THOBER CORADI DETOFENO

PRIORIZAÇÃO DA DÍVIDA TÉCNICA - PRIORTD

Proposta de tese apresentada ao Programa de Pós-Graduação em Informática da Pontifícia Universidade Católica do Paraná para obtenção do título de Doutor em Informática.

Curitiba
2022

THOBER CORADI DETOFENO

PRIORIZAÇÃO DA DÍVIDA TÉCNICA - PRIORTD

Proposta de tese apresentada ao Programa de Pós-Graduação em Informática da Pontifícia Universidade Católica do Paraná para obtenção do título de Doutor em Informática.

Área de concentração: Ciência da Computação

Orientadora: Profa. Dra. Sheila Reinehr
Coorientadora: Profa. Dra. Andreia Malucelli

Curitiba
2022

AGRADECIMENTOS

Ao Deus de Israel, que por meio de Jesus Cristo pude buscar forças nos momentos de desânimo e luz nos momentos obscuros. *"Confia no Senhor de todo o teu coração, e não te estribes no teu próprio entendimento."* Provérbios 3:5

A minha esposa que soube conduzir por meio da compreensão e amor essa caminhada, estando ao meu lado em todos os momentos. A minha filha que me renovava a esperança com sua alegria.

Aos meus pais Nelson e Inézia Detofeno, meus maiores mestres, cujos ensinamentos guardo com grande carinho e consideração para toda vida.

Aos meus irmãos Thercia e Thiago Detofeno, que de maneira sincera sempre estamos unidos e nos apoiando.

As minhas orientadoras, Profa. Dra. Sheila Reinehr e Profa. Dra. Andreia Malucelli, pela dedicação e orientação durante todo o período do trabalho; mas não apenas por isso, ainda mais, pelos exemplos de pessoas com elevadas qualidades intelectuais e morais.

A família de Ivo Farias de Souza pelo apoio, amizade e carinho.

Ao grupo de Engenharia de Software do PPGIA, muitos amigos e colegas que se propõem a discordar, criticar e apoiar com o objetivo de ajudar no crescimento do outro. O caminho é longo, os desafios são grandes e neste grupo sempre teve alguém disposto a ajudar. Muito obrigado.

RESUMO

A Dívida Técnica é uma metáfora que expressa a imaturidade de artefatos de software e seus impactos nas atividades de manutenção e evolução do produto. Conforme as organizações conseguem identificar e compreender os impactos da Dívida Técnica, surgem desafios na hora de tomar as decisões sobre os investimentos para pagá-la. Embora diversos estudos teóricos e práticos tenham sido desenvolvidos recentemente, ainda existe uma carência de estudos empíricos acerca do gerenciamento e da priorização da Dívida Técnica. O objetivo deste estudo é desenvolver um método para priorizar o pagamento dos Itens de Dívida Técnica do código-fonte mais relevantes para o projeto. Devido às características deste projeto, o estudo utiliza o método de pesquisa-ação, no qual foram executados três ciclos com duração total de 3,5 anos. Neles foram executadas as atividades para coletar, analisar os resultados e propor a abordagem para a priorização da Dívida Técnica de código-fonte. Os resultados principais desta pesquisa são um processo de Gestão da Dívida Técnica, um método de priorização da Dívida Técnica denominado PriorTD e uma guilda de Dívida Técnica. O processo de gestão da Dívida Técnica é a sintetização e a formalização dos resultados obtidos nos três ciclos da pesquisa, fornecendo diretrizes e orientações que podem ser personalizadas ou adaptadas para outras organizações ou contextos. O método PriorTD apresenta os artefatos candidatos para o pagamento Dívida Técnica, possuindo conexão entre o pagamento da Dívida Técnica e os interesses de negócio. A guilda de Dívida Técnica é a parte central deste estudo, que apoia o planejamento e a execução de todas as atividades e ações da Gestão da Dívida Técnica e do PriorTD. A pesquisa apresenta evidências empíricas do uso dos modelos propostos em uma empresa de software, com o envolvimento direto de 82 profissionais e mais de 3 milhões de linhas de código-fonte. O PriorTD forneceu orientações e suporte para compreender e guiar as decisões na priorização da Dívida Técnica.

Palavras-chaves: Dívida Técnica, Gestão da Dívida Técnica, Priorização da Dívida Técnica, Guilda de Dívida Técnica.

ABSTRACT

Technical Debt is a metaphor that expresses the immaturity of software artifacts and their impacts on maintenance activities and product evolution. As organizations become able to identify and understand the effects of the Technical Debt, challenges arise when making investment decisions to pay off it. Although several theoretical and practical studies have been developed recently, there is still a lack of empirical studies on the management and prioritization of the Technical Debt. The objective of this study is to develop a method to prioritize the payment of the Technical Debt Items of source code most relevant to the project. Due to the characteristics of the project, this study uses the action research method. Three cycles with a total duration of 3,5 years were executed. In them, the activities to collect, analyze the results, and propose the approach for the prioritization of the source-coded Technical Debt were carried out. The main findings of this research are a Technical Debt management process, a method for prioritizing the Technical Debt called PriorTD, and Technical Debt Guild. The Technical Debt management process is the synthesis and formalization of the results obtained in the three research cycles, providing guidelines and guidelines that can be customized or adapted for other organizations or contexts. The PriorTD method presents the candidate artifacts for the payment of the Technical Debt, with a connection between the payment and business interests. The Technical Debt Guild is the central part of this study, which supports the planning and execution of all activities and actions of Technical Debt Management and PriorTD. The research shows empirical evidence of the use of the proposed models in a software company, with the direct involvement of 82 professionals and over 3 million lines of code. The PriorTD provided guidance and support to understand and guide the decisions in prioritizing the Technical Debt.

Keywords: Technical Debt, Technical Debt Management, Technical Debt Prioritization, Technical Debt Guild.

SUMÁRIO

RESUMO ..... **III**

ABSTRACT ..... **V**

SUMÁRIO ..... **VI**

LISTA DE FIGURAS ..... **X**

LISTA DE TABELAS ..... **XII**

LISTA DE QUADROS ..... **XIII**

LISTA DE ABREVIATURAS E SIGLAS ..... **XIV**

CAPÍTULO 1 - INTRODUÇÃO **15**

1.1 MOTIVAÇÃO 17
1.2 OBJETIVO 22
1.3 QUESTÃO DE PESQUISA 22
1.4 DELIMITAÇÃO DA PESQUISA 23

CAPÍTULO 2 - REVISÃO DE LITERATURA **25**

2.1 FUNDAMENTAÇÃO TEÓRICA 25
2.1.1 Refatoração 25
2.1.2 Dívida Técnica 27
2.1.3 Gestão da Dívida Técnica 31
2.1.4 Guilda e Comunidade de Prática 36
2.2 TRABALHOS RELACIONADOS 38

CAPÍTULO 3 - ESTRUTURAÇÃO DA PESQUISA **44**

3.1 CARACTERIZAÇÃO DA PESQUISA 44
3.2 PESQUISA-AÇÃO 45
3.3 PROJETO DE PESQUISA 48

CAPÍTULO 4 - CONTEXTO DA PESQUISA **52**

CAPÍTULO 5 - 1º CICLO **57**

5.1 COLETA E ANÁLISE DE DADOS 57
5.1.1 Guilda de Dívida Técnica 58
5.1.2 Ferramentas 60

5.1.3 Análise da prioridade 62
5.1.4 Valor principal da Dívida Técnica 64
5.1.5 Medição da Dívida Técnica 68
5.2 PLANEJAMENTO 73
5.3 IMPLEMENTAÇÃO 74
5.3.1 Ação 1: Elaborar e acompanhar as metas para pagamento da Dívida Técnica 74
5.3.2 Ação 2: Treinamento interno 75
5.3.3 Ação 3: KPIs para o acompanhamento dos resultados 76
5.3.4 Ação 4: Divulgação do projeto pela diretoria 76
5.4 AVALIAÇÃO 77
CAPÍTULO 6 - 2º CICLO 82
6.1 COLETA E ANÁLISE DE DADOS 82
6.1.1 Guilda de Dívida Técnica 82
6.1.2 Ferramentas 83
6.1.3 Identificação da Dívida Técnica 84
6.1.4 Análise da Dívida Técnica 87
6.1.5 Medição da Dívida Técnica 90
6.2 PLANEJAMENTO 92
6.2.1 Priorização 93
6.3 IMPLEMENTAÇÃO 100
6.3.1 Ação 1: Elaborar e acompanhar as metas para pagamento da Dívida Técnica 100
6.3.2 Ação 2: KPIs para o acompanhamento dos resultados 102
6.3.3 Ação 3: Definir e divulgar o padrão de codificação em PHP 102
6.3.4 Ação 4: Definir e divulgar o padrão de documentação do código-fonte em PHP 103
6.3.5 Ação 5: Divulgação do projeto pela diretoria 105
6.3.6 Ação 6: Questionário para avaliação do projeto 105
6.4 AVALIAÇÃO 107
6.4.1 Primeira fase da avaliação 107
6.4.2 Segunda fase da avaliação 110
6.4.3 Terceira fase da avaliação 113

CAPÍTULO 7 - 3º CICLO 114

7.1 COLETA E ANÁLISE DE DADOS 114
7.1.1 Guilda de Dívida Técnica 114
7.1.2 Ferramentas 115
7.1.3 Identificação de Dívida Técnica 115
7.1.4 Medição de Dívida Técnica 116
7.2 PLANEJAMENTO 116
7.3 IMPLEMENTAÇÃO 117
7.3.1 Ação 1: Elaborar e acompanhar as metas para pagamento da Dívida Técnica 117
7.3.2 Ação 2: KPIs para o acompanhamento dos resultados 118
7.3.3 Ação 3: Divulgação do projeto pela diretoria 118
7.3.4 Ação 4: Revisar a documentação dos casos de testes 118
7.3.5 Ação 5: Definir um padrão para o desenvolvimento de Teste Automatizado 119
7.3.6 Ação 6: Monitorar a execução dos Testes Automatizados 120
7.3.7 Ação 7: Identificar a Dívida de *Build* 120
7.4 AVALIAÇÃO 121
7.4.1 Meta da Dívida Técnica identificada no código-fonte 122
7.4.2 Dívida de Teste 122
7.4.3 Dívida de *Build* 123
7.4.4 KPI´s 125

CAPÍTULO 8 - RESULTADOS 129

8.1 GUILDA DE DÍVIDA TÉCNICA 129
8.1.1 Fatores de sucesso 130
8.1.2 Principais desafios 130
8.1.3 Orientações para criar uma guilda de Dívida Técnica 131
8.2 PROCESSO DE GESTÃO DA DÍVIDA TÉCNICA 133
8.2.1 Contexto 136
8.2.2 Avaliação da Dívida Técnica 137
8.2.3 Pagamento da Dívida Técnica 142
8.2.4 Prevenção 144
8.2.5 Revisão 145

8.2.6 Comunicação e Monitoramento 145
8.3 PRIORTD: MÉTODO DE PRIORIZAÇÃO DA DÍVIDA TÉCNICA 147
8.3.1 Características do PriorTD 148
8.3.2 Funil de priorização do PriorTD 151
CAPÍTULO 9 - CONSIDERAÇÕES FINAIS 154
9.1 LIMITAÇÃO DA PESQUISA 154
9.2 TRABALHOS RELACIONADOS 155
9.2.1 Guilda de Dívida Técnica 156
9.2.2 Processo de Gestão de Dívida Técnica 157
9.2.3 PRIORTD – Método de Priorização da Dívida Técnica 159
9.3 CONTRIBUIÇÃO DA PESQUISA 160
9.4 CONCLUSÃO 162
9.5 TRABALHOS FUTUROS 163
REFERÊNCIAS BIBLIOGRÁFICAS 165
APÊNDICE A. ANÁLISE DE PRIORIDADE 176
APÊNDICE B. PRIORIDADE, COMPLEXIDADE, IMPACTO E ESFORÇO DAS REGRAS DE QUALIDADE 183
APÊNDICE C. LISTA DE DÍVIDA TÉCNICA DO 2º CICLO 189
APÊNDICE D. ITENS DE DÍVIDA TÉCNICA POR EQUIPE 196
APÊNDICE E. MODELO DE DADOS DA PRIORIZAÇÃO DO ARQUIVO-FONTE 231
APÊNDICE F. RESULTADO DO QUESTIONÁRIO DE AVALIAÇÃO DO 2º CICLO 235
APÊNDICE G. RESULTADOS DA AÇÃO 1 DO 2º CICLO 238
APÊNDICE H. RESULTADOS POR EQUIPE E PRIORIDADE DO 3º CICLO 241

LISTA DE FIGURAS

Figura 1. Modelo conceitual da Dívida Técnica. Fonte: Rios; Mendonça e Spínola (2018). 29
Figura 2. Quadro geral da gestão da Dívida Técnica: macro atividades e atividades. Adaptado de Rios; Mendonça e Spínola (2018). 31
Figura 3. Diferentes tipos de membros em uma guilda. Adaptado de Smite et al. (2020) 38
Figura 4. Ciclo simplificado da Pesquisa-ação. Adaptado de Dick (2000). 46
Figura 5. Fases da Pesquisa-Ação. Adaptado de Coughlan e Coghlan (2002). 47
Figura 6. Ciclos da pesquisa-ação. Fonte: o Autor. 49
Figura 7. *Timeline* da pesquisa 51
Figura 8. Organograma da empresa. Fonte: o Autor. 53
Figura 9. Quantidade de linhas de código-fonte PHP por equipe. Fonte: o Autor. 54
Figura 10. Resumo do contexto da pesquisa. Fonte: o Autor. 56
Figura 11. SonarQube versão 6.5. Fonte: o Autor. 60
Figura 12. Código da regra “Check use $_REQUEST in folder include”. Fonte: o Autor. 61
Figura 13. Exemplo da visualização do *job* SonarQube sendo executado no GitLab. Fonte: o Autor. 62
Figura 14. Quantidade de Item de Dívida Técnica por Prioridade. Fonte: o Autor. 69
Figura 15. Esforço e quantidade de Item de Dívida Técnica *Grave* por equipe. Fonte: o Autor. 70
Figura 16. Esforço e quantidade de Item de Dívida Técnica *Alta* por equipe. Fonte: o Autor. 70
Figura 17. Quantidade de Item de Dívida Técnica *Média* por equipe. Fonte: o Autor. 71
Figura 18. Quantidade de linhas de código-fonte por Item de Dívida Técnica. Fonte: o Autor. 72
Figura 19. Quantidade de linhas de código fonte por Item de dívida técnica. Fonte: o Autor. 92
Figura 20. Estrutura básica da integração entre as ferramentas no 2º Ciclo. Fonte: o Autor. 94
Figura 21. Funil de priorização dos cenários. Fonte: o Autor. 98

Figura 22. Portal de priorização dos arquivos-fonte. Fonte: o Autor. 100
Figura 23. Priorização da equipe T08. Fonte: o Autor. 102
Figura 24. Exemplo para documentar as funções e métodos. Fonte: o Autor. 104
Figura 25. Regra de documentação personalizada. Fonte: o Autor. 104
Figura 26. Formulário do questionário para avaliar o projeto de Gestão da Dívida Técnica. Fonte: o Autor. 106
Figura 27. Resultado da codificação por característica de qualidade. Fonte: o Autor. 112
Figura 28. Execução dos testes por status e por dia. Fonte: o Autor. 123
Figura 29. Quantidade de passos de testes automatizado e manual. Fonte: o Autor. 123
Figura 30. Percentual de *build*s executados com sucesso. Fonte: o Autor 124
Figura 31. Tempo médio de execução de um *Build* por mês. Fonte: o Autor 125
Figura 32. Análise de tendência do KPI2. Fonte: o Autor 127
Figura 33. Processo de Gestão da Dívida Técnica. Fonte: o Autor. 135
Figura 34. Características do PriorTD. Fonte: o Autor. 149
Figura 35. Funil de priorização do PriorTD. Fonte: o Autor 152

LISTA DE TABELAS

Tabela 1. Tipos de Dívida Técnica. Adaptado de Rios, Mendonça e Spínola (2018). 30
Tabela 2. Critérios de decisão. Adaptado de Ribeiro et al. (2017). 35
Tabela 3. Quantidade de regras por prioridade. Fonte: o Autor. 64
Tabela 4. Exemplo do cálculo do esforço e esforço ajustado. Fonte: o Autor. 67
Tabela 5. Regras *Alta* priorizadas. Fonte: o Autor. 72
Tabela 6. Regras *Alta* priorizadas por equipe. Fonte: o Autor. 72
Tabela 7. Meta para as exceções das regras *Alta.* Fonte: o Autor. 74
Tabela 8. Resultado dos KPIs do primeiro ciclo. Fonte: o Autor. 79
Tabela 9. Total de itens de Dívida Técnica por criticidade e versão. Fonte: o Autor. 80
Tabela 10. Quantidade de regras por Tipo e Dívida Técnica. Fonte: o Autor. 87
Tabela 11. Quantidade de Itens por Dívida Técnica. Fonte: o Autor. 90
Tabela 12. Quantidade de itens de Dívida Técnica por equipe e prioridade. Fonte: o Autor. 91
Tabela 13. Percentual de redução da Dívida Técnica da equipe T08. Fonte: o Autor. 101
Tabela 14. Resultados por prioridade do pagamento dos itens de Dívida Técnica. Fonte: o Autor. 110
Tabela 15. Resultado dos KPIs do 2º Ciclo. Fonte: o Autor. 111
Tabela 16. Quantidade de regras por Prioridade e Tipo de dívida. Fonte: o Autor. 115
Tabela 17. Quantidade de itens de Dívida Técnica por Prioridade. Fonte: o Autor. 116
Tabela 18. Metas por equipe e por Prioridade da Dívida Técnica. Fonte: o Autor. . 118
Tabela 19. Resultados por prioridade do pagamento dos itens de Dívida Técnica. Fonte: o Autor. 122
Tabela 20. Resultado dos KPIs do 3º Ciclo. Fonte: o Autor. 126

LISTA DE QUADROS

Quadro 1. Lista de prioridades. Fonte: o Autor. 63
Quadro 2. Complexidade da alteração. Fonte: o Autor. 65
Quadro 3. Impacto da alteração. Fonte: o Autor. 66
Quadro 4. Lista de prioridade do segundo ciclo. Fonte: o Autor. 88
Quadro 5. Complexidade da alteração do segundo ciclo. Fonte: o Autor. 89
Quadro 6. Impacto da alteração do segundo ciclo. Fonte: o Autor. 89
Quadro 7. Características e atributos utilizados na priorização. Fonte: o Autor. 95

LISTA DE ABREVIATURAS E SIGLAS

DT – Dívida Técnica

IDT – Item de Dívida Técnica

KPI – Key Performance Indicator

PO – Product Owner(s)

RSL – Revisão Sistemática da Literatura

N/A – Não aplicável

PSR – PHP Standards Recommendation

PHP-FIG – PHP Framework Interop Group

SGBD – Sistema de Gerenciamento de Banco de Dados

SM – Scrum Master(s)

SQALE - *Software Quality Assessment based on Lifecycle Expectations*

SQuaRE - *Software product Quality Requirements and Evaluation*

CAPÍTULO 1 - INTRODUÇÃO

A Dívida Técnica (DT) tornou-se uma das metáforas mais importantes para expressar os atalhos no desenvolvimento de software, tomados por conveniência, mas que causam a degradação da qualidade interna do software (IZURIETA et al., 2016; AVGERIOU et al., 2016; AVGERIOU et al., 2020).

A metáfora da Dívida Técnica surgiu em 1992 por Ward Cunningham para descrever, em termos financeiros, o incremento nos custos do desenvolvimento de software que são causados pela baixa qualidade do código-fonte (CUNNINGHAM, 1992). De acordo com Brown et al. (2010) a Dívida Técnica é uma metáfora para caracterizar a diferença entre o estado atual de um sistema de software e seu estado ideal hipotético. Entende-se por estado ideal hipotético aquele estabelecido pelo contexto em que o software está inserido.

A Dívida Técnica apresenta o passivo do artefato entregue, que pode tornar mais caras ou impossibilitar as manutenções ou evoluções futuras do software. Ela consiste em um conjunto de itens que apontam para estes artefatos imaturos e indicam a diferença entre o estado atual e o desejado. (AVGERIOU et al., 2016; STOCHEL; WAWROWSKI; RABIEJ, 2012; STOCHEL; CHOŁDA; WAWROWSKI, 2020). O processo para eliminação da Dívida Técnica de código-fonte, ou seja, pagamento dos itens da Dívida Técnica, é chamado de refatoração.

A Dívida Técnica afeta negativamente a produtividade e a viabilidade do desenvolvimento de software. Em muitos casos, os desenvolvedores são forçados a introduzir mais Dívida Técnica devido às dívidas existentes (BESKER; MARTINI; BOSCH, 2019).

Se não gerenciada, a Dívida Técnica pode resultar em excedentes de custo significativos, problemas graves de qualidade, redução da moral dos desenvolvedores (GHANBARI et al., 2017), limitação da capacidade de adicionar novos recursos (SEAMAN et al., 2012) e até mesmo chegar a um ponto de crise quando uma refatoração enorme e cara ou uma substituição completa do sistema precisa ser realizada (MARTINI; BOSCH; CHAUDRON, 2014).

De acordo com Martini et al. (2015), um objetivo importante na pesquisa e na indústria é identificar qual é o melhor momento para realizar a refatoração e, assim, pagar a Dívida Técnica antes de gerar mais juros. As empresas não podem se dar ao luxo de evitar ou reembolsar toda a Dívida Técnica que é gerada continuamente e pode ser desconhecida (RIOS; MENDONÇA; SPÍNOLA, 2018).

A gestão da Dívida Técnica tem por objetivo possibilitar a tomada de decisão sobre a necessidade de eliminar um item de Dívida Técnica e o momento mais apropriado para fazê-lo (GUO; SPÍNOLA; SEAMAN, 2016).

A gestão da Dívida Técnica inclui atividades de identificação, monitoramento e pagamento dos itens de Dívida Técnica incorridos em um sistema (GRIFFITH et al., 2015). O gerenciamento da Dívida Técnica não inclui apenas atividades de desenvolvimento técnico, mas também organizacionais, como comunicação e tomada de decisão (YLI-HUUMO; MAGLYAS; SMOLANDER, 2016).

