais como: SonarQube, Findbugs, CheckStyle, PMD, JUnit, FXCop, StyleCop, Klocwork, Understand, C/C++test, Coverity, Polyspace, Logiscope e JaCoCo. E, por fim, é possível integrar o SQuORE a diversas outras ferramentas, tais como: Eclipse, Jenkins, CruiseControl, ClearCase, Synergy, Git, Svn e MKS.

4.5 Mia-Quality

Mia-Quality é um software de análise de qualidade que permite analisar as principais linguagens de programação e ajusta a análise de acordo com necessidade do cliente. Após definir um modelo de qualidade que atenda as necessidades do problema, o usuário pode visualizar os resultados da verificação no site da plataforma Sonar Qube (Source, 2016) de forma intuitiva e ergonômica.

O software oferece indicadores macroscópicos para a visualização geral do software e, também, permite visualizar o aplicativo a nível de código-fonte, mostrando a aplicação detalhada dos indicadores.

A ferramenta ajuda a identificar defeitos no código-fonte para ajudar a reduzir a dívida técnica da aplicação. O modelo de dívida técnica utilizado pelo software é totalmente compatível com a norma ISO 9126, padrões de qualidade, e também implementa a metodologia SQALE (Letouzey , 2012a).

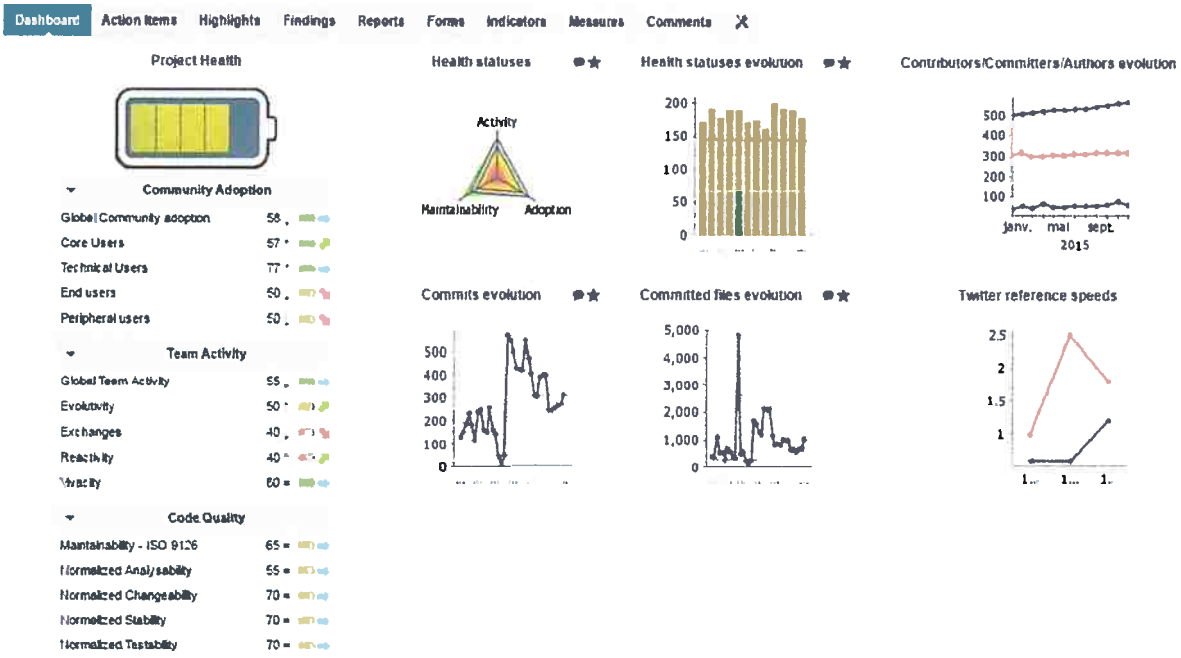

Figura 4.33: *Painel de análise do Jenkins no SQuORE.*

Figura 4.34: *Tela de abertura do Mia-Quality Studio.*

O Mia-Quality oferece como funcionalidades:

- Governança da qualidade real de aplicações legadas.
- Uma estimativa da dívida técnica.
- Permite elevar o nível de maturidade técnica da equipe de desenvolvimento.
- Suporte as linguagens de programação Java, C#.NET e COBOL.
- Facilidade de integração de novas linguagens e de novas medidas.
- Integração nativa com o Sonar Qube.
- Acompanhar diariamente a qualidade das atividades de desenvolvimento de software.
- Ajudar a priorizar as atividades de manutenção.

A ferramenta ainda permite medir a qualidade do software personalizável. O processo de análise de qualidade ocorre em três fases consecutivas que são:

- Definição de métricas e padrões de qualidade a serem cumpridas.
- Análise automática do código dos aplicativos.

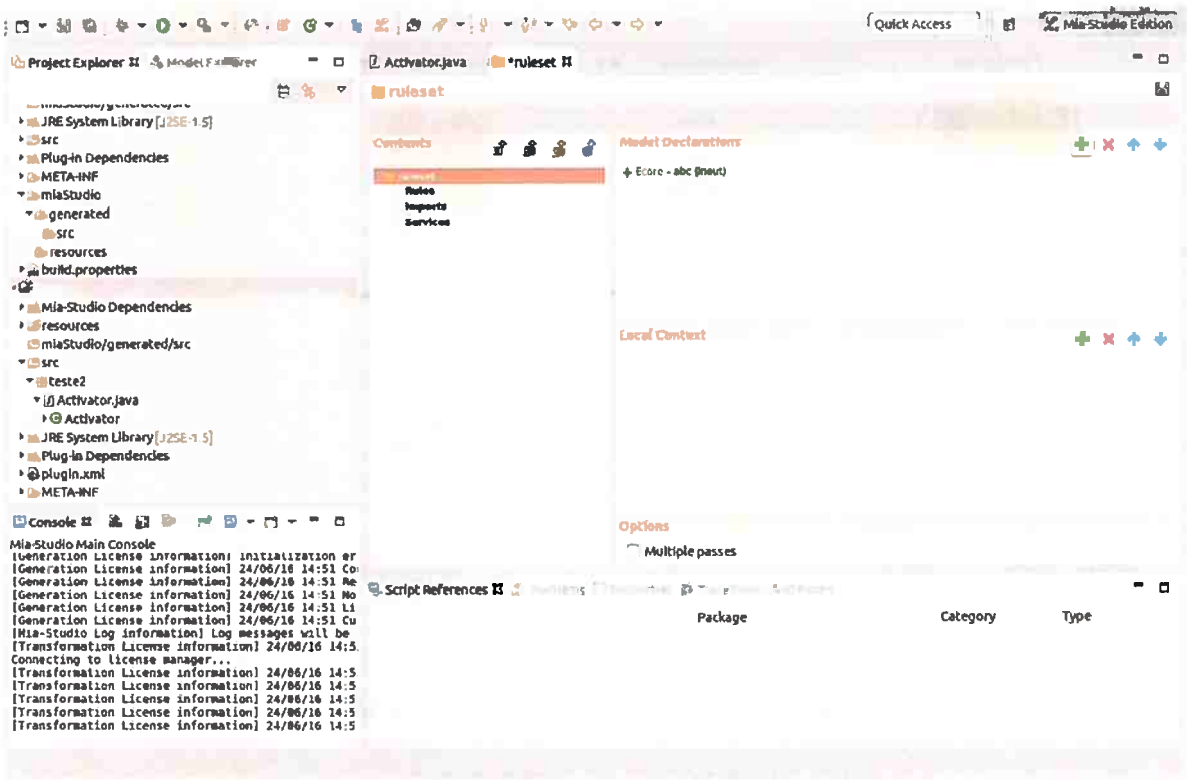

Figura 4.35: *Software Mia-Quality Studio baseado no Eclipse.*

- Visualização dos resultados e históricos.

As medidas de qualidade pode ser visualizadas a nível de código-fonte, mas também pode ser projetadas para fornecer informações de rastreamento, pedidos de manutenção e de incidentes de produção. Além disso, a análise está disponível em vários idiomas e ambientes. Este processo de análise é personalizável e escalável e baseado em modelos de qualidade padrão.

4.6 Quality Reviewer

Quality Reviewer é uma ferramenta que avalia regressivamente os software e entende as mudanças no código-fonte usando visualização automática, tais como: complexidade, métricas de tamanho, métricas estruturais, métricas de *Halstead*, manutenção da ISO 9126, ISO 25010, métricas de *Chidamber & Kemerer* e SQALE (Letouzey, 2012a). Além disso, ele estima esforço, tais como WMFP, FP, COCOMO e Revic, e possuí relatórios. Com isso, a ferramenta ajuda a manter a qualidade do código sob controle. A figura 4.36 mostra o painel principal do Quality Reviewer.

Em seu processo, o software coleta, analisa e visualiza a informação utilizando a metodologia SQALE que é facilmente compreendida e oferece um conhecimento da aplicação desenvolvida. Estas facilidades de comunicação estão presentes em todos os níveis, desde de os diretores de tecnologia até aos desenvolvedores da aplicação.

Quality Reviewer contribui para que gerentes possam entender os efeitos dos desenvolvedores sobre a qualidade do software e tomar decisões para manter a qualidade e produtividade. Neste sentido, a metodologia SQALE aumenta as funcionalidades de relatório disponibilizando uma visão geral da aplicação de forma que pessoas não técnicas possam entender facilmente. As pessoas responsáveis por tomar decisão podem ver evidencias de qualidade do software e o aumento da produtividade e pode, portanto, mais facilmente se convencer da importância de garantir a qualidade do software. Por outro lado, a equipe de desenvolvimento pode mostrar as outras partes envolvidas a qualidade que eles precisam atingir.

O gráfico de Kiviat, como mostra a figura 4.37, pode ajudar a mostrar onde os itens de dívida

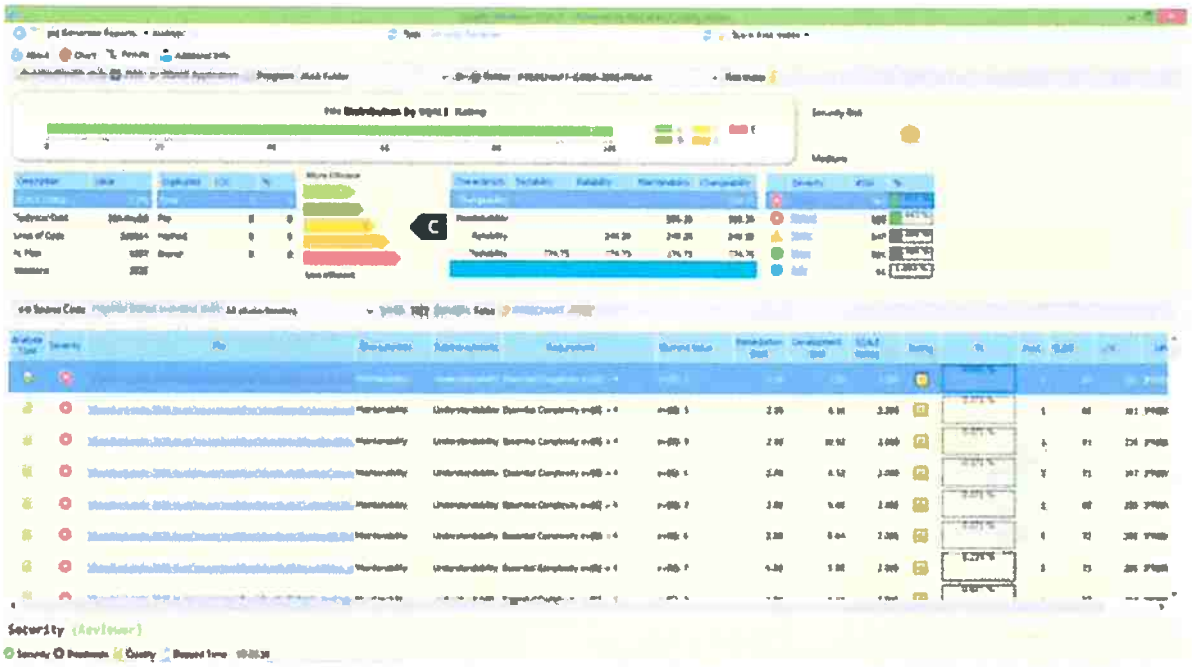

Figura 4.36: *Painel principal do Quality Reviewer. Retirado de http://www.securityreviewer.com/.*

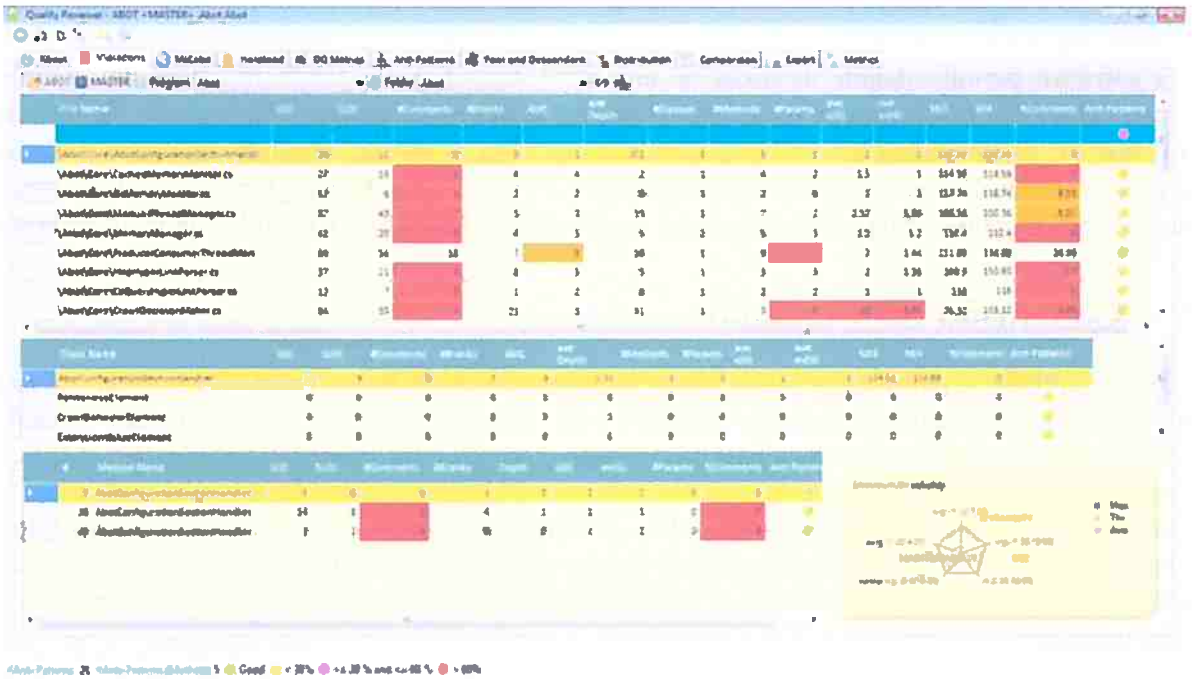

Figura 4.37: *Painel de métricas por arquivos do Quality Reviewer. Retirado de http://www.securityreviewer.com/.*

técnica estão principalmente localizados (manutenção, testes ou tamanho). Para cada arquivo de código-fonte, todas as classes relacionadas ou programas são listados com detecção de anti-padrões.

