FABIANO SHOJI YOSCHITAKI

ANÁLISE DA RELEVÂNCIA DA DÍVIDA TÉCNICA IDENTIFICADA EXCLUSIVAMENTE POR MÉTODO AUTOMÁTICO

Monografia apresentada ao PECE – Programa de Educação Continuada em Engenharia da Escola Politécnica da Universidade de São Paulo como parte dos requisitos para conclusão do curso de MBA em Tecnologia de Software.

São Paulo
2016

FABIANO SHOJI YOSCHITAKI

ANÁLISE DA RELEVÂNCIA DA DÍVIDA TÉCNICA IDENTIFICADA EXCLUSIVAMENTE POR MÉTODO AUTOMÁTICO

Monografia apresentada ao PECE – Programa de Educação Continuada em Engenharia da Escola Politécnica da Universidade de São Paulo como parte dos requisitos para a conclusão do curso de MBA em Tecnologia de Software.

Área de Concentração: Tecnologia de Software

Orientador: Prof. Dr. Fábio Levy Siqueira

São Paulo
2016

Catalogação-na-publicação

Yoschitaki, Fabiano Shoji
ANÁLISE DA RELEVÂNCIA DA DÍVIDA TÉCNICA IDENTIFICADA EXCLUSIVAMENTE POR MÉTODO AUTOMÁTICO / F. Yoschitaki -- São Paulo, 2016.
89 p.

Monografia (MBA em Tecnologia de Software) - Escola Politécnica da Universidade de São Paulo. PECE – Programa de Educação Continuada em Engenharia.

1.Engenharia de Software 2.Qualidade de Software I.Universidade de São Paulo. Escola Politécnica. PECE – Programa de Educação Continuada em Engenharia II.t.

DEDICATÓRIA

Dedico este trabalho inteiramente aos meus pais, Massao e Mieka, pelo apoio incondicional que me dão todo o tempo.

AGRADECIMENTOS

Aos meus pais, Massao e Mieka, que são minhas maiores referências pessoais e à minha namorada, Sayuri, pessoas que sempre me apoiam e me acompanham em todos os aspectos da minha vida.

À Universidade de São Paulo – USP, à Escola Politécnica da Universidade de São Paulo – EPUSP e ao PECE – Programa de Educação Continuada em Engenharia que abriram as portas para a realização deste curso.

Ao meu orientador, Prof. Dr. Fábio Levy Siqueira, pela atenção, objetividade, interesse e dedicação ímpares em todos os momentos. Tornou-se para mim uma grande referência pessoal e profissional.

Ao colega de curso e amigo Everton Nakaharada, pelas diversas vezes em que me incentivou e ajudou para a conclusão deste trabalho.

A todos os bons professores que me guiaram durante o curso.

RESUMO

No universo do desenvolvimento de software, dívida técnica é uma metáfora da dívida financeira que ajuda a simplificar o entendimento dos problemas que podem surgir ao optar por fazer as coisas rapidamente, mas de forma não ideal, tornando tarefas como a manutenção e evolução do software mais custosas. Há diversas ferramentas e abordagens existentes que apoiam o gerenciamento de dívida técnica em um software, tais como as que identificam dívida técnica. Este trabalho tem como objetivo analisar se o conjunto de dívida técnica identificado exclusivamente por meio de um método automatizado é relevante para desenvolvedores. Isto foi realizado por meio de um estudo de caso em que duas abordagens de identificação de dívida técnica – manual e automática – foram aplicadas em um software real e seus resultados foram analisados e discutidos. Os resultados deste estudo mostraram que a utilização conjunta de ferramentas e humanos para identificação de dívida técnica traz melhores resultados do que somente uma das abordagens, além de ter fornecido uma resposta inicial ao objetivo principal do trabalho ao concluir que o conjunto de dívida técnica identificado somente pela ferramenta foi considerado relevante pelos desenvolvedores que participaram neste estudo.

ABSTRACT

In the context of software development, the term technical debt is a metaphor used to simplify the understanding of some problems that may arise when one chooses to do things the quick way, making future software maintenance and evolution harder – the tradeoff between short and long term. There are several tools and approaches aiming to support technical debt management, such as the ones focused on identifying technical debt. This work aims to analyze whether the unique set of technical debt items identified by an automated approach is relevant for developers. This task was done by conducting a case study that applied two different technical debt approaches to a real software in order to analyze and discuss the results. The results of this study showed that using both tools and humans in the technical debt identification process yields better results than one of the approaches alone. Furthermore, this work provided an initial answer to the question “is there a value in using tools to identify technical debt that developers are not aware of?” by showing that developers classified this set of technical debt relevant.

LISTA DE ILUSTRAÇÕES

Figura 1 - Classificações da dívida técnica na visão de Fowler 16
Figura 2 - Os 7 eixos da qualidade 31
Figura 3 - Ciclo de vida do item do SonarQube 34
Figura 4 - Quantidade de itens por tipo 49
Figura 5 - Custo de dívida técnica por tipo 50
Figura 6 - Quantidade de juros por tipo 51
Figura 7 - Probabilidade de juros por tipo 52
Figura 8 - Dívida intencional por tipo 53
Figura 9 - Resultado da análise do módulo EJB pelo SonarQube 55
Figura 10 - Classificação de importância dos problemas no módulo EJB 55
Figura 11 - Resultado da análise da classe do módulo Web 56
Figura 12 - Importância dos itens por tipo 57
Figura 13 - Análise dos resultados entre os levantamentos manual e automático 59
Figura 14 - Quantidade de itens identificados por ambos os levantamentos separados por tag 60
Figura 15 – Percentagem de tags consideradas como dívida técnica por número de participantes 62
Figura 16 - Relevância de tags consideradas como dívida técnica por um participante 64
Figura 17 - Motivo de não ter identificado tags consideradas dívida técnica por um participante 65
Figura 18 - Relevância de tags consideradas como dívida técnica por dois participantes 66
Figura 19 - Motivo de não ter identificado tags consideradas dívida técnica por dois participantes 67
Figura 20 - Relevância de tags consideradas como dívida técnica por três participantes 68
Figura 21 - Motivo de não ter identificado tags consideradas dívida técnica por três participantes 69

Figura 22 - Relevância de tags consideradas como dívida técnica por todos os participantes 70
Figura 23 - Motivo de não ter identificado tags consideradas dívida técnica por todos participantes 71
Figura 24 - Motivos da não identificação dos itens de dívida técnica no levantamento manual 72

LISTA DE TABELAS

Tabela 1 - Modelo do item de dívida técnica por Seaman e Guo (2011) 39
Tabela 2 - Itens duplicados do levantamento manual no escopo de classe 45
Tabela 3 – Item do levantamento manual no escopo de método 46
Tabela 4 - Item do levantamento automático 46
Tabela 5 - Quantidade de itens identificados pelos participantes 48
Tabela 6 - Estrutura do questionário enviado aos participantes com os problemas encontrados somente pelo levantamento automático 61
Tabela 7 – Conjunto de tags que não foi considerado como dívida técnica pelos participantes 63
Tabela 8 – Conjunto de tags que foi considerado como dívida técnica por um participante 63
Tabela 9 - Conjunto de tags que foi considerado como dívida técnica por dois participantes 66
Tabela 10 - Conjunto de tags que foi considerado como dívida técnica por três participantes 68
Tabela 11 - Conjunto de tags que foi considerado como dívida técnica por todos os participantes 69

SUMÁRIO

1 INTRODUÇÃO 10
1.1 Motivação 10
1.2 Objetivo 10
1.3 Justificativa 11
1.4 Estrutura do Trabalho 11

2 DÍVIDA TÉCNICA 13
2.1 Conceito 13
2.2 Classificações 15
2.3 Causas 18
2.4 Consequências 20
2.5 Considerações do Capítulo 22

3 GERENCIAMENTO DE DÍVIDA TÉCNICA 23
3.1 Gerenciar o acúmulo de dívida técnica 23
3.2 Levantar a dívida técnica 24
3.3 Gerenciar o pagamento de dívida técnica 26
3.3.1 Decidir se a dívida conhecida deve ser paga 26
3.3.2 Pagar dívida inesperada quando encontrar 27
3.3.3 Priorizar o pagamento de dívida com maiores juros 27
3.3.4 Pagar dívida de forma incremental 27
3.3.5 Pagar dívida durante o desenvolvimento de novas funcionalidades 28
3.4 Considerações do Capítulo 28

4 MÉTODOS DE LEVANTAMENTO DE DÍVIDA TÉCNICA 29
4.1 Métodos Automatizados 29
4.1.1 Análise Estática Automática com FindBugs 29
4.1.2 Detecção de Violações de Modularidade com CLIO 30
4.1.3 SonarQube 30
4.2 Métodos Manuais 34
4.2.1 Monitoração da Atividade de Compreensão de Código do Desenvolvedor 35
4.2.2 Lista de Itens de Dívida Técnica 36
4.3 Considerações do Capítulo 39

5 ANÁLISE DE MÉTODOS DE LEVANTAMENTO DE DÍVIDA TÉCNICA 41
5.1 Trabalhos Relacionados 42

5.2 Contexto....44
5.3 Critérios de Análise....44
5.4 Aplicação do método de levantamento manual de dívida técnica....47
5.5 Resultados do método de levantamento manual de dívida técnica....48
5.6 Aplicação do método de levantamento automático de dívida técnica....53
5.7 Resultados do método de levantamento automático de dívida técnica....54
5.8 Análise dos Resultados....58
5.9 Apresentação da dívida técnica identificada exclusivamente pelo método automatizado aos participantes....60
5.10 Resultados da análise da dívida técnica encontrada exclusivamente pelo levantamento automático....62
5.11 Discussão....72

6 CONSIDERAÇÕES FINAIS....76
6.1 Conclusões....76
6.2 Contribuições do Trabalho....78
6.3 Trabalhos Futuros....79

REFERÊNCIAS....80
APÊNDICE A – LEVANTAMENTO MANUAL DE DÍVIDA TÉCNICA NO SISTEMA ACESSO A DADOS....82
APÊNDICE B – RESULTADO DO LEVANTAMENTO MANUAL DE DÍVIDA TÉCNICA NO SISTEMA ACESSO A DADOS....85
APÊNDICE C – RESULTADO DO LEVANTAMENTO AUTOMÁTICO DE DÍVIDA TÉCNICA NO SISTEMA ACESSO A DADOS....87

1 INTRODUÇÃO

1.1 Motivação

As tarefas de construção e manutenção de sistemas de software podem conter elementos, muitas vezes de forma despercebida, que trazem impactos negativos para as empresas, principalmente no contexto econômico. Um deles, tema central deste trabalho, é a dívida técnica.

A dívida técnica é uma metáfora dentro do universo de desenvolvimento de software que simplifica o entendimento, principalmente para pessoas não técnicas, dos problemas que podem surgir ao optar por fazer as coisas rapidamente, mas de forma não ideal – o *tradeoff* entre o curto e o longo prazo -, assim como adquirir uma dívida financeira: o custo de ter o benefício adiantado ao fazer um empréstimo traz consigo juros. Neste caso, os juros sobre a dívida técnica vêm na forma de esforço extra a ser realizado em atividades futuras.

No entanto, além de ser útil como metáfora que facilita a comunicação do problema, a dívida técnica também tem inspirado o desenvolvimento de diversas ferramentas e técnicas com o objetivo de apoiar na identificação, medição, monitoramento e gestão como um todo da dívida técnica.

1.2 Objetivo

O objetivo deste trabalho é analisar quanto da dívida técnica identificada exclusivamente por meio de um método automatizado é considerada relevante para os desenvolvedores. Para essa análise, serão aplicados dois métodos de levantamento de dívida técnica em um sistema de software existente, um de forma automatizada por meio da ferramenta SonarQube e o outro manual, baseado na criação da lista de itens de dívida técnica proposto por Seaman e Guo (2011), comparando seus resultados e tentando responder se o subconjunto de dívida técnica encontrado pela ferramenta automatizada, porém não pelo método manual,

é considerado real e relevante para os desenvolvedores, ou seja, se é dívida técnica que traz reais impactos ao desenvolvimento futuro.

1.3 Justificativa

No trabalho conduzido por Zazworka et al. (2013), em que o principal objetivo foi avaliar se ferramentas de identificação automática de dívida técnica têm a capacidade de encontrar itens identificados manualmente por desenvolvedores, foi levantada a seguinte questão: quanto da dívida técnica potencial apontada pelas ferramentas, porém não pelos desenvolvedores, é real? Há benefícios em utilizar ferramentas que identificam dívida técnica que os desenvolvedores não têm conhecimento ou este subconjunto é irrelevante? Partindo desta dúvida, o presente trabalho tem como proposta prover uma resposta inicial.

1.4 Estrutura do Trabalho

Este trabalho está estruturado da seguinte forma:

O Capítulo 1, INTRODUÇÃO, apresenta as motivações, o objetivo, as justificativas e a estrutura do trabalho.

O Capítulo 2, DÍVIDA TÉCNICA, apresenta o conceito de dívida técnica, suas classificações, causas e consequências na construção e manutenção de software.

O Capítulo 3, GERENCIAMENTO DE DÍVIDA TÉCNICA, apresenta algumas abordagens e tarefas voltadas para o gerenciamento da dívida técnica, mostrando sua importância e sugerindo diretrizes gerais.

O Capítulo 4, MÉTODOS DE LEVANTAMENTO DE DÍVIDA TÉCNICA, descreve algumas abordagens – manuais e automáticas – utilizadas para a tarefa de identificação de dívida técnica.

O Capítulo 5, ANÁLISE DE MÉTODOS DE LEVANTAMENTO DE DÍVIDA TÉCNICA, apresenta um estudo de caso realizado em uma empresa em que dois métodos de identificação de dívida técnica – um manual e outro automático – foram aplicados em

um software e seus resultados foram analisados um em relação ao outro, com o intuito de entender a relevância dos resultados do método automático.

O Capítulo 6, CONSIDERAÇÕES FINAIS, apresenta as conclusões, as contribuições do trabalho e analisa possíveis trabalhos futuros.

REFERÊNCIAS relaciona as citações e referências utilizadas no trabalho.

APÊNDICE A apresenta um documento enviado aos respondentes do levantamento manual de itens de dívida técnica no realizado no sistema do estudo, tendo como objetivo auxiliá-los na tarefa de levantamento realizada no Capítulo 5.

APÊNDICE B apresenta uma tabela com o resultado do levantamento manual realizado pelos participantes no Capítulo 5.

APÊNDICE C apresenta uma tabela com o resultado do levantamento automático realizado pela ferramenta SonarQube no Capítulo 5.

2 DÍVIDA TÉCNICA

Este capítulo tem como objetivo a apresentação de referências teóricas sobre o conceito de dívida técnica, suas classificações, causas e consequências no contexto da construção e manutenção de software.

2.1 Conceito

O termo dívida técnica é uma metáfora definida por Ward Cunningham em 1992 (CUNNINGHAM, 1992), publicada em um relatório na conferência OOPSLA (*Object-Oriented Programming, Systems, Languages & Applications*), na tentativa de facilitar o entendimento do problema que pode ser causado pela escolha de fazer as coisas de modo rápido ao invés de devagar, no contexto de desenvolvimento de software. O modo rápido gera dívida técnica, pois, apesar do aspecto positivo de acelerar o desenvolvimento, faz com que a qualidade do software diminua por conta dos atalhos tomados, resultando no aumento da dificuldade das tarefas de alteração e manutenção futuras. O modo devagar provavelmente é mais organizado e melhor estruturado, não deixando obstáculos para as atividades de alteração e manutenção futuras (FOWLER, 2003a).

Nugroho et al. consideram a dívida técnica como o custo necessário para elevar a qualidade do software a um nível considerado ideal e os juros da dívida técnica como o custo adicional utilizado na manutenção do software causados pela baixa qualidade (NUGROHO; VISSER; KUIPERS, 2011).

De acordo com Rubin (2012), além da dívida técnica se referir aos atalhos propositais para acelerar o desenvolvimento, a indústria de software atribuiu outros significados à definição inicial de Cunningham, tais como:

- **Projeto inadequado:** projeto que não agrega mais valor, seja pelas mudanças ocorridas no negócio ou na tecnologia;
- **Defeitos:** existência de problemas já conhecidos no software que ainda não foram solucionados por falta de investimento de tempo;

- **Falta de testes de cobertura:** trechos do software onde mais testes deveriam ser realizados;
- **Excesso de testes manuais:** testar de forma manual quando o correto seria automatizar o processo;
- **Falta de gestão da integração e liberação do software:** execução destas atividades de forma demorada e propensa à ocorrência de erros e
- **Falta de experiência:** falta de profissionais com conhecimento e experiência mínimos necessários para garantir a qualidade do software.

Outra definição para a dívida técnica é a de Seaman e Guo (2011), a qual se refere aos efeitos negativos causados por qualquer artefato imaturo, incompleto ou inadequado durante o desenvolvimento de software, de modo que o desenvolvimento futuro e a manutenção são prejudicados.

A metáfora se assemelha à dívida financeira quanto ao pagamento de juros. Na dívida financeira, fazer um empréstimo de capital gera um custo adicional – os juros – para o devedor na forma do próprio capital emprestado. No caso da dívida técnica, os juros vêm na forma de trabalho extra no desenvolvimento futuro. Assim como na dívida financeira, pode-se optar por pagar juros enquanto a dívida existe, ou pagar toda a dívida de uma vez (na forma de refatoração do projeto), livrando-se da dívida e dos juros relacionados a ela (FOWLER, 2003a).

Assim como uma empresa ou pessoa opta por fazer um empréstimo, adquirindo também uma dívida, seja para obter vantagens de oportunidade de mercado ou comprar um produto com preço promocional, a metáfora também explica porque há casos onde pode ser vantajoso escolher pelo modo rápido, como atingir um importante prazo de entrega, ou seja, há um benefício no curto prazo. O problema são os casos em que se perde o controle da situação, prejudicando e desperdiçando os recursos no desenvolvimento futuro por conta do pagamento de juros (FOWLER, 2003a). Assumir dívida técnica pode produzir valor mais cedo, porém é necessário livrar-se dela o mais rápido possível (FOWLER, 2009).

Apesar das semelhanças com a dívida financeira, uma grande diferença com esta é que a dívida técnica é difícil de ser mensurada de forma eficaz (FOWLER, 2003b). O pagamento de juros – trabalho e esforço extras no desenvolvimento – de fato prejudica uma equipe, mas a dificuldade para medir produtividade (FOWLER, 2003b) torna obscuro o entendimento das verdadeiras consequências da dívida técnica.

Embora não seja necessário pagar a dívida em uma data fixa – e talvez ela nunca seja paga –, normalmente é recomendado fazer isso (reescrevendo o código ou resolvendo o problema) para as partes que afetam negativamente a equipe ou o cliente de forma significativa (ALLMAN, 2012). Isso é recomendado pois a dívida técnica traz riscos ao projeto, podendo complicar o gerenciamento do software quanto à tomada de decisão de quais dívidas, qual quantidade e quando devem ser pagas (SEAMAN; GUO, 2011).

Do ponto de vista de Zazworka et al. (2013), além de a metáfora ter facilitado a comunicação e discussão entre profissionais e pesquisadores sobre os *tradeoffs* técnicos em sistemas de software, o termo também ajudou a criar um vocabulário familiar ligado ao contexto financeiro e tem o potencial de se estabelecer como linguagem comum para este fim (ZAZWORKA et al., 2013). Outro ponto é que, além de servir somente como metáfora que facilita o entendimento do problema, a dívida técnica também inspirou e abriu portas para que ferramentas e técnicas fossem desenvolvidas com o objetivo de apoiar nas tarefas de gerenciamento da dívida técnica, como as ferramentas e métodos que a identificam, essenciais para torna-la explícita e gerenciável.

2.2 Classificações

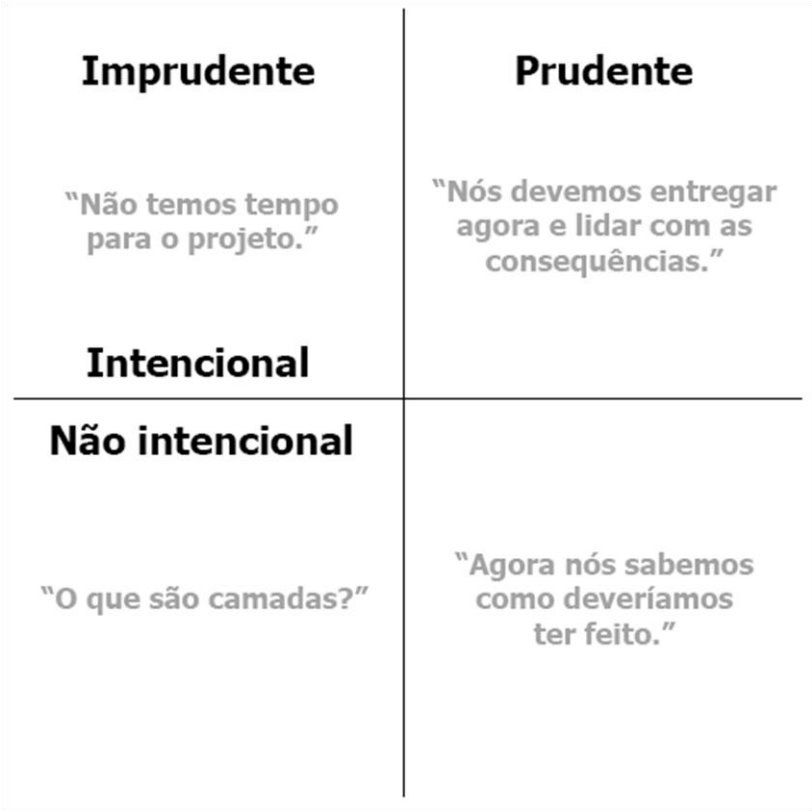
Fowler classifica as origens da dívida técnica em um quadrante, como apresentado na Figura 1, classificando-as como dívidas prudentes ou imprudentes (FOWLER, 2009). Há, além desta distinção, a diferença entre dívida intencional e não intencional.

No caso da dívida prudente e intencional, a equipe tem consciência de que está adquirindo a dívida e também das vantagens e desvantagens de obtê-la, sendo

comuns casos em que a vantagem de entregar mais rápido seja maior do que a desvantagem de entrar em dívida.

Em outra situação, uma equipe pode adquirir dívidas de modo imprudente sem nem mesmo ter consciência disto, ou seja, dívida não intencional, seja por não ter conhecimento de melhores práticas de projeto, falta de atenção, não aderência a padrões de desenvolvimento, falta de casos de teste etc. (FOWLER, 2009).