A Dívida Técnica pode ser um bom investimento, desde que a equipe do projeto saiba de sua presença e do aumento dos riscos que a Dívida Técnica impõe ao projeto. Se a Dívida Técnica for gerenciada adequadamente, poderá ajudar o projeto a atingir seus objetivos mais cedo ou com menor custo. Assim, a gestão da Dívida Técnica foca na redução do seu impacto negativo, se tornando um fator decisivo para o sucesso dos projetos de software (SEAMAN; GUO, 2011; KRUCHTEN; NORD; OZKAYA, 2012; RIOS; MENDONÇA; SPÍNOLA, 2018).

Para um gerenciamento eficiente da Dívida Técnica é requerida a priorização das atividades de pagamento, uma vez que o pagamento completo da Dívida Técnica não é viável tendo em vista recursos limitados (CHARALAMPIDOU et al., 2017).

O pagamento refere-se às atividades realizadas com o objetivo de apoiar a tomada de decisão sobre o momento mais adequado para eliminar os itens de Dívida Técnica. Neste momento é feita a priorização de qual item de Dívida Técnica deve ser eliminado. Os critérios utilizados na tomada de decisão são a base para gerar a priorização no pagamento dos itens de Dívida Técnica (RIOS; MENDONÇA; SPÍNOLA, 2018).

A priorização da Dívida Técnica é considerada uma das atividades mais importantes no gerenciamento da Dívida Técnica. O processo de priorização da Dívida Técnica é usado para definir a ordem e/ou a programação das tarefas de refatoração com base na prioridade de cada item de Dívida Técnica identificado em relação ao

impacto do item no software (LENARDUZZI et al., 2021). Portanto, a priorização da Dívida Técnica é essencial para alocar melhor os recursos e para determinar quais itens de Dívida Técnica devem ser pagos primeiro e quais itens devem ser adiados para as próximas liberações (ALFAYEZ et al., 2020).

A priorização dos itens de Dívida Técnica orientada pelos negócios pode melhorar o alinhamento e a comunicação entre as partes interessadas técnicas e de negócios. Os critérios, ferramentas e abordagens usados para priorizar a Dívida Técnica carecem de uma perspectiva de negócio (DE ALMEIDA, 2019; STOCHEL; CHOŁDA; WAWROWSKI, 2020). Segundo Lenarduzzi et al. (2021) as pesquisas sobre a priorização da Dívida Técnica são ainda preliminares e não há consenso sobre quais são os fatores importantes e como medi-los.

Nos estudos sobre a priorização da Dívida Técnica predominam o uso de abordagens especificas e de natureza empírica. A maioria das pesquisas se concentra no desenvolvimento de métodos, modelos e métricas para gerar argumentos para a tomada de decisão de um grupo pequeno de Dívida Técnica. Algumas das abordagens trazem uma proposta mais completa, onde pretendem atender um grupo maior de critérios e fatores de decisão, porém muitas delas carecem de comprovação prática ou evidências do uso.

1.1 Motivação

A análise realizada pela Cast Software em 1.400 aplicativos contendo 550 milhões de linhas de código-fonte enviados por 160 organizações, concluiu que a Dívida Técnica de um aplicativo de tamanho médio de 300.000 linhas de código-fonte é de U$ 1.083.000. Isso representa que, na média, a refatoração da Dívida Técnica custa 3,61 dólares por linha de código (CAST, 2019).

O relatório publicado em setembro de 2018, a partir da pesquisa de Evans Data Corp, CIA Factbook e Stripe, estimam que os desenvolvedores gastam aproximadamente 4 horas por semana em “código ruim”. Quase 80% dos participantes acreditam que a presença da Dívida Técnica prejudica na sua moral, e mais da metade dos participantes acredita que a Dívida Técnica atrapalha na sua produtividade. Aproximadamente dois terços dos participantes acreditam que a priorização clara da Dívida Técnica melhoraria a produtividade (STRIPE, 2018).

Os tipos de Dívida Técnica relacionados ao código-fonte (arquitetura, *Design*, código, defeito, teste) tendem a causar efeitos que podem ser sentidos mais

rapidamente pela equipe de desenvolvimento. No entanto, é importante estar ciente que a Dívida Técnica pode ocorrer em diferentes artefatos gerados durante o desenvolvimento de software (ALVES et al., 2016; ALI SHAH et al., 2014).

As dívidas de *Design*, código e arquitetura são os tipos mais citados nos estudos publicados sobre Dívida Técnica. Uma possível explicação para essa concentração de estudos nestes tipos é que existem várias ferramentas de análise de código-fonte que permitem identificar problemas como código complexo, *code smell*, código duplicado, entre outros, que muitas vezes servem como indicadores da presença de Dívida Técnica (RIOS; MENDONÇA; SPÍNOLA, 2018).

Muito do que foi proposto em termos de ferramentas para apoiar a Gestão da Dívida Técnica está focado apenas na dívida relacionada ao código-fonte (RIOS; MENDONÇA; SPÍNOLA, 2018; AMPATZOGLOU et al., 2015; FERNÁNDEZ-SÁNCHEZ et al., 2017; ALVES et al., 2016; LI; AVGERIOU; LIANG, 2015). Já existem diversos trabalhos que investigam a qualidade do software a partir de indicadores coletados do código-fonte. Ferramentas disponíveis para esse fim podem ter sido usadas como ponto de partida, pela comunidade cientifica, para analisar como a Dívida Técnica pode afetar os projetos de software (RIOS; MENDONÇA; SPÍNOLA, 2018).

As ferramentas e os estudos mais focados em identificar a Dívida Técnica no código-fonte ajudaram no descobrimento de uma grande quantidade de Dívida Técnica, principalmente a Dívida Técnica em código-fonte legado. Porém, em muitos casos, as organizações podem não ter tempo suficiente para pagar toda a Dívida Técnica, ou mesmo para reduzi-la aos limites aceitáveis definidos para o projeto. Como a Dívida Técnica não pode ser eliminada completamente, é preciso reduzir sua carga (KRISHNA; BASU, 2012; FALESSI; VOEGELE, 2015; CHARALAMPIDOU et al., 2017).

Para um determinado item de Dívida Técnica, é preciso determinar se é melhor manter esse item por enquanto ou pagá-lo, ou seja, livrar-se dele. Para decidir isso, é preciso determinar qual é o provável benefício, efeitos e custos no pagamento do item de Dívida Técnica. (GUO; SEAMAN, 2011; YLI-HUUMO; MAGLYAS; SMOLANDER, 2016).

Para ter visibilidade e transparência entre o pagamento da Dívida Técnica e os interesses de negócio é preciso criar mecanismos que permitam a rastreabilidade entre os itens de Dívida Técnica e os artefatos do projeto (LI; AVGERIOU; LIANG,

2015). Porém, a mensuração formal ou informal das decisões para contrair a Dívida Técnica é quase inexistente, estas decisões são frequentemente tomadas sem uma análise minuciosa dos riscos envolvidos (CODABUX et al., 2017).

É preciso identificar estratégias que aumentem a visibilidade do negócio e forneça uma comunicação transparente sobre o valor e o retorno do pagamento da Dívida Técnica para que as decisões na Gestão da Dívida Técnica envolvam os interesses dos desenvolvedores e de todos os interessados (RIOS; MENDONÇA, SPÍNOLA, 2018; BEHUTIYE et al., 2017).

A Dívida Técnica permite pensar em qualidade de software traduzindo para o negócio da organização (TOM; AURUM; VIDGEN, 2013). Porém, os critérios de decisão utilizados para o pagamento dos itens de Dívida Técnica estão em cenários e objetivos distintos entre as organizações, então, qual é o momento mais apropriado para eliminar um item de Dívida Técnica e onde está acontecendo este pagamento da Dívida Técnica? (RIOS; MENDONÇA; SPÍNOLA, 2018)

Normalmente, a Dívida Técnica não é paga com a finalidade de eliminá-la, mas com o objetivo de fazer um investimento lucrativo no software. Por exemplo, não se deseja pagar a Dívida Técnica de um sistema legado que é garantido que ficará inalterado no futuro. Em muitos projetos de software, o custo e o benefício da refatoração não são facilmente quantificados ou estimados. Isso dificulta para os desenvolvedores justificarem as necessidades de refatoração para os gestores, porque não está claro como priorizar as refatorações e/ou se a refatoração valerá a pena (ZAZWORKA; SEAMAN; SHULL, 2011). É necessário entender quando é o momento de priorizar a refatoração da Dívida Técnica em relação à implementação de novos recursos ou até mesmo de correção de bugs (LENARDUZZI et al., 2021).

Alguns gerentes, especialmente aqueles com formação técnica, entendem os benefícios da Gestão da Dívida Técnica e a necessidade de refatoração do código-fonte (MARTINI; BOSCH; CHAUDRON, 2015; SHARMA; SURYANARAYANA; SAMARTHYAM, 2015). Um próximo passo importante é fornecer mais orientações para a tomada de decisão sobre se deve ou não pagar determinados itens de Dívida Técnica em um determinado ponto no tempo. Essa decisão depende de vários fatores, como o valor da dívida, a taxa de juros que atualmente se paga sobre a dívida e o impacto potencial que a dívida tem sobre o desenvolvimento futuro (BROWN et al., 2010).

Conjuntos de critérios que podem ser usados para apoiar na tomada de decisão sobre o momento mais apropriado para eliminar os itens de Dívida Técnica foram desenvolvidos em diversos trabalhos (SNIPES et al., 2012; FERNÁNDEZ-SÁNCHEZ et al., 2017; RIOS; MENDONÇA; SPÍNOLA, 2018). Ribeiro et al. (2016) apresentam uma lista de 14 critérios de decisão para facilitar a priorização no pagamento dos itens de Dívida Técnica. O impacto da Dívida Técnica no projeto e o custo-benefício são os critérios mais explorados pelos estudos analisados. Porém, não se sabe os benefícios e as limitações dos critérios de decisão propostos para apoiar a decisão sobre quando uma Dívida Técnica deve ser paga. Por exemplo, não é possível indicar se todos os critérios de decisão propostos podem ser usados para apoiar as decisões relativas a todos os tipos de Dívida Técnica (RIBEIRO et al., 2016).

A decisão de pagar a Dívida Técnica depende do conhecimento relacionado aos juros. Se os juros forem (ou serão) altos, então deve-se priorizar o seu pagamento (MARTINI et al., 2017; MARTINI; BOSCH, 2016; SCHMID, 2013). Porém, faltam evidências empíricas sobre como medir o valor principal e os juros da Dívida Técnica, além disso, faltam ferramentas validadas e amplamente utilizadas para ajudar na priorização da Dívida Técnica (AVGERIU et al., 2020; LENARDUZZI et al., 2021). Segundo Khomyakov et al. (2019) as pesquisas em medição do valor principal são muito ativas, porém, a maioria sugere novas abordagens utilizando critérios diferentes das pesquisas anteriores, não há uma evolução e validação dos modelos, isso demonstra um baixo nível de maturidade e que ainda não está claro qual abordagem deve ser seguidas.

Aspectos associados ao gerenciamento da Dívida Técnica em um processo de desenvolvimento de software ainda constituem uma área pouco explorada. Guo et al. (2016) conduziram um estudo que investigou alguns desses aspectos, como o custo e o impacto do gerenciamento das Dívidas Técnicas em um processo de desenvolvimento de software. Embora relevante, esse trabalho ainda é uma iniciativa isolada.

Por outro lado, já existem algumas estratégias que suportam diferentes atividades de gestão em conjunto nas quais as atividades consideradas são monitoramento, medição e priorização. No entanto, essas mesmas estratégias (com exceção do método SQALE [1]) ainda não possuem uma ferramenta para suportar sua

[1] O método SQALE será explicado no Capítulo 2 e pode ser encontrado em: http://www.sqale.org/wp-content/uploads/2010/08/SQALE-Method-EN-V1-0.pdf

execução (RIOS; MENDONÇA; SPÍNOLA, 2018). Embora ainda existam poucas ferramentas que suportem a priorização no pagamento de itens de Dívida Técnica, existem várias estratégias que podem ser utilizadas para esse fim, conforme relatado no capítulo 2.2 deste trabalho.

As estratégias de Gestão da Dívida Técnica ainda requerem mais estudos e avaliações empíricas. Há poucas evidências da indústria de software sobre como lidar com diferentes tipos de Dívida Técnica, como priorizar os itens de Dívida Técnica para maximizar o benefício e quais fatores devem ser considerados durante sua priorização (FERNÁNDEZ-SÁNCHEZ et al., 2017; ALVES et al., 2016; BESKER; MARTINI; BOSCH, 2018; BEHUTIYE et al., 2017; LI; AVGERIOU; LIANG, 2015).

O processo para tomada de decisão não é padronizado e na maioria dos casos depende do contexto da organização. Pode ser observado que a maioria das pesquisas se concentram no desenvolvimento de métodos, modelos e métricas para alimentar abordagens de tomada de decisão, sem examinar como essas decisões são tomadas. Porém, as decisões são amplamente baseadas nas experiências dos gerentes, ou mesmo na intuição, em vez de dados coletados por meio de medições adequadas (GUO; SPÍNOLA; SEAMAN, 2016; LEPPÄNEN et al., 2015; BECKER et al., 2018).

Um desafio para as equipes de desenvolvimento é quantificar os problemas de manutenção de seus projetos, como forma para justificar o investimento em refatoração da Dívida Técnica (CAI; KAZMAN, 2019; SHARMA; SURYANARAYANA; SAMARTHYAM, 2015). Argumentos convincentes são necessários sobre quando e por que se deve remover a Dívida Técnica. Usar um modelo que foca apenas em um problema é muito restrito para apoiar a tomada de decisão (FERNÁNDEZ-SÁNCHEZ et al., 2017). Os resultados até agora mostram que a priorização de itens de Dívida Técnica orientada pelos negócios pode melhorar o alinhamento e a comunicação entre as partes interessadas técnicas e de negócios (DE ALMEIDA, 2019).

Do ponto de vista teórico, o processo de priorização da Dívida Técnica é relativamente bem representado por vários estudos acadêmicos. No entanto, existe uma escassez atual de pesquisas empíricas com foco em como a priorização é realizada na prática. Existem vários desafios diferentes relacionados à priorização da Dívida Técnica, como a falta de informações quantitativas sobre os itens de Dívida Técnica e a concorrência com a implementação dos requisitos do cliente (BESKER; MARTINI; BOSCH, 2019).

Lenarduzzi et al. (2019) conduziram uma revisão sistemática da literatura sobre priorização da Dívida Técnica e destacam que, com base na maioria das pesquisas realizadas com profissionais, os fatores de clientes e negócios são os mais importantes a serem considerados ao priorizar a Dívida Técnica. A priorização de itens de Dívida Técnica orientada pelo negócio pode melhorar o alinhamento e a comunicação entre as partes interessadas técnicas e de negócios. No que diz respeito à priorização da Dívida Técnica, é comum constatar que os critérios, ferramentas e abordagens usadas carecem de uma perspectiva de negócios (DE ALMEIDA, 2019).

Segundo Lenarduzzi et al. (2021), às características e métricas utilizadas durante a atividade de priorização da Dívida Técnica requerem uma visão holística que envolva vários fatores. A avaliação da Dívida Técnica necessita de várias informações, que podem mudar dependendo do contexto, e na maioria dos casos a Dívida Técnica é priorizada sem seguir uma abordagem padronizada.

Segundo Pina et al. (2021) nenhum método de priorização encontrado na literatura é adaptável ao contexto, trabalha com diversas linguagens de programação, cobre vários tipos de Dívida Técnica e pode ser avaliado na prática.

Os estudos apresentam uma lacuna entre a priorização na Gestão da Dívida Técnica com as necessidades de negócio da organização. Um modelo de priorização deve considerar os objetivos e as necessidades da empresa na decisão do pagamento da Dívida Técnica. Os recursos escassos e as prioridades dos projetos devem fazer parte da análise das partes interessadas antes de pagar a Dívida Técnica. O risco de pagamento de Dívida Técnica desnecessária deve ser mitigado, mantendo o foco em partes do sistema que estão diretamente ligadas aos objetivos de negócio.

1.2 Objetivo

O objetivo desta tese é: **Desenvolver um método para priorizar a refatoração da Dívida Técnica no código-fonte mais relevante para o contexto.**

1.3 Questão de pesquisa

O alcance do objetivo visa responder à questão principal deste trabalho que é: **Como pode ser priorizado o pagamento da Dívida Técnica de código-fonte?**

1.4 Delimitação da pesquisa

A proposta deste estudo é apresentar soluções para a priorização no pagamento da Dívida Técnica identificada no código-fonte. Nem todos os tipos de Dívida Técnica possuem a sua origem no código-fonte, desta maneira, este estudo foca nos tipos de Dívida de *Design* e Dívida de Código.

Dentre as atividades na Gestão da Dívida Técnica, identificadas por Rios, Mendonça e Spínola (2018), a pesquisa atua diretamente na atividade de Priorização no macroprocesso de Monitoramento.

Este estudo apresenta o código-fonte mais relevante para a priorização no pagamento dos itens de Dívida Técnica, ou seja, o código-fonte mais importante no presente momento do projeto. Por exemplo, em uma situação específica, pode ser mais importante para a equipe priorizar as demandas dos clientes, desta maneira os códigos-fonte mais importantes são aqueles que estão ligados diretamente com os defeitos (*bugs*) e melhorias solicitadas pelos clientes. Assim, os critérios relacionados à característica de cliente podem ser aplicados para realizar o gerenciamento dos itens de Dívida Técnica. Em outra situação a empresa deseja dar prioridade para a evolução do produto e focar em novas funcionalidades, neste caso os códigos-fonte mais importantes são aqueles que irão sofrer a maior quantidade de modificações devido à implementação dos novos requisitos. E assim, poderíamos citar várias situações em que npos objetivos do projeto ou da equipe definem qual é o código-fonte mais importante e o momento certo para pagar a Dívida Técnica.

Este estudo propõe orientações para que as decisões no pagamento da Dívida Técnica tenham ligação com os objetivos de negócio. O modelo de análise deve mitigar o risco de pagamento de Dívida Técnica desnecessária e apresentar dados estruturados de maneira que forneça suporte para a tomada de decisão.

O propósito deste estudo é fornecer uma solução para as situações que possuem uma grande quantidade de itens de Dívida Técnica de código-fonte, e nas quais não é possível, ou mesmo recomendável, o pagamento de toda a Dívida Técnica.

Este trabalho está estruturado da seguinte maneira: o capítulo 2 apresenta a revisão da literatura; o capítulo 3 apresenta o método de pesquisa utilizado; o capítulo 4 descreve o contexto da pesquisa; nos capítulos 5, 6 e 7 estão descritos os três ciclos da pesquisa-ação; o capítulo 8 apresenta os resultados referentes à proposta de um

processo de gestão de Dívida Técnica e um método de priorização da Dívida Técnica; a conclusão é apresentada no capítulo 9.

CAPÍTULO 2 - REVISÃO DE LITERATURA

Nessa seção são introduzidos os principais conceitos utilizados como base para a execução desta pesquisa. Inicialmente, na fundamentação teórica, discute-se os conceitos de refatoração, Dívida Técnica, item de Dívida Técnica, Tipos de Dívida Técnica e as principais atividades de Gestão da Dívida Técnica. A seção é finalizada com a apresentação dos estudos relacionados à pesquisa deste trabalho.

2.1 Fundamentação Teórica

2.1.1 Refatoração

O primeiro uso conhecido do termo refatoração na literatura foi em um artigo escrito por William Opdyke e Ralph Johnson em setembro de 1990 (Opdyke e Johnson, 1990). Um ano depois, William Opdyke (1992) também publicou seu Ph.D. sobre a refatoração de programas orientados a objetos (Opdyke, 1992).

Em 1999, Martin Fowler publicou o primeiro livro sobre refatoração que tem como título *Improving the Design of Existing Code*. Este livro popularizou a prática de refatoração de código-fonte, definiu seus fundamentos e teve um grande impacto no mundo do desenvolvimento de software. O autor definiu Refatoração como uma sequência de pequenas mudanças (chamadas operações de refatoração) feitas na estrutura interna do código-fonte sem alterar seu comportamento externo. O objetivo dessas operações de refatoração é melhorar a legibilidade e a capacidade para reutilizar o código-fonte, bem como reduzir sua complexidade e custos de manutenção (Abid et al., 2020).

Atualmente, a refatoração se tornou uma parte crucial da prática de desenvolvimento de software, especialmente com o cenário em constante mudança de requisitos. É um elemento central das metodologias ágeis, e a maioria dos ambientes de desenvolvimento profissionais inclui ferramentas de refatoração. (Abid et al., 2020).

As empresas estão se tornando cada vez mais cientes da importância da refatoração e incentivam seus desenvolvedores a refatorar continuamente seu código para definir uma base limpa para atualizações futuras (Abid et al., 2020).

Pode ser difícil para um desenvolvedor ter justificativa para gastar tempo aprimorando uma parte do código para ter a mesma funcionalidade. No entanto, pode ser visto como um investimento para desenvolvimentos futuros. Especificamente, a refatoração é uma tarefa crucial em software com vida útil mais longa, com vários desenvolvedores que precisam ler e entender os códigos (Abid et al., 2020).

A refatoração pode melhorar a qualidade do software e a produtividade de seus desenvolvedores. O aumento da qualidade do software se deve à diminuição da complexidade do código-fonte causada pela refatoração, conforme demonstrado nos estudos Stroggylos e Spinellis (2007) e Kaur e Kaur (2016). O efeito a longo prazo da refatoração melhora a produtividade dos desenvolvedores, aumentando a compreensibilidade e a capacidade de manutenção dos códigos, especialmente quando um novo desenvolvedor se junta a um projeto existente (Abid et al., 2020).