Com a ferramenta é possível criar anti-padrões personalizados baseados em consultas de métricas, usando gráficos para interpretar o impacto dos valores. As métricas baseadas em consultas disponibilizam um rápido acesso a elementos de interesse, salvar estas consultas podem servir de entrada para análises personalizadas. A firura 4.38 mostra o relatório dos anti-padrões.

O Quality Reviewer analisa uma grande quantidade de linguagens de programação que são: C#, Vb.NET, ASP, ASPX, JAVA, JSP, JavaScript, Ruby, Python, Groovy, Flex, ActionScript, HTML, XML, XPath, C, C++, PHP, Objective-C, COBOL, ABAP, PL/SQL, T/SQL, Teradata SQL, ANSI SQL, IBM DB2, IBM Informix, MySQL, FireBire, PostGreSQL, SQLite.

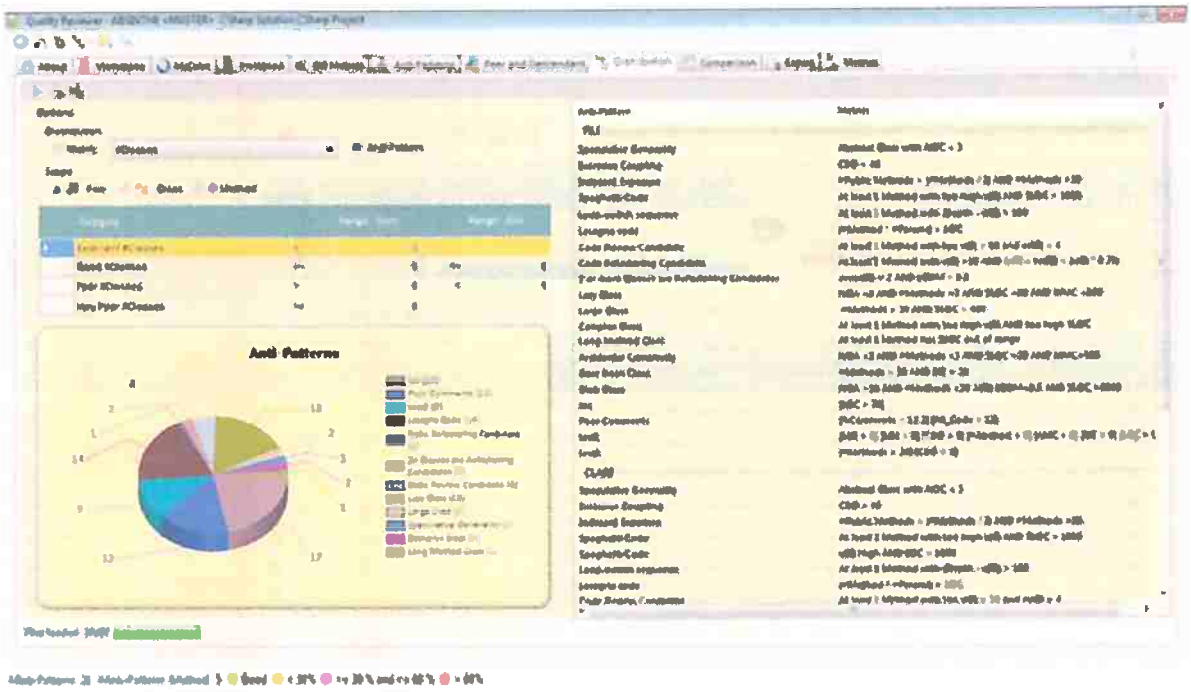

Figura 4.38: *Painel de defeitos Quality Reviewer. Retirado de http://www.securityreviewer.com/.*

O software permite definir métricas de alto-nível especificando arbitrariamente um conjunto de transformações matemáticas realizadas sobre as métricas primitiva. Um número de métricas primitivas são disponibilizadas por padrão, tais como: taxa de coesão, tamanho da classe, tamanho não ponderado de classe, métricas de especialização e reuso, taxa de complexidade lógica, taxa de complexidade da classe, fluxo de informação, complexidade estrutural e complexidade arquitetural.

4.7 Conclusão do Capítulo

Neste capítulo, a plataforma Sonar Qube foi apresentada de forma geral. Além disso, foi mostrado em detalhes como a plataforma realiza a identificação, medida e monitoramento da dívida técnica. Também as vantagens adicionais da extensão comercial *Technical Debt Evaluation* (SQALE) em relação à plataforma nativa.

O Sonar Qube é uma plataforma que lida de forma específica com processo de identificação, medida e monitoramento da dívida técnica porém de forma limitada. Mesmo como a extensão específica para tratar a dívida técnica, o Sonar ainda possui limitações, em especial, na medida da dívida técnica.

A ferramenta Cast AIP é outra ferramenta que cuida de todo o processo que envolve a dívida técnica, porém nem todas as violações de propriedades de qualidade são consideradas dívida técnica. Já o Kiuwan - *System Code* é uma ferramenta que está a bastante tempo no mercado e possui boas visualizações da dívida técnica.

Por sua vez, o *SQuORE Technical Debt* é a ferramenta que melhor implementa a metodologia SQALE, seguindo todas as sugestões do texto de definição. A ferramenta Mia-Quality é a mais limitada das ferramentas analisadas e é útil para ter uma visão geral da dívida técnica. Por fim o *Quality Reviewer* é uma ferramenta que implementa diversos conceitos da metodologia SQALE porém é uma ferramenta mais focada em qualidade de código, em geral, do que em dívida técnica.

No próximo estudo será apresentada uma revisão da literatura a fim de mostrar e classificar o estado da arte em relação a medida dívida técnica.

Capítulo 5

Estado da Arte

Neste capítulo é apresentado trabalhos relacionados a métodos de medida da dívida técnica agrupados conforme as técnicas e abordagens utilizadas, tais como abordagens abstratas, análise de métricas, violações de propriedades de qualidade, aprendizagem de máquia, ferramentas e pesquisa qualitativa.

5.1 Trabalhos Relacionados

Vários trabalhos já foram feitos para estudar formas de predizer a dívida técnica. Alguns são mais conceituais e abordam o tema de forma mais abstrata como em (Letouzey, 2012b), (Seaman e Guo , 2011), (Holvitie e Leppänen, 2013) e (Kruchten *et al.*, 2012). Outros utilizam analisam métricas de código para definir o valor da dívida técnica como em (S. Chin e Gat, 2010), (Nugroho *et al.*, 2011) e (Marinescu, 2012). Também são usadas a análise de violações de propriedades de qualidade como os artigos escritos por Curtis e outros (Curtis *et al.*, 2012a). Outra abordagem utilizada é a de aprendizagem de máquina como em (Akbarinasaji *et al.*, 2016). Outros artigos focam mais em ferramentas como em (Tomas *et al.*, 2013), (Falessi e Reichel, 2015) e (Ramasubbu e Kemerer, 2013). Por fim, ainda existe uma pesquisa qualitativa com o objetivo de entender o que a indústria pensa sobre o custo da dívida técnica (Ernst *et al.*, 2015).

5.1.1 Métodos de Medida Conceituais

O artigo *The SQALE method for evaluating technical debt* (Letouzey, 2012b), como visto na seção 2.6, apresenta a metodologia e modelo de qualidade SQALE para avaliar a dívida técnica de projetos de software. O modelo divide a dívida técnica em características e subcaracterísticas, que ajudam na organização e na visualização da dívida técnica. Neste modelo a dívida técnica é definida como violação de propriedade de qualidade e o seu custo é dado pela soma do custo de pagar todos os itens de dívida técnica. Para calcular o custo de pagar um item de dívida técnica, a metodologia apresenta as funções de remediação de custo associadas a cada propriedade de qualidade, que permite calcular o custo de pagar um item de dívida técnica da propriedade referida. Este trabalho cuida da descrição da metodologia e do modelo, logo, ele não se preocupa em definir as funções de remediação de custo em termos matemáticos. A metodologia SQALE, também só trata do custo principal da dívida técnica e não do juros.

Outro artigo que faz definição de modelo e descrições abstratas é o *Measuring and Monitoring Technical Debt* (Seaman e Guo, 2011). Neste artigo é proposto um arcabouço para gerenciar a dívida técnica. Este arcabouço é centrado em uma listas de itens de dívida técnica, definindo um modelo de dados para armazenar cada um destes itens. A medida da dívida técnica é feita através da estimativa de três parâmetros: o principal, quantidade de juros e a probabilidade de juros. Num primeiro momento estes parâmetros são definidos em uma escala que consiste de: baixo, médio e alto. Conforme os itens forem melhor compreendidos, a escala pode ser substituída por valores numéricos. Logo, uma estimativa mais refinada para a quantidade de juros poderia ser:

$$QuantidadedeJuros = W * C$$

onde C é o custo médio das últumas N modificações, baseado no histórico de dados e W é a probabilidade do juros ser cobrado.

Outro artigo que aborda o mesmo tema do artigo acima é o *Exploring the costs of technical debt management–a case study* (Guo *et al.*, 2016). Porém, neste artigo a ênfase é em um estudo de caso que valida o arcabouço proposto em (Seaman e Guo, 2011). Por ser um artigo com 12 páginas, os autores tiveram espaço para detalhar melhor o arcabouço e, na parte do estudo de caso, exemplificar com mais detalhes a estimativa da dívida técnica.

Seguindo a mesma abordagem, o artigo *DebtFlag: Technical debt management with a development environment integrated tool* (Holvitie e Leppänen, 2013) faz extensões ao arcabouço proposto em (Seaman e Guo, 2011). A principal extensão está relacionada ao armazenamento do item de dívida técnica, neste artigo é proposto que a dívida técnica seja armazenada como um conjunto de DebtFlag's com estrutura similar a do arcabouço, porém permitindo que um item de dívida técnica possa conter mais de uma violação de regra de propriedade de qualidade.

Ainda nesta abordagem abstrata de dívida técnica, o artigo *Technical Debt: From Metaphor to Theory and Practice* (Kruchten *et al.*, 2012) discute desde a definição básica de dívida técnica até maneiras de organizar os tipos de dívida técnica, possíveis causas e o processo tomada de decisão, avaliando o custo e valor da dívida técnica sob diferentes pontos de vista.

5.1.2 Métodos de Medida por Métricas de Código

O artigo *The Economics of technical debt* (S. Chin e Gat, 2010) dividiu a dívida técnica entre desenvolvimento e manutenção:

$$TD_{desenvolvimento} = \sum_{n=0}^{a-1} RI * (1 + CI)^n$$

$$TD_{manuteno} = RI * (1 + CI)^a * n$$

onde RI (*Recurring Interest*) representa a quantidade de dinheiro gasto para pagar a dívida técnica, CI (*Compounding Interest*) representa o custo adicional por não pagar a dívida técnica sobre o tempo, a e m representam o número de anos de desenvolvimento e manutenção, respectivamente.

Nesta definição de dívida técnica, os juros podem ser vistos como o custo de manutenção, logo o sistema sempre deve pagar dívida técnica. Isto pode ser um equívoco, pois se o sistema estiver com a qualidade satisfatória a dívida técnica é 0, portanto não deveria pagar juros, porém está definição vale para a maioria dos projetos do mundo real.

Já no artigo *An empirical model of technical debt and interest* (Nugroho *et al.*, 2011) a dívida técnica é estimada como Esforço de Reparação (ER), que é dado pela seguinte fórmula:

$$ER = FR * VR$$

onde FR é a Fração de Retrabalho, ou seja, uma estimativa da porcentagem de linhas de código que precisão ser alteradas para melhorar a qualidade do código para um nível acima; e VR é Valor de Reconstrução que é a estimativa do esforço, em meses de trabalho, que precisa ser gasto para reconstruir um sistema com uma certa tecnologia.

O valor de reconstrução é dado por

$$VR = TS * FT$$

onde TS é o Tamanho do Sistema em linhas de código e FT é o Fator Tecnologia que representa o fator de produtividade da linguagem de programação.

O valor da dívida técnica ainda pode ser refinado multiplicando o Esforço de Reparação por por um Ajuste de Refatoração (AR), tentando ajustar a estimativa aos aspectos do contexto do

projeto.

O artigo *Assessing technical debt by identifying design flaws in software systems* (Marinescu , 2012) também utiliza a abordagem de análise de métricas de código, porém associada a falhas de projeto. Ele define Influência (I) como o fator que expressa o quanto uma falha de projeto afeta os critérios de bom projeto, define Granularidade (G) como o peso para cada falha de acordo com o tipo de entidade do projeto e Severidade (S) como a criticidade numa escala de 1 a 10. Baseado nestes três fatores, é calculado o Índice de Impacto da Falha (IIF), dado por

$$IIF = I * G * S.$$

Com os valores de IIF para todos os tipos de falha, é possível calcular a dívida técnica de projeto chamada de Índice de Sintomas de Dívida (ISD) que é dado por

$$IDS = \frac{\sum_{todasasfalhas} IIF}{KLOC}.$$

5.1.3 Métodos de Medida por Violações de Qualidade

O artigo *Estimating the principal of an application's technical debt* (Curtis *et al.*, 2012a) mede o custo de pagar a dívida técnica utilizando três parâmetros: o número de violações detectadas na aplicação, o número de horas para consertar cada violação e o custo da hora de trabalho. Para calcular o principal da dívida técnica, basta estimar cada um dos parâmetros e utilizar a seguinte equação

$$\begin{aligned} DT - principal = ((\sum violaesderiscoalto) * (percentagemaserconsertada) * \\ (mdiadehorasnecessriasparaconsertar) * (\$porhora)) + \\ (\sum violaesderiscomdio) * (percentagemaserconsertada) * \\ (mdiadehorasnecessriasparaconsertar) * (\$porhora)) + \\ (\sum violaesderiscobaixo) * (percentagemaserconsertada) * \\ (mdiadehorasnecessriasparaconsertar) * (\$porhora)). \end{aligned}$$

Como estimativa dos parâmetros podemos utilizar os valores da tabela 5.1.

Tabela 5.1: *Valores dos parâmetros para três estimativas de dívida técnica.*

Variáveis	Estimativa 1	Estimativa 2	Estimativa 3
Violações que devem ser consertadas			
Violações de alto risco	50%	100%	100%
Violações de médio risco	25%	50%	
Violações de baixo risco	10%		
Horas para consertar			
Violações de alto risco	1 hora	2,5 horas	10% - 1 hora 20% - 2 horas 40% - 4 horas 15% - 6 horas 10% - 8 horas 5% - 16 horas
Violações de médio risco	1 hora	1 hora	
Violações de baixo risco	1 hora		
$ por hora			
Todas as violações	75	75	75

Já o artigo *Estimating the size, cost, and types of Technical Debt* (Curtis *et al.*, 2012b) é um resumo do artigo (Curtis *et al.*, 2012a) escrito pelos mesmos autores e o artigo *Paying Down the Interest on Your Applications* (Curtis, 2012) é escrito apenas pelo Dr. Bill Curtis e possui mais detalhes e exemplos do que no artigo (Curtis *et al.*, 2012a). Por sua vez, o artigo *The correspondence between software quality models and technical debt estimation approaches* (Griffith *et al.*, 2014) utiliza a método proposto por (Curtis *et al.*, 2012b) para calcular a dívida técnica e utiliza os valores encontrados para verificar a correspondência com modelos de qualidade de software.