Figura 1 - Classificações da dívida técnica na visão de Fowler



Fonte: Adaptado de Fowler (2009)

Ainda de acordo com a classificação de Fowler, há também a dívida imprudente e intencional: a equipe até pode saber e conhecer boas práticas de projeto e até executá-las, mas optam pelo caminho rápido porque acham que podem não ter tempo suficiente para fazer da melhor forma.

Para fechar o quadrante da dívida técnica, Fowler cita a dívida prudente e não intencional que, embora pareça estranha, principalmente no contexto financeiro, ele

acredita que este tipo de dívida técnica seja inevitável mesmo para equipes que tenham excelentes profissionais. No exemplo dado pelo autor, ouviu de um colega um caso em que o projeto de software havia sido entregue com valor: o cliente estava feliz e o código era limpo, entretanto, apesar do bom trabalho que havia sido feito, somente ao final do projeto a equipe visualizava a melhor forma que o projeto poderia ter sido desenvolvido. Isto porque enquanto o desenvolvedor está programando ele também está aprendendo, sendo comuns os casos em que são necessários meses ou anos de programação em um determinado projeto para depois perceber que há melhores abordagens que poderiam ter sido feitas. Isto configuraria uma dívida não intencional, embora prudente (FOWLER, 2009).

A decisão do pagamento de juros ou pagar o total da dívida ainda se aplica, mantendo a metáfora útil também para este caso. Fowler (FOWLER, 2009) cita que este tipo de dívida é difícil de explicar aos gestores porque ela apareceu, reforçando a opinião de que é um tipo de dívida inevitável e, portanto, deve ser esperado que ocorra em algum momento, mesmo dentro das melhores equipes, caracterizando mais um motivo para não adicionar código sujo de forma imprudente.

McConnel (2007) classifica a dívida técnica de forma parecida com Fowler (2009): intencional e não intencional, sendo que a primeira se subdivide em mais dois tipos: curto prazo e longo prazo. A dívida técnica intencional de curto prazo é adquirida para suprir alguma necessidade imediata e, geralmente, não requer muito esforço no planejamento e não produz grandes impactos, podendo ser paga em um curto período de tempo. Já a de longo prazo é normalmente atrelada a um planejamento e está alinhada de forma estratégica, como o lançamento de um novo produto no mercado, em que o tempo é um fator importante. Esta forma de dívida técnica pode ser gerenciada durante anos, embora também possa ser paga de forma rápida, caso seja necessário.

A classificação da dívida técnica também pode ser associada às fases do ciclo de vida do software. Guo e Seaman (2011) classificam a dívida técnica em quatro diferentes tipos:

- **Dívida de documentação:** obtida quando a documentação está desatualizada ou inadequada;
- **Dívida de *design* (ou de código):** associada ao código fonte, tais como *code smells* (FOWLER et al., 1999), violações de modularidade e presença de *grime*;
- **Dívida de teste:** ocorre quando testes que foram planejados não são executados ou não cobrem todos os casos e
- **Dívida de defeito:** problemas conhecidos que não são consertados.

Outra classificação da dívida técnica é a feita por Rubin (2012), considerando a dívida técnica em três categorias:

- **Dívida técnica inesperada:** dívida que a equipe de desenvolvimento encontra casualmente enquanto trabalha em uma parte do sistema (por exemplo, quando uma nova funcionalidade está sendo desenvolvida, a equipe encontra uma solução paliativa desenvolvida há muito tempo por alguém que nem está mais na empresa);
- **Dívida técnica conhecida:** dívida que a equipe de desenvolvimento já sabe de sua existência e
- **Dívida técnica alvo:** dívida técnica conhecida e que foi escolhida para ser tratada ou quitada.

2.3 Causas

Há diversos fatores que favorecem o aparecimento de dívida técnica. Codabux e Wiiliams (2013) citam que a adoção de métodos ágeis, os quais começaram a ganhar popularidade em 2001 a partir do Manifesto Ágil, aumentaram a visibilidade da dívida técnica, pois o foco na funcionalidade – motivado pela entrega rápida do software -, tira de foco questões como *design***,** práticas recomendadas de programação, cobertura de testes e outros fatores que ajudam a evitar dívida técnica (CODABUX; WILLIAMS, 2013). Rubin (2012) descreve alguns motivos comuns que podem gerar dívida técnica:

- **Pressão para cumprir prazos:** dívida técnica pode surgir quando o tempo necessário para que o projeto seja desenvolvido com alta qualidade é menor do que o tempo disponível para sua conclusão. Neste caso, continuar com a velocidade de desenvolvimento, que garanta alta qualidade, resultará em entrega atrasada;
- **Acelerar a velocidade de desenvolvimento:** na situação descrita anteriormente, para que seja mantida a qualidade, é necessário escolher entre diminuir o escopo da entrega ou adiar a data desejada. Entretanto, é frequente o caso em que ambas as opções são rejeitadas pela gerência e a equipe de desenvolvimento tenha de acelerar o trabalho para entregar o escopo inteiro no prazo desejado, fazendo com que dívida técnica seja acumulada por conta dos atalhos escolhidos a fim de cumprir o cronograma de modo acelerado;
- **Testar pouco:** há um mito que considera que a realização de testes é uma despesa adicional, de modo que, reduzindo-a, fará com que a velocidade de trabalho seja maior, entregando o projeto mais rapidamente. Na realidade, quando se opta por testar menos (ou não testar), problemas que não são detectados pelos testes permanecerão no software e o custo para resolvê-los no futuro demandará mais esforço e tempo do que se tivessem sido detectados antes e
- **Dívida gera mais dívida:** a existência de dívida técnica facilita o acúmulo de mais dívida sobre o montante, pois ao prejudicar a velocidade do desenvolvimento das entregas futuras, além da pressão para que todas as funcionalidades sejam entregues no prazo desejado, o resultado provável será uma entrega de baixa qualidade, desenvolvido pela equipe pressionada pelo tempo. Caso esse cenário mantenha se repetindo, o sistema poderá chegar a um ponto em que a dívida técnica acumulada é tão grande que se torna inviável evoluir ou alterar o software.

2.4 Consequências

Projetos que não têm sua dívida técnica monitorada e controlada podem sofrer diversas consequências negativas, tais como: custo acima do previsto, problemas de qualidade, dificuldade para adicionar novas funcionalidades sem causar efeitos colaterais e até mesmo se tornarem inutilizáveis de forma antecipada (RUBIN, 2012).

De acordo com Rubin (2012), a quantidade da dívida técnica é proporcional à gravidade das consequências. Algumas destas consequências são:

- **Ponto de Virada imprevisível**: um aspecto importante da dívida técnica é que ela cresce de uma forma imprevisível. Cada pedaço de dívida técnica, quando somada no montante de dívida, pode trazer prejuízos maiores do que se fossem consideradas de forma separada. Em algum ponto, a dívida técnica atinge um tipo de “massa crítica”, onde o software atinge o Ponto de Virada e não se mantém mais gerenciável. No Ponto de Virada, mesmo pequenas alterações no produto se tornam grandes incertezas. Isso traz risco significativo ao negócio;
- **Alto tempo de entrega**: Assumir dívida técnica significa obter um empréstimo hoje contra o tempo necessário para fazer o trabalho futuro. Quanto maior a dívida hoje, mais lenta a velocidade amanhã. Quando a velocidade diminui, leva mais tempo para entregar novas características e correções aos clientes. Então, na presença de muita dívida técnica, o tempo entre as entregas aumenta em vez de diminuir. Em mercados altamente competitivos, dívida técnica vai ativamente contra os melhores interesses;
- **Alta quantidade de defeitos**: softwares que acumulam quantidade significativa de dívida técnica se tornam mais complexos, dificultando as tarefas de manutenção e alteração, criando um cenário propício para o aumento de defeitos que, por sua vez, pode causar falhas críticas frequentemente. Além disso, a sobrecarga gerada por ter que gerenciar muitos defeitos consome tempo e recursos disponíveis que poderiam ser utilizados para a evolução do sistema;

- **Altos custos de desenvolvimento e de apoio:** conforme a dívida técnica aumenta, os custos de desenvolvimento e apoio começam a subir, isto porque o que costumava ser simples e barato de fazer se torna complicado e caro, mesmo que sejam simples e pequenas mudanças;
- **Atrofia do software:** devido à dificuldade de evoluir o software por causa da dívida técnica, novas funcionalidades deixam de ser adicionadas e defeitos existentes permanecem, fazendo com que o software se torne cada vez menos interessante ou viável para os clientes, que na primeira oportunidade não hesitarão em trocá-lo por outra opção;
- **Previsibilidade reduzida:** a presença de alta dívida técnica torna mais difícil realizar previsões, mesmo que sejam realizadas por times experientes. Nesse cenário de muita incerteza, as estimativas de prazo falham frequentemente, prejudicando o comprometimento da equipe com relação às entregas e prazos;
- **Baixo desempenho no desenvolvimento:** conforme a dívida técnica vai se acumulando, também aumenta a expectativa de que o desempenho da equipe de desenvolvimento seja cada vez pior, limitando as possibilidades;
- **Frustração:** o acúmulo da dívida técnica gera frustração a todos os envolvidos. A equipe de desenvolvimento faz de tudo para não ter que lidar com o software, já que qualquer atividade se torna dolorosa. Essa frustração pode até fazer com que os melhores profissionais busquem oportunidades em outros lugares, o que só faz o cenário da dívida técnica piorar, já que eles seriam os recursos mais adequados para resolver o problema. No entanto, pessoas da área de negócio também são afetadas por não conseguirem manter o comprometimento com os clientes. Os clientes, igualmente frustrados com erros e prazos perdidos, perdem a confiança no produto; e
- **Baixa satisfação do cliente:** do ponto de vista do cliente, quanto maior a frustração, menor a satisfação com relação ao software. Isto mostra que as consequências da dívida técnica não se restringem somente à equipe de desenvolvimento, de modo geral, todos são prejudicados.

2.5 Considerações do Capítulo

Este capítulo contextualizou o conceito de dívida técnica, tema central deste trabalho, explicando a metáfora e fazendo paralelos com características da dívida no contexto financeiro.

Também apresentou suas classificações de acordo com diferentes autores, mostrando que não há uma única visão dos tipos de dívida técnica, além de suas causas e consequências, sendo importantes especialmente para a discussão dos resultados dos levantamentos realizados no Capítulo 5 e para a conclusão do trabalho no Capítulo 6.

3 GERENCIAMENTO DE DÍVIDA TÉCNICA

A fim de mitigar o prejuízo que a dívida técnica pode causar ao projeto, dificultando sua manutenção e/ou evolução – podendo chegar ao ponto de cancelá-lo – é necessário ter o controle sobre a dívida. Isto pode ser obtido através de seu gerenciamento, mantendo-a baixa o suficiente para causar o menor impacto possível no desenvolvimento futuro. É importante notar que enquanto o software estiver ativo, dificilmente estará totalmente livre de dívida técnica, até porque em alguns casos, a relação custo-benefício para removê-la não é justificável do ponto de vista econômico. Durante o gerenciamento, é importante envolver, além dos profissionais técnicos, pessoas da esfera de negócio, pois essa combinação facilita chegar a decisões mais equilibradas, evitando que o resultado final atenda exclusivamente uma perspectiva (RUBIN, 2012).

Algumas abordagens para o gerenciamento da dívida foram propostas, como o *framework* de gerenciamento de dívida técnica de Seaman e Guo (2011), em que o gerenciamento foi dividido em três estágios: identificar, medir e monitorar dívida técnica. Tal mecanismo foi concebido de forma a ser flexível e possibilitar a inclusão de julgamento humano em quaisquer estágios, permitindo, desta forma, que possa ser estendido para futuros trabalhos.

Neste capítulo, serão apresentadas as atividades da abordagem de gerenciamento de dívida técnica proposta por Seaman e Guo (2011) e também algumas diretrizes da abordagem de Rubin. O presente trabalho tem como foco principal a atividade de identificação de dívida técnica.

3.1 Gerenciar o acúmulo de dívida técnica

Rubin (2012) sugere a tarefa de gerenciar o processo de acúmulo de dívida técnica, acompanhando seu tamanho de modo a evitar que o tamanho não ultrapasse os limites saudáveis dentro do sistema. Como importante iniciativa, é preciso evitar que a dívida técnica ingênua se acumule no sistema. Isso pode ser feito através do uso de boas práticas de desenvolvimento, tais como manter o design simples,

desenvolvimento orientado a testes, integração contínua, testes automatizados (RUBIN, 2012).

Para o caso da dívida técnica que já existe no sistema, utilizar a técnica de refatoração de código ajuda a organizá-lo, diminuindo sua complexidade, trazendo benefícios diretos como o aumento da manutenibilidade e extensibilidade.

Um importante fator para o surgimento de dívida técnica é quando, durante a construção de uma funcionalidade, uma tarefa que deveria ter sido concluída acabou sendo postergada. Para mitigar este problema, Rubin (2012) propõe que, quanto mais sólida e clara for a definição de que uma tarefa está de fato concluída, menor a possibilidade de acumular dívida técnica e também o esforço para trabalhar no projeto.

Outra medida importante para gerenciar o acúmulo de dívida técnica é entender corretamente a importância e o impacto da dívida técnica do ponto de vista econômico. De acordo com Rubin (2012), é comum que a dívida técnica seja acumulada no sistema de forma desnecessária porque as empresas não compreendem bem as implicações que a dívida pode gerar. Isto acontece porque muitos fatores não são tão óbvios e/ou não são considerados na decisão. Por exemplo, se a equipe decide pagar uma parte da dívida técnica, durante este período os recursos não estarão mais disponíveis para trabalhar em outro sistema ou em novas funcionalidades, representando um custo de oportunidade que provavelmente não será considerado na tomada de decisão. Outro fator não considerado é a probabilidade que o negócio favoreça e incentive o desenvolvimento de novas funcionalidades do sistema – que de fato trazem valor ao negócio – em vez de dispender esforço, pagando dívida técnica, em funcionalidades já existentes.

3.2 Levantar a dívida técnica

Para levantar a dívida técnica, é importante que ambas as equipes de desenvolvimento e de negócio tenham visibilidade do problema, cada qual sob sua perspectiva, possibilitando a comunicação e a tomada de decisão mais efetivas.

Uma das formas utilizadas para quantificar a dívida técnica, deixando visíveis suas consequências para o negócio, é acompanhar a velocidade ao longo do tempo (RUBIN, 2012). Caso a equipe de desenvolvimento tenha a velocidade diminuída, por consequência o tempo para completar determinada tarefa será maior, representando, desta forma, o custo financeiro do pagamento de juros sobre a dívida técnica acumulada.

Do lado técnico, geralmente a equipe sabe onde estão as maiores quantidades de dívida técnica presentes no sistema. Entretanto, não organizar essas informações torna difícil a análise e tomada de decisão sobre tais itens. Rubin (2012) apresenta três iniciativas para manter a dívida técnica visível no âmbito técnico:

- **Utilizar sistema para acompanhamento de defeitos:** esta abordagem considera itens de dívida técnica como se fossem defeitos, adicionando-os a algum sistema para acompanhamento de defeitos, tendo como vantagem o uso de ferramentas ou técnicas já conhecidas;
- **Adicionar dívida técnica no *product backlog*:** neste caso, itens de dívida técnica são tratados como itens de *product backlog*, tendo tanta visibilidade quanto às novas funcionalidades. Neste caso, quando o custo de dívida técnica é significativamente alto, pode-se envolver o *Product Owner* (PO) para priorizar e ordenar itens da dívida técnica no *product backlog* juntamente com os itens de novas funcionalidades; e
- **Criar *backlog* apenas para dívida técnica:** outra abordagem é criar uma lista somente com itens de dívida técnica e adicioná-los conforme novos itens são descobertos ou inseridos propositalmente no sistema. Desta forma, cria-se visibilidade à equipe técnica, possibilitando estabelecer quando e quais itens de dívida técnica serão pagos.

Há diversos trabalhos que aplicaram abordagens e utilizaram ferramentas para a identificação da dívida técnica, tanto de forma automática – como a plataforma SonarQube e ferramentas como FindBugs, CodeVizard e CLIO - utilizadas e comparadas no trabalho de Zazworka et al. (2014) -, quanto de forma manual, como o questionário para avaliação subjetiva de *code smells* (MANTYLA; VANHANEN;

LASSENIUS, 2004), a estratégia de elicitação manual do caso de uso aplicada por Zazworka et al. (2013) e a criação lista de itens de dívida técnica sugerida por Seaman e Guo (2011). No capítulo 4, algumas dessas abordagens serão apresentadas e discutidas.

3.3 Gerenciar o pagamento de dívida técnica

Rubin (2012) sugere algumas diretrizes para gerenciar o pagamento da dívida técnica, considerando sua própria classificação de tipos de dívida técnica. A seguir, serão apresentadas essas diretrizes: decidir se a dívida conhecida deve ser paga, pagar dívida inesperada quando encontrar, priorizar o pagamento de dívida com os maiores juros, pagar dívida de forma incremental e pagar dívida durante o desenvolvimento de novas funcionalidades.

3.3.1 Decidir se a dívida conhecida deve ser paga

Antes de pagar dívida técnica, é necessário avaliar o contexto para decidir se há boas razões para isto, pois diferentemente da dívida financeira há alguns casos em que a dívida técnica acumulada no sistema não deve ser quitada. Alguns destes casos são (RUBIN, 2012):

- **Sistema será desativado em breve:** se um sistema está prestes a ser desativado, não há benefícios em pagar sua dívida técnica, principalmente nos casos em que o sistema oferece pouco valor ao usuário;
- **Protótipo descartável:** o processo de desenvolvimento de um protótipo tende a acumular dívida técnica, porém a própria condição de ser protótipo sugere que ele seja feito para depois ser descartado, não havendo necessidade de pagar a dívida técnica e
- **Sistema desenvolvido para operar no curto prazo:** sistemas que são desenvolvidos para operar por pouco tempo e que precisam ser lançados rapidamente ao mercado para obter vantagem competitiva. Para estes casos, pagar a dívida técnica não é economicamente viável.

3.3.2 Pagar dívida inesperada quando encontrar

Durante o desenvolvimento, é benéfico para todos que, ao encontrar dívida técnica inesperada, retire-a do sistema. Este benefício não é somente para quem o fez, mas para toda a equipe de desenvolvimento e a organização. Entretanto, como a tarefa pagar a dívida técnica demanda esforço, é necessário que esse esforço seja limitado, caso contrário o objetivo do atual trabalho pode não atingir seu prazo.

Para ajudar a resolver este problema, uma solução é reservar parte do tempo no orçamento para poder resolver essas dívidas inesperadas quando forem encontradas. Caso não seja possível resolver estas dívidas inesperadas, deve-se deixa-las visíveis e classifica-las como dívidas conhecidas.

3.3.3 Priorizar o pagamento de dívida com maiores juros

Nem todas as dívidas técnicas existentes no sistema têm a mesma relevância no aspecto do impacto negativo que elas podem causar. Uma parte do código do sistema que serve de base para diversas outras partes e que é frequentemente modificada, tornando-se cada vez mais complexa e difícil de entender, acumula cada vez mais dívida técnica – aumentando o valor dos juros – do que módulos que são raramente utilizados ou modificados. Portanto, deve-se analisar e classificar a dívida técnica, priorizando as partes em que a dívida mais prejudica o sistema na forma do pagamento de juros.

3.3.4 Pagar dívida de forma incremental

Deixar que a dívida chegue a uma situação em que seja necessário dedicar recursos exclusivos para o seu pagamento atrasa a evolução do sistema, já que a equipe de desenvolvimento ficará dedicada para resolver um grande problema que poderia ter sido resolvido aos poucos, em vez de trabalhar no desenvolvimento de novas funcionalidades que trazem valor ao cliente.

Para evitar isto, quando a quantidade de dívida técnica acumulada no sistema é alta, é mais fácil e vantajoso pagar a dívida aos poucos, de forma incremental, do que paga-la toda de uma só vez, assim como acontece com a dívida financeira. Através

deste método, parte da dívida conhecida é qualificada como alvo para ser paga durante a próxima s*print.*

3.3.5 Pagar dívida durante o desenvolvimento de novas funcionalidades

Uma forma para reduzir a quantidade de dívida técnica conhecida com juros altos de forma incremental é fazê-la em paralelo com o desenvolvimento de novas funcionalidades que agregam valor ao sistema.

Se esta abordagem for executada em conjunto com o compromisso de não adicionar mais dívida técnica de forma ingênua, além de exercer o hábito de pagar dívida inesperada quando encontrar, Rubin (2012) cita, por exemplo, os seguintes benefícios: a consciência de que a dívida técnica afeta a todos e sua redução é responsabilidade de todo o time de desenvolvimento, melhor identificação das partes do código que são mais prejudicadas com a alta dívida técnica, evitar o desperdício de recursos ao pagar dívida técnica desnecessariamente.

3.4 Considerações do Capítulo

Este capítulo apresentou abordagens e atividades envolvendo o gerenciamento de dívida técnica. Em especial, a atividade de levantamento de dívida técnica é considerada central neste trabalho, pois será aprofundada no Capítulo 4 ao apresentar alguns dos métodos de levantamento de dívida técnica existentes, dois dos quais serão aplicados no Capítulo 5.

4 MÉTODOS DE LEVANTAMENTO DE DÍVIDA TÉCNICA

Abordagens existentes para levantar dívida técnica em sistemas de software podem ser divididas basicamente de duas formas: identificação manual, ou seja, dívida técnica extraída pelas pessoas, tais como desenvolvedores e outros *stakeholders*, e identificação automatizada, realizada por intermédio de ferramentas, por exemplo, a partir de análise de código fonte (ZAZWORKA et al., 2013). Neste capítulo, serão descritos alguns destes métodos e ferramentas, tanto manuais quanto automáticos.

4.1 Métodos Automatizados

Alguns tipos de dívida técnica podem ser detectados de forma automática por meio de ferramentas de análise estática no código. Resultados do trabalho de (ZAZWORKA et al., 2014) mostraram que a aplicação conjunta de diversas ferramentas para detecção de dívida técnica no mesmo projeto foi eficaz, pois cada abordagem encontrou problemas de diferentes naturezas no software, com pouca sobreposição em relação às outras. Entretanto, é importante citar que muitas ferramentas existentes podem ser eficientes para levantar dívida técnica de defeito, porém não de outros tipos (ZAZWORKA et al., 2013). Nesta seção, serão apresentadas algumas destas ferramentas.