No estudo de Abid et al. (2020) é proposto um ciclo de vida da refatoração que é composto por seis estágios:

1. Detecção de refatoração: identificar as oportunidades de refatoração é um estágio importante que antecede o processo de refatoração. Os pesquisadores dessa área normalmente propõem técnicas automáticas ou semiautomáticas para identificar as oportunidades de refatoração. Essas técnicas podem ser aplicáveis a diferentes artefatos e devem ser avaliadas empiricamente;
2. Priorização de refatoração: O número de oportunidades de refatoração geralmente excede a quantidade de problemas com os quais o desenvolvedor pode lidar, principalmente quando o esforço disponível para realizar a refatoração é limitado. Além disso, nem todas as oportunidades de refatoração são igualmente relevantes para os objetivos do sistema ou seu funcionamento. Nesta fase, as operações de refatoração são priorizadas usando diferentes critérios de acordo com as necessidades dos desenvolvedores. Por exemplo, maximizar a refatoração de classes com muitos bugs;
3. Recomendação de refatoração: várias ferramentas para recomendar a refatoração foram propostas. Elas se adaptam e sugerem dinamicamente

as refatorações para os desenvolvedores. A saída são sequências de refatorações que os desenvolvedores podem aplicar para melhorar a qualidade dos sistemas;

4. Teste de Refatoração: Após escolher as refatorações a serem aplicadas, os testes precisam ser feitos para garantir a exatidão das transformações dos artefatos e evitar *bugs* futuros. Isso é feito verificando a satisfação das pré-condições e pós-condições das operações de refatoração e a preservação do comportamento do sistema;
5. Documentação de refatoração: depois de aplicar e testar as refatorações, deve-se documentar as refatorações, porque foram aplicadas e as melhorias de qualidade;
6. Previsão: é interessante para os desenvolvedores saber quais locais provavelmente exigirão refatoração em versões futuras de seus produtos de software. Isso ajuda os desenvolvedores para se concentrar nos artefatos relevantes que passarão por mudanças no futuro, a prepará-los para mais melhorias e extensões de funcionalidades e para otimizar os recursos e tempo limitados. A previsão de refatoração pode ser realizada usando o histórico de desenvolvimento.

Os esforços de refatoração estão diretamente associados ao custo de manutenção do sistema de software (Singh; Bindal; Kumar, 2019). Como as refatorações não são visíveis para o usuário e exigem grande esforço, sua necessidade é difícil de justificar para os gestores (Rachow, 2019).

Refatorar é uma tarefa cara e a priorização pode economizar tempo, permitindo considerar apenas (ou primeiro) os problemas mais críticos. Mas, as decisões de refatoração são influenciadas por muitos fatores contextuais que não podem ser decididos considerando apenas o código-fonte (Fontana et al., 2016).

2.1.2 Dívida Técnica

Seaman e Guo (2011), Kruchten; Nord; Ozkaya (2012) e Izurieta et al. (2012) representam a Dívida Técnica como os efeitos de artefatos imaturos na evolução do software que trazem benefícios a curto prazo em termos de aumento de produtividade e redução de custos, porém têm que ser ajustados mais tarde com juros.

O conceito, cujo escopo originalmente se limitava a problemas com código-fonte e artefatos relacionados, foi expandido para considerar diferentes estágios de

um projeto de desenvolvimento de software (ALVES et al., 2016). Avgeriou et al. (2016) definiram a Dívida Técnica como:

> *“A Dívida Técnica é um conjunto de artefatos de Design ou de implementação que são entregues a curto prazo, mas que podem tornar as manutenções futuras mais caras ou impossíveis. A Dívida Técnica apresenta um passivo real ou contingente cujo impacto é limitado às qualidades internas do sistema, principalmente à capacidade de manutenção e evolução.”* (AVGERIOU et al., 2016, p. 112, tradução nossa)

Mcconnell (2007) categoriza a Dívida Técnica em "Não Intencional", incorrida de forma involuntária, e "Intencional", dívida estratégica que geralmente ocorre após uma decisão consciente para realizar um desenvolvimento claramente não ideal. Fowler (2009) utiliza um modelo baseado em quadrantes para categorizar a Dívida Técnica, dividindo em “Prudente” e “Imprudente”, e para cada um destes tipos, distingue entre “Intencional” e “Inconsciente”.

Este trabalho se orienta pelo modelo conceitual de Dívida Técnica ilustrado na Figura 1, que foi inicialmente definido por Avgeriou et al. (2016), descrito por Izurieta et al. (2016) e expandido por Rios; Mendonça e Spínola (2018).

O modelo conceitual, ilustrado na Figura 1, representa as propriedades, artefatos, elementos relacionados aos itens de Dívida Técnica, tipos de Dívida Técnica, estratégias, atividades e ferramentas para apoiar a gestão da Dívida Técnica.

O modelo conceitual, definido por Rios; Mendonça e Spínola (2018), fornecem uma taxonomia com 15 tipos de Dívida Técnica. O estudo apresenta uma definição para os tipos de Dívida Técnica e uma lista de situações em que os itens de Dívida Técnica podem ser encontrados no software. Porém, é observado em estudos posteriores, a publicação do modelo conceitual da Figura 1, o surgimento de novos tipos de Dívida Técnica, tais como: dívida de *Design* de banco de dados (AL-BARAK; BAHSOON, 2016) e dívida herdada (YLI-HUUMO et al., 2016).

Na Figura 1, o modelo conceitual define que um sistema tem um contexto e muitas preocupações. A Dívida Técnica é uma das preocupações de um sistema, que é composta por itens da Dívida Técnica.

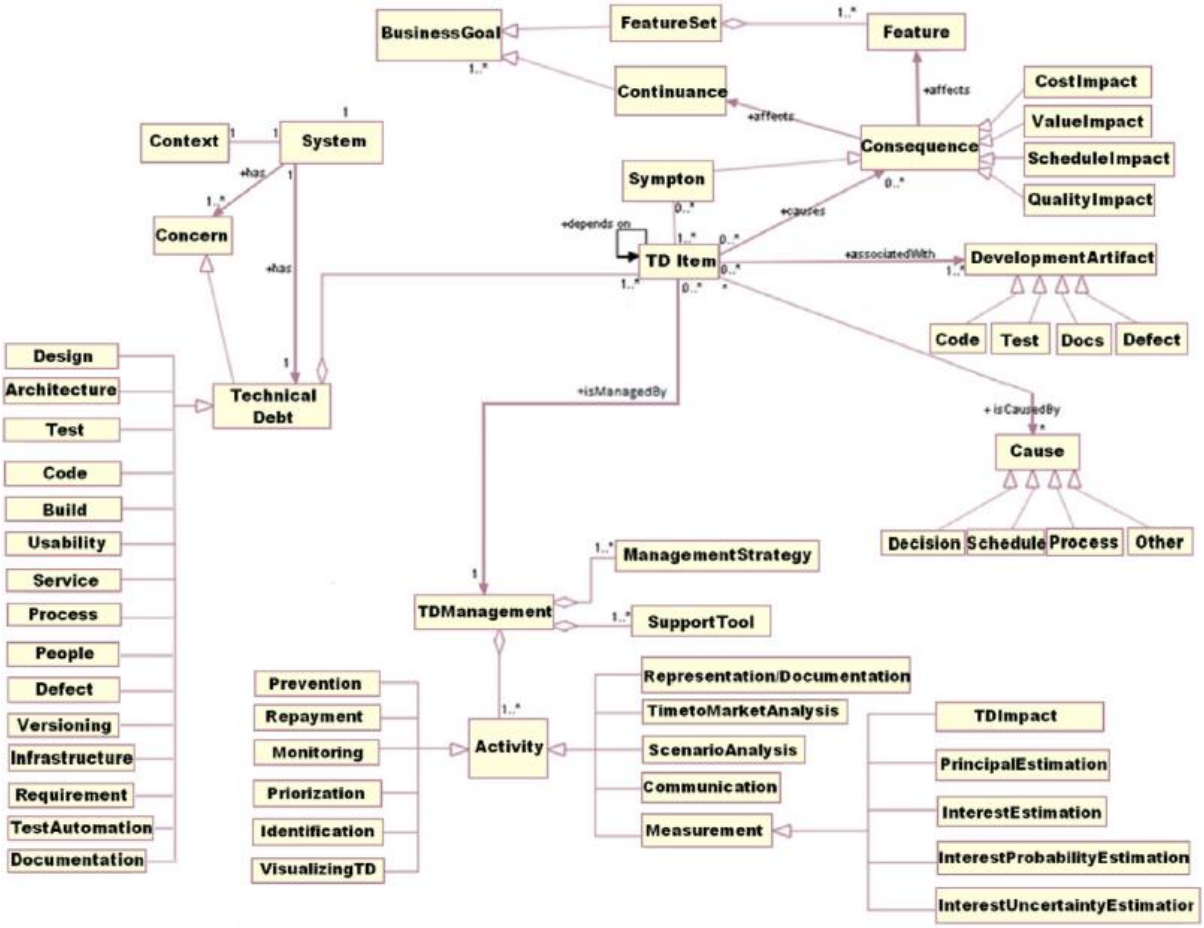


Figura 1. Modelo conceitual da Dívida Técnica. Fonte: Rios; Mendonça e Spínola (2018).

Um Item de Dívida Técnica representa uma instância da Dívida Técnica e possui diversas causas (fatores que levam à ocorrência do Item de Dívida Técnica) e consequências (efeitos produzidos pelas causas que afetam as funcionalidades e/ou objetivos de negócio). Um Item de Dívida Técnica pode estar associado a um ou mais artefatos do processo de desenvolvimento de software (RIOS; MENDONÇA; SPÍNOLA, 2018).

Segundo Rios; Spínola e Mendonça (2018), compreender as causas e efeitos da Dívida Técnica, na perspectiva dos desenvolvedores, é fundamental para orientar as direções da pesquisa. No estudo de Rios et al. (2020) é realizado uma compreensão mais detalhada das causas e efeitos da Dívida Técnica, a partir das evidências do mundo real, este estudo coletou 107 respostas no qual foi possível identificar 78 causas e 66 efeitos. O estudo de Rios et al. (2020) amplia o conhecimento sobre as causas e efeitos da Dívida Técnica, identificando as causas que mais podem contribuir para gerar os itens de Dívida Técnica bem como os efeitos mais comuns que ocorrem

em decorrência da Dívida Técnica gerada. A Tabela 1 apresenta a lista e a descrição dos 17 tipos de Dívida Técnica descritos na literatura.

Tabela 1. Tipos de Dívida Técnica. Adaptado de Rios, Mendonça e Spínola (2018).

Tipos de Dívida	Descrição
Dívida de arquitetura	refere-se aos problemas encontrados na arquitetura do software, que podem afetar os requisitos de arquitetura. Geralmente, a dívida de arquitetura pode ser o resultado de soluções iniciais abaixo do ideal ou soluções que ficam abaixo do ideal à medida que as tecnologias e os padrões são substituídos, comprometendo alguns aspectos internos da qualidade, como a manutenção.
Dívida de automação de teste	refere-se ao trabalho de automação de testes de funcionalidades desenvolvidas que suportam a integração contínua e os ciclos de desenvolvimento mais rápidos.
Dívida de *build*	refere-se a problemas que dificultam a criação da versão compilada e consomem tempo desnecessariamente.
Dívida de código	refere-se a problemas encontrados no código-fonte (código mal escrito que viola as boas práticas ou regras de codificação) que podem afetar negativamente a legibilidade do código, dificultando a sua manutenção.
Dívida de controle de versão	refere-se a problemas no controle de versão do código-fonte.
Dívida de defeito	refere-se a defeitos conhecidos que a equipe de desenvolvimento concorda que devem ser corrigidos, geralmente identificados por atividades de teste ou pelo usuário e relatados em sistemas de rastreamento de *bugs*, porém devido às prioridades concorrentes e aos recursos limitados, sua correção é adiada para um período posterior.
Dívida de *Design*	refere-se à dívida que pode ser descoberta analisando o código-fonte e identificando violações de princípios de *Design* orientado a objetos.
Dívida de *Design* de banco de dados	refere-se a decisões imaturas ou subótimas de *Design* de banco de dados que se manifestam em problemas estruturais ou comportamentais futuros, tornando as mudanças inevitáveis e mais caras de realizar.
Dívida de documentação	refere-se aos problemas encontrados na documentação do projeto de software.
Dívida de infraestrutura	refere-se a problemas de infraestrutura que podem atrasar ou dificultar algumas atividades de desenvolvimento. Tais questões afetam negativamente a capacidade da equipe de produzir um produto de qualidade.
Dívida de pessoa	refere-se a problemas com recursos humanos envolvidos na organização, equipe ou projeto que podem atrasar ou dificultar a execução das atividades de desenvolvimento de software.
Dívida de processo	refere-se a processos ineficientes, por exemplo o que o processo foi projetado para lidar pode não ser mais apropriado.
Dívida de requisito	refere-se às compensações feitas com relação aos requisitos que a equipe de desenvolvimento precisa implementar ou como implementá-los. Em outras palavras, refere-se à distância entre a especificação de requisitos ideal e a implementação real do sistema.
Dívida de serviço	refere-se à seleção e substituição inadequadas de serviços da Web que levam à incompatibilidade entre os recursos de serviço e os requisitos dos aplicativos. Esse tipo de dívida é relevante para sistemas com arquiteturas orientadas a serviços.
Dívida de teste	refere-se a problemas encontrados nas atividades de teste que podem afetar a qualidade dessas atividades.
Dívida de usabilidade	refere-se a decisões inadequadas de usabilidade que precisarão ser ajustadas.
Dívida herdada	refere-se à situação em que uma tecnologia antiga precisa ser reescrita ou substituída por nova tecnologia.

A Dívida Técnica afeta todos os envolvidos no projeto, independentemente da causa. O nível de comunicação referente à Dívida Técnica varia. Os membros da equipe geralmente discutem a Dívida Técnica entre si, mas entendem que há dificuldades ao apresentar evidências de Dívida Técnica para a gerência. Os clientes não têm conhecimento sobre a quantidade e os efeitos da Dívida Técnica no sistema. Mesmo as empresas que conhecem a sua dívida, não têm evidências que estas informações tenham sido disponibilizadas para os clientes (CODABUX et al., 2017).

2.1.3 Gestão da Dívida Técnica

A gestão da Dívida Técnica inclui atividades de identificação, monitoramento e pagamento dos itens de Dívida Técnica incorridos em um sistema (GRIFFITH et al., 2015). Seu principal objetivo é possibilitar a tomada de decisão sobre a necessidade de eliminar um Item de Dívida Técnica e o momento mais apropriado para fazê-lo (GUO; SPÍNOLA; SEAMAN, 2016).

Na visão consolidada da gestão da Dívida Técnica, apresentada na Figura 1, é apresentado o conhecimento sobre a gestão da Dívida Técnica organizado em estratégias de gestão, atividades e ferramentas de apoio. Segundo Rios; Mendonça e Spínola (2018), existem várias estratégias de gestão e ferramentas para apoiar a implementação das atividades de gestão da Dívida Técnica, porém, as ferramentas e estratégias que foram propostas são geralmente focadas em lidar com apenas uma atividade de gestão da Dívida Técnica.

A Figura 2 representa o quadro geral da gestão da Dívida Técnica, dividido em 4 macro atividades e 11 atividades para realizar a gestão da Dívida Técnica.

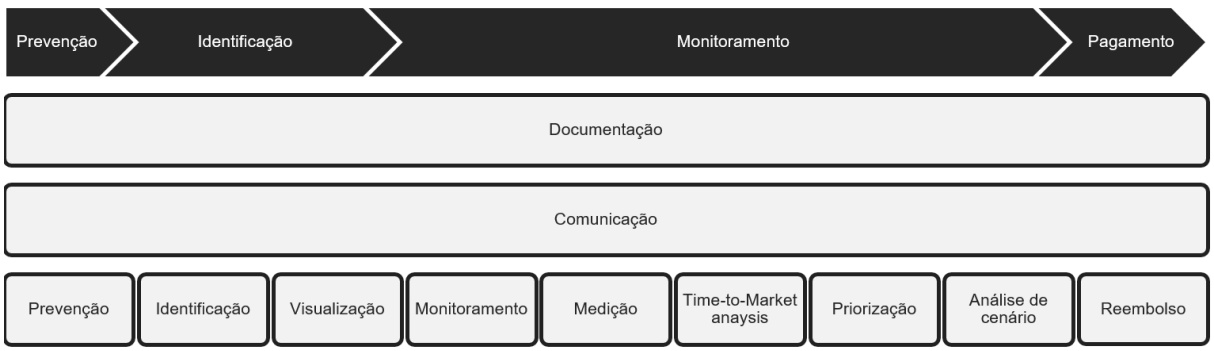


Figura 2. Quadro geral da gestão da Dívida Técnica: macro atividades e atividades. Adaptado de Rios; Mendonça e Spínola (2018).

As macros atividades representam um grupo de atividades para o gerenciamento da Dívida Técnica, tais como:

- Prevenção: refere-se às atividades para prevenir a ocorrência da Dívida Técnica;
- Identificação: refere-se às atividades realizadas para de identificar os itens de Dívida Técnica;
- Monitoramento: refere-se às atividades para o monitoramento dos itens da Dívida Técnica durante a evolução do projeto;
- Pagamento: refere-se às atividades realizadas com o objetivo de apoiar a tomada de decisão sobre o momento mais adequado para eliminar os itens da Dívida Técnica.

Conforme apresentado na Figura 2, as atividades de documentação e comunicação são realizadas durante todo o gerenciamento da Dívida Técnica. Existem poucas estratégias e/ou ferramentas para apoiar as atividades de Prevenção, Visualização da Dívida Técnica, análise do *time-to-market*, Análise de cenário e Reembolso.

Algumas atividades como a identificação (por exemplo detecção da Dívida Técnica por análise estática de código-fonte), medição (quantificação da Dívida Técnica usando estimativas) e pagamento (resolução da Dívida Técnica por técnicas como reengenharia ou refatoração) recebem mais atenção com suporte de ferramentas e abordagens apropriadas (LI; AVGERIOU; LIANG, 2015). Segundo Rios; Mendonça e Spínola (2018), as atividades de Identificação, Monitoramento e Medição possuem diversas iniciativas para apoiar a sua execução.

Segundo Avgeriou et al. (2016), o conceito de Dívida Técnica baseia-se nos conceitos emprestados da economia, de valor principal e juros. No estudo de Rios; Mendonça e Spínola (2018) a estimativa do valor principal, juros e do impacto são atribuídos na atividade de Medição na gestão da Dívida Técnica. Desta maneira, os custos dos itens de Dívida Técnica podem ser entendidos como a junção dos conceitos de valor principal, valor dos juros e probabilidade de juros.

O valor principal da Dívida Técnica refere-se ao custo para eliminar a dívida (ou seja, o esforço necessário para concluir a tarefa). Dependendo do tipo de Dívida, isso pode se traduzir em diferentes tipos de atividades, como a atualização de documentação desatualizada, refatoração de código difícil de manter ou definição de novos casos de teste para melhorar a sua cobertura. O valor dos juros é o custo

potencial em termos de aumento de esforço e diminuição da produtividade que deverá ser pago no futuro como resultado da não conclusão dessas tarefas no presente. A probabilidade de juros é a probabilidade de que a dívida, se não for paga, tornar outros trabalhos mais caros durante um determinado período ou de uma liberação (SEAMAN; GUO, 2011; RIOS; MENDONÇA; SPÍNOLA, 2018).

Embora ainda existam poucas ferramentas que apoiem a priorização do pagamento da Dívida Técnica, existem várias estratégias para esse fim (RIOS; MENDONÇA; SPÍNOLA, 2018).

A priorização é uma atividade realizada para apoiar a tomada decisão de qual Item de Dívida Técnica deve ser eliminado (RIOS; MENDONÇA; SPÍNOLA, 2018). Em muitos casos a priorização é baseada principalmente em palpites, experiência e conhecimento sobre o código-fonte, sem suporte de um método ou valores numéricos exatos (YLI-HUUMO; MAGLYAS; SMOLANDER, 2016). Para este estudo, a atividade de pagamento da Dívida Técnica se refere às ações realizadas para eliminar os itens de Dívida Técnica, levando em consideração as restrições e objetivos da empresa e de cada equipe.

O gerenciamento eficiente da Dívida Técnica requer a priorização das atividades de pagamento, uma vez que o pagamento completo da Dívida Técnica não é viável quando se tem recursos limitados. (CHARALAMPIDOU et al., 2017).

Um modelo para a Gestão de Dívida Técnica deve prever os contextos em que a Dívida Técnica é identificada e avaliada, a fim de que as decisões possam ajudar as organizações a aproveitar as oportunidades e se antecipar em satisfazer as necessidades do mercado (KRUCHTEN; NORD; OZKAYA, 2012). Caso os itens de Dívida Técnica não sejam gerenciados, eles podem causar problemas financeiros e técnicos, aumentando a manutenção do software e os custos de evolução, e podendo levar a um ponto de crise onde todo o futuro do software por ficar comprometido (MARTINI; BOSCH, 2016; NORD et al., 2012; SPÍNOLA et al., 2013; BAVANI, 2012). Não é suficiente que as equipes estejam cientes do que constitui a Dívida Técnica, elas devem estar alinhadas para gerenciar a Dívida Técnica, a fim de agregar valor aos negócios. O simples conhecimento da Dívida Técnica não resulta necessariamente em valor para o software (BAVANI, 2012).

Os critérios para a tomada de decisão são a base para gerar a priorização no pagamento dos itens de Dívida Técnica. O gerenciamento da Dívida Técnica deve se

basear em uma abordagem racional para a tomada de decisão, levando em consideração o desenvolvimento futuro planejado e potencial (SCHMID, 2013).

Nesse sentido, Ribeiro et al. (2016) realizaram um estudo de mapeamento da literatura e identificaram 14 critérios de decisão que podem ser usados para auxiliar na tomada de decisão sobre o pagamento da Dívida Técnica. Este conjunto de critérios foi avaliado quanto à sua pertinência e relevância por meio de uma pesquisa relatada em Ribeiro e Spínola (2016).

A lista de critérios de decisão, apresentada na Tabela 2, foi organizada em quatro categorias:

- Natureza: critérios relacionados às propriedades da Dívida Técnica, como gravidade e tempo em que a dívida foi contraída;
- Cliente: os critérios nessa categoria dizem respeito ao impacto que os itens de Dívida Técnica têm sobre os clientes;
- Esforço: critérios relacionados ao custo da Dívida Técnica, como o impacto da Dívida Técnica no projeto e o esforço necessário para pagar o Item de Dívida Técnica;
- Projeto: critérios relacionados às propriedades do projeto, como vida útil e possibilidade de evolução.

Ribeiro e Spínola (2016) realizaram uma pesquisa para caracterizar os critérios de decisão no que diz respeito à sua pertinência (coluna P da Tabela 2) e relevância (coluna R da Tabela 2) no apoio à Gestão da Dívida Técnica em projetos de desenvolvimento de software. O *ranking* de relevância foi obtido pela importância do critério no momento da tomada de decisão sobre o pagamento de um Item de Dívida Técnica.