5.1.4 Métodos de Medida por Aprendizagem de Máquina

Em uma abordagem de aprendizagem de máquina, o artigo *Measuring the principal of defect debt* (Akbarinasaji *et al.*, 2016) categoriza os *bugs* em regulares e propensos e utiliza os dados do histórico dos regulares para treinar um modelo preditor para calcular a dívida técnica principal nos *bugs* propensos. Além disso, é proposto que o juros são proporcionais à variância entre o tempo real para consertar *bug* e o tempo estimado. O método utilizado para fazer a predição foi o KNN, que retorna uma média dos valores dos k-vizinhos mais próximos.

Ainda nesta abordagem de analisar o histórico de *bugs* o artigo *Predicting effort to fix software bugs* (Weiß *et al.*, 2006) também usa o método KNN para predizer o esforço de consertar *bugs*, neste caso os *bugs* são separados entre novos e consertados. Já o artigo *Bug-fix time prediction models: can we do better?* (Bhattacharya e Neamtiu, 2011) faz uma análise estatística do histórico de *bugs* utilizando regressão simples e multivariada para verificar se reputação dos desenvolvedores está associado a complexidade de consertá-los.

5.1.5 Métodos de Medida por Ferramentas

Já em relação a ferramentas que analisam dívida técnica, o artigo *Practical considerations, challenges, and requirements of tool-support for managing technical debt* (Falessi *et al.*, 2013) indica 12 tipos de problemas com as ferramentas que medem dívida técnica e propõe 10 requisitos para o desenvolvimento de ferramentas que oferecem suporte ao gerenciamento da dívida técnica.

O artigo *Open source tools for measuring the Internal Quality of Java software products. A survey* (Tomas *et al.*, 2013) realizou uma pesquisa para descobrir ferramentas que medem a qualidade interna de produtos de software em Java, após listar ferramentas como o jCosmo, QALab, FindBugs, Xradar, Sonar, squale e outros, foi feita uma descrição e uma análise descritiva breve de cada uma das ferramentas.

No artigo *Towards an open-source tool for measuring and visualizing the interest of technical debt* (Falessi e Reichel, 2015) os autores analisam as características da ferramenta MIND, desenvolvida por eles. Esta ferramenta analisa o histórico de versões do código-fonte e no gerenciador de histórias para conseguir estimar, através de regressão linear, a tendência de defeitos de uma classe e, com isso, determinar a possibilidade de uma dada classe em um dada versão ter pago juros.

Por fim, o artigo *Towards a model for optimizing technical debt in software products* (Ramasubbu e Kemerer , 2013) propõe um modelo para otimizar a dívida técnica em produtos de software. Esta otimização é feita através da curva entre a quantidade de funções adicionadas (valor de negócio) e tempo de crescimento do produto, identificando os pontos onde a dívida técnica pode ser mantida em níveis mais altos e outros em níveis mais baixos, otimizando o tempo da equipe de desenvolvimento e manutenção.

5.1.6 Métodos de Medida por Pesquisa Qualitativa

Com uma abordagem de pesquisa qualitativa o artigo *Measure it? Manage it? Ignore it? Software practitioners and technical debt* (Ernst *et al.*, 2015) realiza uma pesquisa primária com 1.831 engenheiros e arquitetos e software legados. Nesta pesquisa for constatado que a dívida técnica é uma metáfora útil e comumente entendida num nível abstrato para transmitir urgência sobre custos de software acumulado. Foi constatado também que questões arquiteturais são as maiores fontes

de dívida técnica e que é difícil de tratar e, por isso, é importante monitorá-la e rastreá-la. Por fim foi constatado que os profissionais acham que as ferramentas não capturam as principais áreas de acúmulos de problemas de dívida técnica e não demonstram o valor de gerenciá-la.

5.2 Conclusão do Capítulo

Neste capítulo foram apresentados trabalhos relacionados que propõe metologias e arcabouços ((Letouzey, 2012b), (Seaman e Guo, 2011), (Holvitie e Leppänen, 2013)), análise de métricas de código ((S. Chin e Gat, 2010), (Nugroho *et al.*, 2011), (Curtis *et al.*, 2012a)), análises de violações de qualidade((Akbarinasaji *et al.*, 2016), (Weiß *et al.*, 2006)), ferramentas ((Tomas *et al.*, 2013), (Ramasubbu e Kemerer, 2013)) e questões gerais ((Ernst *et al.*, 2015)) referentes a medida da dívida técnica.

No próximo capítulo será apresentado uma proposta de quatro métodos de medida de dívida técnica que, se utilizados conjuntamente, podem tornar o cálculo do custo de pagar cada item de dívida técnica mais flexível, preciso e contextualizável. Além disso, é mostrada a proposta de uma ferramenta que implementa os quatro novos métodos de medida.

Capítulo 6

Métodos e Ferramenta Propostos

Este capítulo apresenta uma série de limitações relacionadas com métodos de medida de dívida técnica, tais como, complexidade, qualidade de código-fonte, capacidade técnica e ferramentas.

Por fim, para suprir as limitações expostas, serão propostos quatro métodos para medir a dívida técnica que foram implementados para funcionarem de forma complementar na ferramenta Technical Debt Analyser (TDA). Estes métodos são: padrão do Sonar Qube, regressão estatística, linguagem de remediação de custo e classe de remediação de custo. Questões de implementação da ferramenta TDA serão discutidas na última seção deste capítulo.

6.1 Limitações dos Métodos de Medida da Dívida Técnica

Uma das grandes limitações do Sonar Qube 5.1 é a medida da dívida técnica, neste caso, o cálculo para saber quanto tempo é necessário para pagar um item de dívida técnica. A principal causa dessa limitação são as funções de remediação de custo,as quais são constantes, lineares e lineares com um parâmetro, ou seja podem ser representadas por uma função f onde $f(x) = Ax+B$. Esta linearidade pode fazer com que a função não modele corretamente o custo de pagar a dívida técnica, pois alguns custos podem ter modelagens exponenciais, logarítmicas, trigonométrica ou qualquer forma diferente da linear. Além disso, o fato de ter apenas uma variável pode fazer com que o contexto do software não seja refletido no custo da dívida técnica.

Segundo o Sonar Source (Source), foi seu time de programadores experientes que definiram, para cada linguagem de programação, as funções de remediações de custo para padrões muito razoáveis. Entretanto, mesmo sendo programadores experientes é fácil ver que estas funções podem variar bastante de software para software e, assim, gerar erros de medida, pelos seguintes fatores:

- **Complexidade do projeto:** Pagar uma dívida técnica de um determinado tipo pode ser mais, ou menos, custoso dependendo da complexidade do projeto. Além disso, mesmo que o código-fonte esteja com uma qualidade satisfatória um sistema complexo tende a ser mais difícil de ser lido e compreendido. Por exemplo, para pagar a dívida técnica de teste pode levar um tempo menor em um sistema de gerenciamento de usuários do que em um sistema de gerenciamento e planejamento de empresas.

- **Qualidade do código-fonte:** A qualidade do código-fonte está diretamente ligada ao custo de pagar a dívida técnica. Isso ocorre pois se o código está com má qualidade ele fica difícil de ser lido,o que pode acarretar em custos muito altos para pagamentos de dívidas técnicas que, em geral, possuíam um custo baixo.

- **Capacidade técnica dos desenvolvedores:** As funções de remediação de custo do Sonar Qube foram criadas baseados em uma média, porém equipes de desenvolvimento diferentes possuem capacidades técnicas diferentes, ou seja, uma equipe menos capacitada pode ter um custo maior para pagar a dívida técnica, enquanto equipes mais capacitadas podem ter um custo menor do que a média para realizar o mesmo pagamento.

- **Influência externa:** Fatores externos ao projeto podem influenciar diretamente no custo da dívida técnica, pois eles podem ditar o ritmo e a qualidade do desenvolvimento. Por exemplo, quando as equipes de desenvolvimento são pressionadas para realizar entregas em prazos menores do que os necessários para manter a qualidade. Posteriormente, o custo de pagar a dívida técnica pode ser maior do que se tivesse feito o projeto dentro dos prazos. Por outro lado, um consultor externo a equipe de desenvolvimento pode dar suporte para que o custo da dívida técnica possa se tornar mais baixo.
- **Ferramentas de desenvolvimento:** As ferramentas de desenvolvimentos influenciam diretamente no custo da dívida técnica, pois é através delas que a dívida técnica é paga. Por exemplo, uma ferramenta que fornece apoio a refatoração pode diminuir o custo de alguns tipos de dívida técnica em relação a outras ferramentas que não possuem este suporte.

A junção dos itens acima pode fazer com que o custo da dívida técnica varie de diferentes formas, tanto para mais quanto para menos. Ou seja, o custo da dívida técnica pode variar dependendo do contexto em que o software é desenvolvido e utilizado, sofrendo assim a influência de fatores internos e externos ao projeto.

6.2 Proposta de Métodos de Medida da Dívida Técnica

Para tornar mais preciso, flexível e contextualizável o cálculo do custo de pagar um item de dívida técnica, para cada regra de conformidade pode-se associar quatro tipos diferentes de métodos para calcular o custo da dívida técnica que completam ou substituem a função de remediação de custo. Estes métodos são: Padrão Sonar Qube, Regressão Estatística, Linguagem de Remediação de Custo e Classe de Modelagem.

Para implementar e analisar estes métodos a ferramenta Technical Debt Analyser (TDA) foi implementada. Esta é uma ferramenta integrada a plataforma Sonar Qube e permite visualizar e alterar, completar os itens de dívida técnica com informações como, por exemplo, criticidade, tempo gasto para o pagamento do item e resolução.

6.2.1 Padrão do Sonar Qube

Para cada *issue* o Sonar Qube, com ou sem extensão, calcula o custo em minutos de corrigir tal defeito. Este custo é chamado de dívida técnica. Cada *issue* está associada a uma regra de conformidade. Para cada uma das regras é associada previamente uma função de remediação de custo que é utilizada para calcular a dívida técnica das *issues* associadas.

Apesar deste cálculo ser limitado, pode ser interessante armazenar esta medida de dívida técnica, pois tendo a função de remediação de custo associada a regra de conformidade, o custo do item de dívida técnica será sempre calculado. Assim, é possível ter um custo para o item de dívida técnica, mesmo não seja possível calcular a dívida técnica por algum dos outros três métodos.

Este método pode ser classificado como análise de violações de propriedades de qualidade, já que para cada regra/propriedade de qualidade é criado um item de dívida técnica e o custo total é dado pela soma de todos esses itens.

6.2.2 Regressão Estatística

Como em sua origem a dívida técnica foi definida em termos utilizados pela economia, pode-se utilizar a econometria para poder medi-la. Segundo (Hoffmann e Vieira, 1977), a análise de regressão é o método mais importante para econometria, pois é interessante conhecer os efeitos que algumas variáveis exercem, ou parecem exercer, sobre outras. Ainda que as variáveis não exerçam efeitos sobre as outras, é possível utilizar uma expressão matemática para relacioná-las. Esta expressão pode ser útil para estimar o valor de uma das variáveis quando conhecemos os valores das outras.

A regressão é uma abordagem para aprendizagem supervisionada que utiliza a relação entre duas ou mais variáveis que permita predizer uma variável a partir de outra(s). Este método pode

ser classificado com aprendizagem de máquina já que regressão é uma abordagem muito utilizada pela área.

As funções de regressão podem ser descritas por

$$Y = f(X_1, X_2, ...X_n)$$

onde Y é a variável dependente e X_n, para $n = 1, 2, ..., k$, são as variáveis independentes ou preditores.

Um dos modelos de regressão mais conhecidos é o modelo de regressão linear, nele a função de regressão é descrita pela seguinte função:

$$Y = a_{00} + \sum_{i=1}^{n} a_{ij}.$$

Por ser um dos modelos que está bem definido, ele possui diversos métodos para encontrar uma solução, ou seja, encontrar os coeficientes a_{ij}. Uma abordagem simples de ver o problema é utilizando matriz, como apresentado em (Draper *et al.*, 1966). Draper faz a transformação do problema para a notação matricial e apresenta um método para resolver o sistema linear.

Porém, o modelo de regressão linear pode não expressar corretamente a relação devido a sua forma linear. Para resolver este problema, é possível utilizar o modelo de regressão polinomial com uma variável dependente, que é um modelo não estritamente linear, representado por

$$Y = a_{00} + \sum_{i=1}^{n} \sum_{j=1}^{m} (a_{ij}) * (X_i)^j.$$

Segundo (Freund e Minton, 1979), este tipo de regressão é atraente, pois é fácil de implementar o seu cálculo e também é possível aproximá-la a quase todas as funções, das quais a exata forma é desconhecida, pela expansão da série de Taylor, que é uma forma de função polinomial.

Porém, algumas restrições e problemas devem ser levados em consideração no cálculo da regressão polinomial utilizando computador:

- Devido aos expoentes, o cálculo da regressão pode extrapolar os limites da representação numérica feita pelo computador. Ou seja, para um dado expoente n, o valor de x^n pode ser tão grande, ou tão pequeno, que não possa ser representado pelo computador, gerando um erro na computação da regressão.
- A aproximação de outras funções feitas pelos polinômios são válidas apenas na região dos dados, assim, a extrapolação pode não ser ideal.

A ferramenta TDA utiliza o modelo de regressão polinomial para estimar o custo da dívida técnica. Para tanto, a variável Y representa a dívida técnica e $X_1, X_2, ..., X_n$, n representam as variáveis de contexto da dívida técnica e m é o grau máximo do polinômio.

Para resolver os problemas e limitações, primeiro, a ferramenta utiliza um número baixo para o grau máximo do polinômio, fazendo com que todos os valores possam ser computados sem erros, porém essa restrição pode fazer com que a função não se aproxime tão bem da função verdadeira que modela o custo da dívida técnica. Segundo, apesar de em alguns casos a ferramenta fazer uso de extrapolação por falta de dados, conforme ela vai sendo utilizada, mais dados são coletados e menos extrapolação são realizadas, isso acontece pois a ferramenta utiliza todas as instâncias possíveis da dívida técnica para calcular a regressão polinomial.

A TDA utiliza o software estatístico R para calcular de fato a regressão polinomial. Dados os valores conhecidos da variável dependente e das variáveis independentes e a forma de um polinômio que relaciona estas variáveis, o R calcula os coeficientes a_{ij} deste polinômio, resultando assim em um polinômio que modela um dado tipo de dívida técnica.

Para cada item de dívida técnica, a ferramenta TDA lista todas as dívidas técnicas que estão relacionadas a mesma regra de conformidade e já foram pagas. Se esta lista não for suficientemente

grande, no caso, não possuir pelo menos 5 itens de dívida técnica a ferramenta não calcula a regressão, pois com poucos dados a chance da regressão estatística refletir a realidade é baixa. Caso a lista tenha mais de 5 itens, a ferramenta gera um código em R com os dados dos itens de dívida técnica pagos para encontrar a função de regressão polinomial e por fim, com os dados de contexto do item desejado, estimar o custo da dívida técnica.