4.1.1 Análise Estática Automática com FindBugs

Ferramentas de análise estática automática encontram problemas no nível de linha de código, tanto a partir do código fonte quanto do código compilado, não exigindo que o software a ser analisado esteja rodando no momento do levantamento (ZAZWORKA et al., 2014). Essas ferramentas procuram por violações de práticas recomendadas de programação que podem gerar problemas ou afetar a qualidade do software, sendo considerado um método barato de se aplicar. Tais ferramentas, além de apontar e identificar os problemas, também sugerem tratamentos e soluções (SEAMAN; GUO, 2011).

FindBugs[1] é uma ferramenta de análise estática automática que analisa o *bytecode* do sistema em busca de defeitos baseados em uma lista de padrões, tais como códigos que violam boas práticas de desenvolvimento, erro de concorrência, código maliciosos, problemas de desempenho e pontos de vulnerabilidade. Todos os potenciais defeitos apontados pela ferramenta são apresentados em uma lista, onde é possível averiguar e decidir quais são problemas de fato.

4.1.2 Detecção de Violações de Modularidade com CLIO

Grandes sistemas de software são formados por módulos, ou seja, subsistemas desenvolvidos que evoluem de forma independente. Quando componentes de módulos distintos evoluem juntos, há uma discrepância. Esta discrepância pode ser gerada, por exemplo, quando requisitos são alterados de forma que a arquitetura original não consegue mais se adaptar facilmente (ZAZWORKA et al., 2014). Com a presença destas discrepâncias, pode ocorrer um desvio da estrutura modular do software, que são as violações de modularidade.

Um exemplo deste tipo de dívida técnica é quando há necessidade de alterações em classes pertencentes a módulos que não são dependentes entre si. É desta forma que a ferramenta CLIO (WONG et al., 2011) detecta violações de modularidade: a partir da comparação entre o relacionamento de fato dos componentes e de como eles poderiam se relacionar, por exemplo, como citado anteriormente, quando componentes, dados como independentes, têm dependência para realização de alguma tarefa.

4.1.3 SonarQube

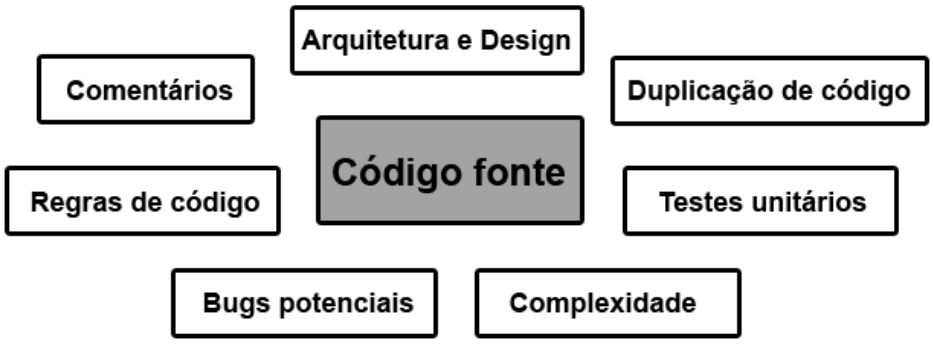
A ferramenta SonarQube, anteriormente chamada somente de Sonar, também realiza análise estática de código, sendo uma plataforma *web* de código aberto voltada para o gerenciamento da qualidade de código do software e da dívida técnica e capaz de realizar análise e medição da qualidade técnica (CAMPBELL; PAPAPETROU; GAUDIN, 2014). A plataforma é escrita na linguagem Java, mas

[1] http://findbugs.sourceforge.net/

permite analisar projetos em mais de 20 linguagens de programação (ABAP, Android, C/C++, C#, COBOL, Flex, Groovy, Java, JavaScript, Objective-C, PHP, PL/I, PL/SQL, Python, RPG, Swift, VB.NET, Visual Basic 6, HTML, JSP/JSF e XML) através do uso de diversos plug-ins, pagos e gratuitos, disponibilizando diferentes métricas sobre o projeto. Ferramentas como PMD, FindBugs e Checkstyle são utilizadas internamente pelo SonarQube para identificação de dívida técnica (CAMPBELL; PAPAPETROU; GAUDIN, 2014).

Além de apontar o que está de errado com o software e os possíveis problemas que poderão aparecer, a ferramenta também fornece meios para corrigi-los. Não somente a identificação de *bugs* é coberta pela ferramenta, mas também regras de código, duplicação de código, documentação, complexidade, testes e arquitetura. Os criadores do SonarQube chamam estes itens de sete eixos da qualidade ou sete pecados mortais dos desenvolvedores (SONARSOURCE, 2016), como pode ser visualizado pela Figura 2, devido à ocorrência de problemas a eles relacionados ser diretamente proporcional ao tamanho da dívida técnica no sistema, sendo possível rastrear esta dívida técnica através de regras e do mecanismo de problemas do SonarQube. Dos sete, o item *Bugs* é considerado o de maior importância, pois representa problemas existentes ou potenciais no software.

Figura 2 - Os 7 eixos da qualidade



Fonte: Adaptado de www.sonarqube.org

O princípio fundamental do SonarQube é trabalhar com dívida técnica, disponibilizando um conjunto de métricas e medições de esforço baseadas em moeda ou quantidade de dias (CAMPBELL; PAPAPETROU; GAUDIN, 2014). Até a versão 4.0 do software, o cálculo da dívida técnica era realizado através de um *plug-in* separado (*Technical Debt Plugin*[2]). A partir desta versão, o cálculo foi implementado no *core* da plataforma utilizando os fundamentos da metodologia SQALE (INSPEARIT, 2016) e disponibilizado de forma gratuita.

Após a instalação da plataforma, para calcular a dívida técnica dos projetos é necessário instalar os analisadores de código fonte. Ao executar a primeira análise, um projeto é criado e adicionado automaticamente à plataforma juntamente com os dados de medições e problemas oriundos da análise. Em todas as linguagens suportadas pelo SonarQube, é realizada a análise estática de código fonte do projeto, sendo também possível, em algumas linguagens, realizar análise de código compilado ou mesmo de forma dinâmica.

Após a análise, itens podem ser criados e classificados. O SonarQube considera um item todo componente que viola alguma regra de codificação estabelecida para o projeto. As regras de codificação podem ser definidas por meio do perfil de qualidade atrelado ao projeto, com a possibilidade de criar, alterar e remover regras conforme a necessidade. Há três tipos de itens considerados pelo SonarQube:

- ***Code Smell*:** problema relacionado à manutenibilidade do sistema, impactando negativamente as tarefas de manutenção por representarem obstáculos que reduzem a velocidade para efetuar mudanças e que, no pior caso, podem facilitar a introdução de novos erros ao sistema;
- ***Bug*:** item que aponta um problema real ou potencial no código e que precisa ser corrigido o mais rápido possível;
- **Vulnerabilidade:** item relacionado a pontos de falha de segurança no sistema que podem ser utilizados para atacar o sistema.

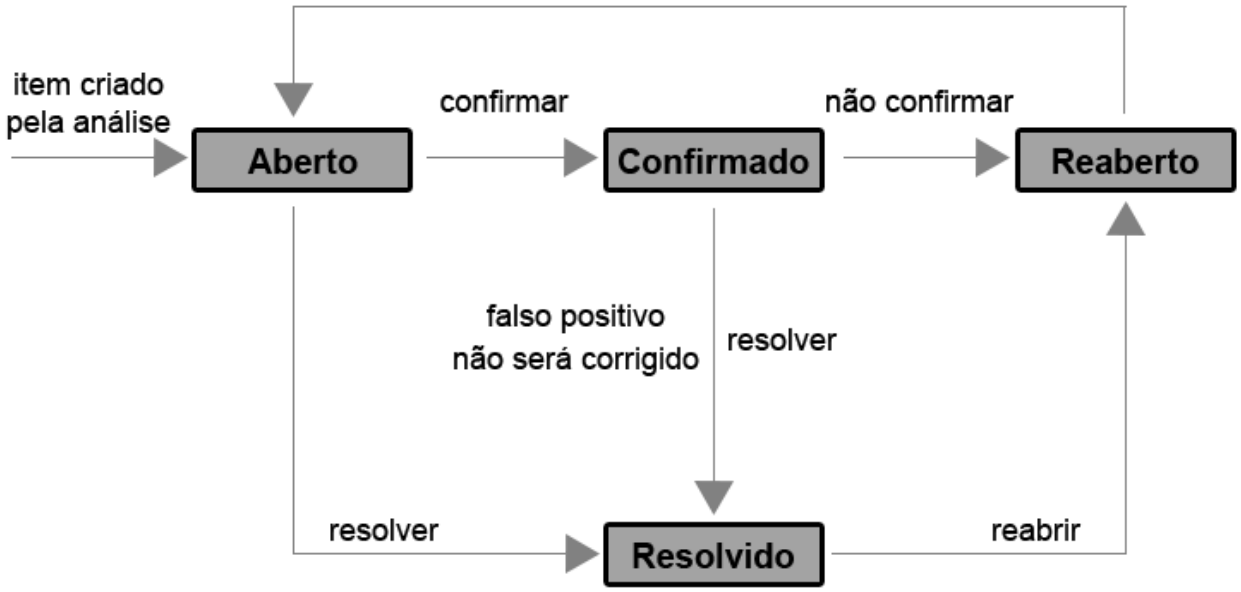
[2] **http://docs.sonarqube.org/display/PLUG/Technical+Debt+Plugin**

Além dos tipos de itens, o SonarQube também utiliza cinco níveis de importância (SONARSOURCE, 2016) para classificar todos os itens encontrados:

- ***Blocker*:** o nível mais grave de todos, exige que o código seja corrigido imediatamente. Problemas classificados neste nível representam *bugs* com alta probabilidade de impactar o comportamento do sistema, tais como vazamento de memória, conexões com o banco que não são fechadas;
- ***Critical*:** problemas desta categoria devem ser revisados imediatamente. Eles representam tanto *bugs* com baixa probabilidade de causar impactos no comportamento do sistema ou falhas de segurança, tais como injeção de SQL ou blocos de tratamento de exceção vazios;
- ***Major*:** problemas que representam falhas de qualidade que podem causar alto impacto à produtividade de desenvolvimento, tais como código não coberto por testes, blocos duplicados, parâmetros não utilizados;
- ***Minor*:** problemas que representam falhas de qualidade que podem causar pouco impacto à produtividade de desenvolvimento, tais como linhas de código muito compridas; e
- ***Info*:** itens que não são *bugs* nem falhas de qualidade, apenas constatações e avisos, tais como métodos que se tornaram obsoletos ou que tiveram seu uso desencorajado e que não foram removidos do sistema.

Após a criação e classificação, todo item segue um ciclo de vida constituído de cinco possíveis estados e ações a serem tomadas, conforme apresentado na Figura 3.

Figura 3 - Ciclo de vida do item do SonarQube



Fonte: Adaptado de http://docs.sonarqube.org/display/SONAR/Issue+Lifecycle

Após a análise, os resultados dos problemas encontrados são disponibilizados na forma de gráficos, listas de problemas, painéis e medições e métricas de código como: complexidade, documentação, duplicações, itens, manutenibilidade, confiabilidade, segurança e testes (SONARSOURCE, 2016). Métricas podem ser qualitativas, como densidade de linhas duplicadas, ou quantitativas, como número de linhas de código.

4.2 Métodos Manuais

Métodos manuais para identificar dívida técnica são propensos a demandar mais tempo para serem aplicados se comparados aos automáticos (ZAZWORKA et al., 2013), porém podem contar com vantagens como:

- **Maior precisão:** o fator humano pode ajudar na precisão ao avaliar se a dívida é ou não relevante, enquanto ferramentas automáticas podem apontar dívida técnica que não tenham tanta importância; e

- **Informações adicionais:** o processo de identificação manual pode contar com informações adicionais contextuais fornecidas por *stakeholders* que são praticamente impossíveis de se obter com ferramentas automáticas, tais como estimativas de esforço e de impacto e lógica de decisão.

Isto faz com que a aplicação destes métodos, apesar de consumir mais tempo e recursos, possa trazer resultados mais precisos quanto à relevância da dívida técnica, enquanto análises automáticas têm maior tendência para apontar falsos positivos neste quesito (ZAZWORKA et al., 2013). Nesta seção, serão apresentados dois métodos de levantamento manual de dívida técnica.

4.2.1 Monitoração da Atividade de Compreensão de Código do Desenvolvedor

O método proposto por Singh et al. (2014) combina métricas relacionadas à estrutura de código com o esforço de compreensão de código pelos desenvolvedores. Foram selecionadas métricas de estrutura de código relacionadas à manutenibilidade e dívida técnica a partir do trabalho realizado por (NUGROHO; VISSER; KUIPERS, 2011), em que por meio de métricas de código estático, foram detectados problemas potenciais de manutenção de código. Quanto à análise estática, as métricas utilizadas são:

- Contagem de classes acopladas;
- Contagem de classes bases;
- Contagem de classes derivadas;
- Contagem de linhas de código da classe; e
- Contagem de métodos declarados na classe.

Estas métricas no nível de classe são relacionadas às métricas de compreensão de código para estimar os juros. Para calcular as métricas de compreensão do desenvolvedor, conforme dito anteriormente, a ferramenta de monitoramento Blaze é utilizada gravando a atividade do desenvolvedor de forma anônima.

Uma sessão – intervalo de tempo fixo em que o desenvolvedor investiga uma classe – é iniciada na primeira vez que o desenvolvedor visita uma classe, a qual é

chamada de classe central, terminando quando ele visita esta mesma classe na última vez antes da sessão finalizar. A partir disto, as seguintes métricas ligadas ao esforço na compreensão do código são calculadas:

- **Sessões:** número de sessões (intervalos de tempo fixos) para cada classe;
- **Visita de classe:** número de vezes que a classe central foi visita dentro do intervalo da sessão;
- **Acessos a outras classes:** número de classes, diferentes da classe central, visitadas no intervalo da sessão;
- **Tempo gasto na classe:** tempo utilizado para investigação da classe central em uma sessão; e
- **Tempo gasto em outras classes:** tempo utilizado, com exceção da classe central, na investigação de todas as outras classes.

Com base nestas métricas, o cálculo de juros de dívida técnica é calculado considerando a diferença entre o tempo de compreensão da classe, a estrutura atual e o tempo que essa a compreensão levaria caso a classe fosse estruturada de forma ideal.

O cálculo do pagamento de juros de dívida técnica ligado à estrutura do código usa medições de tempo dispendidos em classes com métricas estáticas que indicam a existência de dificuldade de manutenção do código ou a presença de *code smells*, indicando possibilidade de dívida técnica em classes.

Esta abordagem permite visualizar estimativas de pagamento de juros de dívida técnica, possibilitando e facilitando a priorização de remoção das dívidas.

4.2.2 Lista de Itens de Dívida Técnica

A abordagem para o gerenciamento de dívida técnica proposta por Seaman e Guo (2011) é composta por três principais atividades: identificar, medir e monitorar dívida técnica. Neste trabalho, somente as duas primeiras atividades serão consideradas. Após a identificar manualmente a dívida técnica e medir sua importância (baseando-se em estimativas do custo da dívida, quantidade de juros e probabilidade de

ocorrência dos juros), o resultado gerado é uma lista de itens de dívida técnica, sendo cada item representado por uma tarefa que não foi concluída e que tem o potencial de afetar o trabalho futuro, causando algum problema ou prejuízo se não for finalizada. A partir da lista de itens de dívida técnica identificados e medidos, a dívida técnica pode ser monitorada e decisões sobre quais e quando os itens devem ser pagos – e se devem – podem ser tomadas pela equipe.

Em cada liberação de nova versão do sistema, a lista de itens de dívida técnica deve ser revisada e atualizada, pois novos itens podem ter sido introduzidos, bem como outros podem ter sido removidos. Tanto a equipe que mantém o software quanto os gestores podem adicionar itens à lista de dívida técnica dependendo do tipo do item de dívida técnica.

Os itens de dívida técnica, seguindo o modelo apresentado na Tabela 1, podem ser trechos de código que precisam ser refatorados, testes que não foram feitos, revisões e inspeções de código pendentes, falta ou desatualização da documentação ou comentários de código, problemas de conformidade arquitetural, problemas já conhecidos etc.

O modelo de item de dívida técnica apresentado contém propriedades financeiras (custo estimado, quantidade estimada de juros e probabilidade estimada de juros) e não financeiras (responsável, tipo, localização, descrição e se foi ou não adquirida intencionalmente) sobre a dívida técnica:

- **Responsável:** nome da pessoa ou papel responsável por consertar a dívida técnica, sendo importante seu preenchimento para fins administrativos e com o potencial de ajudar na construção de uma base histórica de avaliação de custo da dívida e dos juros;
- **Tipo:** os possíveis valores para o tipo da dívida técnica seguem a classificação de Guo e Seaman (2011), isto é, pode ser de defeito, *design* (ou código), teste ou documentação, embora esteja aberto a outros valores, como no trabalho de Zazworka et al. (2013), em que dívida de usabilidade – descrevendo a falta de um modelo comum para interface do usuário – foi introduzida durante a aplicação da pesquisa por um dos participantes;

- **Localização:** o local onde a dívida técnica foi encontrada (como arquivos, classes, métodos, páginas, documentos) é relevante para o entendimento do impacto, relações entre os itens de dívida técnica e possíveis efeitos em cascata que podem ser desencadeados ao pagar a dívida;
- **Descrição:** qual é o problema e os possíveis impactos que podem ser desencadeados na manutenção futura do sistema;
- **Custo estimado:** refere-se ao esforço total necessário para pagar a dívida, isto é, eliminar o problema do sistema. Dependendo do tipo do item de dívida técnica, estimar o custo pode ser mais ou menos simples, por exemplo, a atividade de adicionar documentação tende a ser mais simples do que uma refatoração em módulo complexo. Dados históricos de esforço podem ajudar a estimar o custo de forma mais precisa e confiável, por exemplo, o histórico de custo de modificação de classes que precisam de refatoração ajuda a estimar o custo desta tarefa;
- **Quantidade estimada de juros:** esforço extra que será necessário despender caso o item de dívida não seja removido do sistema. Dados históricos como esforço de mudanças e problemas similares podem ser utilizados com o objetivo de melhorar a precisão da estimativa;
- **Probabilidade estimada de juros:** este campo representa a probabilidade de ocorrência de juros (probabilidade de realizar o esforço extra), caso o item de dívida não seja removido do sistema. Assim como o custo e quantidade de juros estimados, dados históricos também podem ajudar na estimativa para chegar a valores mais confiáveis; e
- **Intencional:** saber se o item foi introduzido no sistema de forma intencional ou não ajuda a entender como as decisões sobre a dívida técnica estão sendo gerenciadas.

Tabela 1 - Modelo do item de dívida técnica por Seaman e Guo (2011)

ID	Número de identificação do item
Responsável	Pessoa ou papel responsável por consertar o item
Tipo	*Design*, documentação, defeito, testes etc
Localização	Arquivos, classes, métodos ou páginas e documentos onde o item está localizado
Descrição	O que é o item e seus impactos
Custo estimado	Quanto esforço é necessário para remover o item por completo (pagar a dívida), inicialmente estimado em uma escala de três valores: Baixo/Médio/Alto
Quantidade estimada de juros	Esforço extra que terá de ser feito no futuro, caso este item não seja removido, inicialmente estimado em uma escala de três valores: Baixo/Médio/Alto
Probabilidade estimada de juros	Probabilidade de que este item cause o esforço extra no futuro, caso não seja removido, inicialmente estimada em uma escala de três valores: Baixo/Médio/Alto
Intencional	Sim/Não/Não se sabe

Fonte: Seaman e Guo (2011)

4.3 Considerações do Capítulo

Este capítulo apresentou alguns dos métodos e abordagens que podem ser utilizados para o levantamento de dívida técnica, tanto de forma manual quanto automática.

Para as abordagens automáticas, apresentou brevemente ferramentas como FindBugs, CodeVizard, CLIO e em especial a plataforma SonarQube, que será utilizada no Capítulo 5 para levantamento de dívida técnica em um sistema real.

Com relação aos métodos manuais, apresentou o método de monitoração da atividade do desenvolvedor e em especial a lista de itens de dívida técnica proposta por Seaman e Guo (2011), abordagem fundamental que será utilizada neste trabalho no mesmo sistema que o SonarQube para que os resultados de ambas as abordagens sejam analisados.

5 ANÁLISE DE MÉTODOS DE LEVANTAMENTO DE DÍVIDA TÉCNICA

Neste capítulo será descrito detalhadamente como foi projetado e conduzido o estudo em que foram analisados os resultados do levantamento manual de dívida técnica realizado por uma equipe de desenvolvedores em relação aos do levantamento automático realizado pela ferramenta SonarQube, apresentada no capítulo anterior, em um sistema de software real e operante.

Este estudo teve como principal foco ajudar a responder um dos questionamentos surgidos após a conclusão do trabalho realizado por Zazworka et al. (2013): “quanto da dívida técnica reportada por ferramentas, mas não pelos desenvolvedores, pode ser considerada real? Ou seja, existe valor em utilizar ferramentas para identificar dívida técnica que os desenvolvedores não estão cientes?” (ZAZWORKA et al., 2013, p. 7).

Para organizar o procedimento em torno desta dúvida, foram definidas cinco principais atividades:

1. **Aplicar o método de levantamento manual de dívida técnica:** foi solicitado aos participantes que analisassem os artefatos do sistema escolhido em busca de casos que julgassem ser dívida técnica e que registrassem estes casos em um segundo documento – uma planilha de preenchimento seguindo o modelo de item de dívida técnica proposto por Seaman e Guo (2011);
2. **Aplicar o método de levantamento automatizado de dívida técnica:** em paralelo à execução da primeira atividade, a ferramenta SonarQube foi usada no mesmo sistema analisado pelos desenvolvedores, sem que estes tivessem conhecimento de seu uso e principalmente dos resultados gerados;
3. **Analisar e consolidar os resultados de ambos os levantamentos:** concluídas as atividades de levantamento de dívida técnica pelos dois métodos, os itens resultantes dos dois métodos foram consolidados por meio da remoção de duplicações seguindo um critério de igualdade apresentado

na seção 5.3. Em seguida, o resultado consolidado de um levantamento foi analisado em relação ao outro com o objetivo de separar os itens em três subconjuntos - itens encontrados exclusivamente pelo método manual, itens encontrados exclusivamente pelo método automatizado e itens encontrados por ambos.