Os resultados indicaram que os critérios relacionados ao impacto que uma dívida pode ter no cliente está entre os critérios mais pertinentes e relevantes. Entre os critérios avaliados, cinco deles não foram considerados pertinentes, ou seja, não foram considerados importantes para apoiar a tomada de decisão sobre o pagamento de um Item de Dívida Técnica, portanto, foram excluídos do conjunto final de critérios (itens destacados em cinza na Tabela 2) (RIBEIRO et al., 2017).

De maneira geral, os resultados apontaram que aspectos que afetam o cliente ou se referem ao impacto da dívida em um projeto tendem a ser considerados válidos para apoiar a tomada de decisão sobre o pagamento de itens de Dívida Técnica.

Tabela 2. Critérios de decisão. Adaptado de Ribeiro et al. (2017).

Categoria	Critério de decisão	P	R
Cliente	Impacto no cliente: itens de Dívida Técnica que impactam diretamente no cliente devem ser priorizados (SNIPES et al., 2012; CODABUX; WILLIAMS, 2013)	100,0%	1
Natureza	Gravidade da Dívida Técnica: itens de Dívida Técnica que possuírem nível alto de gravidade devem ser pagos (SNIPES et al., 2012; CODABUX; WILLIAMS, 2013).	97,0%	2
Esforço	Impacto da Dívida Técnica no projeto: itens de Dívida Técnica que oferecem maior impacto para o projeto (exemplo: esforço extra para dar continuidade à evolução do software) devem ser eliminados (ALLMAN, 2012; LIM; TAKSANDE; SEAMAN, 2012; HOLVITIE; LEPPÄNEN, 2013; LETOUZEY; ILKIEWICZ, 2012; MARINESCU, 2012; SNIPES et al., 2012; GUO; SPÍNOLA; SEAMAN, 2016; CODABUX; WILLIAMS, 2013).	87,9%	3
Projeto	Natureza do projeto: em projetos críticos, a Dívida Técnica deve ser eliminada mais rapidamente (ALLMAN, 2012; DAVIS, 2013).	66,7%	4
Cliente	Visibilidade: se o Item de Dívida Técnica puder ser percebido pelo usuário, então ele deve ser pago (LIM; TAKSANDE; SEAMAN, 2012).	78,8%	5
Cliente	Análise de quando a parte refatorada será utilizada: itens de Dívida Técnica que estão em partes muito utilizadas do sistema devem ser pagos (SEAMAN et al., 2012; AL MAMUN; BERGER; HANSSON, 2014).	66,7%	6
Natureza	Localização da Dívida Técnica: se a Dívida Técnica está localizada em um Item do projeto que será objeto de trabalho devido a uma atividade de desenvolvimento ou manutenção, o engenheiro de software deve aproveitar a atividade para pagar a Dívida Técnica (GUO; SPÍNOLA; SEAMAN, 2016);	60,6%	7
Esforço	Esforço para implementar a proposta de correção: itens de Dívida Técnica que exigem menor esforço para serem pagos devem ser eliminados primeiro (SNIPES et al., 2012; CODABUX; WILLIAMS, 2013; HOLVITIE; LEPPÄNEN, 2013)	60,6%	8
Esforço	Custo-benefício: itens de Dívida Técnica com má relação custo/benefício (é mais caro manter a dívida no projeto do que eliminá-la) devem ser pagos (SEAMAN et al., 2012; ALLMAN, 2012; BUSCHMANN, 2011; BROWN et al., 2010; ZAZWORKA; SEAMAN; SHULL, 2011; SNIPES et al., 2012; GUO et al., 2011; CODABUX; WILLIAMS, 2013).	57,6%	9
Projeto	Necessidade de evolução do sistema ou funcionalidade: itens de Dívida Técnica presentes em sistemas estáveis ou que não irão evoluir não devem ser pagos (KRUCHTEN et al., 2013);	30,3%	10
Esforço	Escopo dos testes: itens de Dívida Técnica que possuem menor escopo de testes para avaliar a sua correção devem ser priorizados (SNIPES et al., 2012; CODABUX; WILLIAMS, 2013).	36,4%	11
Projeto	Tempo de vida do sistema: itens de Dívida Técnica presentes em sistemas que serão descontinuados em breve não devem ser pagos (SIEBRA et al., 2012; BUSCHMANN, 2011; ALLMAN, 2012);	45,4%	12
Natureza	Tempo de permanência da Dívida Técnica no projeto: itens de Dívida Técnica que estão há muito tempo no projeto devem ser pagos (CODABUX; WILLIAMS, 2013).	42,4%	13
Natureza	Existência de uma solução alternativa: o pagamento de Dívida Técnica que não possuem uma solução alternativa deve ser realizado (BUSCHMANN, 2011; SNIPES et al., 2012; CODABUX; WILLIAMS, 2013)	24,2%	14

O gerenciamento da Dívida Técnica exige alguns investimentos que são desconhecidos pelos profissionais e são difíceis de serem justificados pela relação

custo/benefício. Consequentemente, sem investir em processos e ferramentas para identificar a Dívida Técnica, é difícil torná-la visível. E é difícil defender a refatoração da Dívida Técnica invisível. Isso representa um ciclo vicioso: as empresas sofrem os efeitos negativos da Dívida Técnica e tentam contê-la, mas, ao mesmo tempo, não encontram motivações suficientes para investir em um processo de gestão mais sistemático (MARTINI; BESKER; BOSCH, 2016). Na prática, um Item de Dívida Técnica deve ser removido quando for lucrativo. Para isso a Dívida Técnica deve ser quantificada, no entanto, é difícil quantificar o impacto da execução ou não de uma decisão de refatoração (FERNÁNDEZ-SÁNCHEZ et al., 2014). A priorização é essencial na gestão da Dívida Técnica, pois requer a decisão de pagar para eliminar os itens da Dívida Técnica ou continuar pagando os juros dos itens da Dívida Técnica.

2.1.4 Guilda e Comunidade de Prática

Na Idade Média, as guildas desempenharam um papel essencial na sustentabilidade econômica. Uma guilda era formada hierarquicamente por mestres, oficiais e aprendizes e contava com especialistas experientes e renomados em seu campo de artesanato. Esses especialistas eram chamados de mestres artesãos. Havia uma troca de conhecimento nessas guildas para tornar o trabalho mais eficiente e produtivo (Wolek, 1999).

Usando esses tipos de fenômenos mais antigos como referência, Leave e Wenger, em 1991, cunharam o termo Comunidade de Prática (CoP) (Lave e Wenger, 1991). No conceito mais atual, abordado por Wenger e Wenger-Trayner (2015), as CoPs são formadas por pessoas que compartilham uma preocupação ou uma paixão por algo e se engajam no aprendizado coletivo em um domínio compartilhado de esforço para fazê-lo, interagindo regularmente.

Para Wenger, McDermott e Snyder (2002), domínio, comunidade e prática são os três elementos essenciais que caracterizam uma CoP. O domínio constrói a comunidade e sua identidade, corresponde à área de interesse que atrai e mantém os membros. Por outro lado, a comunidade é o elemento central, composta por indivíduos e suas interações baseadas na aprendizagem conjunta.

As CoPs se destacam pela gestão de ativos de conhecimento nas organizações, criando valor para os membros e para a organização, como sendo uma ferramenta de competitividade. Pode desenvolver novas habilidades e gerar oportunidades estratégicas por meio de inovações (Wenger et al., 2002). Acredita-se que as CoPs

sejam utilizadas em organizações de diversas naturezas, com outras terminologias, como redes de aprendizagem, grupos temáticos, clubes de tecnologia e guildas.

A literatura profissional sobre como escalar o desenvolvimento ágil de software sugere CoPs como uma possível solução para aprendizado e compartilhamento de conhecimento entre indivíduos com funções semelhantes, como *Testers* ou *Scrum Masters* (SM) (Larman e Vodde, 2010).

A experiência de quatro CoPs da Ericsson mostra que os fatores de sucesso incluem: um bom tópico, um líder apaixonado, um cronograma adequado, autoridade para tomada de decisões, abertura, suporte de ferramentas, um ritmo adequado e participação entre grupos/organizações, quando necessário. As CoPs da Ericsson tinham três papéis principais: apoiar a transformação ágil, fazer parte da implementação do Scrum em larga escala e apoiar a melhoria contínua. As CoPs tornaram-se um mecanismo central por trás do sucesso da implementação ágil em larga escala na organização do caso que ajudou a mitigar alguns dos problemas mais prementes da transformação ágil (Paasivaara e Lassenius, 2014).

Para Smite et al. (2019), implementar comunidades que funcionem bem não é uma tarefa fácil. Experiências da Oracle Corporation, UK National Health Service, Hewlett-Packard, Wipro Technologies, Alcatel e DaimlerChrysler sugerem que o cultivo da cultura do conhecimento requer atenção organizacional, apoio e patrocínio para CoPs.

Inspiradas nas CoPs, as guildas no Spotify são projetadas além das estruturas formais e unem membros com interesses comuns, sejam relacionados ao lazer (ciclismo, fotografia ou consumo de café) ou engenharia (desenvolvimento web, desenvolvimento back-end, engenharia C++, ou *coaching* ágil).

A Figura 3 apresenta os cinco tipos de membros identificados por Smite et al. (2020) nas guildas do Spotify, com base no número de membros cadastrados nos canais de comunicação e engajados nas atividades. Semelhante a Wenger et al. (2002), Smite et al. (2020) identificaram um grupo de membros centrais (patrocinadores e coordenadores), membros ativos e membros periféricos (membros passivos e assinantes). Este último grupo forma a maioria dos membros da comunidade (Smite et al., 2020).

Smite et al. (2020) notaram que os níveis de atividade dos membros individuais mudam ao longo do tempo por vários motivos: o coordenador muda, alguns membros

ativos tornam-se passivos e vice-versa, e aqueles que mudam de especialização tornam-se usuários inativos apenas acompanhando as últimas notícias.

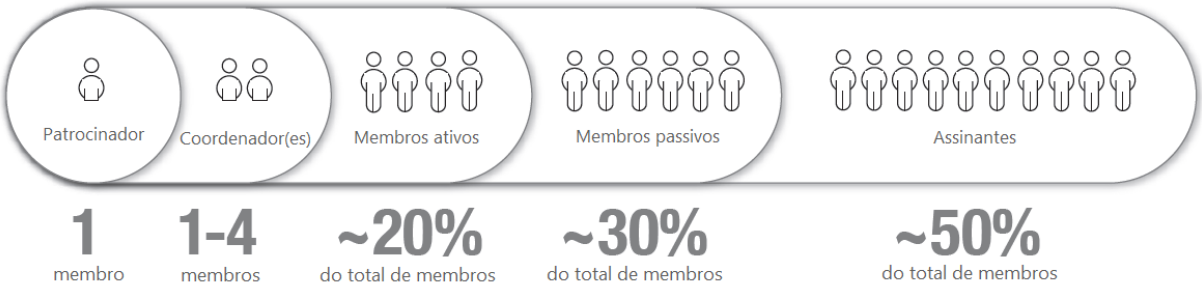


Figura 3. Diferentes tipos de membros em uma guilda. Adaptado de Smite et al. (2020)

Algumas guildas surgem de interesses compartilhados, enquanto outras são estruturadas ou patrocinadas e podem até ter um orçamento específico. A manutenção e geração de valor para a organização de uma guilda é um desafio.

2.2 Trabalhos Relacionados

Nos trabalhos relacionados é apresentada uma lista dos estudos propostos sobre a priorização da Dívida Técnica ou da refatoração de código-fonte independentemente das abordagens utilizadas.

Para a seleção dos trabalhos relacionados foram utilizados como base os estudos secundários de Pina et al. (2021), Lenarduzzi et al. (2021), Abid et al. (2020) e Alfayez et al. (2020), os quais podemos destacar:

- Pina et al. (2021):
    - Título: *Technical Debt Prioritization: Taxonomy, Methods Results, and Practical Characteristics.*
    - Resumo: Este estudo é um mapeamento sistemático de literatura com o objetivo de identificar e classificar os métodos de priorização da Dívida Técnica. O resultado é uma taxonomia de dois níveis com 10 categorias.
    - Artigos publicados até 2020.
    - 51 artigos selecionados.
- Lenarduzzi et al. (2021):
    - Título: *A systematic literature review on Technical Debt prioritization: Strategies, processes, factors, and tools*.

  - Resumo: O estudo conduz uma revisão sistemática de literatura para identificar as abordagens de priorização da Dívida Técnica que foram propostas na academia e na indústria. O estudo investiga as estratégias, processos, fatores e ferramentas utilizadas na priorização da Dívida Técnica.
  - Artigos publicados até dezembro de 2019.
  - 44 artigos selecionados.
- Abid et al. (2020):
  - Título: *30 Years of Software Refactoring Research: A Systematic Literature Review*
  - Resumo: O artigo fornece uma revisão sistemática da literatura, onde coleta e sintetiza 3.183 artigos sobre refatoração. Os resultados são a criação de uma taxonomia para classificar os artigos, as tendências de pesquisas na área de refatoração e as lacunas para pesquisas futuras.
  - Artigos publicados até maio de 2020.
  - 220 artigos foram classificados na etapa de priorização da refatoração para os artefatos de código-fonte.
- Alfayez et al. (2020):
  - Título: *A Systematic Literature Review of Technical Debt Prioritization*.
  - Resumo: Este estudo realiza uma revisão sistemática de literatura para identificar e analisar as abordagens de priorização da Dívida Técnica considerando as restrições de valor, custo e recursos.
  - Artigos publicados de 1992 até 2018.
  - 23 artigos selecionados.

As principais abordagens sobre priorização da Dívida Técnica e da refatoração código-fonte podem ser obtidas a partir destas revisões de literatura. Nelas foi possível identificar os estudos realizados até dezembro de 2020.

Para complementar a identificação do estado atual do conhecimento sobre a priorização no pagamento da Dívida Técnica de código-fonte, para o período de janeiro de 2020 a dezembro de 2021, foram elaboradas pesquisas exploratórias nas bases de dados ACM digital library, IEEE Xplore, Science Direct, Springer Link, Scitepress, Engineering Village e Scopus.

Para as pesquisas exploratórias, foram utilizados os termos relacionados à Dívida Técnica propostos por Li et al. (2015) mais os termos para representar a

refatoração de código-fonte*,* desta maneira a *string* de pesquisa ficou definida com os seguintes termos: *("technical debt") OR ("Design debt") OR ("architect* debt") OR ("test* debt") OR ("implem* debt")OR ("docum* debt") OR ("requirement debt")OR ("code debt") OR ("Infrastructure debt") OR ("versioning debt") OR ("defect debt") OR ("build debt") OR (“refactor) OR (“refactoring”).*

A partir das revisões de literatura de Pina et al. (2021), Lenarduzzi et al. (2021), Abid et al. (2020), Alfayez et al. (2020) e as pesquisas exploratórias, buscou-se conhecer todas as abordagens, ferramentas e tipos de Dívida Técnica utilizadas até o momento na priorização da Dívida Técnica.

Algumas abordagens não foram avaliadas e outras não foram avaliadas especificamente na indústria. Essa falta de avaliações na indústria prejudica a capacidade de prever o desempenho das abordagens (Alfayez et al., 2020). Desta maneira, os trabalhos considerados nesta seção são sobre a priorização na refatoração da Dívida Técnica identificada no código-fonte, que tiveram à comprovação da abordagem na indústria ou que é possível de avaliar na prática.

Zazworka et al. (2011) propõe uma abordagem baseada na análise de custo/benefício para priorizar o pagamento da Dívida Técnica, classificando o valor e os juros da dívida de *Design*, causadas pelas *god classes*. A abordagem sugere calcular a quantidade de correções de defeitos que afetaram estas classes minerando o repositório do software. Basicamente é contada a quantidade de defeitos que foram corrigidos em um determinado período para uma *god classe*, e então são priorizadas as classes que foram alteradas com mais frequência no passado, pois têm maior probabilidade de serem alteradas no futuro. Este estudo foi demonstrado em um produto de software comercial em uma empresa de desenvolvimento.

O estudo de Mkaouer et al. (2014) propõe um modelo baseado no algoritmo genético multiobjetivo NSGA-II, para sugerir a refatoração de código-fonte baseado nos níveis de gravidade dos *code smells* a serem corrigidos e a importância das classes. A importância da classe é definida pelas métricas de quantidade de comentário, relacionamento e métodos implementados. Esta abordagem foi avaliada em seis sistemas *open source*.

O objetivo principal do estudo de Ouni et al. (2015) é priorizar a refatoração do *code smell* mais importante utilizando a meta-heurística proposta por Lam e Li (2010) que é inspirada nas reações químicas. Foram utilizadas quatro métricas para formular o problema de priorização: (1) Prioridade: os desenvolvedores classificam os *code*

smells de acordo com as suas preferências; (2) Gravidade: impacto negativo relativo do *code smell*; (3) Risco: desvio das boas práticas - cada *code smell* detectado corresponde ao desvio de um código-fonte bem projetado; (4) Importância: a importância é dada pela frequência de alterações durante o processo de desenvolvimento, ou seja, as classes com mais alterações são as mais importantes. Esta abordagem foi aplicada nos projetos *open-source*: Xerces-J, JFreeChart, GanttProject, ArtOfIllusion e JHotDraw.

O método SQALE, proposto por Letouzey (2012) é o mais utilizado e o mais referenciado entre os estudos. O objetivo principal do método SQALE é a avaliação e a orientação na melhoria no código fonte, implementando conceitos da Dívida Técnica. A priorização é feita pela classificação da Dívida Técnica no seu modelo de qualidade, propondo índices e indicadores SQALE. O método SQALE ajuda na priorização de qual Dívida Técnica deve ser paga primeiro e ajuda na criação de indicadores de qualidade dos projetos. O método SQALE utiliza a classificação dos requisitos não funcionais que definem o “código correto” como elemento principal na priorização do pagamento da Dívida Técnica.

O estudo de Siverland et al. (2015) investiga 5 variáveis de qualidade de código-fonte para priorizar a refatoração que são comumente utilizadas: Linha de Código, Complexidade, Duplicação de código, Dívida Técnica e *code-churn* (tendência de alteração do código). Todos os dados utilizados no artigo são fornecidos por ferramentas de análise de código-fonte e controle de versão. Durante 20 meses foi examinada uma base de código-fonte com 1.300.000 linhas de código e o resultado apresenta que o *code-churn* é a variável mais forte, seguida pela Linha de Código e Dívida Técnica.

A abordagem de Choudhary e Singh (2016) cria uma lista de priorização de classes orientadas a objetos, onde no topo da lista estão as classes mais relevantes para a refatoração. A proposta combina as informações dos dados históricos, relevância arquitetônica e a severidade das classes para gerar a lista de priorização. A relevância arquitetônica é definida a partir de *code smells* que possuem relação direta com problemas arquitetônicos. A severidade representa a quantidade de *code smells* presente na classe.

Sae-Lim et al. (2017) é um dos primeiros estudos identificados na literatura que busca utilizar o contexto dos desenvolvedores para priorizar a refatoração de *code smells*. No estudo, os dados extraídos de um software de gerenciamento de incidentes

são utilizados para definir o contexto dos desenvolvedores. Desta maneira, os dados de entrada do processo de priorização são: uma lista de informações obtidas do software de gerenciamento de incidentes; o código-fonte do projeto e informações do repositório de código-fonte para análise de impacto. Esta abordagem foi avaliada nos projetos *open-source*: ArgoUML, Jabref, jEdit e muCommander.

A abordagem de Rani e Chhabra (2017) propõem priorizar as *smally classes* com base no seu nível de severidade. A severidade da classe é estimada a partir da medição do acoplamento, do histórico de alteração do software e a quantidade de *code smells*. Esta abordagem foi avaliada em um software composto por 49 classes.

O objetivo do estudo de caso de Charalampidou et al. (2017) é priorizar o pagamento da Dívida Técnica que é mais propensa a produzir juros da dívida. Para calcular a previsão dos juros foram utilizados os dados da quantidade de *code smells* e a frequência de alteração nas versões anteriores (ou seja, histórico do software). Este estudo de caso foi realizado em 47.751 métodos extraídos dos sistemas *open-sources* Spring Framework e AndEngine.

A abordagem de Alfayez e Boehm (2019) utiliza o Algoritmo Evolucionário Multi-objetivo NSGA-II para indicar quais itens de Dívida Técnica devem ser pagos considerando as restrições de valor e custo. Na avaliação da abordagem, foi definido o valor de cada Item de Dívida Técnica sendo a multiplicação entre a severidade e um peso que representa a importância do arquivo no projeto, e o custo foi representado pelo tempo necessário para corrigir cada Item de Dívida Técnica. A abordagem foi avaliada em 40 projetos *open-sources* e em uma empresa de desenvolvimento de software que possui um projeto com Java com 45.980 linhas de código-fonte.

Stochel, Chołda e Wawrowski (2020) propõem a abordagem *Continuous Debt Valuation Approach* (CoDVA), que incluiu aspectos de negócios na tomada de decisão para a priorização da Dívida Técnica. A partir da previsão de vendas de uma determinada funcionalidade do software é calculado o retorno sobre o investimento, para isso é considerado no custo do investimento os valores para o pagamento dos itens de Dívida Técnica. Desta maneira a priorização da Dívida Técnica é alinhada com uma perspectiva de negócio, mais especificamente com processo de vendas.

Almeida et al. (2021) propõem uma priorização da Dívida Técnica considerando a perspectiva dos processos de negócio do sistema. A abordagem é construída em torno do *"priority canvas",* que é utilizado para ajudar os stakeholders na visualização dos ativos de TI e as fontes de valor (processos de negócios) e seus relacionamentos.

A estrutura da abordagem utiliza uma lista de Dívida Técnica, itens de configuração e os ativos de TI para identificar os aspectos de negócios afetados. As fontes de valor e o impacto nos negócios são os elementos que representam o valor nos negócios. No final a abordagem orienta os itens de Dívida Técnica priorizados e o impacto da Dívida Técnica nos negócios.

Os trabalhos de Stochel, Chołda e Wawrowski (2020) e Almeida et al. (2021) buscam incorporar a perspectiva de negócio na tomada de decisão para o pagamento da Dívida Técnica identificada no código-fonte. Observa-se que os estudos tratam a perspectiva de negócio de maneiras diferentes, os quais ficam limitados a cenários de negócio mais específicos. Assim, Stochel, Chołda e Wawrowski (2020) considera a perspectiva de negócio sendo a previsão de vendas e Almeida et al. (2021) trata a perspectiva de negócio como sendo os processos de negócio do produto.