Segue abaixo, um exemplo de um código R gerado pela ferramenta para calcular a regressão linear, encontrando assim o polinômio:

```
1 X1 = c(1, 2, 3, 4, 5)
2 X2 = c(2, 3, 3.6, 4, 5)
3 X3 = c(1.1, 2.2, 2.8, 4.4, 5)
4 X4 = c(0.8, 2, 3.2, 4.3, 5)
5 X5 = c(0.9, 1.9, 2.9, 3.9, 6.5)
6 X6 = c(1.1, 2, 3.2, 4, 5)
7 X7 = c(1, 1.7, 3, 4.3, 5)
8 X8 = c(1, 2.2, 3, 4, 5.1)
9 X9 = c(1.1, 1.9, 3.1, 4.1, 5)
10
11 Y = c(3, 3.4, 3.4, 3.3, 3.2)
12
13 mydata = data.frame(Y, X1, X2, X3, X4, X5, X6, X7, X8, X9)
14
15 fit = lm(formula = Y ~ I(X1 ^ 1) + I(X2 ^ 1) + I(X3 ^ 1) + I(X4 ^ 1) + I(X5 ^ 1)
       + I(X6 ^ 1) + I(X7 ^ 1) + I(X8 ^ 1) + I(X9 ^ 1) +
16     I(X1 ^ 2) + I(X2 ^ 2) + I(X3 ^ 2) + I(X4 ^ 2) + I(X5 ^ 2) + I(X6 ^ 2) + I(X7
           ^ 2) + I(X8 ^ 2) + I(X9 ^ 2), data=mydata)
17
18
19 fit
20
21 #Call:
22 lm(formula = Y ~ I(X1^1) + I(X2^1) + I(X3^1) + I(X4^1) + I(X5^1) +
23     I(X6^1) + I(X7^1) + I(X8^1) + I(X9^1) + I(X1^2) + I(X2^2) +
24     I(X3^2) + I(X4^2) + I(X5^2) + I(X6^2) + I(X7^2) + I(X8^2) +
25     I(X9^2), data = mydata)
26
27 #Coefficients:
28 (Intercept)      I(X1^1)      I(X2^1)      I(X3^1)      I(X4^1)      I(X5^1)
29      2.1259      -1.6000       0.8148       0.1481       0.8519           NA
30     I(X6^1)      I(X7^1)      I(X8^1)      I(X9^1)      I(X1^2)      I(X2^2)
31          NA           NA           NA           NA           NA           NA
32     I(X3^2)      I(X4^2)      I(X5^2)      I(X6^2)      I(X7^2)      I(X8^2)
33          NA           NA           NA           NA           NA           NA
34     I(X9^2)
35          NA
```

Na linha 1 e 2 são definidas as variáveis independentes, que são as variáveis de contexto. Na linha 4 é definida a variável dependente que é a variável de dívida técnica. Na linha 6 é criado um *data frame* necessário para a função lm. Na linha 8 a regressão polinomial é calculada através da função lm, que precisa da forma do polinômio e de um *data frame* para encontrar os coeficientes deste polinômio. Por fim, é exibido o conteúdo da variável que representa a regressão polinomial.

Segue abaixo a continuação do código R gerado pela ferramenta TDA para estimar o valor da dívida técnica:

```
1 degree = 2
2 numVars = 2
3 params = c(14, 1)
4
5 result = coef(fit[1])
6 for (i in 1:degree) {
7   for (j in 1:numVars) {
8     pos = (i-1) * numVars + j + 1
9     c = coef(fit)[pos]
```

```
10
11      if (is.na(c))
12        c = 0
13
14      result = result + c * params[j] ^ i
15    }
16 }
```

Na linha 1, 2, 3 são definidas variáveis que indicam o grau o polinômio gerado, o número de variáveis independentes e um vetor com os dados de contexto (valor das variáveis dependentes) do item de dívida técnica a ser calculado. Por fim, o código realiza o cálculo aplicando cada um dos coeficientes obtidos pela regressão polinomial ao seu respectivo X_i, inclusive contabiliza o coeficiente a_{00}.

6.2.3 Linguagem de Remediação de Custo

A linguagem de remediação de custo foi projetada para que a equipe de desenvolvimento que utiliza a ferramenta TDA possa, de maneira simples, modelar o custo de pagar a dívida técnica. A ferramenta permite que, para cada regra de conformidade, o usuário possa escrever um modelo matemático na linguagem de remediação de custo que calculará o custo de pagar itens de dívida técnica associados aquela regra de conformidade.

Este método pode ser classificado como um método conceito, pois ele não mede o custo de pagar a dívida técnica em si, mas sim prove uma forma abstrata para modelar o custo do pagamento para um dado tipo de dívida técnica.

Para escrever a linguagem de remediação de custos foi utilizado Python 3 e a biblioteca PLY (Python Lex-Yacc) (Beazley, 2016). O PLY é uma implementação do lex e do yacc para Python, o que permite escrever de forma simples uma nova linguagem de programação.

A descrição da linguagem na notação *Backus Normal Form* (BNF) é:

- <Afirmação> ::= <Expressão> | <Atribuição>
- <Atribuição> ::= Palavra = <Expressão>
- <Expressão> ::= <Expressão> + <Expressão>
- <Expressão> ::= <Expressão> - <Expressão>
- <Expressão> ::= <Expressão> * <Expressão>
- <Expressão> ::= <Expressão> / <Expressão>
- <Expressão> ::= (<Expressão>)
- <Expressão> ::= - <Expressão>
- <Expressão> ::= Palavra(<Expressão>)
- <Expressão> ::= Palavra(<Expressão>, <Expressão>)
- <Expressão> ::= Palavra
- <Expressão> ::= Número

Observação 1: Palavra é uma cadeia de carácter iniciada por uma letra seguida, ou não, de letras e números. Alguns exemplos são: palavra, Palavra, p10 e p.

Observação 2: Número, é qualquer número real utilizando o carácter "."para separar a parte inteira da decimal e sem a utilização de separador de milhar. Alguns exemplos são: 42, 0.123, 1.00 e 1789.42.

A precedência dos operadores é dada pela seguinte ordem: "*", "/", "+", -".

A linguagem de remediação de custo possui uma série de funções matemáticas, baseadas na biblioteca do Python 3, que ajudam a modelar a dívida técnica, estas funções são expressas por Palavra(<Expressão>) ou Palavra(<Expressão>, <Expressão>).

Funções de manipulação numérica:

- **ceil(x):** Devolve o teto de x, ou seja, o menor número inteiro maior ou igual a x.
- **floor(x):** Devolve o piso de x, ou seja, o maior inteiro menor ou igual a x.
- **abs(x):** Devolve o valor absoluto de x.
- **factorial(x):** Devolve $x!$. Obs.: x deve ser um número inteiro e não negativo.

Potências e logaritmos:

- **exp(x):** Devolve e^x.
- **expm1(x):** Devolve e^{-x}.
- **log(x, base):** Devolve $\log_{base} x$.
- **log10(x):** Devolve $\log_{10} x$.
- **ln(x):** Devolve $\log_e x$.
- **pow(x, y):** Devolve x^y.
- **sqrt(x):** Devolve $\sqrt{x}$.

Funções trigonométricas:

- **acos(x):** Devolve o arc cosseno de x, em radianos.
- **asin(x):** Devolve o arc seno de x, em radianos.
- **atan(x):** Devolve o arc tangente de x, em radianos.
- **cos(x):** Devolve o cosseno de x, em radianos.
- **sin(x):** Devolve o seno de x, em radianos.
- **tan(x):** Devolve o tangente de x, em radianos.
- **hypot(x, y):** Devolve a norma Euclidiana, $\sqrt{(x*x)+(y*y)}$. Este é o tamanho do vetor da origem ao ponto (x, y).

Além das funções matemáticas a linguagem de remediação de custos possui variáveis previamente carregadas com informações de contexto da dívida técnica que está sendo analisada. Em geral, essas informações são referentes ao arquivo onde o item da dívida técnica está localizado. Essas variáveis são:

- **lines:** quantidade de linhas do arquivo.
- **ncloc:** quantidade de linhas de código, as linhas em branco não são contadas.
- **classes:** quantidade de classes do arquivo.
- **statments:** quantidade de *statments* do arquivo.
- **function_complexity:** complexidade média por função, ou seja, a complexidade total do arquivo dividido pela quantidade de funções.

- **comment_lines_density:** quantidade de linhas de comentários dividido pela quantidade de linhas do arquivo.
- **duplicated_lines:** quantidade de linhas duplicadas no arquivo.
- **violations:** quantidade de *issues* existentes no arquivo.
- **open_issues:** quantidade de *issues* abertas existentes no arquivo.

Essas variáveis foram escolhidas pois são as que são disponibilizadas pelo Sonar Qube para todos os arquivos de código. Mais variáveis poderiam ser utilizadas para expressar contexto, desde que fosse possível consegui-la para todos os arquivos, evitando erros de cálculo.

Segue abaixo, alguns exemplos da utilização desta linguagem:

```
1 1
2 # 1
3
4 -1
5 # -1
6
7 3 + 2
8 # 5
9
10 7 - 3
11 # 4
12
13 abs(-1)
14 # 1
15
16 floor(comment_line_density)
17 # 9
```

6.2.4 Classe de Modelagem

Para cada item de violação de regra (*issue*), a ferramenta TDA verifica se existe alguma classe de modelagem para calcular a dívida técnica. Caso a classe exista, a ferramenta a preenche com dados sobre a regra de conformidade, a *issue* e dados de contexto do projeto, tais como informações de complexidade, quantidade de linhas de código do arquivo, linhas de código duplicadas e muito mais. Com isso, a classe de modelagem é capaz de calcular o custo de pagar uma dívida técnica de forma mais flexível e precisa.

Essa flexibilização faz com que a TDA se adapte a diversos contextos de desenvolvimento de software. Além disso, a simplicidade de *design* torna fácil a implementação e a modificação das classes que calculam o custo de remediação.

A ferramenta TDA possui uma classe abstrata a qual toda classe de modelagem deve implementar. Esta classe abstrata possui dois atributos de classe e um construtor que recebe dois parâmetros e preenche os atributos. O primeiro atributo representa o item de dívida técnica, já o segundo representa o contexto, contendo métricas do arquivo no qual o item de dívida técnica está inserido. Além disso, esta classe também possui um método abstrato que é utilizado para que a ferramenta padronize a chamada para obter o custo de pagar a dívida técnica.

Este método, também, pode ser classificado como um método conceito, pois, como o método anterior, ele não mede o custo de pagar a dívida técnica em si, mas sim prove uma forma abstrata para modelar o custo do pagamento para um dado tipo de dívida técnica.

Segue abaixo, a classe abstrata de modelagem:

```
1 abstract class TechnicalDebtCalculator {
2   protected $technicalDebt;
3   protected $metrics;
4
5   abstract public function getCost();
```

```
6
7   public function __construct(TechnicalDebt $technicalDebt, $metrics) {
8     $this->technicalDebt = $technicalDebt;
9     $this->metrics = $metrics;
10  }
11 }
```

Segue abaixo, um exemplo de classe de modelagem que calcula o custo de pagar a dívida técnica para a regra S00105 do Sonar Qube:

```
1 class S00105 extends TechnicalDebtCalculator {
2   public function getCost() {
3     $limit = 100;
4     if ($this->metrics->getLines() < $limit) {
5       return $this->metrics->getLine();
6     }
7     else {
8       return $limit + log($this->metrics->getLine());
9     }
10  }
11 }
```

6.3 Questões de Implementação

A ferramenta TDA realiza acesso direto ao banco de dados do Sonar Qube para obter listas sobre os seguintes dados: projetos e arquivos analisados, *issues*, dívida técnica, regras de conformidade, características de dívida técnica e métricas de código.

Os dados sobre os projetos e arquivos são utilizados para que o usuário possa escolher o projeto para detalhar a dívida técnica. Já os dados relativos as *issues* e dívida técnica formam os itens de dívida identificados para cada projeto, junto a cada item está associado os custos de pagar a dívida técnica. As regras de conformidade são utilizadas para identificar o tipo de dívida técnica que foi encontrado, possuindo mensagens referentes aquela categoria de erros e detalhes, tais como a função de remediação de custo da dívida técnica. Cada característica de dívida técnica possui sub-características que, por sua vez, possui regras de conformidade, isso permite categorizar a dívida técnica conforme proposto na metodologia SQALE (Letouzey, 2012b). Por fim, as métricas de código são utilizadas para modelar o contexto do projeto, para isso, para cada item de dívida técnica são associados as métricas do arquivo na qual ela se encontra. Essa associação é atualizada apenas enquanto o item de dívida técnica não foi pago, após o pagamento ele deixa de ser atualizado para refletir o contexto no momento do pagamento.

A ferramenta TDA armazena no banco de dados cinco tipos de dívida técnica que são: o custo real de pagamento informado pela equipe de desenvolvimento, o custo calculado pela classe de modelagem, o custo calculado pela linguagem de remediação de custo, o custo calculado por regressão estatística e o custo calculado pelo Sonar Qube. Os custo são utilizados nesta ordem pela ferramenta. Na ausência de algum dos custos o custo utilizado é o imediatamente seguinte, por exemplo, se não tiver custo calculado pela classe de modelagem é usado o custo calculado pela linguagem de modelagem. Para indicar a ausência do custo a ferramenta armazena o número 0, pois qualquer item de dívida técnica deve possuir um custo de pagamento, por mais baixo que este custo seja.

6.4 Conclusão do Capítulo

Neste capítulo foram apresentados uma séries de problemas que os métodos que calculam a dívida técnica devem levar em conta para conseguir uma medida mais precisa. Com base nestes problemas, foram desenvolvidos quatro métodos complementares para medir o custo de pagar a dívida técnica. Para cada um dos itens itens de dívida técnica, quatro medidas são realizadas. A primeira, é a padrão Sonar Qube, que utilizando a metodologia SQALE, define uma função linear

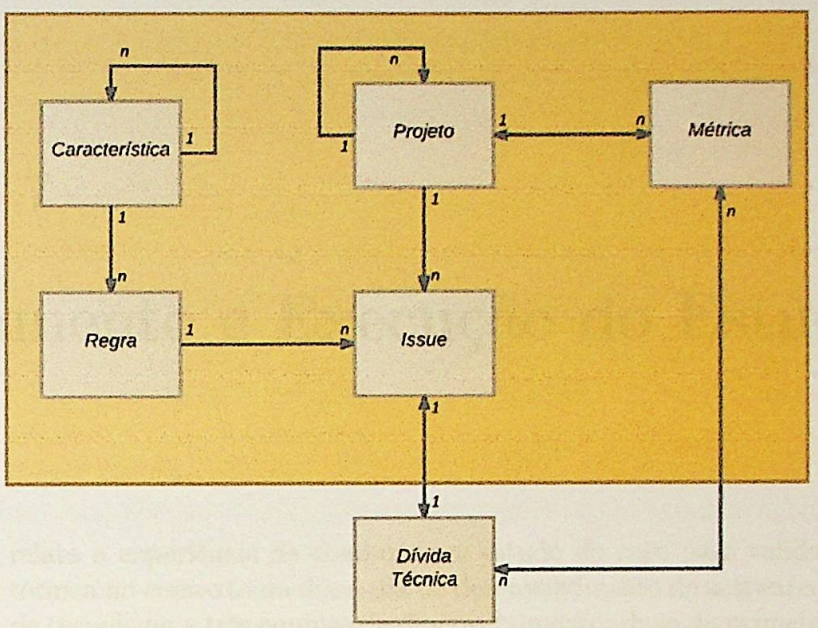

Figura 6.1: *Modelo de entidades do Sonar Qube e da Ferramenta TDA*

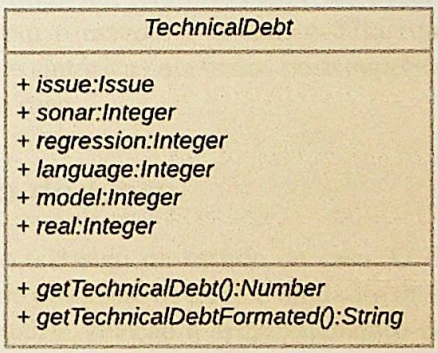

Figura 6.2: *Modelo de dados da ferramenta TDA.*

simples para calcular um dado tipo de dívida técnica. Em segundo, a regressão estatística utiliza dados de pagamento e contexto para gerar um modelo de predição para cada tipo de dívida técnica, os dados de contexto são as métricas de código. Em terceiro, a linguagem de remediação de custo que permite associar um código-fonte para calcular a dívida técnica para cada tipo. E por fim, a classe de remediação de custo que permite criar uma classe na linguagem PHP para modelar o custo de pagar um dado tipo de dívida técnica.