4. **Apresentar aos desenvolvedores o subconjunto de dívida técnica identificado exclusivamente pelo método automatizado e questioná-los:** foi enviada aos participantes uma planilha contendo o subconjunto da dívida técnica encontrado exclusivamente pelo método automatizado com o objetivo de que eles analisassem e respondessem questões pertinentes à relevância deste subconjunto e
5. **Consolidar o *feedback* dado pelos desenvolvedores sobre o subconjunto de dívida técnica não identificado manualmente:** foi feita a análise e consolidação dos dados da segunda planilha preenchidos pelos desenvolvedores, onde os resultados foram apresentados e discutidos.

5.1 Trabalhos Relacionados

O método de levantamento manual utilizado neste capítulo é o proposto por Seaman e Guo (2011), em que o gerenciamento da dívida técnica é realizado em torno de uma lista de itens de dívida técnica criada e manipulada por pessoas ao analisarem manualmente um software. Importante citar que o método não orienta o processo de identificação de dívida técnica no sentido de seguir passos pré-definidos ou diretrizes, mas sim sugere que as tarefas relacionadas ao gerenciamento da dívida técnica (identificar, medir e monitorar) utilizem a lista de itens como principal ponto focal.

Para realizar a análise do resultado da abordagem automática com a manual, este trabalho teve como inspiração o estudo conduzido por Zazworka et al. (2013), em que abordagens de identificação de dívida técnica – automatizada e manual – foram aplicadas em um mesmo sistema, com o objetivo primário de avaliar se ferramentas de identificação automática de dívida técnica têm a capacidade de encontrar itens apontados manualmente. Entretanto, diferentemente deste, o objetivo do presente

estudo é analisar quanto da dívida técnica identificada exclusivamente por meio de um método automatizado é considerada relevante para os desenvolvedores.

No trabalho de Zazworka et al. (2013), o levantamento manual foi realizado por uma equipe de desenvolvimento composta de cinco pessoas, sendo inicialmente treinada para se familiarizar com o conceito da dívida técnica. Durante o treinamento, foram apresentados aos participantes o modelo de preenchimento de item de dívida técnica proposto de Seaman e Guo (2011) e alguns exemplos abstratos de itens de dívida técnica, para evitar vieses na identificação. Esta mesma abordagem de treinamento foi utilizada no estudo deste trabalho.

Ainda sobre o experimento de Zazworka et al. (2013), após o treinamento, foi solicitado à equipe que identificassem, individualmente, casos de dívida técnica no software e preenchessem um questionário com estes itens, onde poderiam utilizar o modelo de item de dívida técnica de Seaman e Guo (2011) caso preferissem. No mesmo questionário também havia perguntas sobre a dificuldade de identificar itens de dívida técnica e de preencher o modelo. Neste sentido, o presente trabalho diferencia-se por levar em consideração apenas os itens de dívida técnica identificados pelos participantes.

Na conclusão do trabalho conduzido por Zazworka et al. (2013), após a comparação dos resultados da abordagem manual com os das abordagens automatizadas, foi apontada a importância do envolvimento tanto de humanos quanto de ferramentas no processo de levantamento de dívida técnica ao mostrar que as abordagens automáticas não foram capazes de encontrar todos os tipos de dívida técnica encontrados pela identificação manual e vice-versa. Além disto, dentre as questões levantadas na conclusão do trabalho, questionou-se e recomendou-se como trabalho futuro quanto da dívida técnica potencial apontada pelas ferramentas, porém não pelos desenvolvedores, é relevante. Fornecer informações e uma resposta inicial a essa dúvida é o principal objetivo deste trabalho.

5.2 Contexto

O estudo foi realizado na Porto Seguro Seguros, uma grande empresa seguradora com sede em São Paulo, Brasil. A área de tecnologia da informação tem mais de mil colaboradores, entre funcionários e prestadores de serviço, e é responsável pelo desenvolvimento dos sistemas de software para clientes internos e externos, sendo em sua maioria aplicações web escritas na linguagem Java.

O sistema utilizado para análise e levantamento de dívida técnica é uma aplicação de grande porte desenvolvida na plataforma Java *Enterprise Edition*, seguindo o modelo *Model-view-controller* (MVC). É uma aplicação *back-end* construída com a finalidade de acelerar o desenvolvimento de projetos ao facilitar a integração de sistemas com as bases de dados próprias da companhia, fornecendo cerca de 650 operações distribuídas em mais de 200 serviços *web*.

O software está em operação há aproximadamente quatro anos e a equipe responsável é composta por cinco desenvolvedores que atuam fazendo o revezamento entre tarefas de manutenção, testes e desenvolvimento de novas funcionalidades, tais como a criação de novos serviços *web* para fornecer dados de novas bases de dados. Na etapa de levantamento manual de dívida técnica, o autor não foi considerado de forma a evitar resultados tendenciados. Entretanto, todos os quatro desenvolvedores participantes do estudo têm conhecimento e familiaridade com o sistema desde sua criação.

5.3 Critérios de Análise

Neste estudo, o escopo de análise do sistema pelos métodos manual e automatizado foi reduzido ao módulo feito em EJB - o mais complexo e importante módulo do sistema, com cerca de 5 mil linhas - e também à classe de um dos serviços *web* contidos no módulo *Web* (escolhido dentre os mais de 200 existentes por ser o maior, mais complexo e mais conhecido pela equipe, com aproximadamente 5.800 linhas). A redução de escopo foi necessária por conta da ordem de grandeza do sistema – constituído por aproximadamente 440 mil linhas de código –, o que tornaria impraticável concluir a tarefa de levantamento manual,

embora não representasse problema algum para a análise realizada pela ferramenta SonarQube.

Para analisar a dívida técnica identificada pelos métodos manual e automatizado a fim de descobrir se um mesmo problema foi identificado por ambos os métodos (sobreposição) ou se houve mais de uma ocorrência do mesmo problema para o mesmo método (duplicação), foi necessário estabelecer um critério de igualdade para os itens de ambos os levantamentos. Isto foi feito por meio da criação de um campo nas planilhas resultantes das aplicações contendo um identificador do problema (ID) para todo item encontrado. O valor deste campo foi composto com dados da localização e uma *tag* resumindo a natureza do problema.

Para exemplificar, a Tabela 2 mostra um caso do levantamento manual em que um mesmo item no escopo da classe foi identificado por três participantes, gerando o mesmo ID para o critério citado. Neste caso, o item foi considerado para contagem uma única vez, já que se tratava do mesmo caso identificado por três participantes.

Tabela 2 - Itens duplicados do levantamento manual no escopo de classe

ID do problema	Participante	Problema	Tipo	Localização
BuscaCorporativaCRM .GRANDE	P1	classe com excesso de linhas	Design	BuscaCorporativaCRM
BuscaCorporativaCRM .GRANDE	P2	classe grande demais	Design	BuscaCorporativaCRM
BuscaCorporativaCRM .GRANDE	P3	Classe muito grande. Muita responsabilidade.	Design	BuscaCorporativaCRM

Para os itens identificados pelos participantes dentro de um método, quando informado, o nome método foi considerado na composição do ID do problema (desconsiderando o número da linha onde o problema foi encontrado, caso fosse informada) com o objetivo de não generalizar o problema para a classe inteira, como apresentado na Tabela 3.

Tabela 3 – Item do levantamento manual no escopo de método

ID do problema	Participante	Problema	Tipo	Localização
ServicoDadosUtil. SEM_DOCUMENTACAO. buildQuery	P1	Código complexo sem documentação.	Documentação	ServicoDadosUtil. buildQuery.191

Para o levantamento automático, o resultado da análise do sistema pelo SonarQube foi exportado para uma planilha contendo informações de todos os problemas identificados pela ferramenta. Esta tarefa foi realizada com o objetivo de facilitar a análise e geração dos IDs de todos os itens identificados, considerando o mesmo critério utilizado para a geração dos IDs dos itens identificados pelos participantes.

A Tabela 4 representa um item da planilha contendo dados de sua descrição, tipo, local onde foi encontrado, *tags* relacionadas ao problema e o ID gerado para o item.

Tabela 4 - Item do levantamento automático

ID do problema	Problema	Tipo	Localização	*Tags*
ServicoDadosUtil. COMPLEXO. buildQuery	The Cyclomatic Complexity of this method "buildQuery" is 47 which is greater than 10 authorized.	*Code Smell*	ServicoDadosUtil - Line 121	*brain-overload*

A partir deste critério de classificação de itens estabelecido para este estudo, a verificação de duplicação ou sobreposição de itens para fins de comparação foi feita exclusivamente a partir dos IDs gerados.

Importante citar que os itens identificados pelo SonarQube que foram considerados duplicados e depois removidos pelo critério de igualdade definido nesta seção não devem ser desconsiderados sob outras perspectivas ou contextos. O critério definido para este estudo foi pensado para facilitar as atividades de análise dos resultados dos métodos.

Trabalhar com itens considerados repetidos neste trabalho dificultaria colher o *feedback* dos participantes quanto à relevância do subconjunto de dívida técnica encontrada exclusivamente pela ferramenta automatizada. Isto poderia levar à conclusão errônea de que a ferramenta automatizada identifica mais tipos de problemas – e com mais relevância – do que o levantamento manual (quando na verdade se trata do mesmo caso), tendenciando o resultado do objetivo deste trabalho.

5.4 Aplicação do método de levantamento manual de dívida técnica

Foi solicitado aos quatro participantes que analisassem e levantassem dívida técnica dentro do escopo do sistema definido no estudo. O processo de levantamento manual de dívida técnica seguiu a abordagem da Seaman e Guo (2011) e teve como primeiro passo a familiarização do conceito da dívida técnica pela equipe. Isto foi realizado por meio de um documento, apresentado no Apêndice A, que descreveu a dívida técnica baseada nas visões de Seaman e Guo (2011), Fowler (2003a) e Cunningham (1992) propositadamente em alto nível, contendo exemplos fora do contexto do desenvolvimento de software, inspirado pelo trabalho de Zazworka et al. (2013) de modo a evitar um resultado tendenciado por exemplos concretos de dívida técnica.

O documento apresentou e explicou o modelo de item de dívida técnica proposto por Seaman e Guo (2011) com o objetivo de que os respondentes preenchessem uma planilha – enviada junto com o documento – conforme encontrassem itens que julgassem ser algum tipo de dívida técnica, estimando o custo, probabilidade e quantidade de juros de cada item levantado.

O modelo de item de dívida técnica foi ligeiramente modificado quanto ao campo “Descrição”, sendo dividido em “Qual o problema” e “Impactos” para uma melhor separação de causa e consequência. Os campos “ID” e “Responsável” foram retirados por ambos não serem necessários no estudo. Para o campo “Tipo”, embora tenha sido sugerida a classificação de tipos de Seaman e Guo (2011), foi dada liberdade para que os participantes não ficassem restritos somente a este domínio

de valores. O procedimento contido no documento também pediu que os participantes informassem o tempo total utilizado para completar a tarefa.

Os participantes foram informados para atuar na tarefa de forma isolada, sem a ajuda, opinião ou consulta dos levantamentos dos demais e também que, ao concluir a tarefa, enviassem a planilha preenchida. Foi-lhes dado cinco dias, a partir da data de entrega do documento e da planilha de preenchimento, como tempo limite para conclusão da tarefa.

5.5 Resultados do método de levantamento manual de dívida técnica

Conforme apresentado na Tabela 5, no término da tarefa de levantamento manual foram identificados 157 itens de dívida técnica pelos participantes – considerando itens duplicados encontrados por dois ou mais participantes – em aproximadamente 18 horas.

Tabela 5 - Quantidade de itens identificados pelos participantes

Participante	Itens identificados	Horas
P1	33	5
P2	50	4
P3	29	3
P4	45	6
Todos	157	18

Após a junção dos resultados individuais em uma única planilha, todos os itens foram analisados com o objetivo de remover ocorrências duplicadas, ou seja, o mesmo problema relatado duas ou mais vezes por participantes diferentes. Isto foi feito por meio da criação de uma nova coluna na planilha unificada contendo o “ID” do problema, conforme apresentado na seção 5.3 deste capítulo.

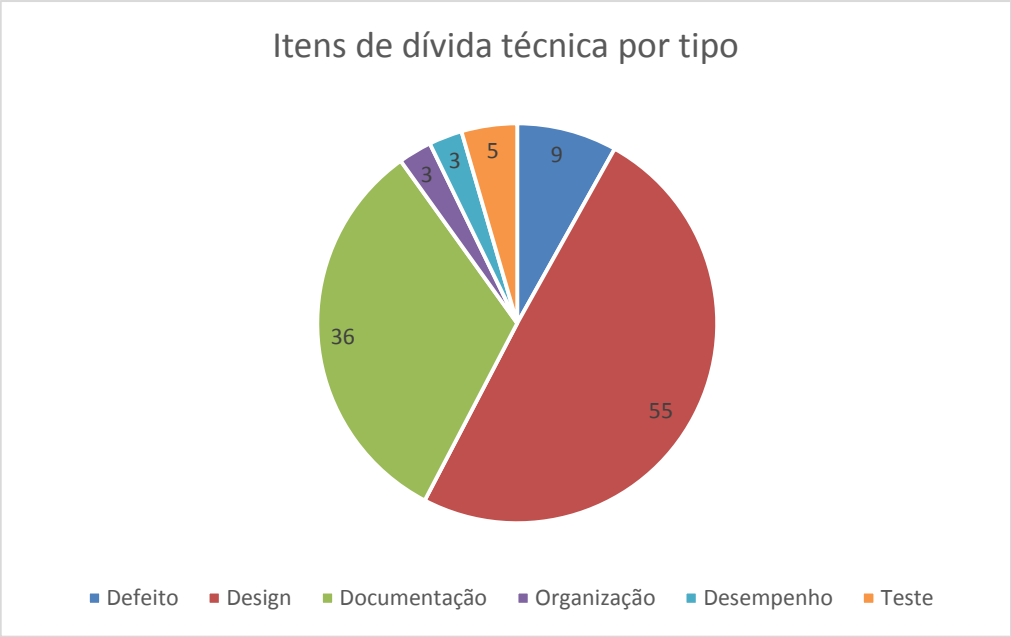
Seguindo o critério de igualdade estabelecido, foram encontrados e removidos 46 itens dos 157 levantados – restando 111 itens considerados diferentes –, mostrando

que houve aproximadamente 29% de sobreposição de itens levantados por ao menos dois desenvolvedores.

Após a remoção dos itens duplicados, os 111 itens restantes foram organizados pelos tipos informados participantes, como apresentado na Figura 4. Interessante citar que, além da classificação de tipos de dívida técnica feita Seaman e Guo (2011), dois diferentes tipos apareceram no estudo:

- **Organização:** dívida referente à falta de identação e limpeza de códigos comentados que impacta negativamente a legibilidade do código; e
- **Desempenho:** dívida referente a trechos de código que podem afetar negativamente o desempenho do sistema, tais como uso do mecanismo de reflexão do Java e criação de objetos desnecessários.

Figura 4 - Quantidade de itens por tipo

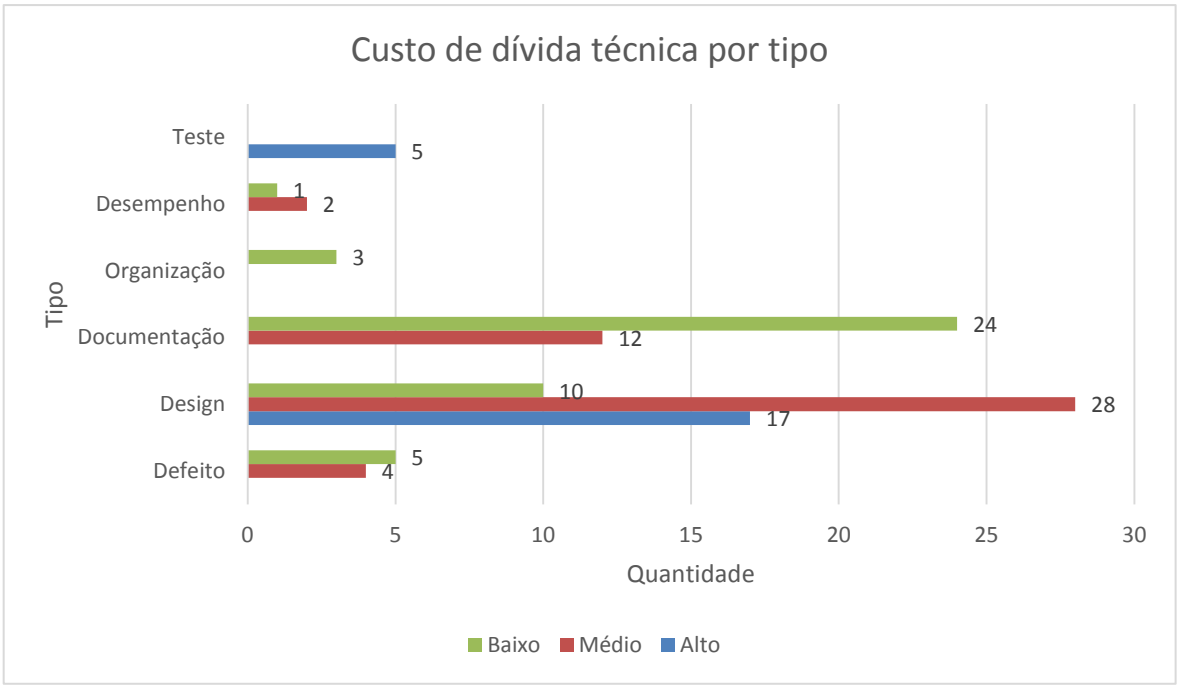


Houve um item que foi classificado como dívida técnica de “Segurança” por um dos participantes, porém este foi reclassificado pelo autor como “Defeito” pois se tratava de um problema potencial que poderia surgir porque não foi implementado controle de sincronismo de *threads* no trecho identificado.

O resultado mostra que a maioria dos itens de dívida técnica encontrados são relacionados aos tipos “*Design*” (50%) e Documentação (32%), evidenciando um problema pertinente a ambos os tipos: código difícil de ler, entender e consequentemente alterar, sendo potencializado pela falta de documentação.

Quanto ao custo da dívida dos itens identificados, isto é, o esforço necessário para remover o problema, 22 itens (19,8%) foram classificados com alto custo, 46 itens (41,4%) com médio custo e 43 (38,8%) com baixo custo. A Figura 5 apresenta o custo da dívida separada pelos seis tipos considerados.

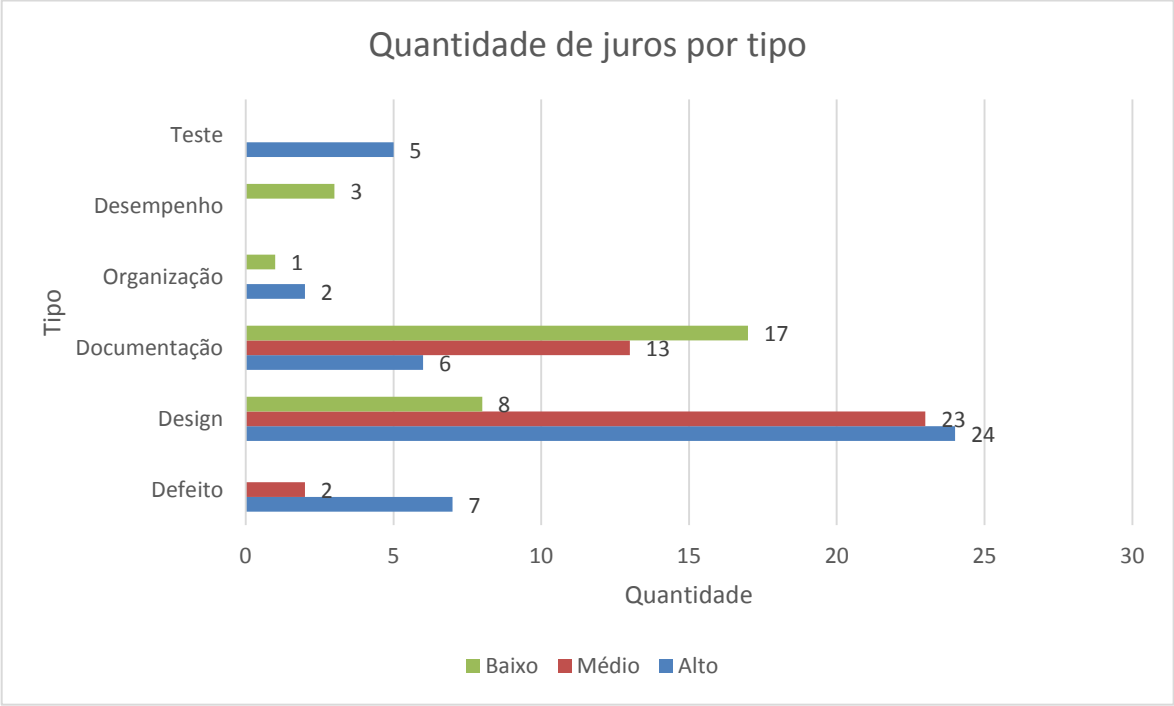
Figura 5 - Custo de dívida técnica por tipo



Os itens classificados com alto custo estavam distribuídos somente entre os tipos de dívida de “Design” e de “Teste”. Para o tipo “Design”, 15 dos 17 itens relatados tinham relação com o tamanho e complexidade da classe. No caso do tipo “Teste”, todos os cinco itens identificados foram classificados com alto custo por não existir código de teste construído para as classes apontadas. Itens do tipo “Organização” foram todos classificados com baixo custo (falta de identação de código e linhas de código comentadas que não foram removidas).

Do ponto de vista da quantidade de juros dos itens identificados, isto é, a quantidade de esforço extra necessária no futuro caso o problema não seja removido, os tipos de dívida de “Defeito” e de “Teste” (72% e 100% respectivamente) são proporcionalmente os que mais impactam a manutenção e evolução do sistema, como indicado na Figura 6. Para o tipo “Defeito”, não havia nenhum item com custo de dívida alto, mostrando que, apesar de causarem altos juros, não foram considerados problemas difíceis de resolver. Para o tipo “Teste”, o motivo para o alto custo foi a ausência de cobertura de testes para as classes informadas, fazendo com que toda modificação se torna mais difícil por não saber se ela pode causar efeitos colaterais.