A análise dos estudos identificados nas revisões de literatura e na pesquisa exploratória revelou uma escassez de abordagens que foram avaliadas na indústria. Essa falta de avaliações do setor diminui a credibilidade das abordagens, conforme destaca Alfayez et al. (2020). Da mesma maneira, Lenarduzzi et al. (2021) destacam que não foi identificado abordagens de priorização da DT que abordam explicitamente a implementação de novos recursos e o esforço necessário na refatoração da Dívida Técnica. Pina et al. (2021) evidenciam que as propostas de priorização não são independentes da linguagem de programação.

As revisões de literatura de Pina et al. (2021), Lenarduzzi et al. (2021) e Alfayez et al. (2020) concordam que a priorização da Dívida Técnica deve ser adaptável ao contexto.

Não foi encontrada uma convergência entre os métodos e abordagens utilizados nos estudos pesquisados sobre priorização da Dívida Técnica, porém os estudos estão buscando, cada vez mais, alinhar os objetivos de negócio com a necessidade de evolução contínua do software.

A evolução na priorização da Dívida Técnica deve buscar atender às necessidades do contexto da organização, examinar quais são as motivações que levam à tomada de decisão para elaborar abordagens que se concentrem em trazer resultados de negócio alinhados com a estratégia do produto de software.

CAPÍTULO 3 - ESTRUTURAÇÃO DA PESQUISA

Neste capítulo é detalhado o percurso metodológico da pesquisa.

3.1 Caracterização da pesquisa

As características e classificações apresentadas a seguir basearam-se nos estudos de Gil (2002). Esta pesquisa, com relação à natureza, é classificada como aplicada e com relação ao envolvimento do pesquisador é classificada como pesquisa-ação.

Este trabalho é caracterizado como uma pesquisa aplicada, pois utiliza teorias, conhecimentos, métodos e técnicas conhecidos da comunidade científica para um propósito específico. Neste trabalho os novos conhecimentos adquiridos são sobre a priorização no pagamento da Dívida Técnica identificada no código-fonte. Isto gera conhecimento prático dirigido à solução de um problema específico (GIL, 2002).

A pesquisa explorou um ambiente complexo de desenvolvimento de software, onde foram desenvolvidas e aplicadas etapas para a Gestão da Dívida Técnica. O ambiente de estudo possui um processo de desenvolvimento de software bem estruturado e com maturidade, porém não foi evidenciada nenhuma iniciativa de Gestão da Dívida Técnica até o início deste trabalho, constituindo este o problema alvo da pesquisa.

A pesquisa-ação procura unir a pesquisa à ação ou prática, isto é, desenvolver o conhecimento e a compreensão como parte da prática. Optou-se pela pesquisa-ação devido ao caráter colaborativo do trabalho e à possibilidade de o pesquisador atuar de forma concorrente à sua execução (COUGHLAN; COGHLAN, 2002). Os seguintes fatores levaram à escolha do método de pesquisa-ação para a execução deste trabalho:

- a iniciativa da pesquisa partiu da necessidade real de uma organização (problema no gerenciamento da dívida técnica de código-fonte);
- os profissionais da empresa participaram junto com o pesquisador na identificação dos problemas e na busca pelas soluções;
- os objetivos da pesquisa foram definidos junto com os interessados;

- os profissionais afetados pela pesquisa participaram e colaboraram nas definições e nas tomadas de decisões;
- as informações foram disponibilizadas e compartilhadas com os grupos interessados, conforme o avanço da pesquisa;
- as ações decorrentes dos resultados coletados foram negociadas com os grupos interessados;
- a flexibilidade no planejamento e execução da pesquisa, não seguiu uma série de atividades pré-definidas, mas sim emergiu de sua execução;
- a partir dos conhecimentos e resultados obtidos, a empresa alterou o seu processo;
- a implementação da pesquisa foi cíclica, onde as fases finais foram usadas para aprimorar os resultados das fases anteriores.

3.2 Pesquisa-ação

Fazer pesquisa-ação significa planejar, observar, agir e refletir de maneira mais consciente, mais sistemática e mais rigorosa, o que se faz na experiência diária (KEMMIS; MCTAGGART, 1988).

A pesquisa-ação tem por finalidade possibilitar aos sujeitos da pesquisa, participantes e pesquisadores, os meios para conseguir responder aos problemas que vivenciam com maior eficiência e com base em uma ação transformadora. Ela facilita a busca de soluções de problemas por parte dos participantes, aspecto em que a pesquisa convencional tem pouco alcançado (THIOLLENT, 2011).

Dick (2000) afirma que a caracterização da pesquisa-ação varia de um autor para outro, no entanto existe um conjunto de pontos comuns a todos, tais como:

- Atuar em uma situação existente com duplo objetivo e aperfeiçoar e ampliar o conhecimento sobre o assunto;
- Possuir uma natureza cíclica: executar uma série de etapas repetidamente. O ciclo varia de acordo com o autor, mas, pelo menos, deve incluir as etapas de Planejamento, Ação e Reflexão, conforme exemplificado na Figura 4 (DICK, 2000);
- Possuir uma natureza reflexiva, ou seja, uma reflexão crítica sobre o próprio processo de pesquisa, bem como sobre os resultados obtidos;

- É predominantemente qualitativa, embora quantificações sejam possíveis em algumas situações.

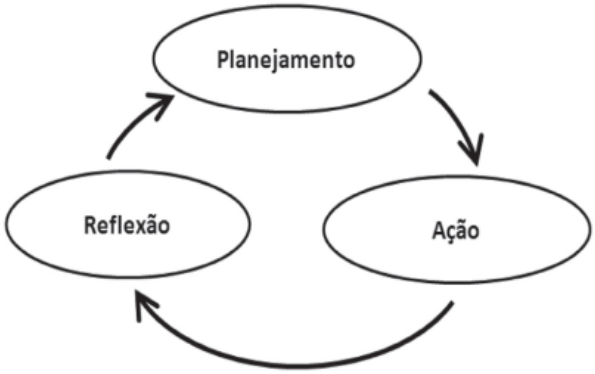


Figura 4. Ciclo simplificado da Pesquisa-ação. Adaptado de Dick (2000).

Estas características devem ser consideradas desde o momento da concepção da pesquisa, que conforme Coughlan e Coghlan (2002), compreende três fases principais: (i) Fase Preliminar; (ii) Fase de Condução; e (iii) Fase de Monitoramento. Os mesmos autores apresentam as fases para o desenvolvimento de um projeto de pesquisa-ação constituído por oito etapas, como pode ser visto na Figura 5.

A primeira fase da pesquisa-ação é a Fase Preliminar que compreende o entendimento sobre o contexto em que a pesquisa será realizada, bem como o propósito da condução do trabalho. Essa fase envolve ainda o estabelecimento de justificativas para a ação requerida (razões pelas quais as ações devem ser conduzidas) e justificativas para a pesquisa em si (razões pelas quais ela deve ser conduzida, que questões a serem endereçadas e qual será a contribuição gerada) (DRESCH; LACERDA; MIGUEL, 2015).

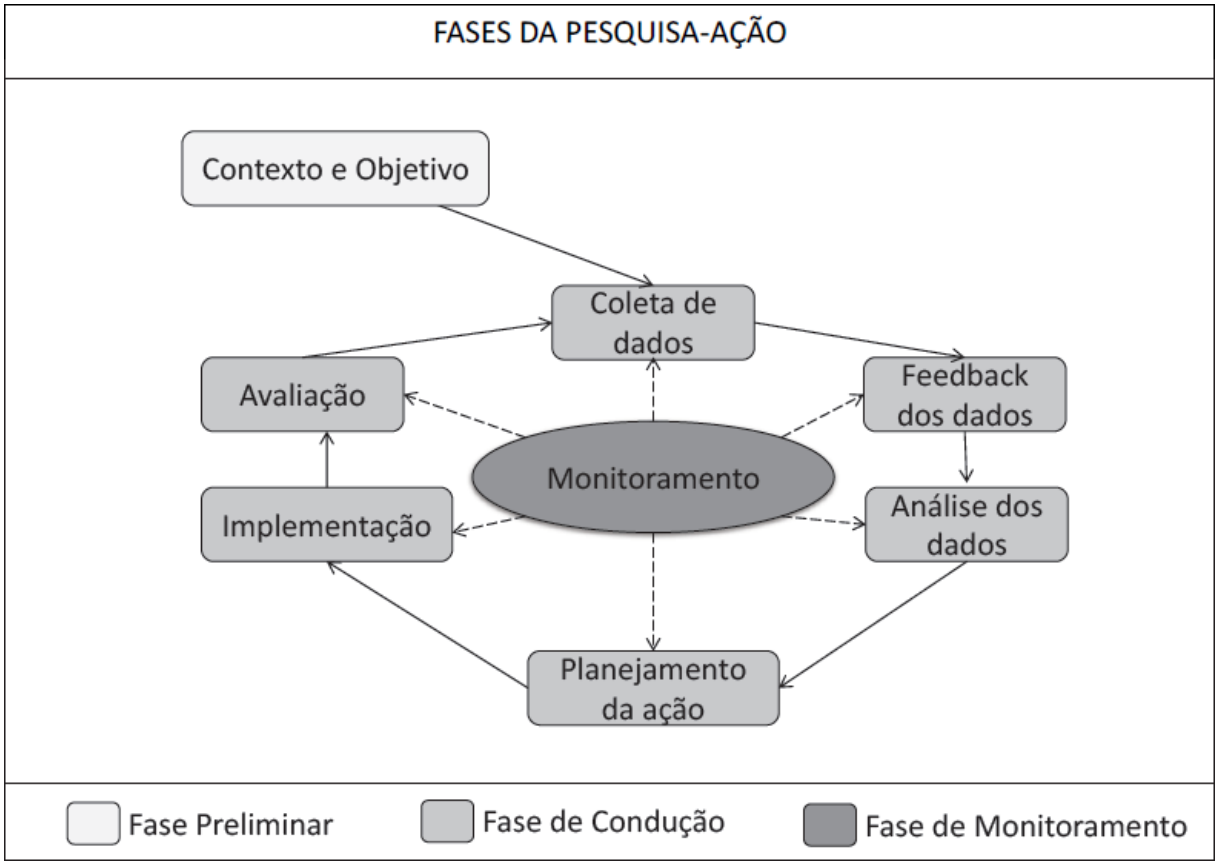


Figura 5. Fases da Pesquisa-Ação. Adaptado de Coughlan e Coghlan (2002).

A Fase de Condução pode ser representada por seis passos e envolve:

- Coleta de Dados: pode ocorrer de diversas maneiras com observações, entrevistas, questionários ou relatórios, coletando dados qualitativos ou quantitativos;
- *Feedback* dos Dados: os dados coletados são submetidos para a organização analisar por meio de relatórios ou reuniões de *feedback*;
- Análise dos Dados: a análise dos dados é realizada de forma conjunta com todos os envolvidos com a pesquisa. Essa análise busca que cada parte contribua com uma visão crítica, seja acerca dos dados coletados, questões internas da empresa, condução da pesquisa ou interação com os conhecimentos do pesquisador;
- Planejamento da Ação: também deve ser realizado em conjunto entre os envolvidos com a pesquisa. Nesta atividade é estabelecido o que vai ser feito e o prazo;

- Implementação: as ações são implementadas, com objetivo de promover as mudanças planejadas em colaboração com os envolvidos e responsáveis da organização;
- Avaliação: segundo Coughlan e Coghlan (2002), a avaliação é uma reflexão dos resultados esperados ou não, decorrentes da implementação da ação, visando melhorias para o ciclo seguinte.

A terceira fase (Fase de Monitoramento), não é exatamente uma fase, mas, sim, uma atividade contínua, que compreende a verificação de cada um dos seis passos anteriores, no sentido de identificar qual é o aprendizado gerado na condução da pesquisa-ação. Esse monitoramento deve estar presente de diferentes maneiras, conforme cada passo do ciclo de condução. Do lado organizacional, pode haver o estabelecimento de um grupo diretivo durante a condução da pesquisa-ação, nesse caso com maior interesse nos resultados práticos do trabalho (COUGHLAN; COGHLAN, 2002). Ainda, segundo os autores, o pesquisador deve, por outro lado, estar interessado não somente na operação do projeto, mas também no monitoramento do processo de aprendizagem, que levará, em última instância, à contribuição teórica desse tipo de desenvolvimento empírico.

3.3 Projeto de pesquisa

Optou-se, nesta pesquisa, por estruturá-la conforme ilustra a Figura 6. Como se pode observar, iniciou-se com uma etapa de compreensão do Contexto e prosseguiu-se com três ciclos da Fase de Condução, compostos pelas etapas de: análise de dados, planejamento, implementação e avaliação.

Cada ciclo foi conduzido com um objetivo, conforme apresentado a seguir:

- 1º Ciclo: o objetivo do primeiro ciclo desta pesquisa foi definir e implementar um modelo para a Gestão da Dívida Técnica no desenvolvimento em código-fonte PHP, que inclui desde a identificação, o monitoramento e o pagamento da Dívida Técnica. Com isso, deveria ser definido um processo para a Gestão da Dívida Técnica que possibilitasse orientar a tomada de decisão no pagamento da Dívida Técnica a partir de um modelo de priorização preliminar. O processo de Gestão da Dívida Técnica de código-fonte deveria ser capaz de identificar eventos em potencial, capazes de afetar a manutenção e a evolução do software e permitir o gerenciamento

da Dívida Técnica de forma compatível com a quantidade de dívida que a organização estivesse disposta a aceitar;

- 2º Ciclo: o segundo ciclo foi uma revisão do ciclo anterior, onde além de implementar e evoluir na Gestão da Dívida Técnica de código-fonte, foram definidos os procedimentos para priorizar o código-fonte conforme a importância e os objetivos de cada projeto ou equipe. Neste ciclo, os critérios de priorização foram revisados e apresentou-se um método para o pagamento da Dívida Técnica do código-fonte mais relevante para o projeto ou equipe, o qual foi nomeado como PriorTD;
- 3º Ciclo: o objetivo do terceiro ciclo desta pesquisa foi evoluir o processo de Gestão da Dívida Técnica considerando outros tipos de Dívida Técnica, tais como Dívida de teste, Dívida de automação de teste e Dívida de *build*.

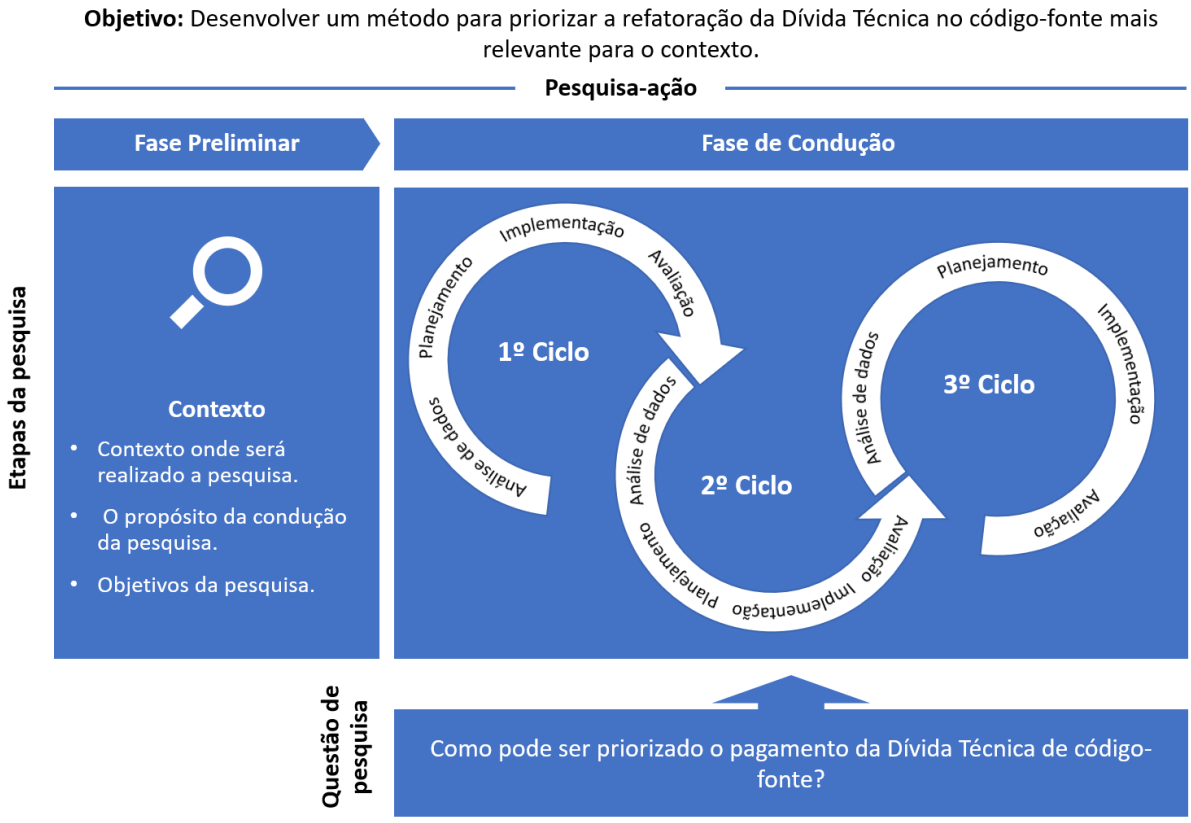


Figura 6. Ciclos da pesquisa-ação. Fonte: o Autor.

Ao final de cada ciclo, os resultados foram utilizados para revisar e evoluir o processo de Gestão da Dívida Técnica e o método para a priorização da Dívida Técnica de código-fonte. Ao final dos três ciclos foi apresentada uma proposta para a priorização da Dívida Técnica de código-fonte (PriorTD).

Cada um dos ciclos seguiu as etapas sugeridas por Coughlan e Coghlan (2002), mas adaptadas para serem executadas em 4 etapas:

- Coleta e análise de dados: esta etapa compreende as etapas de Coleta de dados, *Feedback* dos dados e Análise dos dados, sugeridas pela pesquisa-ação. A coleta e análise das informações foram realizadas por meio de reuniões com grupos de profissionais responsáveis de diversas áreas da empresa e análise das ferramentas que a empresa possui ou podem ser utilizadas neste projeto;
- Planejamento: nesta etapa de planejamento, as ações e prazos foram apresentados e discutidos com os envolvidos no projeto. As ações foram aprovadas pelos interessados antes de seguir para a próxima etapa;
- Implementação: na fase de implementação, as ações planejadas foram executadas. Neste momento foram realizados a priorização e o pagamento para mudar a situação atual da Dívida Técnica. Alguma das ações de pagamento da Dívida Técnica tiveram um período de execução de até um ano;
- Avaliação: a etapa de avaliação do ciclo é a chave para refletir sobre os resultados e construir o conhecimento, visando as melhorias para o ciclo seguinte. Nesta fase foram avaliados os resultados das ações e do monitoramento periódico da Gestão da Dívida Técnica.

A duração de cada ciclo está atrelada ao ciclo anual de gestão da empresa que prevê o planejamento de metas e ações para o ano seguinte. Os ciclos preveem períodos de planejamento e execução das ações que impactam no processo de desenvolvimento de software ou nos objetivos das equipes. A Figura 7 apresenta os períodos em que foram executadas cada uma das etapas nos três ciclos desta pesquisa.

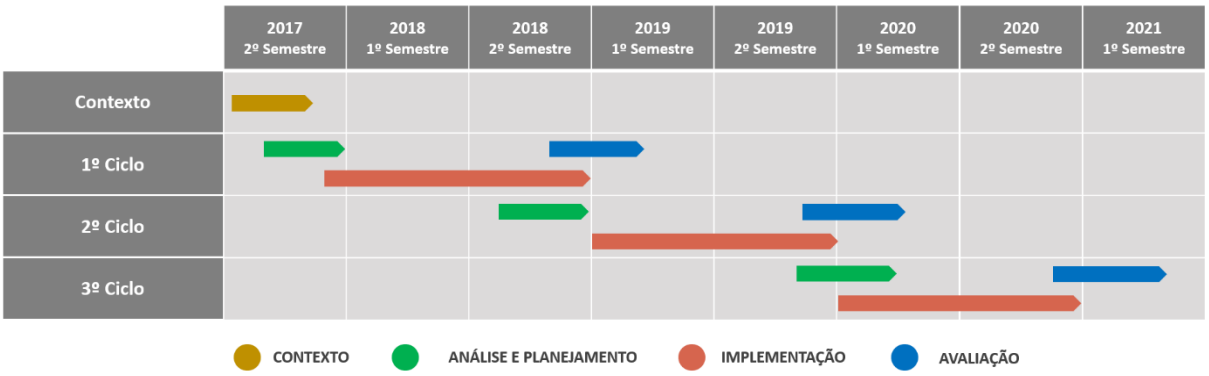


Figura 7. *Timeline* da pesquisa

CAPÍTULO 4 - CONTEXTO DA PESQUISA

O objetivo desta etapa é apresentar o ambiente, as partes interessadas, os objetivos da empresa e os propósitos da condução da pesquisa. O contexto da pesquisa teve pequenas modificações durante os três ciclos da pesquisa, porém os objetivos e propósitos se mantiveram sem alterações. As modificações do contexto se concentraram nas mudanças estruturais das equipes que foram estudadas.

Esta pesquisa foi desenvolvida em parceria com uma empresa de desenvolvimento de software brasileira, localizada em Joinville-SC, na área de Desenvolvimento de Software.

A empresa foi fundada em 1995 e possui mais de 2 mil clientes e 300 mil usuários ao redor do mundo, com soluções em software para gestão e conformidade empresarial. Os produtos são utilizados por empresas dos mais variados portes e ramos de atuação, incluindo manufatura, governo, farmacêutico, hospitais e laboratórios, serviços financeiros, alta tecnologia e TI, educação, energia e utilidade pública, logística, varejo e serviços.

A empresa possui 40 módulos que, juntos, compõem um produto, atendendo a serviços para automação e aprimoramento dos processos de negócio, conformidade regulamentar e governança corporativa. Tecnicamente o produto é totalmente disponível na WEB, com documentação e localização para mais de 10 idiomas e compatível com 3 sistemas gerenciadores de bancos de dados.

A estrutura organizacional da empresa é dividida em 7 áreas: Desenvolvimento de Software, Desenvolvimento de Negócios, Tecnologia, Administrativo Financeiro, Comercial, Serviços e Escritório de Gestão. Cada uma das áreas possui suas atribuições e um diretor responsável. A Figura 8 apresenta o organograma da empresa dividido em áreas e subáreas, cada uma das subáreas possui um responsável que está subordinado ao diretor da área.