No final do capítulo, foi apresentado questões de implementação referentes a ferramenta desenvolvida e chamada de Technical Debt Analyser (TDA). Estas questões abrangem a modelagem de entidades e estrutura de dados da ferramenta e, também, o funcionamento interno.

No próximo capítulo será mostrado um estudo de caso que tem como objetivo principal avaliar os quatro métodos propostos nesta dissertação. Além disso, é proposto um novo método baseado na análise temporal de árvores sintáticas abstratas.

Capítulo 7

Planejamento e Execução do Estudo de Caso

Este capítulo relata a experiência de conduzir um estudo de caso para validar os métodos de medida da dívida técnica no contexto do dia-a-dia de desenvolvimento de software. Os casos vieram de duas empresas de tecnologia e três equipes de desenvolvimento, duas da primeira empresa e uma da segunda.

Na primeira parte do capítulo, é apresentado em detalhes o design do estudo de caso, contendo todas as informações referentes aos objetivos, planejamentos e considerações teóricas referentes ao estudo. Posteriormente, é apresentado um relatório com a descrição da condução do estudo de caso, seguido das lições aprendidas com o estudo. Por fim, é feita uma nova proposta de medida de dívida técnica baseada em árvores sintáticas abstratas para superar os problemas e falhas ocorridos durante a execução do estudo de caso.

7.1 Design do Estudo de Caso

Nesta seção será apresentado o design do estudo de caso. O design foi dividido em: a análise racional, os objetivos e questões de pesquisa, os casos e unidades de análise e sua seleção, o arcabouço teórico, os métodos de coleta, análise, seleção e armazenamento dos dados, o protocolo do estudo de caso, as considerações éticas, estratégias de replicação e os métodos para garantir a qualidade do estudo.

7.1.1 Análise Racional

A principal razão pela qual este estudo de caso foi realizado foi desenvolver e avaliar métodos de medida de dívida técnica que se adaptem ao contexto em que o software é desenvolvido. Este problema foi detectado durante a realização de um trabalho de conclusão de curso em bacharelado em ciência da computação e até o momento com poucas soluções exploradas.

Outra razão encontrada posteriormente é permitir que os times de desenvolvimento possam modelar de forma fácil o custo de pagar um determinado tipo de dívida técnica.

7.1.2 Objetivos

O principal objetivo deste estudo de caso é avaliar os quatro métodos de medida de dívida técnica propostos neste trabalho. Com isso, verificar em quais casos, cada um dos métodos pode ser utilizado.

Para a linguagem de remediação de custos e a classe de modelagem, que exigem uma análise e modelagem de dados e a escrita de código, um segundo objetivo é aplicado. Este objetivo é entender a dificuldade de analisar e modelar dados e, em seguida, transformar em um código para calcular o custo de pagar um item de dívida técnica de um dado tipo. Ou seja, para cada um destes métodos

é preciso saber se os membros da equipe de desenvolvimento são capazes de realizar toda a tarefa de configuração e análise ou é necessário outro profissional, como por exemplo, um matemático ou estatístico.

7.1.3 Questões de Pesquisa

1. Como medir o custo de pagar a dívida técnica em diferentes contextos, tais como, a linguagem de programação e métricas de código?
2. Como utilizar o histórico de pagamentos de itens de dívida técnica para refinar o custo de novos pagamentos?
3. Como identificar em quais casos cada um dos métodos pode ser utilizado?
4. Para os métodos que exigem configuração prévia:
   (a) Em média, quanto tempo demora para configurar a medida para cada tipo de dívida técnica?
   (b) Qual é a dificuldade de configurar estes itens?
   (c) É adequado que a equipe de desenvolvimento configure estes métodos?

7.1.4 Casos e Unidades de Análise

Inicialmente, o estudo foi modelado como um estudo de caso único e embutido, como mostrado em (Runeson *et al.*, 2012), pois ele era composto de um caso chamado de Empresa 1 e duas unidades de análise chamadas de Time 1 e Time 2, como mostra a figura 7.1.

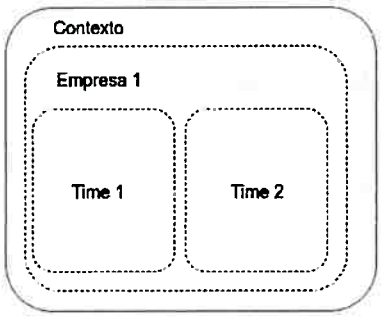

Figura 7.1: *Estudo de caso único e embutido.*

Após algumas semanas do planejamento inicial do estudo de caso, surgiu um novo caso e o estudo foi alterado para um estudo de caso múltiplo e embutido. Este novo caso foi chamado de Empresa 2 e a nova equipe de desenvolvimento foi chamada de Time 3, como mostra a figura 7.2.

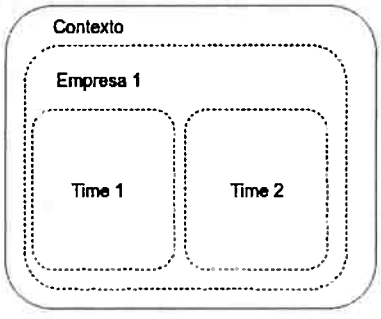

Figura 7.2: *Estudo de caso múltiplo e embutido.*

A Empresa 1 desenvolve uma plataforma de fácil criação de aplicativos com pouco, ou nenhum, conhecimento em programação. O Time 1 desenvolveu a primeira versão da plataforma a há 5 anos.

Eles utilizavam metodologia *scrum* para o gerenciamento do projeto e a equipe era composta por 1 *Product Owner* e 5 programadores com até 5 anos de experiência. O Time 2 desenvolveu a segunda versão da plataforma há 2 meses, no início do estudo. Eles utilizavam princípios de metodologias ágeis para gerenciar o projeto e a equipe era composta por 5 programadores com mais de 5 anos de experiência e 2 *designers*.

Já a Empresa 2, desenvolve novas funcionalidades e realiza manutenção das funcionalidades já existentes de um software de gestão de benefícios previdenciários especializado em Regimes Próprios de Previdência Social (RPPS), ou seja, orientado ao setor público. O software é uma aplicação web formada por módulos como de cadastro (servidores públicos, dependentes, conta corrente, dentre outros), concessão de benefícios previdenciários como aposentadoria e pensão, processamento da folha de pagamento, gestão eletrônica de documentos com certificação digital, auto-atendimento para os aposentados e pensionistas e integração com softwares próprios dos estados, municípios e governo federal. O Time 3 era o responsável pelo desenvolvimento deste software e utilizava scrum e kanban para realizar o planejamento das atividades. A equipe era composta por 7 programadores Java com diferentes níveis de conhecimento e experiência.

7.1.5 Arcabouço Teórico

O arcabouço teórico para este estudo de caso foi constituído principalmente pelo artigo *Guidelines for conducting and reporting case study research in software engineering* (Runeson e Höst , 2009) e pelo livro *Case Strudy Research in Software Engineering* (Runeson *et al.*, 2012). Além disso, foi feito uma revisão da literatura, analisando estudos de casos em engenharia de software, principalmente com o tema de dívida técnica.

7.1.6 Métodos de Coleta de Dados

O estudo de caso foi conduzido dentro de um acordo entre as empresas e os pesquisadores. Questões de confidencialidade e publicações foram regulamentados em um acordo. Os pesquisadores tiveram acesso a trechos de códigos-fonte relacionados aos itens de dívida técnica, e informações relacionadas ao desenvolvimento dos softwares.

A plataforma Sonar Qube, junto com a ferramenta *Technical Debt Analyser* (TDA), foram utilizadas para medir a dívida técnica dos projetos de software. A cada iteração do ciclo de desenvolvimento do software, os integrantes da equipe deveriam utilizar as ferramentas para escolher itens de dívida técnica para pagarem. A escolha poderia ser feita de acordo com a necessidade e a disponibilidade de tempo de cada equipe, definidas junto com os gestores na reunião de planejamento da iteração. Os programadores deveriam marcar o tempo, em minutos, gasto para pagar o item de dívida técnica e informá-lo na ferramenta TDA. O banco de dados da plataforma e da ferramenta poderiam ser utilizados para ajudar os gestores e programadores a escolherem os itens de dívida técnica e obterem mais informações sobre ela.

Inicialmente, não foi definido um tempo de duração para a coleta de dados utilizando a ferramenta, pois parte dos dados deveriam ser gerados pelos programadores conforme suas disponibilidade. Esta coleta de dados seria finalizada quando tivessem dados suficientes para que todos os métodos de medida da dívida técnica pudessem ser utilizados apropriadamente.

Outro método de coleta de dados que planejada para uso é a entrevista semi-estruturada. A entrevista possui a seguinte estrutura:

- Introdução
- Conhecimento pessoal, do grupo e da empresa
- Conhecimento sobre dívida técnica e qualidade de software
- Experiência de uso dos métodos de medida da dívida técnica
- Considerações finais

7.1.7 Métodos de Análise de Dados

Para analisar os dados gerados pela ferramenta, métodos de estatística descritiva deveriam ser utilizados para comparar os valores informados pelos programadores e os valores fornecidos por cada um dos métodos.

As entrevistas seriam gravadas em áudio digital e, posteriormente, transcritas de forma integral, exceto os trechos que obviamente fogem ao escopo desta pesquisa. Após a transcrição, ele deveria ser revisada com cada um dos entrevistados para garantir que a correta interpretação.

Em seguida, para cada frase da entrevista, códigos de agrupamento deveriam ser adicionados, permitindo com que pontos em comum de diferentes entrevistas pudessem ser encontrados. Por fim, com as frases agrupadas, para cada código ou para conjunto de código conclusões e contradições deveriam ser escritas. Neste ponto, conclusões de diferentes códigos poderiam ser comparados a fim de gerar novas conclusões.

7.1.8 Passo-a-passo do Estudo de Caso

O estudo de caso foi dividido em oito tarefas que deveriam ser realizadas na seguinte ordem:

1. Instalação e configuração do Sonar Que e da ferramenta Technical Debt Analyser (TDA).
2. Palestra sobre dívida técnica e o uso das ferramentas no dia-a-dia do desenvolvimento de software.
3. Questionário de caracterização da equipe de desenvolvimento.
4. Pagamento dos itens de dívida técnica utilizando as ferramentas para controle.
5. Atividade em grupo para avaliar o uso dos métodos que exigem modelagem e a escrita da função de remediação da dívida técnica.
6. Atividade em grupo para avaliar qual método consegue o melhor custo de dívida técnica para uma série de itens.
7. Entrevista individual com cada programador.
8. Apresentação dos resultados para toda a equipe.

7.1.9 Definição e Armazenamento de Dados

Os dados quantitativos foram armazenados em bancos de dados relacionais utilizando a estrutura de dados da plataforma Sonar Qube e da ferramenta Technical Debt Analyser (TDA).

Já a gravação digital das entrevistas deveriam ser armazenadas no Google Drive para permitir às pessoas autorizadas. A transcrição e a codificação das frases da entrevistas seriam feitas no editor de texto e na planilha eletrônica do próprio Google Drive, respectivamente.

7.1.10 Seleção de Dados

As entrevistas deveriam ser realizadas com todos os integrantes de cada uma das equipes de desenvolvimento para garantir a multiplicidade dos dados obtidos a partir das entrevistas.

Por outro lado, deveriam ser selecionados apenas os dados qualitativos que permitissem analisar todos os métodos de medida da dívida técnica ao mesmo tempo, ou pelo menos, dois a dois. Isto permitiria, realizar comparações entre os métodos, permitindo a extração de conclusões.

7.1.11 Estratégia de Seleção de Caso

A Empresa 1 selecionada para esta pesquisa é proveniente dos contatos de um dos pesquisadores. A empresa possuía duas equipes trabalhando em projetos similares, porém com tecnologias e equipes com níveis de conhecimento diferentes. Este fato permite obter uma grande variação, especialmente, no contexto de desenvolvimento. Sendo assim, a Empresa 1 pode ser eleita como um caso e as duas equipes como unidades de análise.

A Empresa 2 foi selecionada após o contato através da página de divulgação da pesquisa (http://www.ime.usp.br/~diogojp/). A empresa possui duas equipes, uma que trabalha com desenvolvimento em Java e outra com PL-SQl. Devido a restrição de análise de linguagens de programação, somente a equipe que desenvolve em Java foi selecionada como unidade de análise e a Empresa 2 como um caso.

7.1.12 Protocolo do Estudo de Caso

O protocolo de estudo de caso é o documento onde todas as informações referentes ao planejamento e a execução do estudo de caso. Este protocolo foi utilizado para guiar a coleta e análise de dados, garantindo assim uma padronização.

As principais seções deste documento são:

- Introdução: Descrevendo a proposta do estudo de caso.
- Processos Gerais: Uma visão geral dos processos envolvidos no estudo de caso.
- Instrumentos de Pesquisas: Guias para a realização de todos os processos do estudo de caso.
- Guia de análise de dados: Guia para a realização da análise de dados e extração de conclusões.

Este é um documento que deveria ser constantemente atualizado para refletir a realidade do planejamento e da execução do estudo de caso.

7.1.13 Considerações Éticas

Um acordo foi firmado entre as empresas e os pesquisadores para garantir a confidencialidade dos dados e da empresa. Todo o material como áudios, transcrições e bancos de dados deveriam ser mantidos em sigilo, publicando apenas as partes essenciais e com o consentimento dos envolvidos.

Além disso, para participar da entrevista cada um dos membros da equipe deveria concordar com a publicação das conclusões extraídas. A revisão das entrevistas e conclusões seriam feitas junto a cada um dos entrevistados para garantir a confidencialidade e a veracidade das informações.

7.1.14 Estratégia de Replicação

Com dois casos e três unidades de análises distintas, a intenção era encontrar uma estratégia de replicação entre os casos baseado no arcabouço teórico. Como este estudo de caso é baseado em contexto de desenvolvimento de software é difícil replicá-lo completamente, porém todo o processo pode ser utilizado para a replicação do estudo.

7.1.15 Garantia da Qualidade, Validade e Confiabilidade

Para tentar garantir a qualidade, validade e confiabilidade deste estudo de caso, foi implementado o seguinte processo:

1. Todo o documento de pesquisa gerados neste estudo de caso deveriam ser analisados por, pelo menos, dois pesquisadores.
2. Os dados quantitativos deveriam ser revisados coletivamente com cada uma das equipes.

3. As entrevistas seriam revisadas por cada um dos entrevistados, tanto a transcrição, quanto a codificação e a extração de conclusões.

4. Seriam gerados relatórios para todas as atividades realizadas neste estudo de caso, onde todos envolvidos poderiam revisar o relatório, permitindo melhorias e eventuais correções no processo.