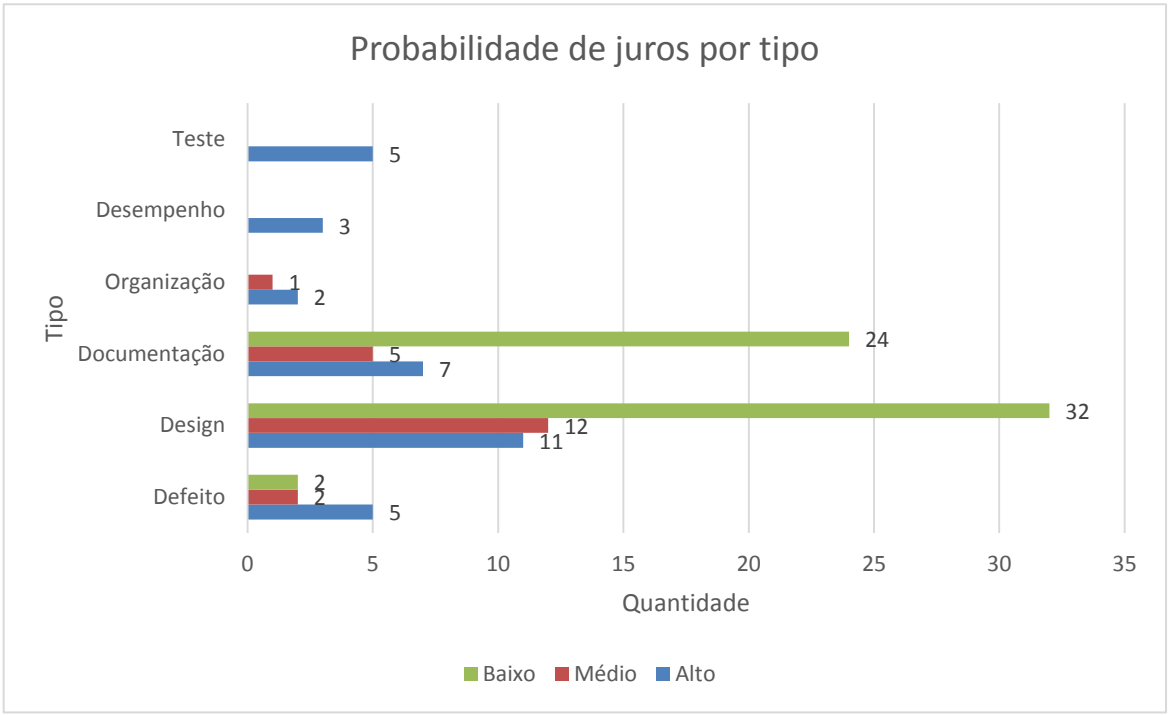
Figura 6 - Quantidade de juros por tipo



A alta probabilidade de juros dos itens levantados, ou seja, a chance de que o problema cause o esforço extra se não for resolvido, ficou distribuída de forma homogênea em todos os tipos de dívida, conforme mostra a Figura 7. Entretanto, para os tipos “Desempenho” e “Teste”, todos os casos foram classificados com alta probabilidade. O impacto informado pelos participantes para o tipo “Desempenho” foi a possibilidade de o sistema executar de forma lenta. A alta probabilidade, neste

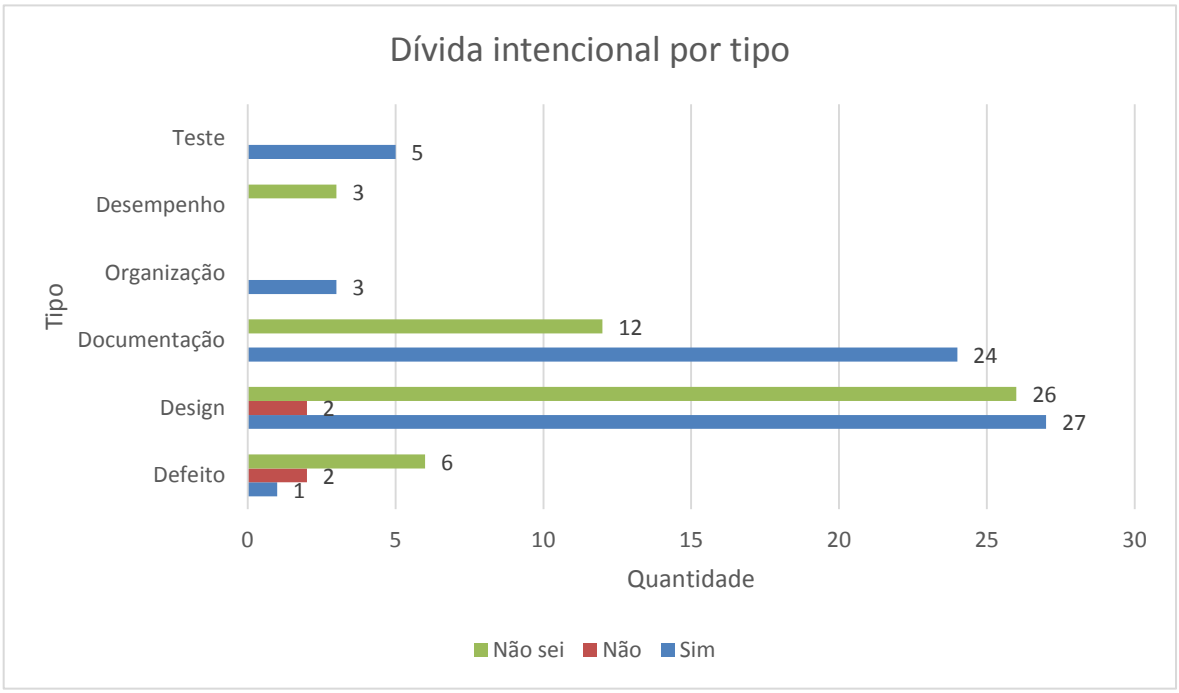
caso, deve-se ao fato de que as classes informadas fazem parte do *core* do sistema e são utilizadas pelo sistema todo o tempo. A alta probabilidade para o tipo “Teste” também está relacionada ao fato de que essas classes são o *core* do sistema e mudanças neste módulo são realizadas todo o tempo, justificando este valor.

Figura 7 - Probabilidade de juros por tipo



Quanto à dívida técnica identificada ser ou não intencional, 54% dos casos foi classificado como “Sim”, 42% como “Não sei” e 4% como “Não” (Figura 8). Proporcionalmente, os tipos “Teste”, “Organização” e “Documentação” apareceram com as maiores ocorrências de itens intencionais. Ao analisá-los, pode-se concluir que, pelo fato das atividades atribuídas a esses tipos – testar, limpar o código e documentar – não serem relevantes nem visíveis para o cliente final, acabaram sendo de alguma forma não priorizadas.

Figura 8 - Dívida intencional por tipo



5.6 Aplicação do método de levantamento automático de dívida técnica

Em paralelo à tarefa de levantamento manual realizada pelos participantes, foi iniciada a análise automática do sistema pela ferramenta SonarQube na versão 5.6. O sistema inteiro foi analisado pela ferramenta, porém somente um subconjunto do resultado encontrado – relativo à mesma porção do sistema analisada pelo levantamento manual – foi utilizado para análise, consolidação e comparação com os resultados do levantamento manual.

Para utilizar o SonarQube é necessário ter instalado um Sistema de Gerenciamento de Banco de Dados (SGBD) que seja suportado pelo SonarQube (SONARSOURCE, 2016). A opção escolhida no estudo foi o MySQL *Community Server* 5.7.12[3]. Outro requisito necessário é ter instalado uma *Java Virtual Machine* (JVM). Dentre as opções, foi escolhida a opção disponibilizada pela Oracle, o *Java Development Kit* (JDK) versão 8[4]. O último requisito é utilizar uma das opções de navegador *web* para

[3] **http://www.mysql.com**
[4] **http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html**

servir como interface de interação com o SonarQube. O navegador utilizado foi o Mozilla Firefox na versão 47.0.

Após instalar e configurar o SonarQube com apoio da documentação do site oficial, para aplicar o método de análise no sistema foi necessário obter e configurar separadamente o analisador de código fonte SonarQube Scanner para Maven. Ele foi a opção escolhida porque o sistema e sua estrutura de pastas já estavam configurados para funcionar com o Maven, uma ferramenta de automação de compilação desenvolvida pela Apache Software Foundation[5]. Antes de executar o comando para que a análise no projeto seja iniciada pelo analisador, é preciso que o servidor do SonarQube esteja funcionando no momento.

A análise utilizando o SonarQube Scanner para Maven foi realizada através da linha de comando, conforme a documentação do analisador. O processo de análise levou cerca de dez minutos para ser concluído. Em seguida, os dados foram enviados ao servidor do SonarQube para serem processados, levando cerca de cinco minutos para conclusão. Após esta tarefa de processamento, os dados já estavam disponíveis, sendo possível visualizar o projeto criado na página inicial do SonarQube através do navegador.

5.7 Resultados do método de levantamento automático de dívida técnica

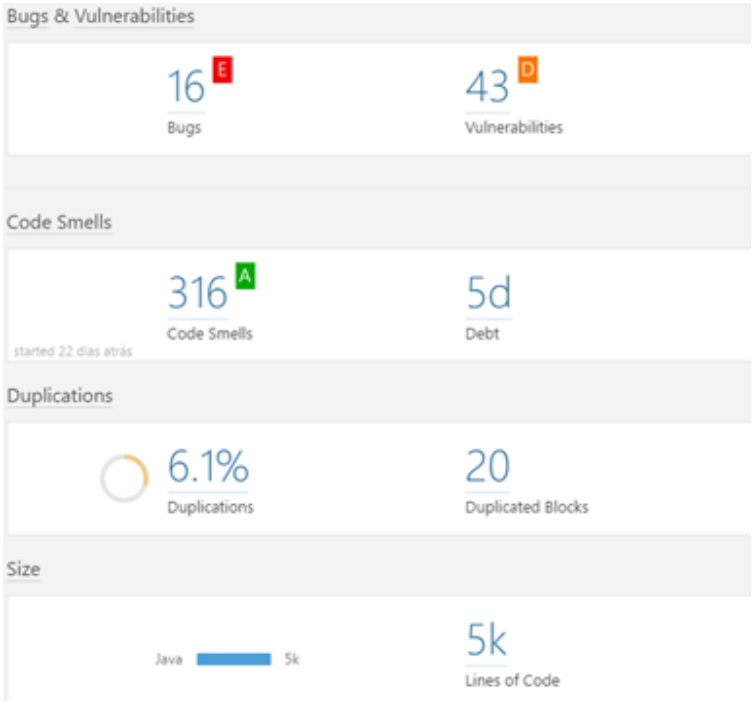
O resultado da análise do projeto foi visualizado na própria ferramenta, onde inicialmente foram apresentados números dos três tipos de problemas considerados pelo SonarQube – c*ode smells*, *bugs* e vulnerabilidades –, quantidade de dívida técnica calculada em dias e métricas de tamanho, como total de linhas de código e porcentagem de duplicação de código.

Entretanto, como o escopo de análise do sistema foi reduzido ao módulo EJB e a um dos serviços *web* contidos no projeto Web, foi necessário realizar uma filtragem no resultado total com o objetivo de separar somente o subconjunto dos problemas localizados dentro do escopo escolhido. Através da navegação da raiz do projeto até o módulo com *Enterprise JavaBeans* (EJB), foi possível visualizar separadamente o

[5] **https://maven.apache.org/**

resultado encontrado somente neste módulo em relação ao sistema inteiro, como apresentado na Figura 9.

Figura 9 - Resultado da análise do módulo EJB pelo SonarQube



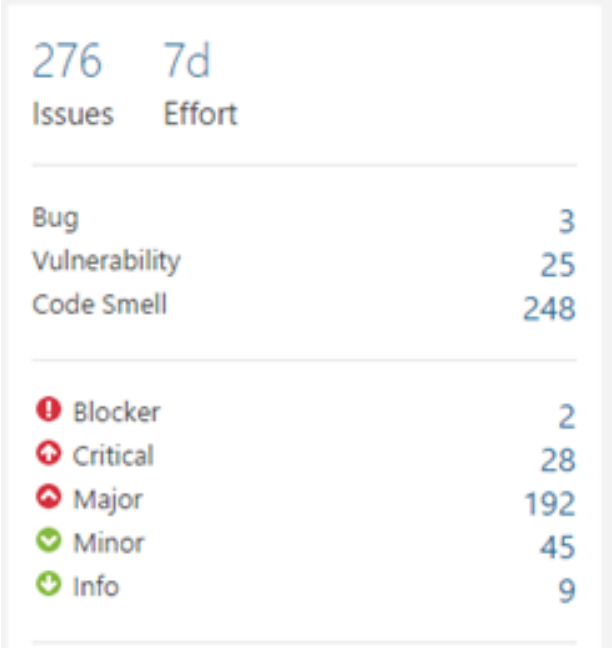
O número de problemas encontrados neste módulo foi de 375 (316 *code smells*, 16 *bugs* e 43 vulnerabilidades), representando cerca de 3% em relação ao sistema inteiro no momento da aplicação do estudo. Conforme mostra a Figura 10, a grande maioria dos itens apontados foram *code smells* (84%) e, quanto ao nível de importância, 81% dos casos foram classificados com alto risco (acima de *Major*).

Figura 10 - Classificação de importância dos problemas no módulo EJB

Para a segunda parte do sistema escolhida para ser analisada, o serviço *web*, foi necessário entrar no escopo do módulo Web pelo SonarQube e navegar até o pacote onde a classe Java estava localizada. Um ponto interessante do SonarQube é que ele disponibiliza as mesmas visualizações e informações independentemente do nível hierárquico do projeto, ou seja, é possível ver os resultados tanto do projeto inteiro, quanto de um módulo, pacote ou classe específica.

Ao localizar a classe Java escolhida, foi possível visualizar o número e os tipos de problemas encontrados (Figura 11): 276 itens (248 *code smells*, 3 *bugs* e 25 vulnerabilidades), representando aproximadamente 2,4% de todos os problemas existentes no sistema inteiro. Cerca de 90% de todos os problemas apontados foram *code smells* e, quanto ao nível de importância, 80% dos casos foram classificados com alto risco (acima de *Major*), proporção semelhante ao encontrado no módulo EJB.

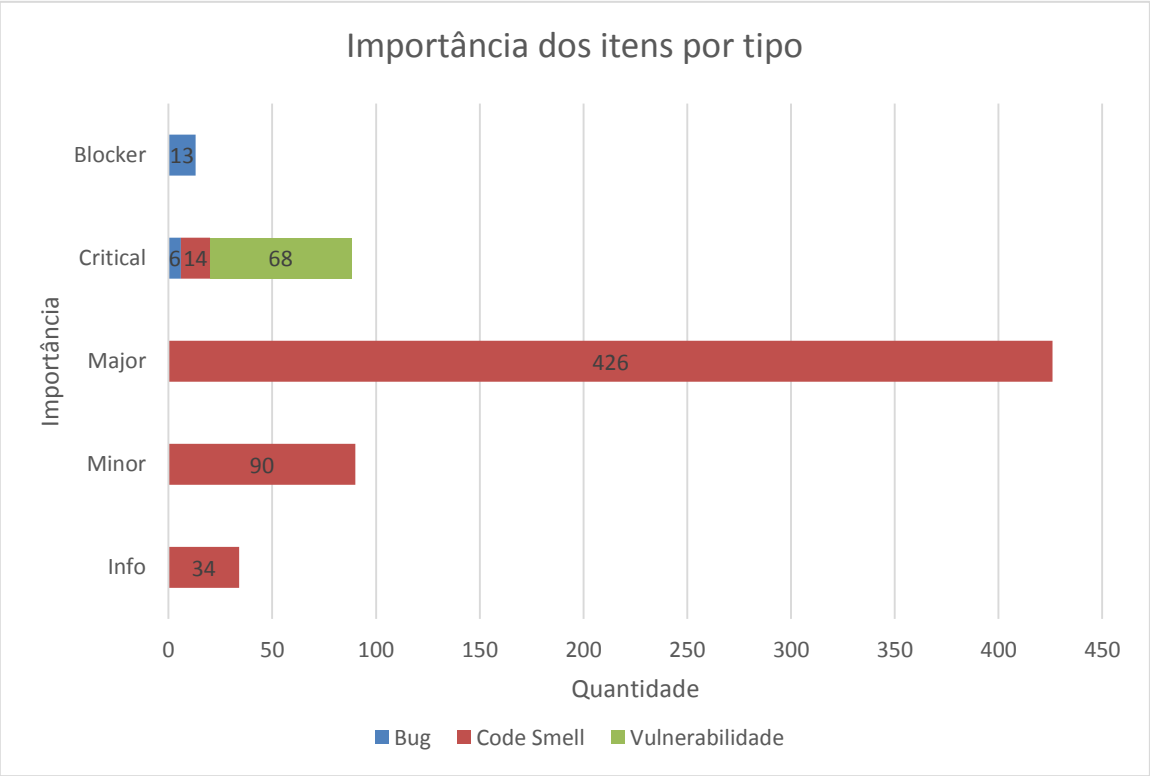
Figura 11 - Resultado da análise da classe do módulo Web



Juntando todos os itens identificados dentro do escopo selecionado do sistema, sem a remoção de duplicações seguindo o critério de igualdade estabelecido, foram encontrados 651 itens, sendo 19 *bugs* (3%), 68 problemas de vulnerabilidade (10%) e 564 *code smells* (87%). Quanto à classificação de importância do resultado,

conforme apresentado pela Figura 12, a importância *Blocker* foi composta inteiramente por *bugs* e todas as vulnerabilidades foram classificadas como *Critical*.

Figura 12 - Importância dos itens por tipo



O próximo passo foi remover as duplicações do resultado do SonarQube a partir da geração de IDs para todos os 651 itens identificados, seguindo a mesma avaliação realizada nos itens resultantes do levantamento manual, em que foram considerados a localização e a natureza do problema.

Desconsiderando todos os itens com IDs repetidos, restaram 349 itens considerados exclusivos pelo critério de igualdade. Quase metade dos itens foram considerados duplicados (46% do total) porque em diversos casos o SonarQube gerou um item diferente para cada linha de código – como, por exemplo, em um método que contém bloco de código comentado que não foi removido. Com o critério de igualdade estes itens foram contados como um único problema.

5.8 Análise dos Resultados

A análise dos resultados do levantamento manual em relação aos do automático teve como objetivo separar os itens em três subconjuntos: itens identificados por ambos os levantamentos, itens identificados exclusivamente pelo levantamento manual e itens identificados exclusivamente pelo levantamento automático. Este último subconjunto de dívida técnica foi utilizado para a próxima etapa, em que foi apresentado e avaliado pelos participantes com a finalidade de colher *feedback* sobre sua relevância.

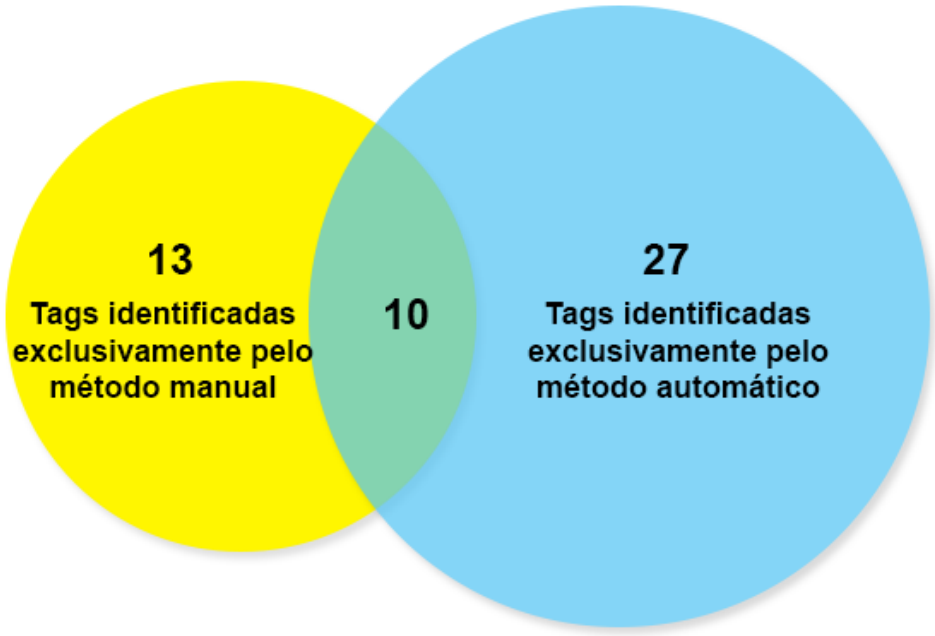
O procedimento realizado foi extrair e reunir todas as diferentes *tags* que compuseram os IDs gerados para os itens dos levantamentos (sem considerar repetições) em uma única planilha e, por meio da comparação dos dois conjuntos de *tags*, fazer a separação dos itens dentro de um dos três subconjuntos citados.

O Apêndice B apresenta o resultado das 23 diferentes *tags* extraídas dos 111 itens resultantes do levantamento manual, juntamente com sua frequência de ocorrência e sua descrição em relação ao item de dívida técnica, de acordo com o que foi informado pelos participantes.

Em seguida, a mesma tarefa foi feita para o levantamento automático. O Apêndice C apresenta os detalhes das 37 diferentes *tags* extraídas dos 349 itens levantados pelo SonarQube, além da frequência de ocorrência e descrição.