A área de Desenvolvimento de Software é dividida nas subáreas de Manutenção Evolutiva, Desenvolvimento sob demanda, Inspeção e Documentação.

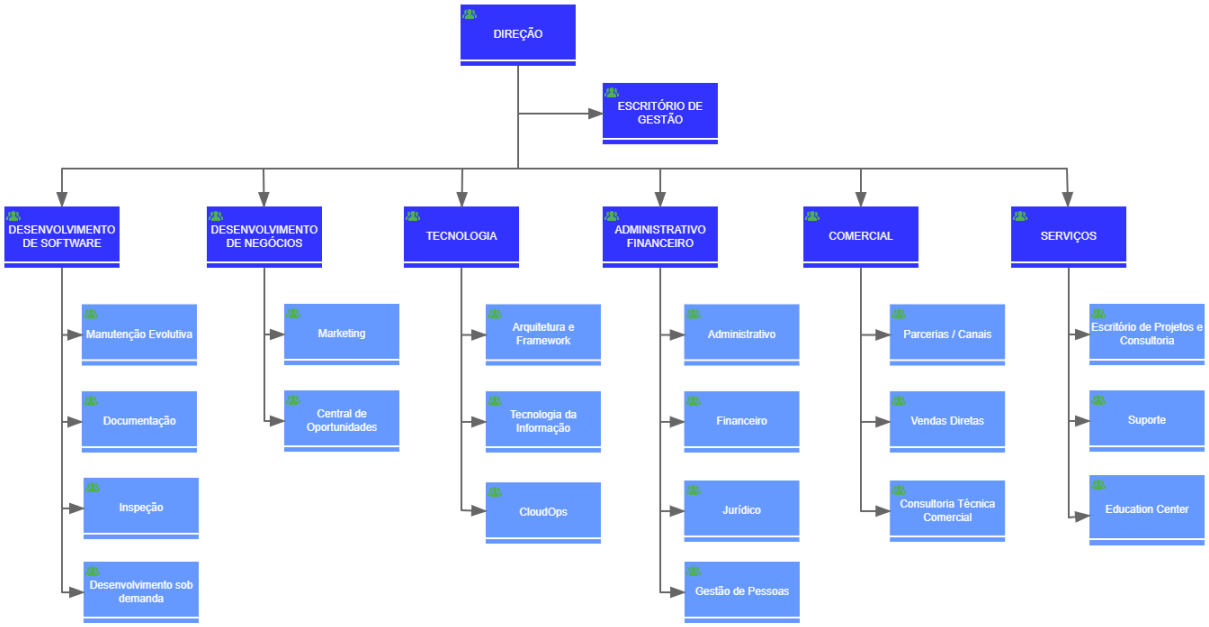


Figura 8. Organograma da empresa. Fonte: o Autor.

A área de Desenvolvimento de Software reúne alguns benefícios da filosofia ágil com a gestão de projetos. A gestão e planejamento de projetos utiliza a metodologia SCRUM, dividindo em ciclos de duas semanas, porém com uma liberação trimestral para o mercado.

Este estudo se concentra na subárea de Manutenção Evolutiva, que tem a responsabilidade de evolução e manutenção de 38 módulos do produto e está dividida entre 12 equipes de desenvolvimento.

A composição mínima de uma equipe é de um *Product Owner* (PO), um SM, um Analista de Teste e dois ou mais desenvolvedores de software. No total esta pesquisa envolveu diretamente 82 profissionais, sendo: 1 Diretor da área, 1 Coordenador da Inspeção, 12 PO, 12 SM, 44 Desenvolvedores e 12 Analistas de Teste.

O produto é desenvolvido em diversas linguagens de programação, sendo a mais utilizada a linguagem PHP. Baseado nisso, este projeto atuou especificamente no código-fonte desenvolvido em PHP. O desenvolvimento em PHP na empresa iniciou em 2002 e até dezembro de 2018 a empresa possuía 3.499.708 milhões de linhas de código-fonte em PHP, que estão sob a responsabilidade das 12 equipes de desenvolvimento. O gráfico ilustrado na Figura 9 apresenta a quantidade de linhas de código-fonte em PHP que cada equipe é responsável. As equipes T01 e T02 possuem as maiores quantidades de código-fonte PHP sob sua responsabilidade. Todas as

equipes têm desenvolvimento em PHP, desta maneira este estudo afeta diretamente a todos os desenvolvedores da área.

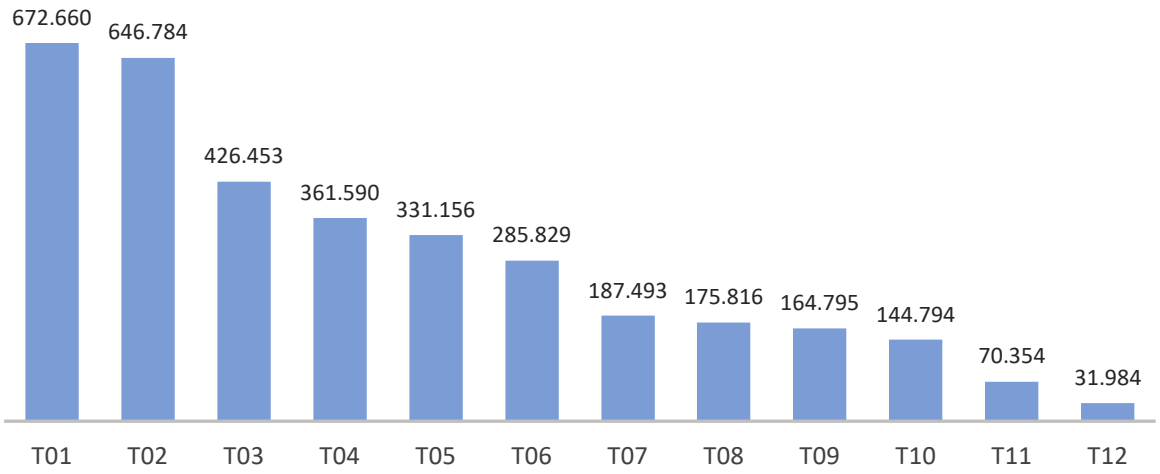


Figura 9. Quantidade de linhas de código-fonte PHP por equipe. Fonte: o Autor.

Todas as iniciativas das áreas devem estar alinhadas com o planejamento estratégico da empresa. Um dos objetivos estratégicos que impacta diretamente a área de Desenvolvimento de Software é o *"Elevar o nível de produtividade e qualidade no desenvolvimento e lançamento dos produtos"*. Este objetivo visa buscar iniciativas para melhorar a qualidade e a produtividade no desenvolvimento de software.

Para avaliar a qualidade do produto a empresa utiliza o *Key Performance Indicator (*KPI) *"Quantidade média de bugs por cliente"* que é medido mensalmente. A empresa, no entanto, não possui um KPI para avaliar a produtividade no desenvolvimento dos produtos. A avaliação da produtividade é feita por demonstrações periódicas das novas funcionalidades do produto e por acompanhamento do *roadmap* estratégico que é aprovado anualmente da mesma maneira que ocorre a revisão do planejamento estratégico.

O diretor da área de Desenvolvimento formou um grupo de profissionais para propor e avaliar iniciativas que impactassem no objetivo estratégico de *"Elevar o nível de produtividade e qualidade no desenvolvimento e lançamento dos produtos"*. Este grupo foi formado por profissionais com destaque nas suas funções, sendo: três PO, três SM, dois Desenvolvedores, dois Analistas de Teste e o Diretor da área.

Como membro do grupo, apresentei os conceitos de Dívida Técnica e as atividades para a Gestão da Dívida Técnica como abordagem que contribui para a

qualidade e a produtividade no desenvolvimento de software. A partir disso, o grupo aprovou a iniciativa estratégica para a Gestão da Dívida Técnica do código-fonte PHP e o início desta pesquisa-ação.

A iniciativa estratégica para Gestão da Dívida Técnica do código-fonte PHP foi desdobrada da seguinte maneira:

1. Definir e implementar um padrão de codificação;
2. Definir e implementar um padrão de documentação do código-fonte;
3. Identificar, medir e gerenciar as principais Dívidas Técnicas;
4. Identificar e propor ações de melhoria no código-fonte mais relevantes para o projeto.

Para monitorar as mudanças provocadas pelas iniciativas buscou-se utilizar KPIs que já estavam sendo monitorados. Desta maneira, foram selecionados os KPIs: Quantidade média de *bugs* por cliente e Eficácia na Detecção de Defeitos (representa a quantidade de *bugs* registrados pela inspeção por versão do módulo). Os dois KPIs são para monitorar a qualidade do software. Para monitorar a produtividade nenhum KPI foi definido, principalmente pelo fato de a empresa não possuir critérios bem definidos para medir o nível de produtividade das equipes.

Os KPIs não possuem uma relação direta com a Dívida Técnica da empresa, porém a partir dos seus dados históricos podemos avaliar se as ações promovidas por este estudo tiveram algum efeito na qualidade ou produtividade das equipes.

A partir do levantamento do ambiente e seus objetivos, foi possível identificar que a pesquisa poderia contribuir na definição e evolução de um processo de Gestão de Dívida Técnica de código-fonte e apresentar soluções para identificar o código-fonte mais relevante para o pagamento da Dívida Técnica, ou seja, para a priorização dos itens de Dívida Técnica de código-fonte.

A Figura 10 apresenta de forma resumida o contexto da pesquisa, que descreve o ambiente da pesquisa, o objetivo da empresa, o objetivo da pesquisa e os envolvidos e interessados.

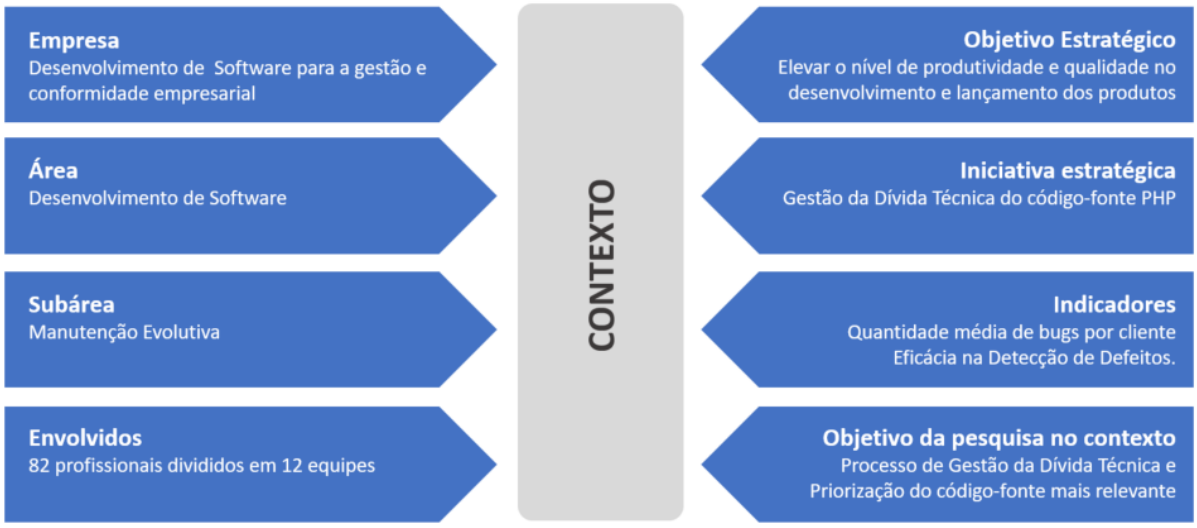


Figura 10. Resumo do contexto da pesquisa. Fonte: o Autor.

CAPÍTULO 5 - 1º CICLO

No primeiro ciclo da pesquisa foram definidas e implementadas diversas atividades para elaborar a Gestão da Dívida Técnica no contexto de estudo.

A definição das atividades para este trabalho partiu da proposta de gerenciamento da Dívida Técnica de Seaman e Guo (2011). Contou-se com as experiências relatadas nos trabalhos de Oliveira, Goldman e Santos (2015) e do seminário de Dagstuhl (2016) para orientar a execução das tarefas de identificação, medição e monitoramento.

Conforme a orientação de estudo de Holvitie et al. (2018) buscou-se envolver os *stakeholders* e estabelecer uma comunicação clara com os envolvidos. Esta prática ajudou no patrocínio e disseminação do projeto na empresa.

O desafio deste ciclo foi implementar os conceitos da Dívida Técnica e da Gestão da Dívida Técnica no contexto deste estudo, que até o momento, não tinha um histórico de projetos desta natureza.

Para ajudar nestes desafios foi criada a guilda de Dívida Técnica. O início da guilda de Dívida Técnica foi marcado por discussões e alinhamentos sobre os propósitos, objetivos, diretrizes para condução das atividades e assuntos de interesse para a organização e seus membros. O objetivo da guilda é estudar e ajudar na implantação e monitoramento da Gestão da Dívida Técnica, com propostas e ações para melhorar a qualidade interna e reduzir os custos de manutenção e evolução do software.

Na etapa de Avaliação são descritos diversos resultados deste ciclo, destacando o desenvolvimento do Processo de Gestão da Dívida Técnica como o resultado mais importante para a continuidade do projeto na empresa.

5.1 Coleta e análise de dados

Para executar esta etapa formou-se um grupo de profissionais denominado guilda da Dívida Técnica. A guilda foi responsável pelas coletas e análises dos dados, levantamento das ferramentas necessárias e aprovação das etapas do projeto com

os interessados. A coleta dos dados foi realizada por meio de observações, entrevistas e de uma ferramenta de análise estática de código-fonte.

A empresa não possui um histórico de projetos que envolvessem os conceitos de Dívida Técnica, desta maneira, foi necessário alinhar os conceitos e as expectativas sobre o assunto.

Em Zazworka et al. (2011) os participantes fizeram uso de ferramentas automáticas para identificar a Dívida Técnica. No presente estudo, além do uso, foi personalizada a ferramenta SonarQube para identificar e analisar a Dívida Técnica. Conforme relatado por Oliveira et al. (2015) é importante fazer uma análise manual das Dívidas Técnicas. No presente estudo, após uma análise manual, foi possível personalizar a ferramenta para automatizar a identificação da Dívida Técnica.

Após a criação da guilda foram definidas as demais atividades da etapa:

- Avaliação e instalação de ferramentas;
- Análise da prioridade das regras de qualidade;
- Definição do valor principal da Dívida Técnica;
- Medição da Dívida Técnica.

5.1.1 Guilda de Dívida Técnica

O método de desenvolvimento ágil, principalmente o Scrum, promove que as equipes sejam auto-organizadas e multidisciplinares, formadas por não mais do que dez pessoas e com um objetivo específico. (YIN; FIGUEIREDO; SILVA, 2011). A formação da guilda seguiu também essa diretriz, buscando-se reunir pessoas de diferentes times com funções e habilidades distintas.

A guilda foi composta por 7 profissionais de diferentes equipes da área de Desenvolvimento, sendo eles: dois PO, três Desenvolvedores Sênior em PHP, um Analista de DevOps e um Analista de Testes. Sendo um PO o coordenador da guilda e mais seis membros ativos.

Guildas, ou Comunidades de Prática, não são um fenômeno novo. O conceito de Comunidades de Prática foi introduzido pela primeira vez em 1991 por Jean Lave e Etienne Wenger. O livro de Emily Webber (2016) apresenta as principais características e experiências para criar uma comunidade de prática em uma organização. Um exemplo bem conhecido de Guildas em empresas de tecnologia é o

Spotify. Henrik Kniberg publicou um vídeo em 2014[2], que popularizou o que é conhecido como “modelo Spotify” (KNIBERG, 2014).

A formação da guilda de Dívida Técnica partiu das orientações para formação de equipes da metodologia Scrum, das orientações de Webber (2016), das características de autonomia e alinhamento com os objetivos estratégicos dadas por Kniberg (2014) por meio do modelo do Spotify de organização das equipes.

A guilda deve estar alinhada com a estratégia da empresa e ter objetivos bem definidos. Portanto, a guilda buscou se orientar pelas seguintes características:

- Estar alinhado com a estratégia da empresa;
- Ter um propósito ou objetivo bem definido;
- Ter autonomia para implementar soluções;
- Comunicar claramente os problemas e as oportunidades para os interessados;
- Escolher os membros de diferentes equipes, porém considerando que os membros deveriam ter conhecimento no contexto do trabalho;
- Ser um incentivador e promotor dentro das equipes para as ações de pagamento da Dívida Técnica;
- Ter influência nas equipes para ajudar no direcionamento e priorização das tarefas de refatoração da Dívida Técnica;
- Manter o foco na qualidade e produtividade no desenvolvimento de software;
- Ajudar na definição das ações de priorização e pagamento da Dívida Técnica;
- Orientar as equipes sobre as melhores práticas e padrões de desenvolvimento interno;
- Ter encontros periódicos para monitorar as ações e propor mudanças.

Ficou definido que a guilda de Dívida Técnica teria o objetivo de implementar e monitorar um programa de Gestão da Dívida Técnica. A guilda seria responsável pelo planejamento, definição dos padrões de desenvolvimento e treinamento das equipes.

Devido à orientação de alcançar resultados específicos, que visam melhorar a qualidade e produtividade no desenvolvimento com foco no código-fonte em PHP, a

[2] O vídeo pode ser encontrado em: https://engineering.atspotify.com/2014/03/27/spotify-engineering-culture-part-1/

guilda foi formada por profissionais que conheciam a arquitetura e os impactos do código-fonte PHP no produto.

A decisão de montar uma guilda com encontros periódicos foi devido ao contexto da empresa. Porém foi percebido que um projeto de Gestão da Dívida Técnica poderia ser implementado de outras maneiras, tais como, a alocação de uma equipe especializada e focada somente neste projeto ou a criação de um projeto compartilhado com outras equipes.

5.1.2 Ferramentas

O levantamento e análise de dados teve o apoio de ferramentas de análise estática do código-fonte em PHP e a personalização da ferramenta para se adaptar a arquitetura do produto. As ferramentas e as necessidades de personalização foram identificadas pela guilda de Dívida Técnica.

Foram selecionados o SonarQube e o plugin SonarPHP para fazer a leitura e monitoramento da Dívida Técnica do código-fonte PHP. Neste ciclo da pesquisa foram instalados o SonarQube versão 6.5.0.27846 e o plugin SonarPHP versão 2.11.0.2485. A Figura 11 demonstra a instalação do SonarQube com plugin SonarPHP.

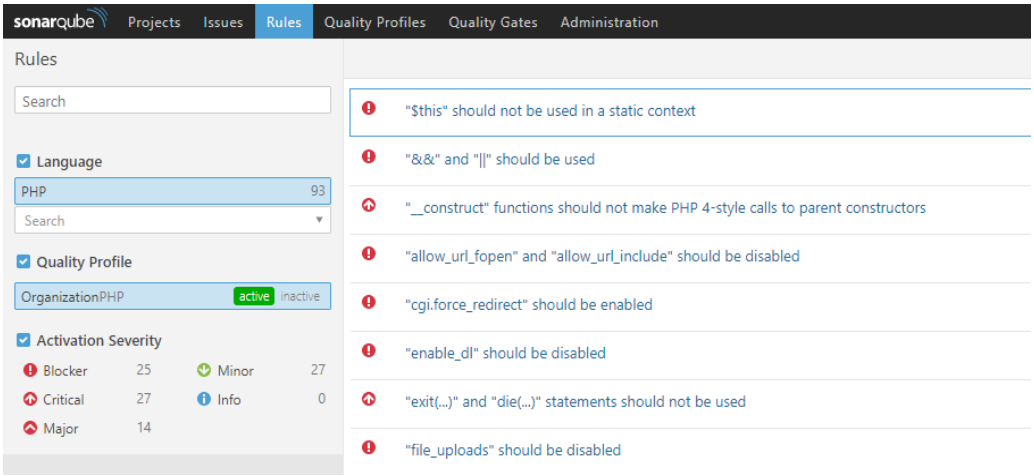


Figura 11. SonarQube versão 6.5. Fonte: o Autor.

A guilda avaliou a necessidade de construir duas regras de qualidade personalizadas no SonarQube. Estas regras têm um alto impacto na qualidade do código-fonte, pois a forma de desenvolvimento atual não possibilita a criação de testes unitários.

A primeira regra de qualidade personalizada no SonarQube tem o objetivo de evitar que o desenvolvedor utilize a variável “*superglobal*” (ou global automática) *$_REQUEST* em uma pasta específica, nomeada como “/include/”, onde possui o código-fonte responsável pela manipulação e validação dos dados. O nome da regra de qualidade foi definido como “*Check use $_REQUEST in folder include*”, a Figura 12 demonstra o código-fonte desenvolvido em Java para atender à regra.

```
/*
 * SonarQube PHP Custom Rules by ____
 * Copyright (C) 2017-2017
 *
 */
package org.sonar.samples.php.checks;

import org.sonar.check.Priority;

@Rule(
  key = "S3",
  priority = Priority.MAJOR,
  name = "Check use $_REQUEST in folder include.",
  tags = {"convention"}
  )
@SqaleConstantRemediation("45min")
public class VerifyRequestInInclude extends PHPVisitorCheck {

    private static final String MESSAGE = "Can not use $_REQUEST in folder include.";
    private static final String REQUEST = new String("$_REQUEST");
    private static final String INCLUDE = new String("include");

  @Override
  public void visitToken(SyntaxToken token) {
      if  ((this.context().file().toString().contains(INCLUDE)) &&  (token.toString().contains(REQUEST))) {
          context().newIssue(this, token, MESSAGE);
      }
  }

  public void visitBlock(BlockTree tree) {
      if  ((this.context().file().toString().contains(INCLUDE)) &&  (tree.toString().contains(REQUEST))) {
          context().newIssue(this, tree, MESSAGE);
      }
  }
}
```

Figura 12. Código da regra “Check use $_REQUEST in folder include”. Fonte: o Autor.

A segunda regra de qualidade tem o objetivo de evitar que o desenvolvedor inclua comando SQL junto com o código-fonte PHP. Essa regra busca orientar o desenvolvedor para incluir os comandos de acesso ao banco de dados no lugar determinado pela empresa. Desta maneira é possível validar todos os comandos SQL nos SGBD, com os quais o produto é compatível, por meio de testes unitários. A regra foi nomeada como “*SQL commands should not be used in this folder”*.