7.2 Relatório

A primeira unidade de análise a entrar no estudo de caso foi o Time 2. Após o acordo ser firmado entre as partes e uma primeira reunião de apresentação do projeto de pesquisa ser realizada, os principais projetos da equipe de desenvolvimento começaram a ser analisados utilizando o Sonar Qube e a ferramenta *Technical Debt Analyser* (TDA). Toda a análise foi realizada num servidor virtual disponibilizado pelos pesquisadores.

Como o Time 2 utilizava o GitHub como repositório de código, foi criado para cada um dos projetos uma rotina no servidor Jenkins para monitorar novos *commits* de código e automaticamente sincronizá-los e realizar as tarefas de análise do novo código. Ou seja, o código era baixado do repositório, analisado com o Sonar Scanner e, por fim, calculado os outros valores de dívida técnica utilizando a ferramenta TDA.

Após pouco mais de um mês das conversas iniciais, o projeto que era financiado por programas do governo federal, teve seu orçamento cortado forçando o término imediato das atividades realizadas pelo Time 2.

Paralelamente as conversas iniciaias com o Time 2, foram iniciadas as conversas com o Time 1. Após o acordo ser firmado entre as partes foram realizadas duas reuniões. Na primeira reunião a pesquisa foi apresentada para pessoas com gerencias na empresa, como o gerente de produtos, o gerente de marketing e o gerente financeiro da empresa. Na segunda reunião a pesquisa foi apresentada para os membros da equipe de desenvolvimento. Nesta reunião, além da apresentação do projeto, foi realizada a configuração e a análise do principal projeto da equipe utilizando o Sonar Qube e a ferramenta *Technical Debt Analyser* (TDA). Toda a análise foi realizada num servidor virtual disponibilizado pelos pesquisadores.

A análise do primeiro projeto despertou o interesse dos membros da equipe de desenvolvimento e outros três projetos foram configurados com a ajuda de um dos desenvolvedores. O Time 1 também utilizava Git como repositório de código, logo a mesma solução do processo de análise foi implementada. Isto é, foram criadas rotinas no servidor Jenkins para baixar as alterações do repositório Git e, em seguida, analisá-las utilizando o Sonar Scanner e com a ferramenta TDA.

Com todas as análises de projeto configuradas, foi realizada uma palestra sobre dívida técnica e o uso das ferramentas. Nesta palestra, os desenvolvedores puderam tirar dúvidas sobre os conceitos de dívida técnica e o uso das ferramentas para poder escolher os itens de dívida técnica a serem pagos.

Na reunião de planejamento após a configuração das ferramentas, a equipe de desenvolvimento definiu alguns itens de dívida técnica para pagar. Porém, um grave problema no software desenvolvido, fez com a equipe abandonasse a iteração para resolvê-lo. O problema levou pouco mais de um mês para ser resolvido, porém o *Product Owner* e o sócio, que apoiava a pesquisa, saíram da Empresa 1. Com isso, o Time 1 acabou deixando de participar do estudo de caso.

Novas equipes de desenvolvimentos foram procuradas para tentar compensar a perda do Time 2 e garantir a qualidade, validade e confiabilidade do estudo de caso. Para tanto, foi criado uma página para a divulgação do projeto de pesquisa (http://www.ime.usp.br/~diogojp). Este site foi divulgado no Facebook e teve três retornos. Um dos retornos foi a Empresa 2, outro foi de uma empresa que não respondeu aos e-mails para um possível acordo e o último foi uma gerente de projetos indicada por um dos membros do Time 3. Foram trocadas várias mensagens com a gerente, porém por problemas pessoais dela, não foi possível entrar em um acordo para realizar a pesquisa.

Após as primeiras conversas com a Empresa 2, uma reunião foi realizada com o gerente geral

de tecnologia, uma das analistas de sistemas e o líder da equipe de desenvolvimento. Nesta reunião foi apresentada a pesquisa e acertados os detalhes sobre a confidencialidade do projeto.

Após o acordo ser firmado, foi agendado um dia para a instalação e configuração do servidor e das ferramentas em um servidor virtual localizado dentro da Empresa 2, porém foram necessários dois dias para a configuração completa do servidor. Como a equipe de desenvolvimento utilizava um repositório de código integrado ao Visual Studo, não foi possível sincronizar automaticamente o código. Para superar este problema, foi desenvolvido um script para sincronizar os arquivos do computador de um dos desenvolvedores com o FTP do servidor. Ao detectar a sincronização do código, um script agendado faz a chamada ao Sonar Scanner e à ferramenta TDA para fazer a análise do código e medir a dívida técnica. Devido ao tamanho do código-fonte e o tempo levado para a análise completa, o agendamento foi realizado apenas em dois horários, um durante a madrugada e o outro durante o horário de almoço.

Após configurar o servidor e as ferramentas, foi realizada uma palestra sobre a dívida técnica e o uso das ferramentas. Durante a palestra surgiram algumas dúvidas, principalmente, em relação ao uso das ferramentas. Além disso, foram feitas sugestões de melhorias da ferramenta TDA. Algumas sugestões foram implementadas, uma delas foi um filtro por arquivos.

Com os ajustes realizados na ferramenta, os desenvolvedores deveriam começar a escolher itens de dívida técnica para realizarem o pagamento. Porém, neste momento, eles optaram por não realizar pagamento da dívida técnica, colocando todos os recursos de pessoal para entregar novas funcionalidades e realizar uma nova entrega do software desenvolvido.

Após passar este período de entrega, apenas um dos desenvolvedores utilizou a ferramenta para pagar a dívida técnica dos projetos. Isso fez com que o volume de dados gerados fosse insuficiente para que a método de regressão e os métodos que precisavam de modelagem funcionassem.

7.3 Lições Aprendidas

Apesar do estudo de caso ter falhado, várias lições foram aprendidas e vários processos devem ser aprimorados em relação ao planejamento, divulgação e condução do estudo de caso.

Em relação ao planejamento do estudo de caso, as lições aprendidas foram:

- As limitações de tempo e a priorização das equipes de desenvolvimento devem ser levadas em conta no planejamento.
- Definir datas mais rígidas ao cronograma de pesquisa pode fazer com que a pesquisa se torne prioridade dentro das equipes de desenvolvimento.
- Planejar em detalhes as possíveis entradas e saídas de casos e unidades de análise ao estudo de caso.

Já em relação a divulgação do estudo de caso:

- Divulgar em mais redes sociais e grupos de interesse pode ajudar a conseguir mais casos para o estudo.
- Preparar um vídeo de curta duração pode despertar o interesse de mais pessoas.
- Preparar um servidor de demonstrações com as ferramentas de pesquisa.
- Permitir que o visitante do site possa analisar o seu projeto em pouco passos. Por exemplo, apenas informando o endereço do repositório Git e uma conta de acesso, se necessário.
- Utilizar ferramentas, como o Google Analytics, para monitor o acesso ao site.
- Ter uma versão em inglês do site, para que empresas do mundo todo possam se juntar a pesquisa.

Por fim, em relação a condução do estudo de caso:

- Manter uma comunicação diária com as equipes de desenvolvimento.
- Realizar mais visitas de acompanhamento para incentivar o uso das ferramentas.

7.4 Medida Baseada em Árvores Sintáticas Abstratas

Um dos problemas que levaram a falha do estudo de caso foi que a maioria dos métodos dependiam que os programadores informassem dados para que eles pudessem funcionar. Logo, seria interessante ter alguma forma de medir o custo de pagar a dívida técnica sem precisar depender de dados dos usuários. Para resolver este problema, um novo método de medida baseado em Árvores Sintáticas Abstratas (Abstract Syntax Tree - AST) está sendo desenvolvido.

Para encontrar uma função de remediação o método de medida analisa o repositório de códigos de um software e realiza a análise das AST's e dos itens de dívida técnica de cada uma das versões do código, como mostra a figura 7.3.

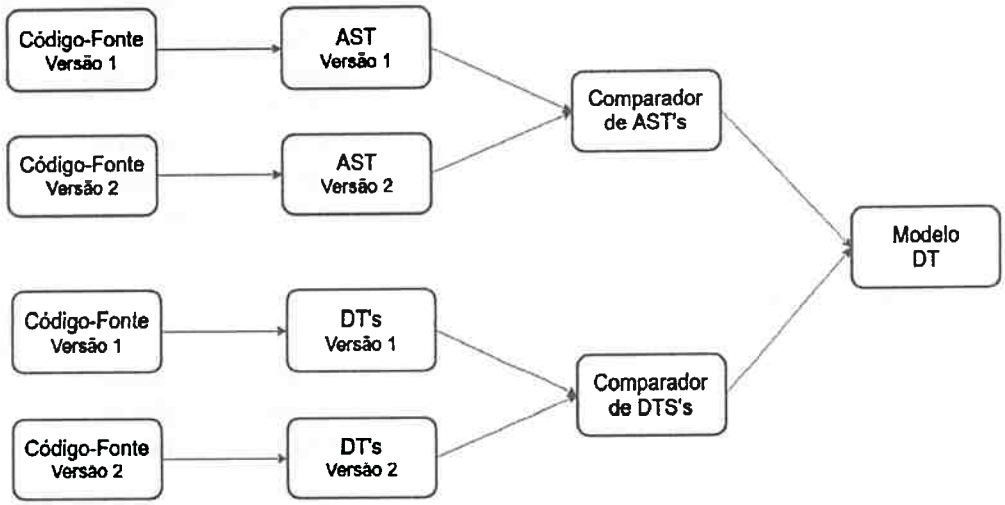

Figura 7.3: *Modelo de medida de dívida técnica baseado em Árvores Sintáticas Abstratas.*

Para cada duas versões seguidas do código-fonte o método de medida faz o seguinte:

- Gera uma AST para cada versão do código.
- Gera a lista de itens de dívida técnica para cada versão do código.
- As AST's são comparadas para verificar quais *statements* foram adicionados e quais foram removidos.
- As listas de itens itens de dívida técnica são comparados para descobrir quais itens de dívida técnica foram pagos de uma versão para outra.
- Para cada item de dívida técnica pago, é feita a comparação das AST's dos arquivos onde estava localizada o item.

Logo, dado que um item de dívida técnica foi pago de uma versão para outra, com a comparação entre as AST's chega-se em uma fórmula da seguinte forma:

$$custo(X) = \sum_{i=1}^{n} SA_i + \sum_{j=1}^{n} SR_i$$

onde n é a quantidade de tipos SA_i é a quantidade de *statement* adicionados do tipo i e SR_i é a quantidade de *statements* removidos do tipo i. Por exemplo, $custo(x) = 2 * adicionarifstatement + 1 * removerwhilestatement$.

Com este modelo de medida, a dívida técnica é dada em termos de adições e remoções de *statements*. Porém, ela também pode ser dada em tempo. Para tanto, é preciso fazer um mapeamento do custo em minutos, por exemplo, de adicionar e remover cada tipo de *statement*. O custo de cada um dos tipos de *statements* pode ser feita por especialistas, assim como é feito no Sonar Qube para cada tipo de dívida técnica, porém fazer isso em termos de linha de código exige uma quantidade muito menor de mapeamento. Além disso, caso novos tipos de dívida técnica sejam adicionados não haveria a necessidade de definir um novo mapeamento para este tipo, ou seja, o método seria capaz de encontrar uma função de remediação utilizando apenas os *statements*.

Outra forma de definir o custo de cada *statement* é através de regressão, porém isso cairia novamente no problema do estudo de caso, ou seja, dependeria que os desenvolvedores informassem o tempo necessário para pagar os itens de dívida técnica.

Como este método de medida foi baseado em processos de decisão de Markov de primeira ordem, pois olha apenas a diferença entre uma versão de código e sua versão anterior, logo, poderia também ser utilizado aprendizado com reforço. Neste caso, seria feito um mapeamento inicial, por especialistas, do custo de cada tipo de *statement* e usando a informação de tempo de pagamento de cada item de dívida técnica pelos desenvolvedores como reforço de aprendizado, ir ajustando o custo de cada tipo de *statemente* e consequentemente o custo de pagar um item de um dado tipo de dívida técnica.

7.4.1 Análise Preliminar

Para realizar uma análise preliminar do método de medida baseado em árvores sintáticas abstratas foi realizada uma análise do projeto Apache Maven (https://github.com/apache/maven). Este projeto foi escolhido, principalmente, por usar Java como linguagem de programação e, também, por ser um projeto *open source* amplamente difundido e com uma comunidade ativa. O Apache Maven possui mais de 10 mil *commits* sendo o primeiro em Agosto de 2003 e possui atualmente 51 colaboradores.

Primeiramente, os *commits* foram divididos em blocos com 500 *commits* cada descartando o primeiro bloco, com uma abordagem similar a adotada em (Aniche *et al.*, 2016). Neste caso, o último *commit* será usado como o representante do bloco, ou seja, ele representa o estado final do código após 500 modificações. Para pagar um item de dívida técnica pode ser necessárias diversas mudanças no arquivo de código até que o item seja totalmente pago, logo a abordagem de dividir em blocos pode contribuir nesse sentido. Além disso, dividir em blocos faz diminuir o custo computacional em cerca de 500 vezes, viabilizando a análise do código em um curto período de tempo.

Após a separação em blocos, cada representante é analisado utilizando o Sonar Qube. Em seguida, para cada par cronologicamente sequencial (por exemplo: (primeiro, segundo), (segundo, terceiro), (terceiro, quarto)) é feita uma comparação entre as versões de código representantes para definir quais itens de dívida técnica mantiveram, foram pagos ou foram adquiridos.

Para cada item de dívida técnica pago de um bloco para outro, as árvores sintáticas abstratas dos arquivos onde se encontrava o item e do arquivo sem o item de dívida técnica são comparados para determinar a quantidade de *statements* de cada tipo que foram inseridos ou removidos durante o pagamento da dívida técnica. Esse conjunto de valores é colocado em uma lista que é associada a regra de conformidade que foi violada e gerou o item de dívida técnica. As árvores sintáticas abstratas são geradas utilizando o Java development tools (JDT).

Após realizar todas as comparações, para cada uma das regras de conformidade são gerados modelos do custo de pagar um item de dívida técnica daquele tipo. Para isso, se a lista de *statements* associada a regra tiver pelo menos 10 itens, então para cada tipo de *statement* é calculado uma média aritmética simples para definir a quantidade daquele *statement* que precisa ser inserida, sinal positivo, ou removida, sinal negativo.

Por exemplo, para a regra: "expressões não deveriam ser tão complexas"precisam adicionar 2,18 expressões, 2,16 declarações de variáveis, 0,58 declaração de métodos e 1,06 *if*. Já para a regra: "elementos obsoletos não deveriam ter ambos anotação e marcação do JavaDoc"precisam remover 1,281 expressões, 1,133 declarações de variáveis, 0,739 declaração de métodos e 0,148 *if*.

Uma tabela completa com os custos de pagar a dívida técnica dado pela análise do Apache Maven se encontra no apêndice A.

Como ameaça a validade deste método de medida temos o fato das comparações das árvores sintáticas abstratas serem feitas por inteiro e não apenas no ramo que foi afetado pelo item de dívida técnica. Outro fator é que os *outliers* não estão sendo descartados, logo ele pode influenciar fortemente a média.

7.5 Conclusão do Capítulo

O capítulo relatou um estudo de caso para avaliar os quatro métodos de medida propostos neste trabalho. A objetivo principal era verificar quais métodos avaliados apresentavam o melhor resultado para diferentes situações e, assim, poder determinar quando utilizar cada um deles.