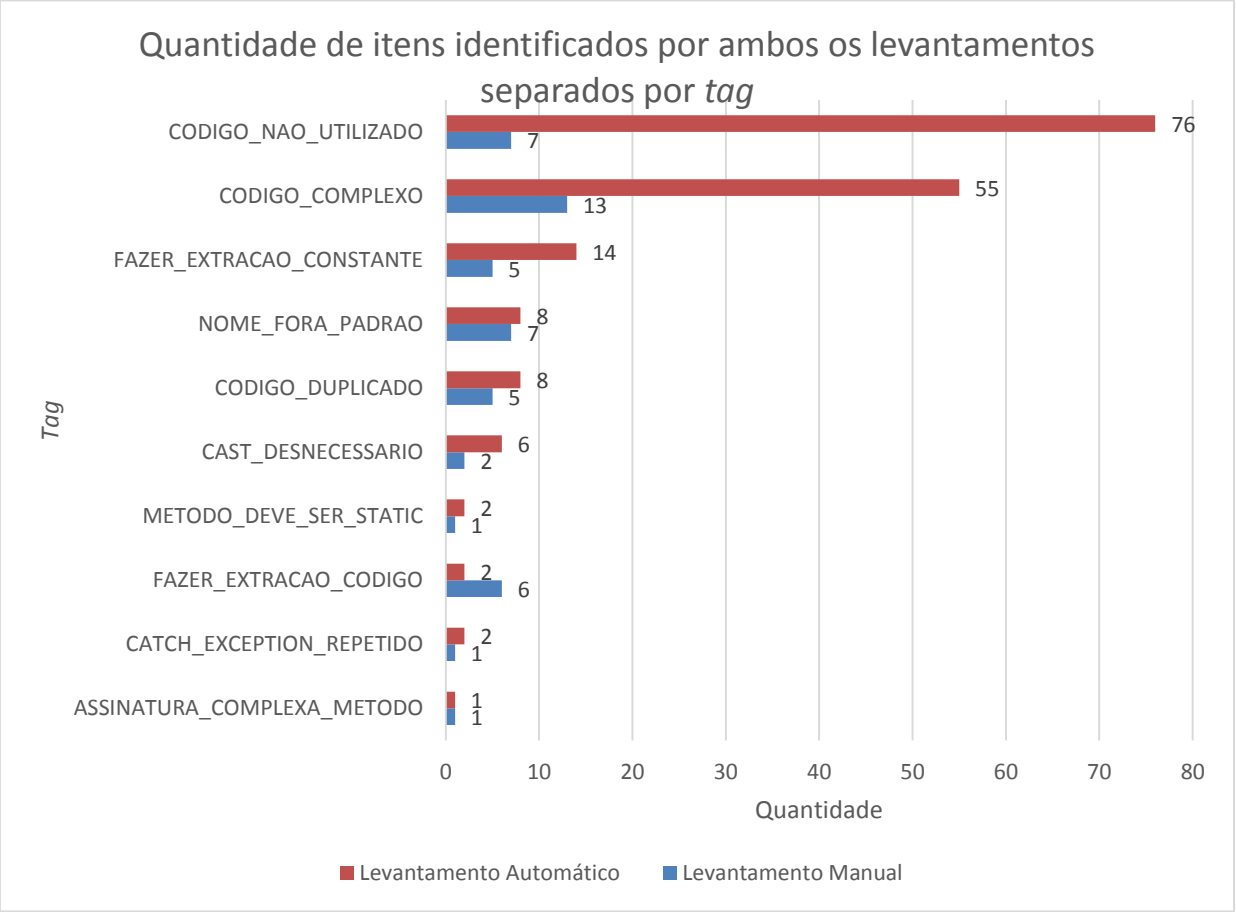
A Figura 13 apresenta o resultado da análise realizada entre as tags de itens identificadas pelo método manual com as do método automático, mostrando que 10 tags foram encontradas por ambos e que o subconjunto de tags identificadas exclusivamente pelo método automático foi um pouco maior do que o dobro do que foi identificado exclusivamente pelo método manual.

Figura 13 - Análise dos resultados entre os levantamentos manual e automático



Considerando o subconjunto das 10 *tags* em comum para os dois levantamentos, a Figura 14 mostra a quantidade de itens identificados com a *tag* para cada levantamento. A única *tag* em que a quantidade de itens identificados pelo método manual foi maior do que a automática foi a FAZER_EXTRACAO_CODIGO. Houve empate de quantidade para a *tag* ASSINATURA_COMPLEXA_METODO, sendo o mesmo método apontado por ambos os levantamentos. Para as demais *tags,* o levantamento automático identificou mais itens do que o manual. Isto mostra que, embora a identificação manual consiga identificar parte das mesmas características de itens de dívida técnica que a identificação automática (ponto de vista qualitativo), esta última tem maior eficiência do ponto de vista quantitativo do que a primeira. Isto significa que, apesar de os participantes saberem dos problemas e impactos gerados no software, analisar o código de forma manual pode fazer com que alguns dos problemas passem despercebidos.

Figura 14 - Quantidade de itens identificados por ambos os levantamentos separados por *tag*



5.9 Apresentação da dívida técnica identificada exclusivamente pelo método automatizado aos participantes

Separados os itens em relação às *tags* nos três subconjuntos, o próximo passo foi apresentar aos participantes do levantamento manual somente o subconjunto das 27 *tags* dos itens que foram identificados somente pelo método automatizado. Isto foi feito por meio do envio de um questionário em formato de planilha para cada participante contendo as descrições dos problemas relacionados às *tags*, além de três perguntas a serem respondidas, como apresentado na Tabela 6. Para a última pergunta, optou-se por deixar a resposta como texto livre no lugar de sugerir respostas pré-definidas (múltipla escolha) com o objetivo de evitar que as respostas pudessem influenciar a resposta de alguma forma.

Tabela 6 - Estrutura do questionário enviado aos participantes com os problemas encontrados somente pelo levantamento automático

Descrição do problema	Descrição do problema retirado do resultado do SonarQube
Amostra de item encontrado com o problema	Trecho de código onde o SonarQube identificou o problema
Você considera este item como sendo dívida técnica?	**Valores:** Sim/Não
Caso tenha considerado este item como dívida técnica, em sua opinião, qual é a relevância deste item em relação ao prejuízo que ele pode causar?	**Valores:** Baixo/Médio/Alto
Caso tenha considerado este item como dívida técnica, qual foi o motivo de não ter sido identificado no levantamento manual?	Texto livre

O objetivo desta tarefa foi analisar a relevância destes itens por meio das respostas fornecidas pelos participantes para dar uma resposta inicial ao questionamento do trabalho de Zazworka et al. (2013), foco deste trabalho: quanto da dívida técnica potencial apontada pelas ferramentas, porém não pelos desenvolvedores, é real? Há benefícios em utilizar ferramentas que identificam dívida técnica que os desenvolvedores não têm conhecimento ou este subconjunto é irrelevante? (ZAZWORKA et al., 2013, p. 7).

Assim como no levantamento de itens de dívida técnica, os participantes foram informados para responder às perguntas de forma isolada, sem a opinião ou consulta das respostas dos demais e também que enviassem a planilha preenchida de volta. Foi-lhes dado três dias, a partir da data de entrega da planilha de preenchimento, como tempo limite para conclusão da tarefa.

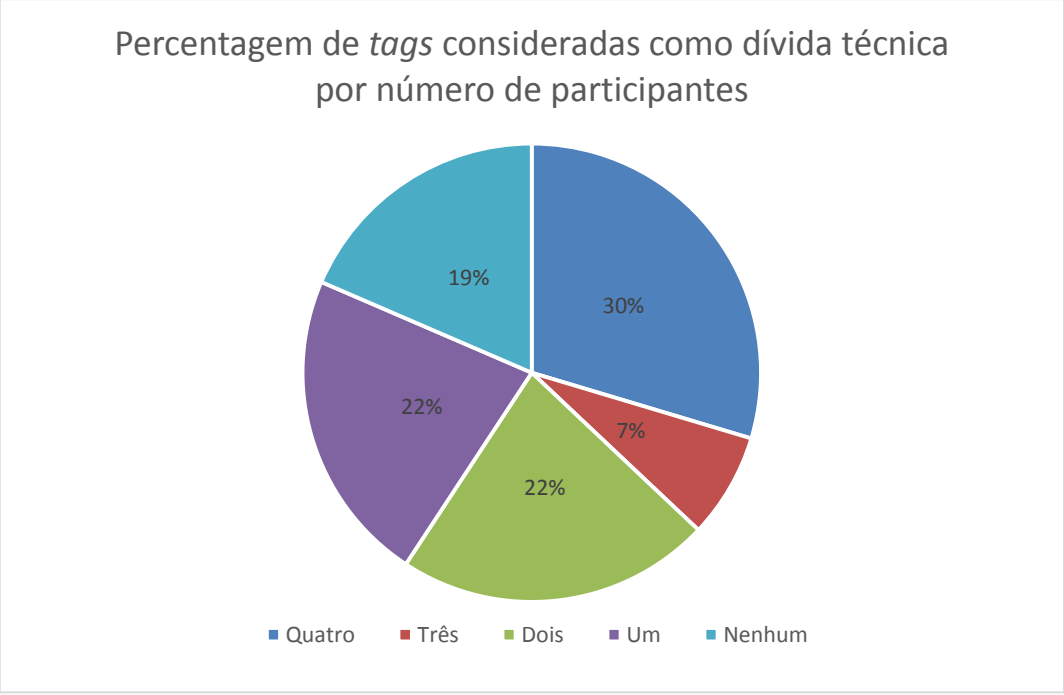
5.10 Resultados da análise da dívida técnica encontrada exclusivamente pelo levantamento automático

Concluídas as tarefas de análise do subconjunto de dívida técnica identificado somente pelo SonarQube, os quatro participantes enviaram as planilhas com as informações solicitadas sobre as 27 *tags* e o resultado foi consolidado para análise.

As respostas da última pergunta – o porquê de não ter identificado o item durante o levantamento manual, caso o considerasse como sendo dívida técnica, foram padronizadas e simplificadas para organizar o resultado, gerando cinco diferentes motivos:

- Não analisou o trecho de código;
- Falta de conhecimento do problema;
- Conhece o problema, mas não percebeu por falta de atenção;
- Conhece o problema, mas considera difícil identificá-lo manualmente e
- Conhece o problema, mas não se lembrou dele para a análise.

Figura 15 – Percentagem de *tags* consideradas como dívida técnica por número de participantes



A Figura 15 apresenta inicialmente o resultado da opinião dos participantes em relação à primeira pergunta, isto é, se consideravam os itens como sendo dívida técnica ou não.

Tabela 7 – Conjunto de *tags* que não foi considerado como dívida técnica pelos participantes

Tag	**Tipo**	**Importância**
AUSENCIA_OVERRIDE_ANNOTATION	*Code Smell*	*Major*
CLASSE_UTIL_SEM_CONSTRUTOR	*Code Smell*	*Major*
MUDAR_COMPARACAO_BOOLEAN	*Code Smell*	*Minor*
CONVENCAO_ARRAY_DESIGNADOR	*Code Smell*	*Minor*
CONVENCAO_ORDEM_METODO	*Code Smell*	*Minor*

Para o conjunto considerado dívida técnica por apenas um participante, seis *tags* (22% do total) foram apontadas, não necessariamente pelo mesmo participante.

A Tabela 8 apresenta o resultado mostrando que, com exceção de BUG_SEMPRE_TRUE_OU_FALSE, todos os itens foram classificados como *code smells* pelo SonarQube, sendo em sua maioria classificada com importância *minor*.

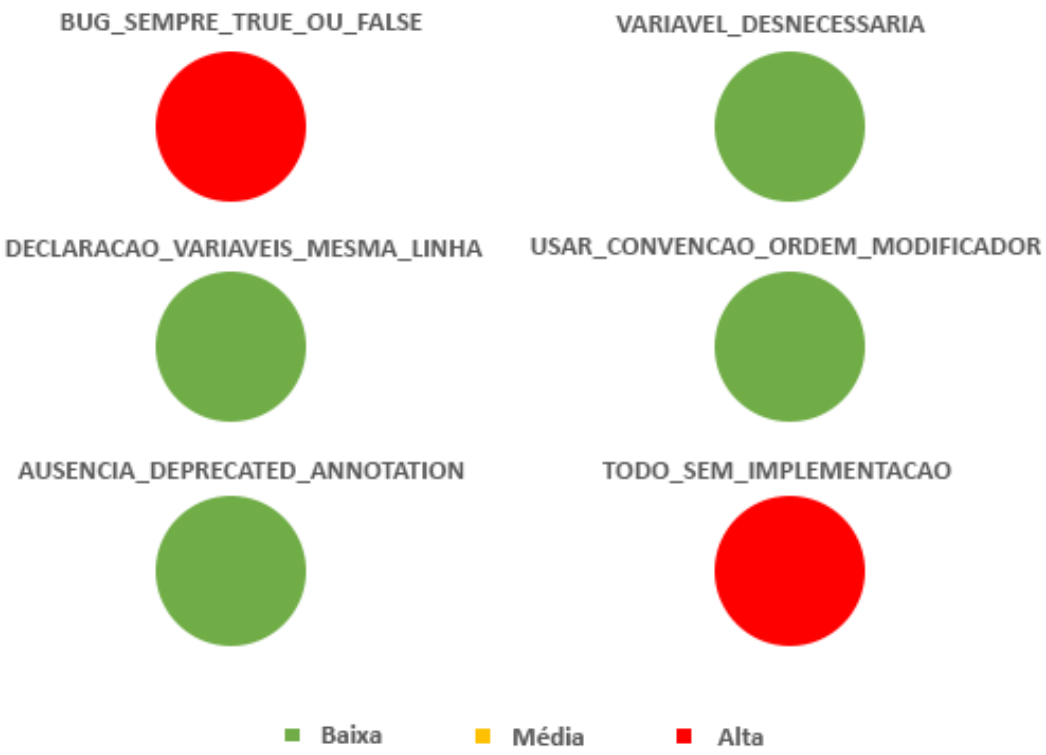
Tabela 8 – Conjunto de *tags* que foi considerado como dívida técnica por um participante

Tag	**Tipo**	**Importância**
BUG_SEMPRE_TRUE_OU_FALSE	*Bug*	*Blocker*
DECLARACAO_VARIAVEIS_MESMA_LINHA	*Code Smell*	*Minor*
AUSENCIA_DEPRECATED_ANNOTATION	*Code Smell*	*Major*
TODO_SEM_IMPLEMENTACAO	*Code Smell*	*Info*
USAR_CONVENCAO_ORDEM_MODIFICADOR	*Code Smell*	*Minor*
VARIAVEL_DESNECESSARIA	*Code Smell*	*Minor*

Particularmente para a *tag* BUG_SEMPRE_TRUE_OU_FALSE, os casos identificados pelo SonarQube foram apontados como falsos positivos por três participantes – os que responderam não considerar dívida técnica para esse item – ao avaliarem o trecho de código identificado pelo SonarQube enviada na planilha, ou seja, a ferramenta apontou um problema que não existia de verdade.

Ainda para o conjunto considerado dívida técnica por apenas um participante, a Figura 16 apresenta o resultado da classificação da relevância dos problemas. Embora BUG_SEMPRE_TRUE_OU_FALSE tenha disso identificado como falso positivo, o participante classificou este tipo de problema com relevância alta, já que esta *tag* representa um *bug* no sistema. TODO_SEM_IMPLEMENTACAO também foi classificado como alta relevância, mostrando que tarefas incompletas devem ser concluídas. As demais *tags* foram classificadas com relevância baixa e nenhuma delas representava um problema.

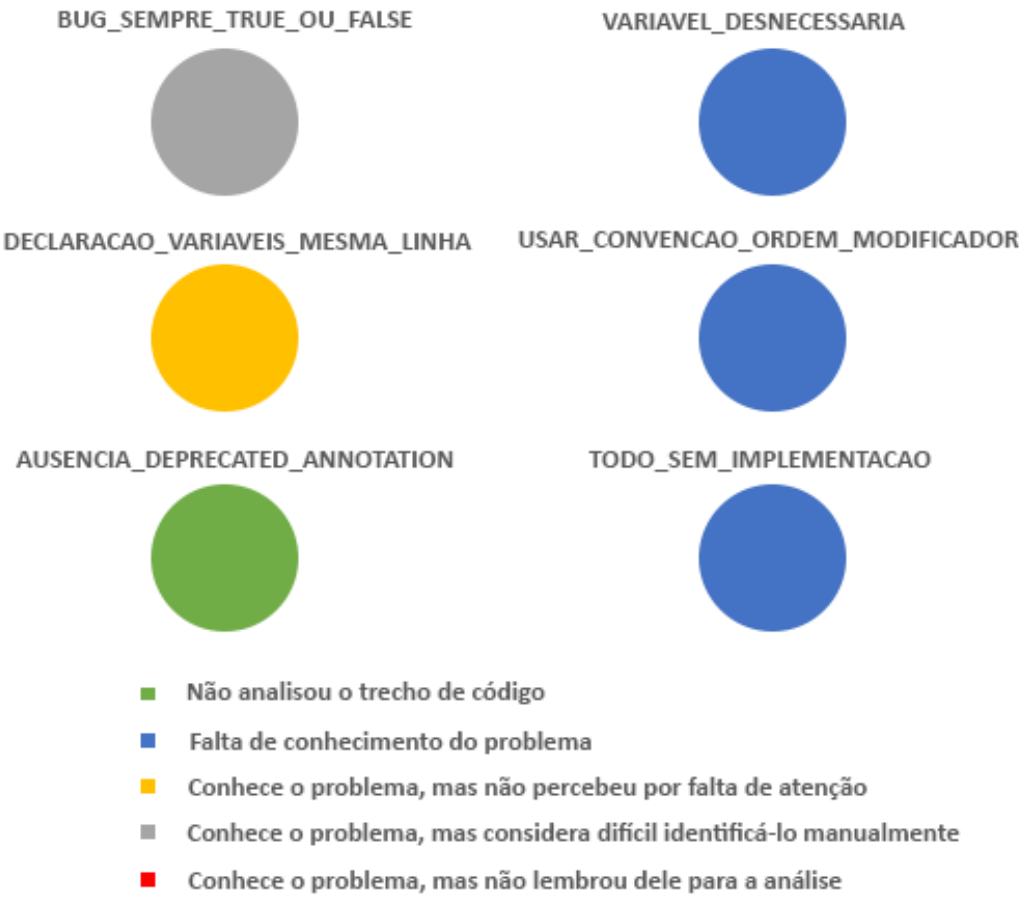
Figura 16 - Relevância de *tags* consideradas como dívida técnica por um participante



Quanto aos motivos informados pelos participantes respondendo o porquê de não terem identificado estes itens na análise manual, a Figura 17 mostra que, para metade dos casos, a causa foi a falta de conhecimento do problema pelos participantes, ou seja, não sabiam ou não tinham parado para analisar que os trechos apontados pelo SonarQube para estes casos poderiam impactar tarefas de manutenção e evolução do sistema.

Para um caso, foi informado que trecho de Javadoc não foi analisado (AUSENCIA_DEPRECATED_ANNOTATION) e para os últimos dois foi informado que, apesar de saber dos problemas, um passou despercebido (DECLARACAO_VARIAVEIS_MESMA_LINHA) na análise e o outro não é fácil de identificar manualmente (BUG_SEMPRE_TRUE_OU_FALSE).

Figura 17 - Motivo de não ter identificado *tags* consideradas dívida técnica por um participante



Para o conjunto considerado dívida técnica por dois participantes, seis *tags* (22% do total) foram apontadas. A Tabela 9 apresenta o resultado, mostrando que todos os itens foram classificados como *code smells* pelo SonarQube, sendo em sua maioria de importância *major*.

A Figura 18 apresenta o resultado da classificação da relevância dos problemas pelos participantes. Três itens foram classificados inteiramente com baixa relevância

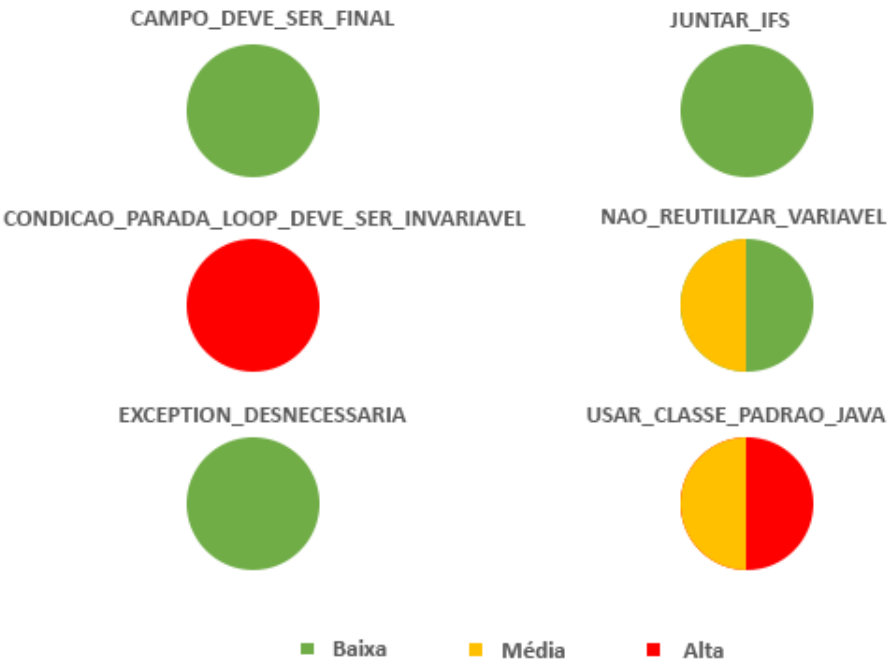
(CAMPO_DEVE_SER_FINAL, JUNTAR_IFS e EXCEPTION_DESNECESSARIA), nenhum destes itens são problemas no sistema, apenas pontos de atenção e de melhoria de legibilidade.

Tabela 9 - Conjunto de *tags* que foi considerado como dívida técnica por dois participantes

Tag	Tipo	Importância
CAMPO_DEVE_SER_FINAL	*Code Smell*	*Major*
CONDICAO_PARADA_LOOP_DEVE_SER_INVARIAVEL	*Code Smell*	*Major*
EXCEPTION_DESNECESSARIA	*Code Smell*	*Minor*
JUNTAR_IFS	*Code Smell*	*Major*
NAO_REUTILIZAR_VARIAVEL	*Code Smell*	*Major*
USAR_CLASSE_PADRAO_JAVA	*Code Smell*	*Major*

O item NAO_REUTILIZAR_VARIAVEL – problema que, além de diminuir a legibilidade do código, pode confundir o desenvolvedor – foi considerado ter relevâncias baixa e média e USAR_CLASSES_PADRAO_JAVA foi apontado com relevâncias média e alta, representando um problema potencial que tem como consequência a incompatibilidade do sistema com novas versões do Java.

Figura 18 - Relevância de *tags* consideradas como dívida técnica por dois participantes



O único item classificado inteiramente com alta relevância foi CONDICAO_PARADA_LOOP_DEVE_SER_INVARIAVEL, sendo este uma má prática de programação que dificulta a legibilidade e facilita a entrada de *bugs*.

Em relação aos motivos que os participantes informaram por não terem identificado estes itens na análise manual, a Figura 19 mostra que a falta de conhecimento do problema se mantém como principal causa, seguida da falta de atenção na análise do código.