Como forma de garantir que todo o novo código-fonte aprovado para ser incluído no pacote de liberação passasse pela análise do SonarQube, foi configurado o GitLab (ferramenta utilizada pela empresa para o controle de versão do código-fonte) para avaliar automaticamente as regras de qualidade do *merge request* (solicitação para enviar um pacote para uma versão do sistema).

A configuração do GitLab tem o objetivo de automatizar o procedimento de *Merge Request*. As equipes de Inspeção e DevOps foram responsáveis pela criação

de um *job* para executar o SonarQube com opção *preview*. A opção *preview* do SonarQube analisa somente as mudanças ocorridas no código-fonte de um pacote, evitando retornar todas as Dívidas Técnicas do código-fonte e apresentando somente as Dívidas Técnicas incluídas nas mudanças realizadas pelo desenvolvedor.

A Figura 13 apresenta um exemplo da execução do *job* SonarQube e a visualização das Dívidas Técnicas geradas por um *Merge Request*.

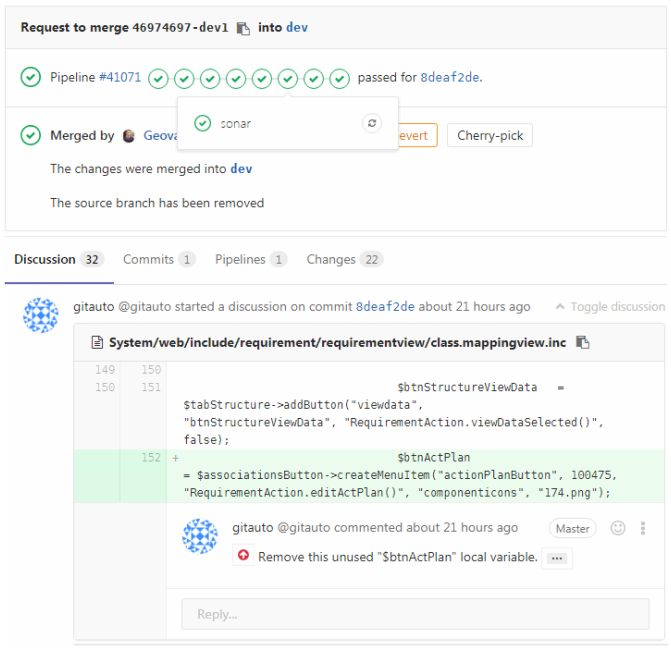


Figura 13. Exemplo da visualização do *job* SonarQube sendo executado no GitLab. Fonte: o Autor.

5.1.3 Análise da prioridade

Na análise de prioridade, as regras de qualidade foram classificadas considerando os objetivos e recursos disponíveis do contexto da pesquisa. Esta análise foi utilizada para priorizar as ações de pagamento da Dívida Técnica. Devido a isso, foi importante que todos os membros da guilda tivessem conhecimento de cada uma das regras de qualidade.

O plugin SonarPHP forneceu 127 regras de qualidade e foram criadas mais 2 regras por meio da plataforma do SonarQube.

Nem todas as regras de qualidade têm a mesma importância, e a importância pode variar entre os contextos. Portanto, outro grande problema prático no uso de ferramentas para gerenciar as Dívidas Técnicas é que as regras de qualidade não são

validadas e priorizadas de acordo com o contexto do aplicativo (FALESSI; VOEGELE, 2015).

Seguindo estas orientações, a análise de prioridade da Dívida Técnica foi realizada de maneira colaborativa entre o pesquisador e a guilda de Dívida Técnica.

As análises das regras de qualidade foram classificadas dentro dos critérios de prioridade definidos pela guilda. O Quadro 1 apresenta a lista de opções para definição da prioridade e uma descrição para orientar a classificação das regras utilizadas para gerar o cálculo da Dívida Técnica.

Quadro 1. Lista de prioridades. Fonte: o Autor.

P – Prioridade	
0 – *Não usado*	regra não utilizada e sem importância para empresa.
1 – *Baixa*	regras de boas práticas que devem ser monitoradas, porém a empresa aceita o desenvolvimento neste ciclo.
2 – *Média*	regras de menor importância e impacto na melhoria na qualidade do produto. Não pode aumentar a quantidade de itens destas regras.
3 – *Alta*	a quantidade de itens destas regras deve ser eliminada ou diminuída durante o período de execução da ação. Regras que a empresa entende que são muito importantes e têm alto impacto na melhoria da qualidade do produto e na padronização do código fonte.
4 – *Grave*	itens destas regras devem ser eliminados e não será permitido fazer *commit* contendo itens destas regras. Regras consideradas como *bugs*, vulnerabilidades do sistema ou comandos que não devem ser utilizados.

Para atribuir a prioridade a uma regra, foram levados em consideração: o contexto da empresa, as limitações técnicas e as limitações dos recursos. As prioridades foram classificadas conforme os objetivos que a empresa desejava atingir e dentro dos recursos disponíveis.

A análise de prioridade das regras de qualidade foi realizada por 5 membros da guilda isoladamente, seguindo as orientações descritas no Quadro 1 para definir a prioridade de cada regra. No Apêndice A está detalhada a avaliação de cada um dos membros da guilda que participaram dessa análise.

Ao agregar as análises das prioridades, realizou-se uma reunião com a guilda de Dívida Técnica para definir em conjunto todas as regras e então chegar a um consenso.

Ao calcular o Fleiss' Kappa (Fleiss, 1981), para avaliar o grau de concordância dos avaliadores, nos resultados das avaliações de prioridade das regras de qualidade, conforme detalhado no Apêndice A, obteve-se o valor de 58,60% que pode ser considerado como nível moderado de concordância entre os avaliadores.

A partir do resultado do Kappa motivou a realização de uma reunião com o grupo para definir em conjunto as prioridades das regras e então chegar a um consenso.

O Apêndice B apresenta a lista das regras de qualidade e os resultados das avaliações de prioridade após o consenso com os membros da guilda.

A Tabela 3 apresenta um resumo da quantidade de regras de qualidade por prioridade. Como se pode observar, 93 das 129 regras foram escolhidas para serem acompanhadas neste ciclo. As regras foram classificadas conforme as orientações do Quadro 1, onde foram classificadas as 93 regras selecionadas em: 25 regras como *Grave*; 27 regras como *Alta*; 14 regras como *Média* e 27 regras como *Baixa*.

Tabela 3. Quantidade de regras por prioridade. Fonte: o Autor.

P - Prioridade	Regras
0 – Não usado	36
1 – Baixa	27
2 – Média	14
3 – Alta	27
4 – Grave	25

5.1.4 Valor principal da Dívida Técnica

O valor principal da Dívida Técnica refere-se ao custo para eliminar um Item de Dívida Técnica. Dependendo do tipo de Dívida Técnica, isso pode se traduzir em diferentes tipos de atividades, como atualização de documentação desatualizada, refatoração de código-fonte difícil de manter ou definição de novos casos de teste para melhorar sua cobertura (SEAMAN; GUO, 2011; CURTIS; SAPPIDI; SZYNKARSKI, 2012).

Neste trabalho, foi considerado que o custo para eliminar a Dívida Técnica é igual ao esforço necessário para resolver cada Item de Dívida Técnica, e que para avaliar a viabilidade das ações para o pagamento da Dívida Técnica é necessário conhecer os esforços necessários para resolver cada Item de Dívida Técnica.

Foi definido que somente as regras de qualidade com prioridade *Grave*, *Alta* e *Média* teriam ações para pagar a Dívida Técnica. Assim, somente estas regras de qualidade passaram pela avaliação dos esforços.

No SonarQube é possível definir o esforço necessário para resolver cada Item de Dívida Técnica. Sua configuração possui valores de esforços para corrigir cada uma das regras. Porém, ao realizar uma análise mais detalhada, descobriu-se que os valores de esforços sugeridos pelo SonarQube não refletiam a realidade do contexto. Desta maneira, para auxiliar no cálculo do esforço da Dívida Técnica foi proposto analisar cada regra de qualidade sob os aspectos de complexidade e impacto. A complexidade é entendida como a dificuldade técnica para solucionar uma regra de qualidade.

No Quadro 2 estão descridos os critérios para avaliar a complexidade em alterar as regras de qualidade. Cada uma das complexidades possui uma orientação para classificar as regras de qualidade, a qual foi definida como: *Very low* (Muito baixa), para alterações simples ou realizadas por ferramentas automatizadas; *Low* (Baixa), alteração simples que deve ser feita manualmente; *Moderate* (Moderado), alteração simples que afeta mais de uma linha de código-fonte; *High* (Alta), alteração arriscada que afeta a funcionalidade da função ou classe; e *Very high*, alteração com alto risco de causar instabilidade no componente.

Quadro 2. Complexidade da alteração. Fonte: o Autor.

C – Complexidade	
1 – Very low	alteração simples e pode ser realizada por ferramentas já homologadas e validadas
2 – Low	alteração simples, porém, deve ser executada manualmente, alterando somente uma linha de código e não afetando outras rotinas.
3 – Moderate	alteração simples, porém, é necessário revisar e/ou alterar mais de uma linha de código do mesmo bloco.
4 – High	alteração arriscada, que é necessário revisar o resultado de toda a função e pode afetar mais de uma classe ou funcionalidade do sistema.
5 – Very high	Alteração arriscada, podendo causar instabilidade e possuindo alto risco de gerar problema.

O impacto de uma alteração é classificado pela extensão da modificação dentro do sistema, ou seja, o potencial da alteração afetar outros módulos ou classes. No Quadro 3 estão descritos os critérios para avaliar o impacto de uma alteração para

atender as regras de qualidade. Os impactos foram classificados em: *Very low* (Muito baixo), para alterações simples que não afetam o resultado; *Low* (Baixo), alterações simples que afetam somente o método ou função onde foi realizado a modificação; *Moderate* (Moderado), alteração que pode afeta o resultado da classe ou arquivo; *High* (Alta), alteração que afeta o resultado de outras classes ou arquivos; e *Very High* (Muito alta), alteração de alto risco que pode causar instabilidade em diversas funcionalidades do sistema.

Quadro 3. Impacto da alteração. Fonte: o Autor.

I – Impacto	
1 – Very low	alterações simples e que não afetam o resultado do código-fonte
2 – Low	alterações simples que afetam somente a função ou método da alteração
3 – Moderate	alteração que pode afetar o resultado da função ou método, porém não afeta outras classes ou arquivos
4 – High	alteração pode afetar outras classes e arquivos, podendo afetar o resultado
5 – Very high	alteração pode causar mudanças em outras classes e arquivos, alto risco de gerar problema, podendo afetar o resultado de mais de uma função

As regras de qualidade foram submetidas à análise de complexidade e de impacto e foram realizados testes com amostras de itens de Dívida Técnica para definir os esforços necessários para calcular o valor principal.

Durante uma semana um desenvolvedor resolveu diversos itens de Dívida Técnica e foram anotados os esforços para resolver cada um dos itens. A partir destas amostras, foi descoberto que o esforço necessário para resolver um Item da Dívida Técnica é diferente da soma dos esforços para resolver vários itens de Dívida Técnica no mesmo arquivo-fonte. Como a orientação para os desenvolvedores era de pagar a maior quantidade de Dívida Técnica do arquivo-fonte que estão trabalhando no momento, a utilização do cálculo de esforço por Item de Dívida Técnica iria superestimar a viabilidade das ações.

Devido ao compartilhamento dos procedimentos internos de liberação (Por exemplo: Code review e Cross test), de testes pela equipe de qualidade e pelo fato do desenvolvedor está imerso no mesmo contexto do código-fonte, foi identificado que a soma dos esforços para pagar um Item de Dívida Técnica é maior que o esforço gasto na prática para pagar todos os itens de Dívida Técnica em um mesmo arquivo-fonte.

A partir dessa constatação, para calcular o valor principal para este contexto é necessário um cálculo de esforço para resolver um Item de Dívida Técnica e um cálculo do esforço para resolver mais de um Item de Dívida Técnica no mesmo arquivo-fonte. Para o cálculo de esforço para todos os Itens de Dívida Técnica do mesmo arquivo-fonte, foi definido o Esforço Ajustado, como sendo o tempo em minutos para eliminar todos os itens de Dívida Técnica de um arquivo-fonte.

O Esforço Ajustado é igual ao esforço para resolver um item de Dívida Técnica, porém considerando que o desenvolvedor irá resolver mais de um item de Dívida Técnica no mesmo arquivo-fonte. Da mesma maneira que foram definidos os esforços para cada uma das regras de qualidade, foi obtido o Esforço Ajustado a partir da medição das amostras do código-fonte.

O cálculo do esforço ajustado total de cada arquivo-fonte ficou definido como sendo a soma do Item de Dívida Técnica com maior esforço com o total do esforço ajustado de todos os outros itens de Dívida Técnica.

A Tabela 4 apresenta um exemplo do cálculo do Esforço e do Esforço Ajustado de um arquivo-fonte. O total do esforço para pagar a Dívida Técnica do arquivo-fonte chartEngine.inc é de 1.009 minutos, porém ao utilizar os valores do Esforço Ajustado, o tempo necessário para pagar a Dívida Técnica do arquivo-fonte diminui para 69,3 minutos. A classe chartEngine.inc possui 322 itens de Dívida Técnica de 8 regras de qualidade diferentes. O maior esforço individual é para a regra de qualidade “Check use $_REQUEST in folder include”. Assim o seu valor de esforço, de 30 minutos, é mantido para o cálculo do esforço ajustado.

Tabela 4. Exemplo do cálculo do esforço e esforço ajustado. Fonte: o Autor.

Regra	Prioridade	Esforço	Esforço Ajustado	Item da DT
Control structures should use curly braces	Média	84	2.8	28
Sections of code should not be "commented out"	Média	5	1.0	1
Source code should comply with formatting standards	Média	786	26.2	262
Tabulation characters should not be used	Média	3	0.1	1
The "var" keyword should not be used	Média	66	2.2	22
Check use $_REQUEST in folder include.	Alta	30	30.0 *	1
Jump statements should not be followed by other statements	Alta	15	3.0	3
Unused local variables should be removed	Alta	20	4.0	4
Total		**1009**	**69,3**	**322**

Quando se analisa isoladamente cada regra de qualidade e seu esforço individual para resolver a regra, o esforço para resolver todas as regras de um arquivo-fonte pode ser superestimado. Neste estudo, onde é recomendado o pagamento de todos itens de Dívida Técnica do arquivo-fonte, o cálculo do esforço ajustado ajuda no planejamento das ações, criando uma previsão mais assertiva sobre os custos no pagamento da Dívida Técnica.

No Apêndice B são apresentados os valores utilizados para as avaliações de Prioridade, Complexidade, Impacto, Esforço e Esforço Ajustado para cada uma das regras de qualidade.

5.1.5 Medição da Dívida Técnica

A medição da Dívida Técnica é realizada para auxiliar na tomada de decisão e na definição das ações para o controle, eliminação ou diminuição da Dívida Técnica. Neste procedimento de medição da Dívida Técnica buscou-se estimar e mensurar a Dívida Técnica a partir das definições de prioridade e dos parâmetros definidos para calcular o valor principal da Dívida Técnica.

A coleta dos dados foi realizada com a execução do SonarQube e posteriormente foi feita a classificação por prioridade e o cálculo dos esforços necessários para então decidir qual Dívida Técnica veria ser eliminada. O cálculo de esforço foi realizado por uma ferramenta desenvolvida pelo próprio pesquisador.

Ao executar o SonarQube no código-fonte PHP foram extraídos o total de 587.347 itens de Dívida Técnica. A Figura 14 ilustra a quantidade de itens de Dívida Técnica agrupada pela prioridade: 8.993 *Grave*; 37.441 *Alta*; 476.572 *Média*; e, 64.341 *Baixa.* A maior quantidade de itens de Dívida Técnica refere-se às Dívidas Técnicas classificadas com a prioridade *Média.* Essa diferença entre a quantidade de itens de Dívida Técnica entre as prioridades foi devido às regras de formatação do código-fonte.

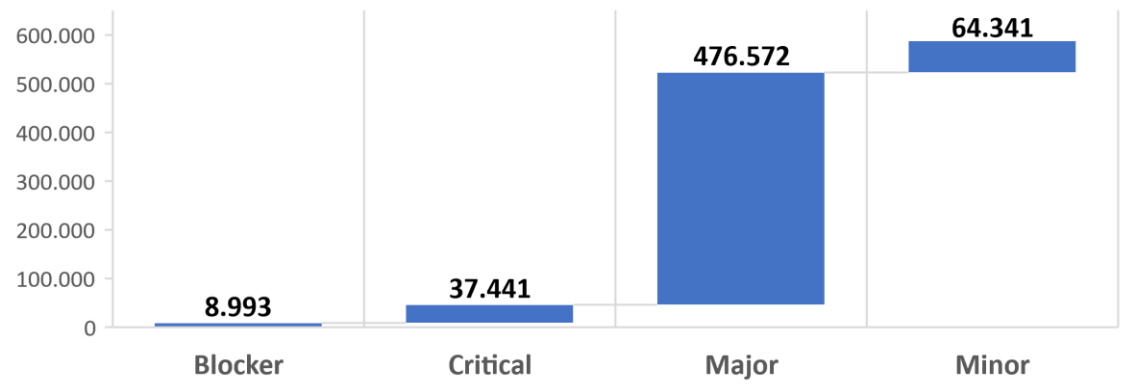


Figura 14. Quantidade de Item de Dívida Técnica por Prioridade. Fonte: o Autor.

Neste ciclo da pesquisa, a área de Desenvolvimento de Software estava dividida em 11 equipes. Desta maneira, para ajudar na tomada de decisão, as análises de quantidade de itens de Dívida Técnica e esforço foram realizadas por equipe.

As Figuras 15 e 16 apresentam o esforço ajustado total (medido em horas) e a quantidade de itens de Dívida Técnica das prioridades *Grave* e *Alta*.

Ao analisar a Figura 15, identificou-se a possibilidade de todas equipes conseguirem eliminar os itens de Dívida Técnica *Grave* durante o período das ações. A equipe T01 era a que mais possuía itens de Dívida Técnica; estimou-se que seriam necessárias 54,24 horas de desenvolvimento para eliminar todos os itens de Dívida Técnica, aproximadamente o dobro do tempo da equipe T02 que é a segunda com maior quantidade de itens de Dívida Técnica. No meio do gráfico tem-se as equipes T03, T04, T05 e T07 com ~20 horas e as outras equipes estão abaixo de ~7 horas de estimativa para eliminar os itens de Dívida Técnica *Grave*.

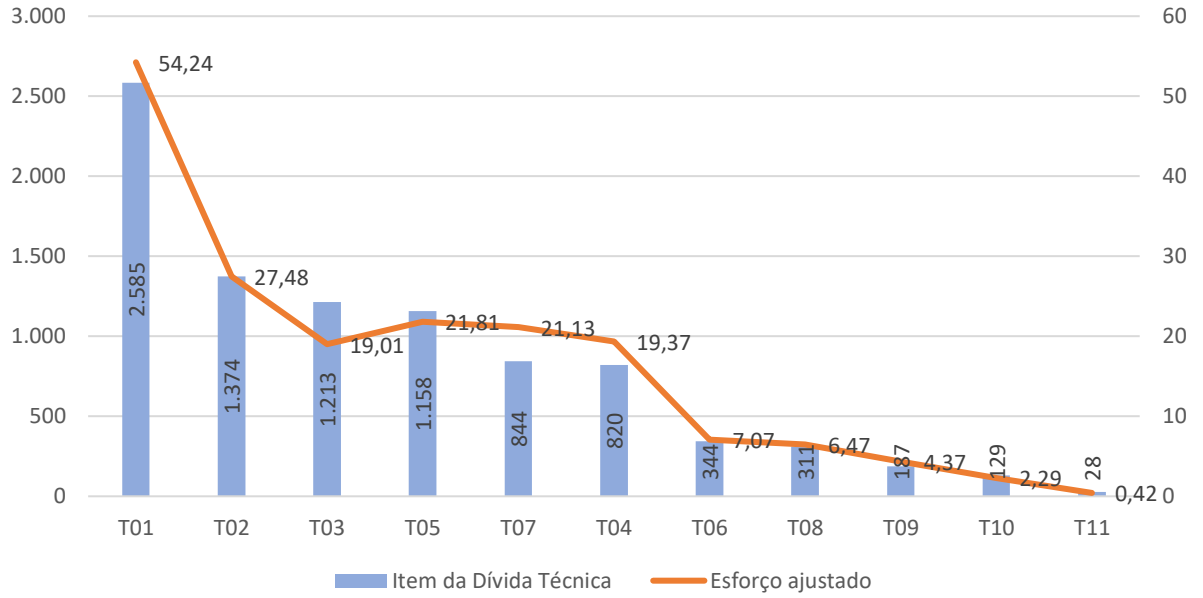


Figura 15. Esforço e quantidade de Item de Dívida Técnica *Grave* por equipe. Fonte: o Autor.

Conforme estimado (Figura 16), as equipes T01 (aproximadamente 627 horas) e T02 (aproximadamente 961 horas) são as que mais necessitavam de recursos para eliminar os itens de Dívida Técnica do tipo *Alta*. Com exceção da equipe T11 (aproximadamente 64 horas), as outras equipes foram estimadas entre aproximadamente 220 e 520 horas para eliminar os itens de Dívida Técnica *Alta.*

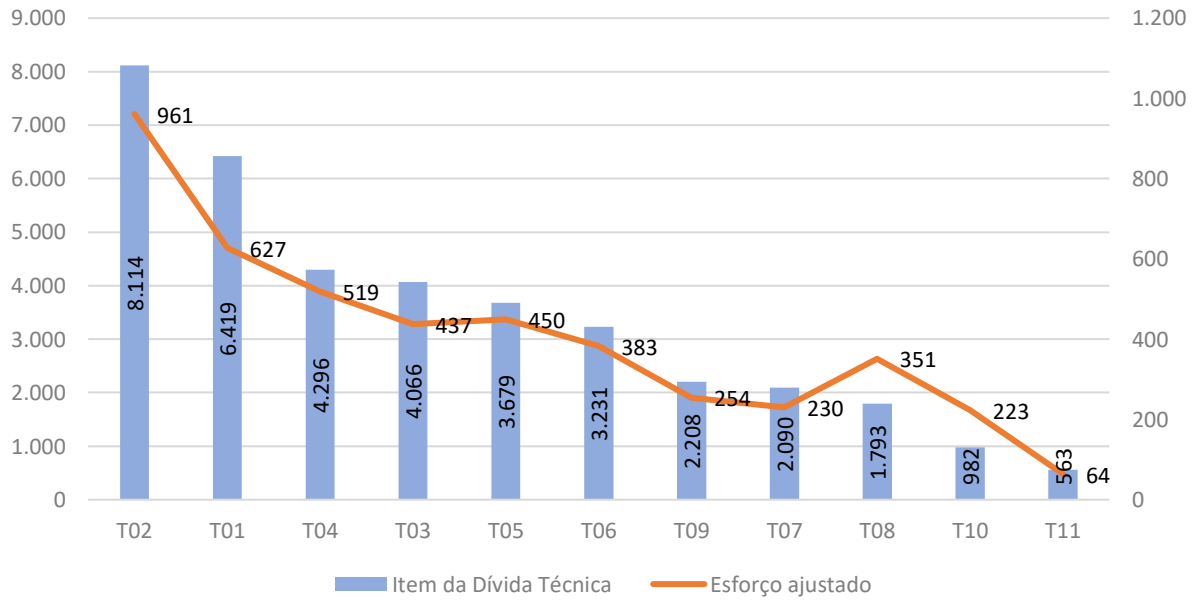


Figura 16. Esforço e quantidade de Item de Dívida Técnica *Alta* por equipe. Fonte: o Autor.