O estudo de caso falhou, porém uma série de lições foram aprendidas com a condução deste estudo de caso. Estas lições são em sua maioria ligadas a aprimoramentos de planejamento, divulgação e condução do estudo de caso.

Com as lições aprendidas, e uma análise dos pontos de falha do estudo de caso, foi proposto uma nova abordagem para medir o custo de pagar a dívida técnica. Este novo método é baseado em análise de versões de código-fonte, comparações de árvores sintáticas abstratas e identificação automáticas de pagamento de itens de dívida técnica. Expressando assim, a dívida técnica em forma de *statements* de código, que podem ser associados a horas de desenvolvimento e, assim, oferecer o custo de pagar a dívida técnica em horas de desenvolvimento.

Capítulo 8

Conclusões

Em outros trabalhos, foram propostas algumas formas de medir o custo de pagar a dívida técnica, porém estas propostas não permitem realizar o calculo de forma flexível, precisa e adaptativa num mesmo método. Os métodos propostos neste trabalho, se usados conjuntamente, pode fornecer essas propriedades, permitindo que uma mesma abordagem possa ser utilizada em diferentes linguagens de programação, equipes de desenvolvimento e metodologias de desenvolvimento.

Este trabalho, também, realizou a análise da literatura sobre métodos de medida de dívida técnica e desenvolveu quatro novos métodos complementares para calcular o custo de pagar cada item de dívida técnica levando em conta o contexto em que o software é desenvolvido.

Os métodos de medida do custo de pagar cada item de dívida técnica desenvolvidos neste trabalho são:

- **Padrão Sonar Qube:** é a medida da dívida técnica realizada pela plataforma Sonar Qube para, em caso de impossibilidade de utilizar algum dos outros três métodos, possuir um custo para dívida técnica. Esta medida é obtida por uma função linear do tipo $f(x) = Ax + B$ associada a cada regra de conformidade. As funções são inicialmente definidas por especialistas experientes, mas podem ser alteradas por meio do plugin do Sonar (Source) ou diretamente no banco de dados.

- **Regressão Estatística:** esta é uma abordagem estatística, que também vem sendo utilizada na área de aprendizagem de máquina. Neste caso, para cada item de dívida técnica é realizada uma regressão polinomial multivariada da forma:

$$Y = A + \sum_{i=1}^{n} \sum_{j=1}^{m} (a_{ij}) * (X_i)^j$$

  As n variáveis independentes $X_1, X_2, ..., X_n$ representam o contexto em que a dívida técnica, de itens do mesmo tipo, foi paga e a variável Y representa o custo da dívida técnica paga. Por fim m é o grau máximo do polinômio, A é uma constante e a_{ij} são os coeficientes de cada fator. O software R (R-Project, 2016) é utilizado para realizar o cálculo da regressão e, este, devolve os coeficientes de cada um dos fatores do polinômio. Dados os coeficientes e o contexto do item de dívida técnica desejado basta estimar o custo de pagar o item de dívida técnica.

  A vantagem deste método é que ele é capaz de modelar o custo de pagar a dívida técnica independente da forma da função que modela o problema. Por outro lado, para conseguir fazer isso ela precisa de uma grande quantidade de dados, que neste caso deveria ser informado pelo programador.

- **Linguagem de Remediação de Custo:** é uma linguagem que permite, de maneira simples, modelar o custo de pagar um item de dívida técnica de um determinado tipo. A linguagem é similar a expressões matemáticas, disponibilizando as quatro operações básicas, precedências

com parenteses e funções matemáticas tais como: seno, cosseno e raiz quadrada. Além disso, é possível definir variáveis para armazenar o valor das expressões e posteriormente utilizá-la dentro de outras expressões. Algumas variáveis são reservadas e possuem em seus valores dados de contexto do projeto, tais como: informações de complexidade, quantidade de linhas de código do arquivo e linhas de código duplicadas.

A vantagem é que para cada regra de violação propriedade de qualidade é possível modelar a função de remediação de maneira a expressar corretamente o custo de pagar um item de dívida técnica associado aquela propriedade. Por outro lado, ela não é uma linguagem tão poderosa quanto as linguagens como C, Java, PHP. Além disso, é preciso ter conhecimento para analisar e inferir uma função de custo.

- **Classe de Modelagem:** para cada tipo de dívida técnica nossa ferramenta verifica se existe uma classe de modelagem, escrita em PHP, associada para calcular o custo de um item de dívida técnica. Caso a classe exista, a ferramenta a preenche com dados sobre o tipo de dívida técnica, o item de dívida técnica e dados de contexto do projeto, tais como: informações de complexidade, quantidade de linhas de código do arquivo e linhas de código duplicadas. Com isso, a classe de modelagem é capaz de calcular o custo de pagar uma dívida técnica de forma mais flexível e precisa.

  A vantagem deste método de medida é a mesma do anterior, para cada regra de violação propriedade de qualidade permitir associar uma classe PHP para modelar a função de remediação. Porém para utilizar a expressividade da linguagem de programação PHP é preciso conhecê-la e, mesmo com isso, ainda continua o problema da necessidade de conhecimento para analisar e inferir a função de remediação de custo.

Para viabilizar a implementação dos métodos de medida de dívida técnica desenvolvidos neste trabalho e, posteriormente, poder analisá-los foi desenvolvida a ferramenta *Technical Debt Analyser* (TDA) que é integrada a plataforma Sonar Qube. Esta ferramenta permite aos membros de equipes de desenvolvimento de software visualizem o custo de pagar a dívida técnica calculados pelos quatro métodos de medida desenvolvidos neste trabalho. Ela permite também filtrar e alterar a criticidade, resolução, responsável por pagamento, regras, diretórios e arquivos. É por meio desta ferramenta que os programadores podem informar o tempo que eles levam para pagar cada item de dívida técnica para ser utilizado pelo método de regressão estatística.

Um estudo de caso foi conduzido para avaliar os quatro métodos de medida de dívida técnica propostos neste trabalho. O objetivo principal da avaliação era verificar em quais situações cada um dos métodos poderia ser utilizado e, com isso, melhorar a previsão de custo de pagar cada item de dívida técnica.

O estudo de caso falhou devido a necessidade de interações com a ferramenta, problemas financeiros e restrições de tempo das três unidades de análise. Apesar da falha, várias lições foram aprendidas com a condução do estudo de caso. Estas lições são em sua maioria ligadas a aprimoramentos de planejamento, divulgação e condução do estudo de caso.

Para superar os problemas enfrentados durante a execução do estudo de caso, principalmente a necessidade dos programadores precisarem informar o tempo gasto para pagar cada item de dívida técnica, foi proposta uma nova abordagem para medir o custo principal da dívida técnica. Este novo método de medida é baseado em aprendizagem de máquina, ele analisa diversas versões do código-fonte, realizando comparações das árvores sintáticas abstratas e identificando automaticamente o pagamento de itens de dívida técnica. Com estas informações, para cada tipo de dívida técnica é criado um modelo para pagar a dívida técnica dada em *statements* de código, cada tipo de *statement* pode ser associado a quantidade de horas de desenvolvimento e, assim, oferecer o custo de pagar um item de dívida técnica em horas de desenvolvimento. O projeto Apache Maven foi analisado utilizando o método proposto e foi obtidos modelos de custo para o pagamento de item de dívida técnica para mais de 40 regras de conformidade.

8.1 Considerações Finais

Em relação aos métodos de medida, pode ser interessante abrir mão de métodos mais precisos, mas que dependam da intervenção de pessoas, para métodos menos precisos, mas que podem ser calculados de forma automática ou com um mínimo de esforço do usuário.

Para a ferramenta, é interessante ter uma ferramenta mais completa de modo que ela possa ser utilizada para ajudar a gerenciar a dívida técnica em todas as etapas, desde a identificação até o monitoramento. Além disso, uma ferramenta que permita a fácil e rápida configuração de análise de novos projetos pode ajudar a trazer mais casos e aumentar a confiabilidade das pesquisas.

Em relação a condução do estudo de caso, é interessante planejar em detalhes a entrada e saída de casos e unidades de analise do estudo. Isso ajuda a prevenir problemas em estudos mais longos e que precisam de diversas interações de pessoas envolvidas nos projetos. Além disso, planejar visitas de acompanhamento constantes pode ajudar a motivar e evitar que o estudo não tenha a devida prioridade ou seja deixado de lado.

8.2 Sugestões para Pesquisas Futuras

Como sugestão para trabalhos futuros em relação ao custo principal da dívida técnica pode-se ter:

- Condução de outro estudo de caso para avaliar os quatro métodos de dívida técnica, porém, desta vez com restrições principalmente em relação a tempo e ao uso da ferramenta.
- Desenvolvimentos de outros métodos de medidas de dívida técnica que não precisem da intervenção dos programadores.
- Aprimorar a ferramenta para que ela possa ser utilizada para suportar todo o processo de gerenciamento da dívida técnica. Implementar formas de visualização e rastreamento podem ajudar a completar as funcionalidades-chave da ferramenta.

Outra parte que pode ser explorada em relação a medida de dívida técnica está relacionada aos juros. Poucos trabalhos exploram este problema devido a incerteza relacionada com está medida, pois o juros só é cobrado quando uma manutenção ou desenvolvimento é realizado e a dívida técnica faz com que este trabalho leve mais tempo do que levaria se os itens de dívida técnica relacionados com esta escrita de código estivessem pagos.

Além do estudo de métodos de medida, outros estudos que utilizam os valores fornecidos por estes métodos podem ser realizados contribuindo para o avanço da área de dívida técnica e motivar a indústria a tratar a dívida técnica dentro de seus projetos de software. Por exemplo, priorizar os itens de dívida técnica informando a ordem na qual eles devem ser pagos e criar uma forma de rastrear e analisar os itens pagos.

Apêndice A

Árvore Sintática Abstrata e Dívida Técnica

Este apêndice possui uma tabela de *statements* necessários para pagar um item de dívida técnica para cada regra de conformidade do Sonar Qube. A tabela A.1 mostra a regra de conformidade e as quantidades de cada tipo de *statement* que precisa ser adicionada ou removida para pagar um item de dívida técnica. Os *statements* mostrados na tabela A.1 são: expressão, declaração de variável, declaração de métodos, *if*, *else*, *for* e *while*. A quantidade de cada tipo de *statement* foi calculada usando o projeto Apache Maven.

Árvores sintáticas abstratas são estruturas de dados utilizadas para representar a estrutura sintática de um código-fonte. A árvore pode ser construída de várias formas, uma delas é armazenar um *statement* em cada nó e ligar os nós conforme a estrutura do código-fonte, sendo a ordem dada pelo filhos da esquerda para a direita. A figura A.1 mostra um exemplo simplificado de uma árvore sintática abstrata.

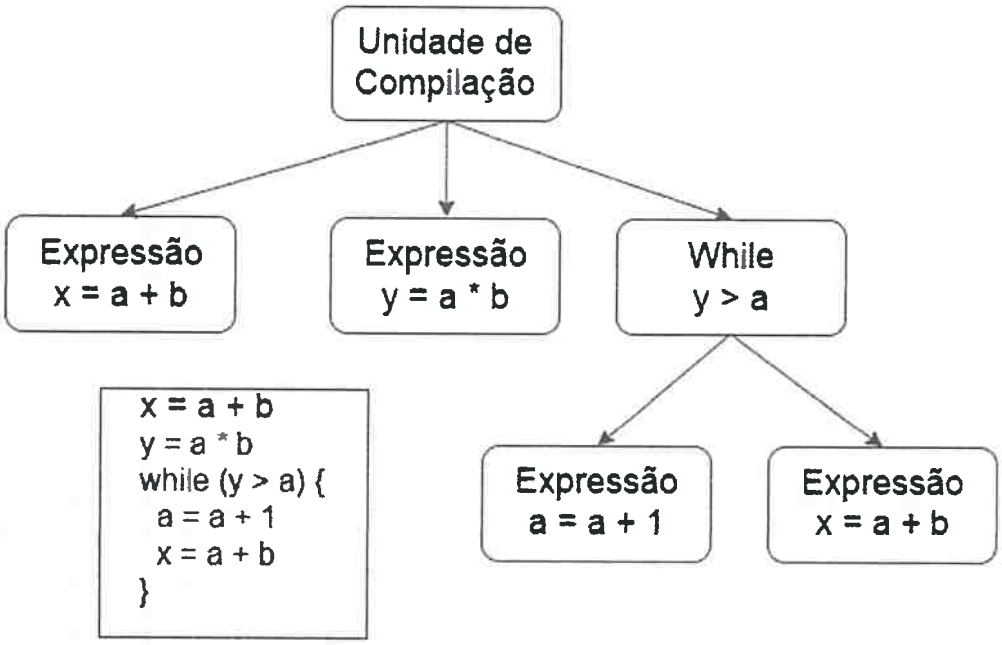

Figura A.1: *Exemplo simplificado de uma árvore sintática abstrata.*

Tabela A.1: *Statements necessários para pagar um item de dívida técnica para cada regra do Sonar Qube.*

Regra do Sonar	Exp	Var	Método	If	Else	For	While
"@Override"annotation should be used on any method overriding (since Java 5) or implementing (since Java 6) another one	1,513	1,234	1,299	0,801	0,138	0,179	0,000
Constants should not be defined in interfaces	0,000	0,007	-0,015	0,000	0,000	0,000	0,000
Useless imports should be removed	-1,405	-1,248	-0,445	-0,317	-0,041	-0,105	0,000
Local variables should not shadow class fields	-0,193	-0,104	-0,275	-0,160	-0,008	-0,035	0,000
"TODO"tags should be handled	0,482	0,457	0,319	0,192	0,019	0,033	0,000
Methods should not be too complex	0,502	0,503	0,238	0,204	0,032	0,025	0,000
Unused local variables should be removed	-0,973	-0,763	-0,356	-0,174	-0,032	-0,014	0,000
Exception handlers should preserve the original exception	0,193	0,339	0,195	0,124	0,016	0,007	0,000
Loops should not contain more than a single "break"or "continue"statement	2,981	2,769	1,173	1,308	0,173	0,077	0,000
Generic exceptions should never be thrown	-0,241	-0,022	0,034	0,027	0,008	0,038	0,000
Methods should not be empty	-0,011	-0,006	0,017	0,019	0,003	0,014	0,000
Synchronized classes Vector, Hashtable, Stack and StringBuffer should not be used	0,342	0,304	0,129	0,196	0,033	0,036	0,000
The members of an interface declaration or class should appear in a pre-defined order	-0,736	0,037	-1,870	-0,227	0,111	-0,273	0,000
Control flow statements "if", "for", "while", "switch"and "try"should not be nested too deeply	0,309	0,392	0,172	0,122	0,019	0,035	0,000
Field names should comply with a naming convention	-0,778	-1,037	-0,333	-0,222	-0,037	-0,222	0,000
String literals should not be duplicated	0,717	0,598	0,225	0,221	0,036	0,033	0,000
Collapsible "if"statements should be merged	0,935	0,761	0,497	0,391	0,048	0,063	0,000
Throwable.printStackTrace(...) should not be called	-1,455	-1,196	-0,259	-0,566	-0,084	-0,119	0,000
Strings literals should be placed on the left side when checking for equality	0,380	0,348	0,173	0,236	0,028	0,017	0,000
Lambdas and anonymous classes should not have too many lines	-5,467	-3,200	-1,733	-0,267	0,000	0,000	0,000
Standard outputs should not be used directly to log anything	0,361	0,298	0,080	0,099	0,020	0,015	0,000
Source files should not have any duplicated blocks	-0,216	1,092	0,097	0,025	0,007	-0,012	0,000
Method parameters, caught exceptions and foreach variables should not be reassigned	1,633	1,599	0,846	0,713	0,116	0,110	0,000
Throws declarations should not be superfluous	-0,443	-0,426	0,036	-0,104	-0,019	-0,033	0,000
Useless parentheses around expressions should be removed to prevent any misunderstanding	0,320	0,274	0,253	0,170	0,033	0,041	0,000