Figura 19 - Motivo de não ter identificado *tags* consideradas dívida técnica por dois participantes

O item USAR_CLASSE_PADRAO_JAVA foi apontado como trecho de código não analisado, pois se tratava de declaração de pacotes e classes. Para o item NÃO_REUTILIZAR_VARIAVEL, um dos participantes declarou não ter reconhecido

como problema na análise e o outro informou que este tipo de problema não é fácil de identificar manualmente.

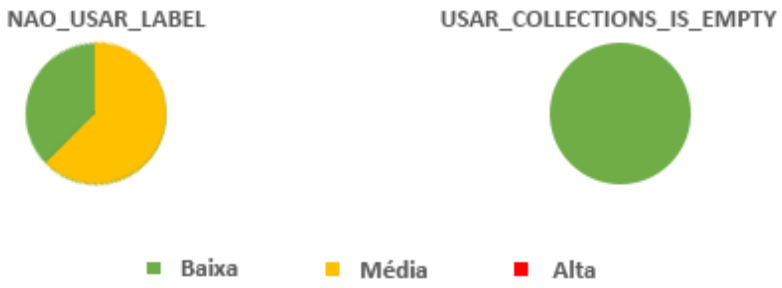
Apenas duas *tags* (7% do total) foram apontadas como dívida técnica por três participantes. A Tabela 10 mostra que ambos os itens foram classificados como *code smells* pelo SonarQube com importância *major*.

Tabela 10 - Conjunto de tags que foi considerado como dívida técnica por três participantes

Tag	**Tipo**	**Importância**
NAO_USAR_LABEL	*Code Smell*	*Major*
USAR_COLLECTIONS_IS_EMPTY	*Code Smell*	*Major*

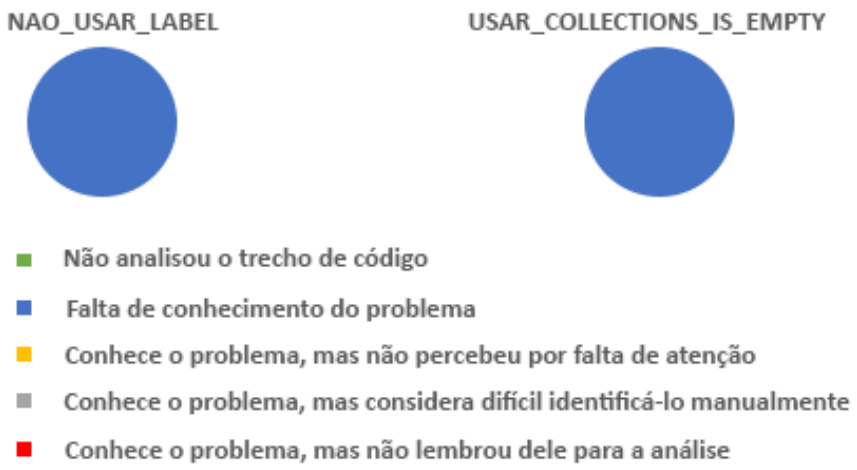
A Figura 20 apresenta a relevância dos problemas informada pelos três participantes. O item NAO_USAR_LABEL foi considerado por dois participantes com relevância média e por um com baixa relevância, representando um problema que reduz a legibilidade do código, abrindo espaço para a introdução de *bugs*. O item USAR_COLLECTIONS_IS_EMPTY foi considerado pelos três participantes com baixa relevância.

Figura 20 - Relevância de *tags* consideradas como dívida técnica por três participantes



Como apresentado na Figura 21, os dois itens compartilharam o mesmo motivo do porquê de não terem sido identificados na análise manual: falta de conhecimento do problema, principalmente em relação ao item USAR_COLLECTIONS_IS_EMPTY, para o qual a análise do SonarQube apontou que a não correção deste item afeta negativamente o desempenho do sistema.

Figura 21 - Motivo de não ter identificado *tags* consideradas dívida técnica por três participantes



O último conjunto, itens considerados dívida técnica por todos os quatro participantes, foi composto por oito *tags* (30% do total). A Tabela 11 apresenta o resultado, mostrando a classificação destes itens pelo SonarQune: três *bugs,* uma vulnerabilidade e 4 *code smells*.

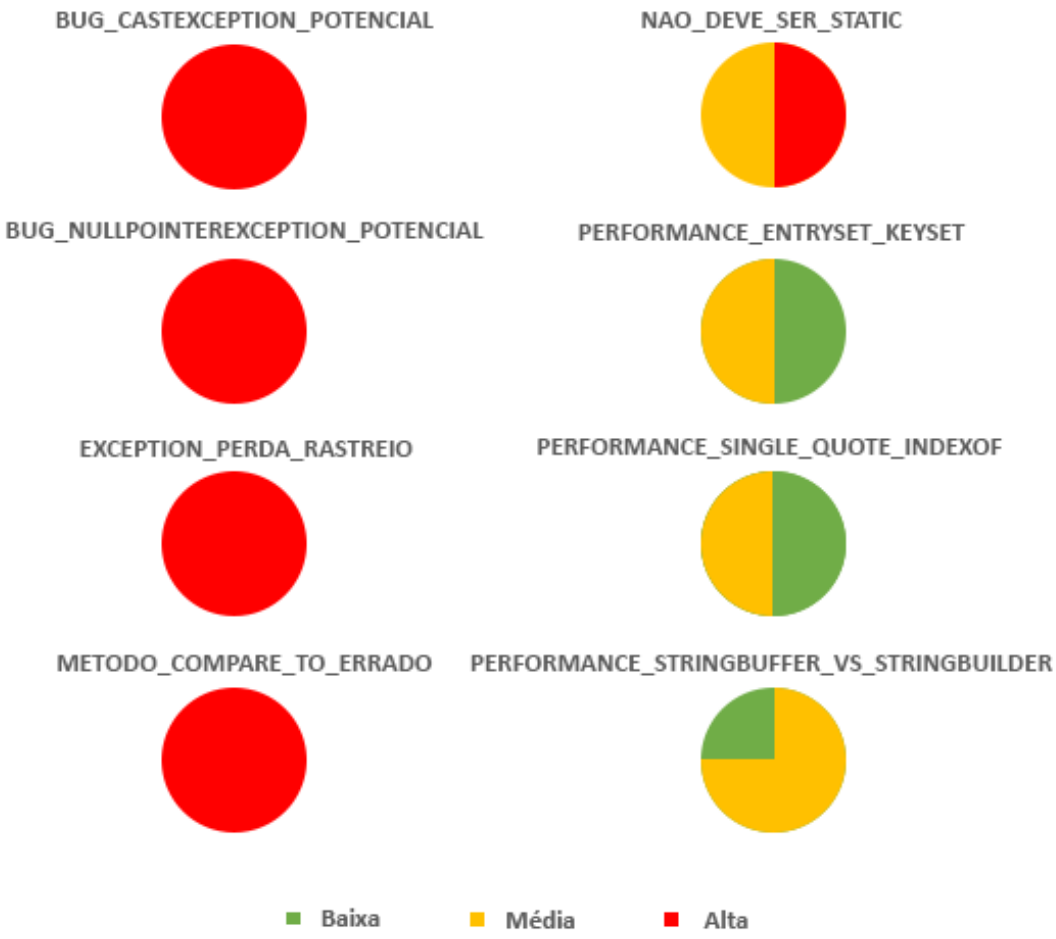
Tabela 11 - Conjunto de *tags* que foi considerado como dívida técnica por todos os participantes

Tag	**Tipo**	**Importância**
BUG_CASTEXCEPTION_POTENCIAL	*Bug*	*Blocker*
BUG_NULLPOINTEREXCEPTION_POTENCIAL	*Bug*	*Blocker*
EXCEPTION_PERDA_RASTREIO	Vulnerabilidade	*Critical*
METODO_COMPARE_TO_ERRADO	*Code Smell*	*Major*
NAO_DEVE_SER_STATIC	*Bug*	*Critical*
PERFORMANCE_ENTRYSET_KEYSET	*Code Smell*	*Major*
PERFORMANCE_SINGLE_QUOTE_INDEXOF	*Code Smell*	*Minor*
PERFORMANCE_STRINGBUFFER_VS_STRINGBUILDER	*Code Smell*	*Major*

A relevância dos itens classificada pelos participantes, conforme apresentada na Figura 22, mostra forte relação com a classificação realizada pelo SonarQube. Os três itens categorizados como *bugs*, a vulnerabilidade e o *code smell*

METODO_COMPARE_TO_ERRADO foram classificados com alta relevância pelos participantes. Os outros três itens – todos *code smells* – tiveram relevâncias baixa e média, tratando-se de itens que afetam o desempenho.

Figura 22 - Relevância de *tags* consideradas como dívida técnica por todos os participantes

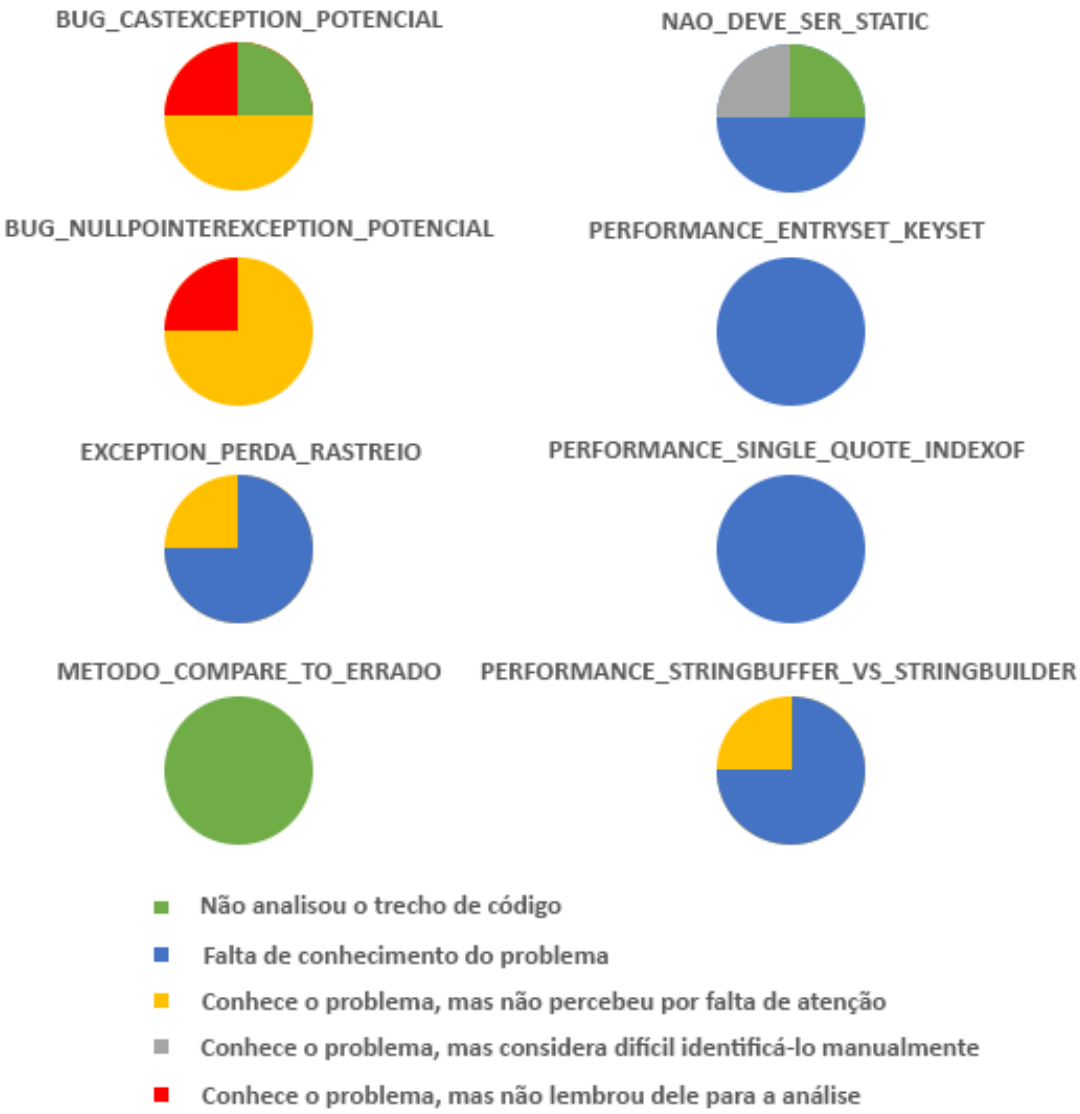


Quanto aos motivos da não identificação no levantamento manual informados pelos participantes, a Figura 23 mostra que, novamente, a falta de conhecimento do problema foi o motivo mais frequente, principalmente em relação aos itens que afetam o desempenho do sistema.

Embora tenham conhecimento de problemas, a falta de atenção ao analisar o código também apareceu em diversos itens, reforçando como importante causa da não identificação de dívida técnica no levantamento automático. Para o item

METODO_COMPARE_TO_ERRADO, todos os participantes informaram que não analisaram este trecho de código.

Figura 23 - Motivo de não ter identificado *tags* consideradas dívida técnica por todos participantes

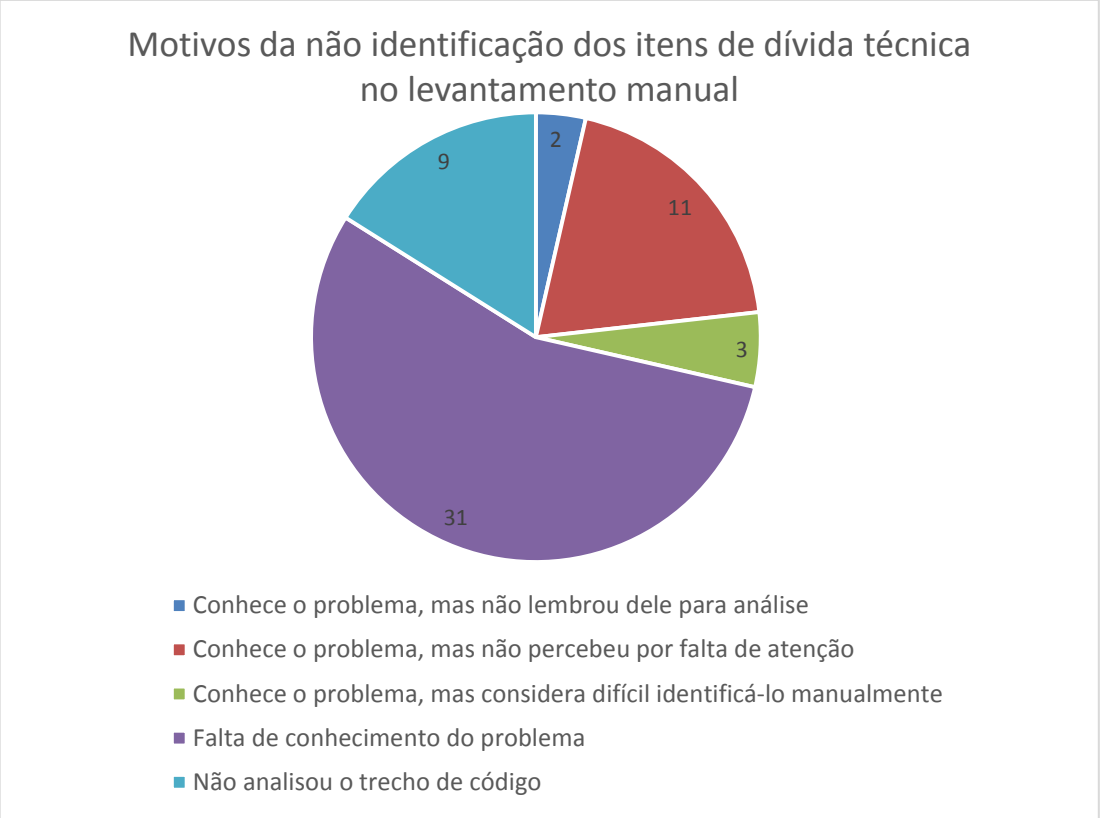


Organizando todas as respostas dos participantes sobre os itens identificados exclusivamente pelo método automático, o resultado final é que, dos 27 itens que compuseram este conjunto, 22 deles (81%) foram considerados relevantes por pelo menos um participante.

Partindo destes 22 itens considerados dívida técnica, foram informadas 56 respostas com relação à relevância dos itens (no máximo seriam 88 respostas caso todos os participantes tivessem considerado todos os itens), sendo 20 delas (36%) com importância baixa, 13 (23%) com média e 23 (41%) com alta importância.

Em relação aos motivos pelos quais os participantes não identificaram os itens considerados dívida técnica, conforme apresentado na Figura 24, o motivo preponderante foi a falta de conhecimento dos problemas pelos participantes.

Figura 24 - Motivos da não identificação dos itens de dívida técnica no levantamento manual



5.11 Discussão

Neste trabalho, a aplicação do método de levantamento manual de dívida técnica realizada de forma isolada por quatro participantes em um software real mostrou que, apesar dos participantes terem identificado diversos problemas e seus impactos no software, houve pouca sobreposição de resultados (29%) considerando o critério

de igualdade definido neste trabalho, mostrando que a experiência e a visão de cada indivíduo sobre o que é considerado problema variam.

Os diferentes pontos de vista dos participantes foram reforçados por meio da análise dos resultados do conjunto de itens de dívida técnica identificado exclusivamente pelo SonarQube, em que cada participante, novamente de forma isolada, respondeu se considerava cada uma das 27 *tags* como dívida técnica ou não (Figura 15). O resultado deste *feedback* mostrou que a concordância total entre eles, isto é, os casos em que todos os participantes concordaram ou discordaram sobre o item apontado pelo SonarQube ser dívida técnica, ocorreu em 49% dos casos (13 dos 27 itens, sendo 5 deles não considerados como dívida técnica e 8 considerados).

Ainda em relação ao resultado do *feedback* dado pelos participantes sobre o conjunto de itens identificados exclusivamente pelo SonarQube, para os itens em que pelo menos dois participantes concordaram ser dívida técnica (16 itens ou 59% do total), a subjetividade dos participantes gerou diferentes respostas em relação à classificação da relevância dos itens e os motivos pelos quais eles não foram identificados durante o levantamento manual. Quanto à relevância informada para estes 16 itens, metade deles foi classificada com o mesmo nível pelos participantes. Para a outra metade, os participantes tiveram opiniões diferentes. Quanto aos motivos informados pelos participantes por não terem identificado manualmente os itens, somente 9 dos 16 tiveram a mesma resposta compartilhada entre os participantes.

Importante notar que, além do conjunto de itens que somente o SonarQube conseguiu identificar, também houve um conjunto de itens considerado dívida técnica que somente os participantes do levantamento manual conseguiram. Parte deste resultado é devido ao fato de que alguns arquivos analisados pelos participantes (como o arquivo pom.xml do Maven) não são analisados pelo SonarQube, além de problemas como documentação desatualizada em relação ao código, itens impraticáveis de serem detectados atualmente por ferramentas como SonarQube. Estes fatos reforçam o resultado do trabalho conduzido por Zazworka et

al. (2013), onde foi citada a importância do envolvimento de humanos no processo de identificação de dívida técnica.

Após a tarefa final do envio do *feedback* dos itens identificados exclusivamente pelo SonarQube, em uma conversa com dois dos quatro participantes foi dito que a falta de conhecimento técnico e prazos apertados – uma das causas citadas por Rubin (2012) – são os principais fatores para o surgimento de dívida técnica nos sistemas.

Considerando que a falta de conhecimento do problema foi o motivo mais informado pelos participantes para explicar a não identificação do conjunto de itens levantados exclusivamente pelo SonarQube (55% dos casos) e que esse mesmo fator foi associado a um dos maiores causadores de dívida técnica no sistema, concluímos que a dívida técnica imprudente e não intencional, conforme o quadrante de classificação de dívida técnica de Fowler (2009), foi o tipo de dívida técnica que mais justifica a utilização de ferramentas como SonarQube.

Ainda na discussão com os participantes, com exceção de problemas que impactam diretamente a operação da companhia, na discussão com os participantes foi dito que grande parte da dívida técnica que eles conhecem – dívida técnica intencional, tanto prudente quanto imprudente, de acordo com Fowler (2009) – se mantém por muito tempo no sistema porque não há tanto espaço para realizar tarefas que ajudam a pagar parte da dívida técnica, tais como refatoração de código ou atualização de documentação, sendo deixadas em segundo plano em relação às tarefas que adicionam valor visível ao cliente, tais como construção de novas funcionalidades.

Outro ponto com relação ao levantamento automático de dívida técnica feito pelo SonarQube, além do conjunto de dívida técnica que somente ele foi capaz de identificar, já citado anteriormente, a ferramenta se mostrou mais eficiente do que os participantes do ponto de vista quantitativo, ou seja, para o conjunto de problemas que foi identificado por ambos os levantamentos (Figura 14), a quantidade total de ocorrência dos itens identificada pelo SonarQube em comparação à identificada pelos participantes foi aproximadamente 3.6 vezes maior (174 apontamentos contra 48). Este fato evidencia a importância da utilização de ferramentas de detecção

automática de dívida técnica como o SonarQube, mesmo para a identificação de itens que as pessoas são capazes de identificar, pois diferente dos humanos, as ferramentas não falham neste quesito quantitativo.

Por fim, em resposta à dúvida originada do trabalho de Zazworka et al. (2013): “quanto da dívida técnica potencial apontada pelas ferramentas, porém não pelos desenvolvedores, é real? Há benefícios em utilizar ferramentas que identificam dívida técnica que os desenvolvedores não têm conhecimento ou este subconjunto é irrelevante?” (ZAZWORKA et al., 2013, p. 7), o resultado consolidado deste trabalho apontou que os desenvolvedores consideraram 81% do resultado encontrado exclusivamente pelo SonarQube como relevante, sendo a falta de conhecimento dos problemas o principal motivo mencionado pelos desenvolvedores pela não identificação no procedimento manual.

6 CONSIDERAÇÕES FINAIS

Este capítulo apresenta as conclusões obtidas com este trabalho, suas principais contribuições e recomendações, bem como os trabalhos futuros.

6.1 Conclusões

Este trabalho apresentou o conceito de dívida técnica, explicando sua metáfora com a dívida financeira, classificações do ponto de vista de diferentes autores, possíveis origens e impactos nas tarefas de manutenção e evolução do software.

Também foram apresentadas linhas gerais sobre abordagens existentes para o gerenciamento de dívida técnica, descrevendo tarefas como identificar, medir e pagar dívida técnica.

Em especial, a tarefa de identificação de dívida técnica foi aprofundada para direcionar o trabalho ao seu principal objetivo: analisar a relevância da dívida técnica identificada exclusivamente por método automático. Partindo disto, foram apresentados alguns métodos de identificação de dívida técnica, tanto manuais quanto automáticos.

Seguindo em direção ao objetivo do trabalho, duas abordagens de identificação de dívida técnica – uma manual e outra automática – foram selecionadas para serem aplicadas em um software real: criação manual de lista de itens de dívida técnica proposta por Seaman e Guo (2011) e identificação automática por meio do SonarQube, ferramenta especializada para lidar com dívida técnica.

Este trabalho apresentou todos os passos, detalhes e resultados da aplicação dos dois métodos escolhidos em um software real. A abordagem manual exigiu a participação de desenvolvedores que trabalham ou já trabalharam com o software para identificarem dívida técnica.

Para a análise manual, o resultado consolidado de todos os participantes mostrou que 23 diferentes problemas foram identificados. Além disto, o resultado evidenciou que eles têm diferentes visões do que consideram dívida técnica no software e,

mesmo para os itens que foram identificados por mais de um participante, em muitos casos a percepção sobre a relevância destes itens divergiram.

Para a análise automática, os resultados obtidos pelo SonarQube mostraram que a ferramenta identificou maior variedade de itens de dívida técnica (foram 37 diferentes problemas) e também maior quantidade do que os identificados pelos desenvolvedores, apresentando mais detalhes sobre os problemas e inclusive formas de resolvê-los.

Concluídas as análises dos resultados de cada abordagem separadamente, o estudo realizou uma análise de caráter comparativo entre os resultados obtidos das duas abordagens. O resultado desta análise gerou três subconjuntos de dívida técnica: dívida técnica identificada exclusivamente pelo método manual (13 itens), dívida técnica identificada exclusivamente pela análise automática (27 itens) e dívida técnica identificada pelos dois métodos (10 itens).

Este resultado mostrou que, embora a ferramenta automática consiga detectar mais tipos de dívida técnica e em maiores quantidades do que os participantes, ela não foi capaz de identificar alguns problemas identificados pelos participantes, evidenciando que uso conjunto de ferramentas e pessoas para a identificação de dívida técnica tem maior valor do que uma única abordagem, como Zazworka et al. (2013) haviam concluído.

Considerando o subconjunto de dívida técnica identificada somente pelo método automático, foi realizada a etapa final deste trabalho: analisar a relevância deste subconjunto do ponto de vista dos desenvolvedores. Isto foi feito por meio da apresentação destes itens aos desenvolvedores, que responderam perguntas relacionadas à relevância do subconjunto e, para os itens que consideravam dívida técnica, os porquês de não terem identificado no procedimento manual.

A análise do *feedback* dos participantes em relação a esse subconjunto gerou algumas importantes conclusões para ajudar a responder o objetivo principal deste estudo. A primeira foi que os participantes consideraram 22 dos 27 itens (81%) relevantes, ou seja, a maioria dos itens foi considerada como dívida técnica. A

segunda, relativa ao nível de relevância dos itens considerados dívida técnica, mostrou que a maior parte das respostas (41%) considerou os itens como muito relevantes. A terceira mostrou que o principal motivo pelo qual os participantes não identificaram os itens de dívida técnica durante o levantamento manual foi porque não sabiam dos problemas apontados. Isso quer dizer que, mesmo desconsiderando a ineficiência da análise manual propensa a falhas humanas (como falta de atenção ou não ter analisado parte do código), eles não identificariam tais problemas.

Desta forma, as conclusões finais obtidas neste trabalho apontaram que o conjunto de dívida técnica identificado exclusivamente pela ferramenta de identificação automática de dívida técnica foi considerado relevante pelos desenvolvedores.

6.2 Contribuições do Trabalho

Este trabalho contribui para o cenário da dívida técnica em relação à sua identificação, descrevendo alguns dos métodos e ferramentas existentes que apoiam esta tarefa. Além disto, o estudo apresentou detalhes da aplicação e análise de dois diferentes métodos – um manual e o outro automático – em um sistema de software real, fazendo uma análise comparativa dos resultados de ambas as abordagens.

A análise comparativa dos resultados de ambas as abordagens contribuiu para o entendimento dos pontos fracos e fortes de cada abordagem ao apontar as diferenças encontradas nos resultados, mostrando que, apesar da tarefa de identificação manual ser propensa a falhas como falta de atenção, ela tem o potencial de identificar dívida técnica que a ferramenta não consegue. Por outro lado, apesar da ferramenta automática encontrar itens irrelevantes ou mesmo falsos positivos, a grande maioria foi considerada relevante e inclusive não seria identificada pelos participantes. Este resultado contribui ao reforçar que o uso conjunto de ferramentas e pessoas gera mais valor do que somente uma abordagem.

Outra contribuição foi em relação à proposta principal deste trabalho, se a utilização de ferramentas de identificação automática de dívida técnica traz benefícios ao

encontrar resultados relevantes para os desenvolvedores, em especial o conjunto de dívida técnica identificado somente pela ferramenta. A análise final dos resultados forneceu uma resposta inicial ao apresentar dados apontando que, sim, o resultado identificado exclusivamente pelo uso da ferramenta automática de identificação de dívida técnica é relevante.

6.3 Trabalhos Futuros

A análise deste estudo foi limitada à opinião de quatro participantes, uma única ferramenta automática e um único sistema de software. Recomenda-se como trabalho futuro realizar uma comparação mais profunda entre as duas abordagens (manual e automática), envolvendo um número maior de pessoas e diferentes ferramentas de identificação de dívida técnica para obter respostas mais sólidas e menos enviesadas.

Outra proposta de trabalho futuro é realizar um estudo sobre o conjunto de dívida técnica identificada pelos desenvolvedores que não foram identificados pela ferramenta, tendo como objetivo entender quais são os fatores e obstáculos que ainda não permitem a identificação deste conjunto pelas ferramentas. Em complemento a isto, recomenda-se também como trabalho analisar ou desenvolver maneiras de tornar essa tarefa possível, de modo que a tarefa de identificação manual – propensa a falhas inerentes aos humanos – se torne menos necessária para a obtenção de resultados mais sólidos.

REFERÊNCIAS

ALLMAN, E. Managing Technical Debt. **Communications of the ACM**, v. 55, n. 5, p. 50–55, 2012.

CAMPBELL, G. A.; PAPAPETROU, P. P.; GAUDIN, O. **SonarQube in action**. Shelter Island, NY: Manning, 2014.

CODABUX, Z.; WILLIAMS, B. **Managing technical debt: An industrial case study**. Proceedings of the 4th International Workshop on Managing Technical Debt. **Anais**...IEEE Press, 2013Disponível em: <http://dl.acm.org/citation.cfm?id=2663299>. Acesso em: 11 maio. 2016

CUNNINGHAM, W. **The WyCash Portfolio Management System**. Addendum to the Proceedings on Object-oriented Programming Systems, Languages, and Applications (Addendum). **Anais**...: OOPSLA '92.New York, NY, USA: ACM, 1992Disponível em: <http://doi.acm.org/10.1145/157709.157715>. Acesso em: 25 jun. 2016

FOWLER, M. et al. **Refactoring: Improving the Design of Existing Code**. 1 edition ed. Reading, MA: Addison-Wesley Professional, 1999.

FOWLER, M. **Technical Debt**. Disponível em: <http://martinfowler.com/bliki/TechnicalDebt.html>. Acesso em: 25 jun. 2016a.

FOWLER, M. **Cannot Measure Productivity**. Disponível em: <http://martinfowler.com/bliki/CannotMeasureProductivity.html>. Acesso em: 25 jun. 2016b.

FOWLER, M. **Technical Debt Quadrant**. Disponível em: <http://martinfowler.com/bliki/TechnicalDebtQuadrant.html>. Acesso em: 4 jun. 2016.

GUO, Y.; SEAMAN, C. **A Portfolio Approach to Technical Debt Management**. Proceedings of the 2Nd Workshop on Managing Technical Debt. **Anais**...: MTD '11.New York, NY, USA: ACM, 2011Disponível em: <http://doi.acm.org/10.1145/1985362.1985370>. Acesso em: 25 jun. 2016

INSPEARIT. **SQALE | Software Quality Assessment based on Lifecycle Expectations**. Disponível em: <http://www.sqale.org/>. Acesso em: 29 jul. 2016.

MANTYLA, M. V.; VANHANEN, J.; LASSENIUS, C. **Bad smells-humans as code critics**. Software Maintenance, 2004. Proceedings. 20th IEEE International Conference on. **Anais**...IEEE, 2004Disponível em: <http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1357825>. Acesso em: 22 jun. 2016

MCCONNELL, S. **Technical Debt**. Disponível em: <http://www.construx.com/10x_Software_Development/Technical_Debt/>. Acesso em: 20 jun. 2016.

NUGROHO, A.; VISSER, J.; KUIPERS, T. **An empirical model of technical debt and interest**. Proceedings of the 2nd Workshop on Managing Technical Debt. **Anais**...ACM, 2011Disponível em: <http://dl.acm.org/citation.cfm?id=1985364>. Acesso em: 11 maio. 2016

RUBIN, K. S. **Essential Scrum: A Practical Guide to the Most Popular Agile Process**. 1 edition ed. Upper Saddle River, NJ: Addison-Wesley Professional, 2012.

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

SINGH, V.; SNIPES, W.; KRAFT, N. A. **A Framework for Estimating Interest on Technical Debt by Monitoring Developer Activity Related to Code Comprehension**. IEEE, set. 2014Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6974886>. Acesso em: 22 mar. 2016

SONARSOURCE. **SonarQube Documentation**. Disponível em: <http://docs.sonarqube.org/display/SONAR/Documentation>. Acesso em: 28 jul. 2016.

WONG, S. et al. **Detecting software modularity violations**. Proceedings of the 33rd International Conference on Software Engineering. **Anais**...ACM, 2011Disponível em: <http://dl.acm.org/citation.cfm?id=1985850>. Acesso em: 23 mar. 2016

ZAZWORKA, N. et al. **A case study on effectively identifying technical debt**. ACM Press, 2013Disponível em: <http://dl.acm.org/citation.cfm?doid=2460999.2461005>. Acesso em: 11 maio. 2016

ZAZWORKA, N. et al. Comparing four approaches for technical debt identification. **Software Quality Journal**, v. 22, n. 3, p. 403–426, 2014.

APÊNDICE A – LEVANTAMENTO MANUAL DE DÍVIDA TÉCNICA NO SISTEMA ACESSO A DADOS

Este apêndice representa o documento enviado aos respondentes do levantamento manual de itens de dívida técnica no sistema objeto do estudo, tendo como objetivo auxiliá-los na tarefa de levantamento – realizada no Capítulo 5 – por meio de breve explicação do conceito de dívida técnica e de exemplos fora do contexto do desenvolvimento de software de modo a evitar um resultado enviesado.

O que é dívida técnica?

Dívida técnica é gerada como resultado de ter realizado tarefas no sistema de modo rápido, reduzindo a qualidade do software e aumentando a dificuldade de tarefas futuras como a manutenção e evolução do sistema, ou seja, são os efeitos negativos causados por qualquer artefato imaturo, incompleto ou inadequado durante o desenvolvimento de software, impactando negativamente o desenvolvimento e manutenção no futuro.

Por que se chama dívida técnica?

O termo é uma metáfora e foi denominada desta forma por se assemelhar com a dívida financeira. No contexto financeiro, realizar um empréstimo de capital gera um custo adicional: os juros - o preço do dinheiro. No contexto de desenvolvimento de software, fazer as coisas do modo rápido – análogo a adquirir uma dívida - também gera juros: trabalho extra no desenvolvimento e manutenção futuros por ter que lidar com artefatos incompletos, imaturos ou inadequados. Da mesma forma que ocorre o com a dívida financeira, pode-se optar por não quitar a dívida, mas será preciso pagar juros enquanto a dívida existir (conviver com o problema no sistema), ou pagar a dívida (como uma refatoração no sistema que corrige o problema), livrando-se da dívida e dos juros.

Quais os tipos de dívida técnica?

Há basicamente quatro diferentes tipos considerados neste estudo (porém não há restrições quanto a novos tipos): dívida de documentação do sistema, de *design* do código, de testes e de defeitos (problemas).

Qual o objetivo?

Analisar todo o módulo EJB do sistema Acesso a Dados – *core* do software e parte mais complexa – e as classes referentes ao *web service* BuscaCorporativaCRM no módulo Web – escolhido por ser o mais conhecido por todos os desenvolvedores e também o maior em número de linhas – a partir da *stream* RELEASE do projeto no RTC em busca de itens que você considera que sejam algum tipo de dívida técnica, com base no modelo abaixo e, para cada item encontrado (não há limite de quantidade), preencher as colunas da planilha enviada Levantamento manual de dívida técnica.xls além de seu nome e o tempo total gasto para a identificação de todos os itens.

Qual o problema	Descrição do item encontrado
Tipo	*Design*, documentação, defeito, testes, segurança etc
Localização	Arquivos, classes, métodos ou páginas e documentos onde o item está localizado
Impactos	Impactos possíveis que o item encontrado pode causar no sistema, desenvolvimento e manutenção futura
Custo estimado	Quanto esforço é necessário para remover o item por completo (pagar a dívida) **Valores:** Baixo/Médio/Alto
Quantidade estimada de juros	Esforço extra que terá de ser feito no futuro, caso este item não seja removido. **Valores:** Baixo/Médio/Alto
Probabilidade estimada de juros	Probabilidade de que este item cause o esforço extra no futuro, caso não seja removido. **Valores:** Baixo/Médio/Alto
Intencional	O item foi adicionado no sistema de forma intencional? **Valores:** Sim/Não/Não se sabe

Exemplos de como seriam itens dívida técnica relacionados a um veículo:

Qual o problema	Pneu careca
Tipo	Físico
Localização	Pneu direito traseiro
Impactos	Estourar ou aquaplanar, causando acidente
Custo estimado	**Baixo** (comprar um novo e trocar)
Quantidade estimada de juros	**Alto** (risco de morte)
Probabilidade estimada de juros	**Médio** (o pneu não está tão careca assim)
Intencional	**Sim** (sabia da situação)

Qual o problema	CNH vencida
Tipo	Documentação
Localização	Documento do motorista
Impactos	Multa, apreensão de veículo, acidente (pode ser que o motorista precise de óculos, mas como ele não renovou, não há com saber)
Custo estimado	**Baixo** (renovar o documento)
Quantidade estimada de juros	**Alto** (impacto financeiro e risco de morte)
Probabilidade estimada de juros	**Alto** (há muitos postos policiais rodoviários no trajeto de rotina)
Intencional	**Sim** (não renovou por desleixo)

APÊNDICE B – RESULTADO DO LEVANTAMENTO MANUAL DE DÍVIDA TÉCNICA NO SISTEMA ACESSO A DADOS

Este apêndice apresenta a tabela com o resultado das 23 diferentes *tags* extraídas dos 111 itens resultantes do levantamento manual realizado no Capítulo 5, juntamente com sua frequência de ocorrência e sua descrição em relação ao item de dívida técnica, de acordo com o que foi informado pelos participantes.

Tag	**Quantidade**	**Descrição**
CODIGO_GRANDE	27	Método ou classe com excesso de linhas de código
CODIGO_SEM_DOCUMENTACAO	16	Método ou classe sem documentação associada ou comentários de código suficientes
CODIGO_COMPLEXO	13	Método ou classe com trechos de código complexos de entender, como alta complexidade ciclomática
CODIGO_NAO_UTILIZADO	7	Método, classe ou variável que não é utilizado e não foi removido
FAZER_EXTRACAO_CODIGO	6	Trechos de código que poderiam ser melhor encapsulados
CODIGO_DUPLICADO	5	Trechos de código encontrados duas ou mais vezes dentro da classe
FAZER_EXTRACAO_CONSTANTE	5	Valores fixos no código como *strings* literais que poderiam virar constantes
CODIGO_SEM_TESTE	5	Método ou classe sem código de teste construído
NOME_FORA_PADRAO	7	Nomes de métodos ou variáveis não seguiram padrão Java
BUG_QUERY	3	*Query* de banco de dados com potencial de causar varredura completa de tabelas
CODIGO_SEM_IDENTACAO	3	Trecho de código desorganizado do ponto de vista visual ou não identado
REFLECTION_PERFORMANCE_LENTA	2	Uso desnecessário da API *Reflection* do Java
CAST_DESNECESSARIO	2	*Casting* de objetos desnecessário
ASSINATURA_COMPLEXA_METODO	1	Assinatura de método com excesso de parâmetros
BUG_TAMANHO	1	Lista de objetos que tem o potencial de crescer indefinidamente

COMENTARIO_DEFASADO	1	Comentário de código que não condiz mais com o código atual
COMENTARIO_INUTIL	1	Comentário de código desnecessário
CATCH_EXCEPTION_REPETIDO	1	Tratamentos de diferentes tipos de exceção Java que fazem a mesma coisa
METODO_DEVE_SER_STATIC	1	Método de instância que deveria ser de classe
POM_COMPLEXO	1	Arquivo pom.xml do Maven considerado complexo ou confuso
CODIGO_SEM_SINCRONIZACAO	1	Método ou código que deveria ter controle de sincronização ou concorrência
SEM_CONTROLE_TRANSACAO	1	Método que deveria ter controle de transação
DESEMPENHO_STRING_VS _STRINGBUFFER	1	Trecho de código que poderia utilizar StringBuffer em vez de String para ter melhor desempenho

APÊNDICE C – RESULTADO DO LEVANTAMENTO AUTOMÁTICO DE DÍVIDA TÉCNICA NO SISTEMA ACESSO A DADOS

Este apêndice apresenta a tabela com o as 37 diferentes *tags* extraídas dos 349 itens resultantes do levantamento automático no Capítulo 5, além da frequência de ocorrência e descrição em relação ao item de dívida técnica, de acordo com o que foi informado pelo SonarQube.

Tag	**Quantidade**	**Descrição**
CODIGO_NAO_UTILIZADO	76	Método, classe ou variável que não é utilizado e não foi removido
CODIGO_COMPLEXO	55	Método ou classe com trechos de código complexos de entender, como alta complexidade ciclomática
EXCEPTION_PERDA_RASTREIO	35	Controle de erros ineficaz, perda do rastreio do problema
AUSENCIA_OVERRIDE_ANNOTATION	23	Método que implementa interface Java, mas está sem a anotação @Override
TODO_SEM_IMPLEMENTACAO	23	*Tag* TODO no código sem completar a tarefa
USAR_COLLECTIONS_IS_EMPTY	17	Trecho de código que poderia usar método isEmpty() em vez de verificar o tamanho pelo método size() para melhor legibilidade
FAZER_EXTRACAO_CONSTANTE	14	Valores fixos no código como *strings* literais que poderiam virar constantes
DESEMPENHO_STRINGBUFFER_VS_STRINGBUILDER	11	Trecho de código que poderia utilizar a classe StringBuilder em vez de StringBuffer para ter melhor desempenho
USAR_CONVENCAO_ORDEM_MODIFICADOR	11	Ordem de declaração de modificadores de atributo que não obedece ao padrão da especificação Java
CODIGO_DUPLICADO	8	Trechos de código encontrados duas ou mais vezes dentro da classe
NOME_FORA_PADRAO	8	Nomes de métodos ou variáveis não seguiram padrão Java
VARIAVEL_DESNECESSARIA	7	Variável temporária sem necessidade
CAST_DESNECESSARIO	6	*Casting* de objetos desnecessário

CLASSE_UTIL_SEM_CONSTRUTOR	6	Classes utilitárias não deveriam ter construtor público
NAO_REUTILIZAR_VARIAVEL	6	Variável temporária que foi reutilizada no código em vez de criar outra
BUG_SEMPRE_TRUE_OU_FALSE	5	Instrução condicional que sempre volta true (ou sempre false) independente da situação
JUNTAR_CONDICOES_IF	5	Instruções condicionais separadas que poderiam ser unidas
BUG_NULLPOINTEREXCEPTION _POTENCIAL	4	Trecho de código em que há probabilidade de acontecer NullPointerException
CAMPO_DEVE_SER_FINAL	3	Variável de classe que deveria ter o modificador de acesso *final*
CONVENCAO_DESIGNADOR_ARRAY	3	Declaração de variáveis do tipo *array* que atrapalha legibilidade (como int matrix[][] em vez de int [][] matrix)
BUG_CASTEXCEPTION_POTENCIAL	2	Trecho de código em que há probabilidade de acontecer CastException
CONDICAO_PARADA_LOOP _DEVE_SER_INVARIAVEL	2	Variável que controla parada de loop não deveria mudar de valor
CATCH_EXCEPTION_REPETIDO	2	Tratamentos de diferentes tipos de exceção Java que fazem a mesma coisa
FAZER_EXTRACAO_CODIGO	2	Trechos de código que poderiam ser melhor encapsulados
METODO_DEVE_SER_STATIC	2	Método de instância que deveria ser de classe
VARIAVEL_NAO_DEVE_SER_STATIC	2	Variável de classe que deveria ser de instância por não ser *thread-safe*
ASSINATURA_COMPLEXA_METODO	1	Assinatura de método com excesso de parâmetros
MUDAR_COMPARACAO_BOOLEAN	1	Mudar a comparação booleana de bool == true para !bool para melhor legibilidade
CONVENCAO_ORDEM_METODO	1	Obedecer à convenção de código Java em que construtores devem vir antes de métodos
DECLARAR_VARIAVEIS _MESMA_LINHA	1	Declaração de variáveis na mesma linha que atrapalham legibilidade (como private int a, b, c)
EXCEPTION_DESNECESSARIA	1	Lançamento de exceções desnecessárias como RuntimeException
AUSENCIA_DEPRECATED _ANNOTATION	1	Método anotado com @Deprecated sem a informação no Javadoc.

METODO_COMPARE_TO_ERRADO	1	Método compareTo que deveria examinar somente o sinal do resultado da comparação, não o seu valor
NAO_ USAR_LABEL_LOOP	1	Trecho de código que usa *label* em *loop* Java, diminuindo a legibilidade
DESEMPENHO_KEYSET_VS _ENTRYSET	1	Trecho de código que poderia utilizar EntrySet em vez de KeySet para ter melhor desempenho
DESEMPENHO_ASPAS_SIMPLES _INDEXOF	1	Método indexOf que poderia utilizar char em vez String para melhor desempenho
USAR_CLASSE_PADRAO_JAVA	1	Trecho de código que poderia usar classes da API Java em vez de classes proprietárias