Tabela A.2: *Statements necessários para pagar um item de dívida técnica para cada regra do Sonar Qube.*

Regra do Sonar	Exp	Var	Método	If	Else	For	While
Public methods should throw at most one checked exception	1,109	0,938	0,630	0,207	0,049	0,057	0,000
Utility classes should not have public constructors	0,078	84,821	0,017	0,047	0,008	0,002	0,000
Constant names should comply with a naming convention	2,056	2,155	0,845	0,592	0,141	0,070	0,000
Modifiers should be declared in the correct order	2,846	2,385	2,077	1,385	0,308	0,385	0,000
Local Variables should not be declared and then immediately returned or thrown	0,833	0,714	0,317	0,310	0,016	-0,024	0,000
Unused method parameters should be removed	0,072	0,063	0,041	0,008	0,000	0,006	0,000
Unused private fields should be removed	2,960	3,103	1,822	0,787	0,103	0,247	0,000
Collection.isEmpty() should be used to test for emptiness	0,091	0,091	0,091	0,152	0,030	0,000	0,000
Collections.emptyList(), emptyMap() and emptySet() should be used instead of Collections.EMPTY_LIST, EMPTY_MAP and EMPTY_SET	-1,645	-1,452	-0,347	-0,071	-0,033	0,004	0,000
Exit methods should not be called	11,500	8,333	2,278	4,111	0,722	0,611	0,000
Methods should not have too many parameters	-0,703	-0,436	-0,085	-0,339	-0,061	-0,073	0,000
"switch case"clauses should not have too many lines	-1,717	-0,891	0,283	-0,826	-0,130	-0,152	0,000
Class variable fields should not have public accessibility	0,018	90,646	0,005	0,007	0,002	0,001	0,000
Nested blocks of code should not be left empty	1,878	1,383	0,374	0,670	0,113	0,096	0,000
"public static"fields should be constant	0,018	90,121	0,005	0,007	0,002	0,001	0,000
Deprecated code should be removed eventually	-1,097	-0,970	-0,633	-0,127	0,000	-0,084	0,000
Deprecated elements should have both the annotation and the Javadoc tag	-1,281	-1,133	-0,739	-0,148	0,000	-0,099	0,000
Overriding methods should do more than simply call the same method in the super class	1,952	1,143	0,619	0,095	0,000	0,000	0,000
Expressions should not be too complex	2,180	2,160	0,580	1,060	0,220	0,180	0,000
Parsing should be used to convert "Strings"to primitives	-1,040	-0,754	-0,206	-0,372	-0,065	-0,055	0,000
Loggers should be "private static final"and should share a naming convention	0,779	0,858	0,300	0,344	0,067	0,071	0,000
Try-catch blocks should not be nested	2,364	2,182	0,818	0,720	0,121	0,197	0,000
"switch"statements should end with a "default"clause	2,643	2,214	1,929	1,286	0,286	0,357	0,000
Classes should not be too complex	4,179	3,179	4,571	2,250	0,179	0,393	0,000
Tabulation characters should not be used	0,511	0,437	0,252	0,081	0,007	0,022	0,000
"for"loop stop conditions should be invariant	5,938	5,313	1,375	2,250	0,250	0,188	0,000
IP addresses should not be hardcoded	1,395	1,372	0,907	0,744	0,116	0,023	0,000
Redundant casts should not be used	7,400	6,200	5,400	3,600	0,800	1,000	0,000
Assignments should not be made from within sub-expressions	1,423	1,192	1,038	0,692	0,154	0,192	0,000

Referências Bibliográficas

Akbarinasaji *et al.* **(2016)** Shirin Akbarinasaji, Ayse Basar Bener e Atakan Erdem. Measuring the principal of defect debt. Em *Proceedings of the 5th International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering*, páginas 1–7. ACM. Citado na pág. 2, 47, 50, 51

Alves *et al.* **(2014)** Nicolli SR Alves, Leilane F Ribeiro, Vivyane Caires, Thiago S Mendes e Rodrigo O Spínola. Towards an ontology of terms on technical debt. Em *Managing Technical Debt (MTD), 2014 Sixth International Workshop on*, páginas 1–7. IEEE. Citado na pág. 7

Ampatzoglou *et al.* **(2015)** Areti Ampatzoglou, Apostolos Ampatzoglou, Alexander Chatzigeorgiou e Paris Avgeriou. The financial aspect of managing technical debt: A systematic literature review. *Information and Software Technology*, 64:52–73. Citado na pág. 9

Aniche *et al.* **(2016)** Maurício Aniche, Christoph Treude, Arie van Deursen e Marco Aurélio Gerosa. A validated set of smells in model-view-controller architectures. *International Conference Software Maintenance and Evolution*, 132. Citado na pág. 71

Beazley (2016) David Beazley. Ply. http://www.dabeaz.com/ply/, 2016. Acesso em 20 junho de 2016. Citado na pág. 57

Bhattacharya e Neamtiu (2011) Pamela Bhattacharya e Iulian Neamtiu. Bug-fix time prediction models: can we do better? Em *Proceedings of the 8th Working Conference on Mining Software Repositories*, páginas 207–210. ACM. Citado na pág. 50

Boegh *et al.* **(1999)** Jorgen Boegh, Stefano Depanfilis, Barbara Kitchenham e Alberto Pasquini. A method for software quality planning, control, and evaluation. *IEEE software*, 16(2):69. Citado na pág. 6, 7, 13

Brown *et al.* **(2010)** Nanette Brown, Yuanfang Cai, Yuepu Guo, Rick Kazman, Miryung Kim, Philippe Kruchten, Erin Lim, Alan MacCormack, Robert Nord, Ipek Ozkaya *et al.* Managing technical debt in software-reliant systems. Em *Proceedings of the FSE/SDP workshop on Future of software engineering research*, páginas 47–52. ACM. Citado na pág. 7

Cunnigham (1992) Ward Cunnigham. Vídeo de ward cunnigham sobre dívida técnica, transcrito por june kim e lawrence wang. http://c2.com/cgi/wiki?WardExplainsDebtMetaphor, 1992. Acesso em 10 junho de 2015. Citado na pág. 1, 5

Curtis (2012) Bill Curtis. Paying down the interest on your applications. Citado na pág. 50

Curtis *et al.* **(2012a)** Bill Curtis, Jay Sappidi e Alexandra Szynkarski. Estimating the principal of an application's technical debt. Citado na pág. 2, 47, 49, 50, 51

Curtis *et al.* **(2012b)** Bill Curtis, Jay Sappidi e Alexandra Szynkarski. Estimating the size, cost, and types of technical debt. Em *Proceedings of the Third International Workshop on Managing Technical Debt*, páginas 49–53. IEEE Press. Citado na pág. 50

Delamaro *et al.* (2007) Márcio Eduardo Delamaro, José Carlos Maldonado e Mário Jino. *Introdução ao teste de software.* Editora Campus. Citado na pág. 16

Draper *et al.* (1966) Norman Richard Draper, Harry Smith e Elizabeth Pownell. *Applied regression analysis*, volume 3. Wiley New York. Citado na pág. 55

Ernst *et al.* (2015) Neil A Ernst, Stephany Bellomo, Ipek Ozkaya, Robert L Nord e Ian Gorton. Measure it? manage it? ignore it? software practitioners and technical debt. Em *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, páginas 50–60. ACM. Citado na pág. 2, 5, 8, 47, 50, 51

Falessi e Reichel (2015) Davide Falessi e Andreas Reichel. Towards an open-source tool for measuring and visualizing the interest of technical debt. Em *Managing Technical Debt (MTD), 2015 IEEE 7th International Workshop on*, páginas 1–8. IEEE. Citado na pág. 2, 47, 50

Falessi *et al.* (2013) Davide Falessi, Michele A Shaw, Forrest Shull, Kathleen Mullen e Mark Stein Keymind. Practical considerations, challenges, and requirements of tool-support for managing technical debt. Em *Managing Technical Debt (MTD), 2013 4th International Workshop on*, páginas 16–19. IEEE. Citado na pág. 50

Fowler (1999) Martin Fowler. *Refactoring: improving the design of existing code.* Pearson Education India. Citado na pág. 15

Fowler (2009) Martin Fowler. Technical debt quadrant. http://martinfowler.com/bliki/TechnicalDebtQuadrant.html, 2009. Acesso em 10 julho de 2015. Citado na pág. 1, 6

Freire () Alexandre Freire. Dívida técnica: precisando de crédito? ou "como evitar que o cobrador bata na sua porta. http://ccsl.ime.usp.br/pt-br/divida-tecnica-precisando-de-credito-ou-como-evitar-que-o-cobrador-bata-na-sua-porta. Acesso em 20 junho de 2015. Citado na pág. 21

Freund e Minton (1979) Rudolf Jakob Freund e Paul D Minton. *Regression methods: a tool for data analysis.* Number 519.536 F889. Citado na pág. 55

Griffith *et al.* (2014) Isaac Griffith, Derek Reimanis, Clemente Izurieta, Zadia Codabux, Ajay Deo e Byron Williams. The correspondence between software quality models and technical debt estimation approaches. Em *Managing Technical Debt (MTD), 2014 Sixth International Workshop on*, páginas 19–26. IEEE. Citado na pág. 50

Guo *et al.* (2016) Yuepu Guo, Rodrigo Oliveira Spínola e Carolyn Seaman. Exploring the costs of technical debt management–a case study. *Empirical Software Engineering*, 21(1):159–182. Citado na pág. 5, 6, 48

Hoffmann e Vieira (1977) Rodolfo Hoffmann e Sônia Vieira. *Análise de regressão: uma introdução à econometria.* Editora da Universidade de São Paulo. Citado na pág. 54

Holvitie e Leppänen (2013) Johannes Holvitie e Ville Leppänen. Debtflag: Technical debt management with a development environment integrated tool. Em *Proceedings of the 4th International Workshop on Managing Technical Debt*, páginas 20–27. IEEE Press. Citado na pág. 47, 48, 51

Kruchten *et al.* (2012) Philippe Kruchten, Robert L Nord e Ipek Ozkaya. Technical debt: From metaphor to theory and practice. *Ieee software*, 29(6). Citado na pág. 47, 48

Letouzey (2012a) Jean-Louis Letouzey. *The SQALE Method - Definition Document.* Citado na pág. 17, 18, 19, 26, 41, 42, 44

Letouzey (2012b) Jean-Louis Letouzey. The sqale method for evaluating technical debt. Em *Proceedings of the Third International Workshop on Managing Technical Debt*, páginas 31–36. IEEE Press. Citado na pág. 1, 9, 10, 11, 16, 47, 51, 60

Marinescu (2012) Radu Marinescu. Assessing technical debt by identifying design flaws in software systems. *IBM Journal of Research and Development*, 56(5):9–1. Citado na pág. 2, 47, 49

McConnell (2007) Steve McConnell. Managing technical debt. http://www.construx.com/10x_Software_Development/Technical_Debt, 2007. Acesso em 10 junho de 2016. Citado na pág. 1, 6

Metrixware (2016) Metrixware. Kiuwan. https://www.kiuwan.com/, 2016. Acesso em 20 junho de 2016. Citado na pág. 37

Nugroho *et al.* (2011) Ariadi Nugroho, Joost Visser e Tobias Kuipers. An empirical model of technical debt and interest. Em *Proceedings of the 2nd Workshop on Managing Technical Debt*, páginas 1–8. ACM. Citado na pág. 2, 47, 48, 51

R-Project (2016) R-Project. R-project. https://www.r-project.org/, 2016. Acesso em 20 junho de 2016. Citado na pág. 73

Ramasubbu e Kemerer (2013) Narayan Ramasubbu e Chris F Kemerer. Towards a model for optimizing technical debt in software products. Em *Managing Technical Debt (MTD), 2013 4th International Workshop on*, páginas 51–54. IEEE. Citado na pág. 2, 47, 50, 51

Runeson e Höst (2009) Per Runeson e Martin Höst. Guidelines for conducting and reporting case study research in software engineering. *Empirical software engineering*, 14(2):131–164. Citado na pág. 65

Runeson *et al.* (2012) Per Runeson, Martin Host, Austen Rainer e Bjorn Regnell. *Case study research in software engineering: Guidelines and examples*. John Wiley & Sons. Citado na pág. 64, 65

S. Chin e Gat (2010) W. Bodwell S. Chin, E. Huddleston e I. Gat. The economics of technical debt. *Cutter IT Journal*, 23(10):11–15. Citado na pág. 2, 47, 48, 51

Seaman e Guo (2011) Carolym Seaman e Yuepo Guo. Measuring and monitoring technical debt. *Advances in Computers*, 82(6810):25–46. Citado na pág. 1, 5, 17, 21, 47, 48, 51

Software (2016) Cast Software. Cast application intelligence platform. http://www.castsoftware.com/, 2016. Acesso em 20 junho de 2016. Citado na pág. 36

Source () Sonar Source. Technical debt evaluation (sqale). http://www.sonarsource.com/products/plugins/governance/sqale/. Acesso em 20 junho de 2015. Citado na pág. 28, 53, 73

Source (2016) Sonar Source. Sonar qube. http://www.sonarqube.org/, 2016. Acesso em 20 junho de 2016. Citado na pág. 25, 26, 42

Tom *et al.* (2013) Edith Tom, AybüKe Aurum e Richard Vidgen. An exploration of technical debt. *Journal of Systems and Software*, 86(6):1498–1516. Citado na pág. 9

Tomas *et al.* (2013) P Tomas, M José Escalona e Manuel Mejias. Open source tools for measuring the internal quality of java software products. a survey. *Computer Standards & Interfaces*, 36(1):244–255. Citado na pág. 2, 47, 50, 51

Weiß *et al.* (2006) Cathrin Weiß, Rahul Premraj, Thomas Zimmermann e Andreas Zeller. Predicting effort to fix software bugs. *Issues*, 11. Citado na pág. 50, 51

Zazworka e Seaman (2013) Nico Zazworka e Carolyn Seaman. Identifying and managing technical debt. http://www.slideshare.net/zazworka/identifying-and-managing-technical-debt, 2013. Acesso em 24 junho de 2015. Citado na pág. 14, 15

Zazworka *et al.* **(2013)** Nico Zazworka, Rodrigo O Spínola, Antonio Vetro, Forrest Shull e Carolyn Seaman. A case study on effectively identifying technical debt. Em *Proceedings of the 17th International Conference on Evaluation and Assessment in Software Engineering*, páginas 42–47. ACM. Citado na pág. 1, 13

Zazworka *et al.* **(2014)** Nico Zazworka, Clemente Izurieta, Sunny Wong, Yuanfang Cai, Carolyn Seaman, Forrest Shull *et al.* Comparing four approaches for technical debt identification. *Software Quality Journal*, 22(3):403–426. Citado na pág. 13, 14, 15

Índice Remissivo