A Figura 17 apresenta uma análise das Dívidas Técnicas do tipo *Média* por equipe. Essa análise ajudou na decisão de não planejar ações para o pagamento das Dívida Técnicas deste tipo, uma vez que as equipes não disporiam de tempo suficiente dentro do ciclo planejado.

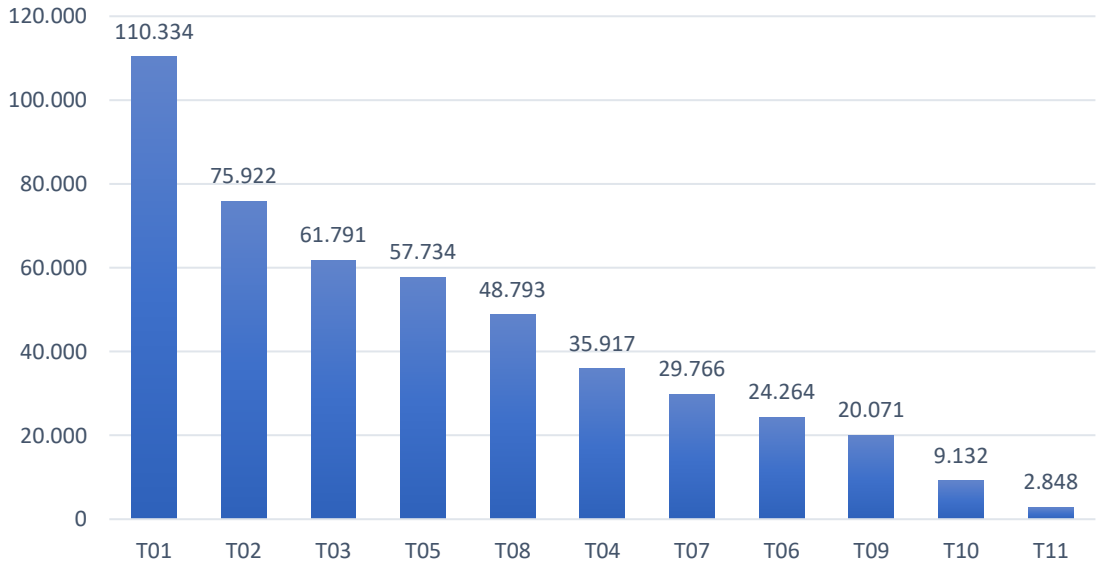


Figura 17. Quantidade de Item de Dívida Técnica *Média* por equipe. Fonte: o Autor.

A diferença na quantidade de Dívida Técnica por equipe é muito grande, o mesmo acontece com os esforços necessários para pagar a Dívida Técnica.

Para a priorização no pagamento da Dívida Técnica foram selecionados os itens com prioridade *Grave* e somente 3 regras de qualidades com prioridade *Alta*, devido à grande quantidade de itens desta prioridade. As 3 regras selecionadas pela guilda foram compreendidas como as mais importantes para possibilitar a implementação de testes unitários no futuro, um dos objetivos do projeto.

As Tabelas 7 e 8 apresentam a análise detalhada das três regras de qualidade selecionadas para fazer parte das ações de pagamento da Dívida Técnica, descritas como: "*Unused local variables should be removed*", "*SQL commands should not be used in this folder*" e "*Check use $_REQUEST in folder include*".

Na Tabela 5 estão agrupados os valores das regras por quantidade de itens de Dívida Técnica e por esforço. Apesar da regra R1 ter a maior quantidade de itens, a correção é mais simples, logo possui um esforço bem menor que as outras regras.

Tabela 5. Regras *Alta* priorizadas. Fonte: o Autor.

Regra	Item da Dívida Técnica	Esforço (hr)
R1 - Unused local variables should be removed	15.338	1.278
R2 - SQL commands should not be used in this folder	7.424	3.712
R3 - Check use $_REQUEST in folder include	5.620	2.810

Na Tabela 6 a quantidade de itens de Dívida Técnica é distribuída entre as onze equipes. Nota-se que as equipes T01 e T02 possuem as maiores quantidades de itens, semelhante ao apresentado nas Figuras 16, 17 e 18.

Tabela 6. Regras *Alta* priorizadas por equipe. Fonte: o Autor.

Regra / Equipe	T01	T02	T03	T04	T05	T06	T07	T08	T09	T10	T11
R1	2.807	3.050	1.790	1.842	1.369	1.306	1.025	625	913	232	379
R2	890	1.644	663	819	771	870	246	564	474	421	62
R3	853	1.180	575	747	661	477	286	199	332	236	74

A última análise desta etapa, conforme ilustrado na Figura 18, apresenta a quantidade de linhas de código-fonte por Item de Dívida Técnica de cada equipe. Nesta análise foram considerados somente os itens de Dívida Técnica com prioridade *Grave*, *Alta* e *Média*. A partir da análise destas informações pode-se concluir que existem Dívidas Técnicas em todas as funcionalidades do produto e na maioria dos arquivos-fontes.

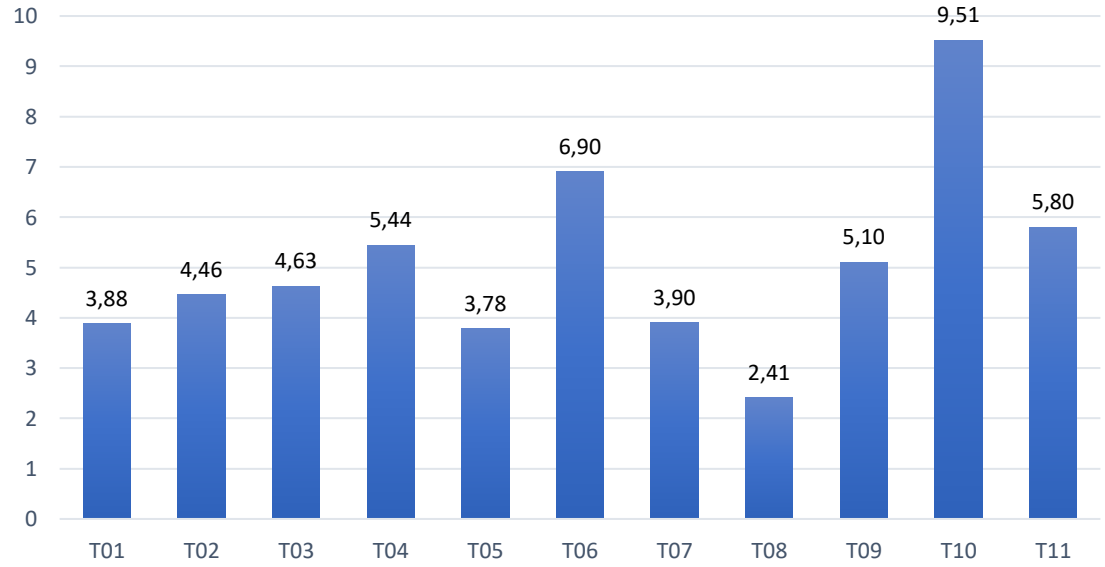


Figura 18. Quantidade de linhas de código-fonte por Item de Dívida Técnica. Fonte: o Autor.

5.2 Planejamento

A etapa de Planejamento das ações foi realizada em conjunto com as equipes e a guilda de Dívida Técnica. Nesta etapa foram estabelecidos os prazos e as metas para executar as ações.

A partir dos resultados das etapas anteriores se teve os subsídios para selecionar e priorizar o pagamento da Dívida Técnica. O primeiro passo desta etapa foi apresentar os resultados das análises para a guilda, onde foram planejadas as seguintes ações:

1. Elaborar e acompanhar as metas por equipe para a eliminação das Dívidas Técnicas priorizadas;
2. Realizar o treinamento interno sobre as melhores práticas de desenvolvimento em PHP e de sugestões de como reescrever o código-fonte para eliminar os itens de Dívida Técnica;
3. Definir os KPIs para o acompanhamento dos resultados das ações;
4. Divulgar amplamente o projeto e as metas de cada equipe pela diretoria.

Conforme as análises de prioridade e de esforços para eliminar um Item de Dívida Técnica foram elaboradas, foram geradas as metas para as equipes diminuírem as Dívidas Técnicas dentro período de janeiro a dezembro de 2018. Para os gestores acompanharem as metas e os resultados alcançados pelas equipes foi disponibilizado um *Portal WEB* com os KPIs de cada equipe.

A Gestão da Dívida Técnica orienta para o desenvolvimento dentro dos padrões e das melhores práticas de *Design* e programação. Desta maneira foi necessário repassar estes padrões para as equipes. O treinamento interno para os desenvolvedores buscou repassar o conhecimento e comunicar efetivamente a todos eles sobre as mudanças nos procedimentos do desenvolvimento do código-fonte em PHP.

Todas as ações deveriam ter impacto nos objetivos estratégicos. Por esse motivo os responsáveis solicitaram que os resultados das ações pudessem ser acompanhados por KPIs internos para avaliar a efetividade do projeto.

A guilda orientou sobre a importância de divulgar o projeto e as metas para todas as equipes, para aumentar a aderência ao projeto. Essa divulgação foi realizada pelo Diretor da área, reforçando o comprometimento da alta gestão com os resultados do projeto.

5.3 Implementação

A fase de implementação das ações deste 1º Ciclo ocorreu no período de janeiro a dezembro de 2018. Cada ação foi desdobrada em tarefas e decisões para realizar o pagamento da Dívida Técnica, considerando as restrições de tempo e recursos das equipes.

5.3.1 Ação 1: Elaborar e acompanhar as metas para pagamento da Dívida Técnica

Esta ação foi dividida em elaborar e acompanhar as metas. Na etapa de elaboração das metas foram determinadas as seguintes orientações:

1. Todas as regras de qualidade com prioridade *Grave* deveriam ser eliminadas;
2. O custo para eliminar todas as Dívidas Técnicas de prioridade *Alta* poderia gerar uma sobrecarga em algumas equipes, assim, com exceção das 3 regras de qualidade selecionadas na Tabela 5, todas as outras regras com prioridade *Alta* deveriam ser eliminadas. Para as exceções, a meta foi definida por regra, conforme descrito na Tabela 7. Para a regra R1 deveria-se eliminar no mínimo 2.000 itens de Dívida Técnica, para as regras R2 e R3, no mínimo 500 itens de Dívida Técnica;

Tabela 7. Meta para as exceções das regras *Alta.* Fonte: o Autor.

Regras (exceções)	Meta por equipe
R1 - Unused local variables should be removed	> 2.000 IDT
R2 - SQL commands should not be used in this folder	> 500 IDT
R3 - Check use $_REQUEST in folder include	> 500 IDT

3. Cada equipe ficou responsável por sua Dívida Técnica. As equipes não deveriam utilizar recursos compartilhados de outras equipes;
4. As ações 1 e 2 seriam as metas das equipes para o pagamento da Dívida Técnica correspondente ao período de um ano, tendo como vigência de janeiro a dezembro de 2018;
5. As metas para eliminar a Dívida Técnica fizeram parte do programa de participação de resultados da empresa.

Para o acompanhamento das metas pelas equipes foi disponibilizado um portal com os resultados da quantidade de Dívida Técnica atual e das metas, agrupando os resultados por equipe, por prioridade e pelas regras de exceção. Todos os desenvolvedores tiveram acesso a este portal e puderam acompanhar a evolução no atingimento das suas metas.

Conforme o estudo de Oliveira et al. (2015) o esforço para gerenciar a Dívida Técnica pode chegar a 25% do tempo total do desenvolvimento. Neste ciclo, os esforços para o pagamento da Dívida Técnica foram dimensionados de maneira empírica, porém cada equipe deveria dispor de no mínimo 20% do seu tempo para pagamento das Dívidas Técnicas priorizadas.

5.3.2 Ação 2: Treinamento interno

O treinamento foi realizado para qualificar e orientar os desenvolvedores sobre as mudanças nos procedimentos de programação e de liberação de código-fonte em PHP, visando o atendimento das metas das equipes e o objetivo da empresa. O treinamento foi realizado em janeiro de 2018 em duas turmas, capacitando 24 desenvolvedores e foi constituído dos seguintes tópicos:

- Conhecer e entender as regras e os critérios de prioridades utilizados na ferramenta SonarQube;
- Como usar o SonarQube e visualizar a Dívida Técnica da equipe;
- Como executar o Sonar Scanner nos projetos em desenvolvimento;
- Como desenvolver teste unitário para o código fonte PHP;
- Procedimento para separar os comandos SQL do código PHP;
- Melhores práticas no uso do comando $_REQUEST do PHP;
- Como monitorar as metas da equipe.

O treinamento foi de 4 horas e em cada turma havia no mínimo um desenvolvedor de cada equipe. No total das duas turmas foram capacitados no mínimo 2 desenvolvedores de cada equipe.

Os desenvolvedores que participaram do treinamento eram responsáveis para repassar o conhecimento para os outros desenvolvedores da equipe. O treinamento foi documentado e publicado nas ferramentas internas de base de conhecimento da empresa.

5.3.3 Ação 3: KPIs para o acompanhamento dos resultados

A empresa desejava avaliar se as alterações no código-fonte para o pagamento da Dívida Técnica alterariam os resultados da qualidade do produto percebida pelo cliente. Para fazer este acompanhamento foram selecionados 3 KPIs, que possuíam um histórico de medição de mais de 5 anos:

- KPI1 - Quantidade de *bugs* registrados pelo cliente: quantidade de *bugs* registrados pelos clientes ativos por versão;
- KPI2 - Quantidade de *bugs* por cliente: indicador estratégico de quantidade de *bugs* registrados pelos clientes ativos divididos pela quantidade de clientes ativos por versão;
- KPI3 - Percentual de eficácia na detecção de defeitos: a eficácia na detecção de defeitos é coletada a cada versão de terceiro dígito do produto (uma nova versão é liberada a cada 3 meses). O percentual de eficácia é a proporção entre o total de defeitos encontrados na versão dividido pelos defeitos encontrados na inspeção interna do produto. Este KPI apresenta a efetividade da equipe de inspeção em encontrar os defeitos no produto.

Na medida que as equipes eliminassem as Dívidas Técnicas para atender às metas divulgadas pela diretoria, seria possível avaliar, por meio destes do KPIs, se as ações de pagamento da Dívida Técnica tiveram algum efeito sobre a qualidade do produto.

5.3.4 Ação 4: Divulgação do projeto pela diretoria

O objetivo desta ação é a divulgação do projeto de Gestão da Dívida Técnica do código-fonte PHP para todos os profissionais da área de Desenvolvimento de Software. A divulgação foi realizada em janeiro de 2018 pelo Diretor da área, onde foram explicadas: a importância do projeto, as metas das equipes para 2018 e a data de início do monitoramento da Dívida Técnica.

Esta ação teve o objetivo de apresentar a importância do projeto para todos os desenvolvedores, como sendo uma iniciativa da área para melhorar a qualidade e a produtividade no desenvolvimento de software.

5.4 Avaliação

A avaliação do 1º ciclo da pesquisa foi realizada por meio de observações do pesquisador e de análise dos KPIs após o período de execução das ações.

A formação da guilda de Dívida Técnica ajudou na disseminação do projeto dentro da empresa e contribuiu para identificar as Dívidas Técnicas adequadas aos objetivos da empresa.

Devido ao conhecimento holístico dos procedimentos adotados pela empresa no desenvolvimento de software, que foi possível obter devido ao nível de competência dos membros da guilda, foram identificadas as práticas do uso da global *$_REQUEST* junto com o código-fonte que possui as regras de negócio e o uso de comandos SQL no meio do código-fonte PHP que impossibilitam a criação de teste unitários.

As ferramentas utilizadas foram fundamentais para dar visibilidade à Dívida Técnica. O SonarQube forneceu uma priorização padrão das regras de qualidade, no entanto, foi necessário desenvolver uma categorização de prioridade específica para o contexto.

O envolvimento direto dos membros da guilda na categorização e classificação das regras de qualidade melhorou a confiança no estudo e no uso das ferramentas. A guilda forneceu segurança para os interessados, pois as decisões eram alinhadas com um grupo experiente e com a diretoria. Nesta fase chegamos ao mesmo resultado do estudo de Falessi e Voegele (2015), onde o engajamento dos especialistas constrói confiança dos desenvolvedores no uso das ferramentas e na categorização da Dívida Técnica.

Para cada regra de qualidade, os desenvolvedores apresentaram diferenças de interesse e importância. Foi necessária uma reunião de consenso para consolidar as classificações de prioridade. Apresentar exemplos de violações das regras para os desenvolvedores ajudou na avaliação do nível de prioridade das regras de qualidade, pois, na maioria das vezes, as descrições fornecidas pelo SonarQube não foram suficientes para entender e avaliar o nível de prioridade.

Nem todas as regras de qualidade do SonarQube podem ser consideradas uma Dívida Técnica. Pode-se entender que algumas das regras que possuem menor importância técnica, não afetam a qualidade ou produtividade da equipe. Porém acredita-se que todas as regras de qualidade selecionadas têm um nível de relevância

que pode contribuir na cultura dos desenvolvedores, tornando-os alinhados com o padrão de desenvolvimento e as melhores práticas propostos pela empresa.

Na etapa de análise de prioridade e no cálculo do valor principal da Dívida Técnica buscou-se o entendimento das Dívidas Técnicas dentro do contexto do estudo. Um dos resultados importantes destas análises foi a compreensão do impacto da Dívida Técnica dentro do contexto. Ao analisar detalhadamente todas as regras e suas prioridades, complexidades, impactos e esforços necessários, foi possível a construção de uma base de informações para apoiar a tomada de decisão na priorização e no pagamento da Dívida Técnica.

Na medição da Dívida Técnica, ou avaliação da Dívida Técnica, as informações foram utilizadas na tomada de decisão, de modo a definir quais Dívidas Técnicas necessitavam de ações e iniciativas para o pagamento. Neste momento, a empresa teve a visibilidade da Dívida Técnica no código-fonte.

A priorização foi realizada principalmente com base na criticidade das regras de qualidade. Esta decisão não obteve um resultado satisfatório no pagamento da Dívida Técnica, pois as equipes relataram que o critério utilizado para pagar a Dívida Técnica não considerou a importância do código-fonte para o projeto. Foram relatados desperdícios de esforço no pagamento de Dívida Técnica em código-fonte que não era utilizado pelos clientes ou que não tinha alteração há mais de dois anos.

A decisão que não obteve resultado satisfatório foi a padronização das metas para todas as equipes. Esta decisão não considerou os esforços versus a quantidade de recursos das equipes e a criticidade dos projetos das equipes. Desta maneira, as equipes não priorizavam o pagamento da Dívida Técnica, justificando que tinham atividades mais relevantes.

A decisão das metas fazerem parte do programa de bonificação da empresa contribuiu para que as equipes incluíssem no planejamento das *sprints* o pagamento da Dívida Técnica. Esta decisão, juntamente com a divulgação do projeto realizado pelo diretor da área, mostrou a todos os envolvidos a importância do projeto dentro da área de Desenvolvimento de Software. Estas ações tiveram um impacto positivo neste projeto.

Com a implementação das ações de divulgação das metas das equipes, da visualização da Dívida Técnica por *commit* e da divulgação dos indicadores para acompanhar os resultados das ações, permitiu a verificação e observação crítica da situação atual, ou seja, permitiu o monitoramento da Dívida Técnica.

Neste 1º ciclo do projeto, a etapa de monitoramento ocorreu até novembro de 2018, quando iniciou o processo de revisão e melhoria contínua da Gestão da Dívida Técnica.

A Tabela 8 apresenta as medições dos KPIs utilizados para avaliar os resultados deste ciclo. A medição foi realizada por versão do produto, o projeto iniciou na versão 8 e foi analisado até a versão 11. Os KPIs não apresentaram diferenças significativas nos resultados. Pela análise dos KPIs não foi possível isolar o resultado obtido deste ciclo da pesquisa. Como no segundo ciclo da pesquisa os resultados dos KPIs tiveram resultados mais positivos, acreditamos que os códigos-fontes liberados com as refatorações não tinham atingido um número representativo dos clientes.

A quantidade de clientes representa o total de contratos ativos que a empresa possui. Não é possível identificar a quantidade de usuários devido à forma de comercialização do produto pela empresa. A empresa possui diversos contratos, sendo que os maiores clientes possuem contrato para uso ilimitado de usuários e do produto.

Tabela 8. Resultado dos KPIs do primeiro ciclo. Fonte: o Autor.

Início	Fim	Versão	KPI1	Bug Interno	Qtd. Cliente	KPI2	KPI3
mar/16	mai/16	1	571	2.310	687	0,83	80,18%
jun/16	ago/16	2	537	2.477	655	0,82	82,18%
set/16	nov/16	3	438	2.700	662	0,66	86,04%
dez/16	fev/17	4	614	2.605	678	0,91	80,93%
mar/17	mai/17	5	543	2.302	698	0,78	80,91%
jun/17	ago/17	6	602	1.966	696	0,86	76,56%
set/17	nov/17	7	557	2.017	717	0,78	78,36%
dez/17	fev/18	8	673	2.140	747	0,90	76,08%
mar/18	mai/18	9	616	2.027	781	0,79	76,69%
jun/18	ago/18	10	577	2.356	798	0,72	80,33%
set/18	nov/18	11	635	2.188	818	0,78	77,51%

KPI1 (Quantidade de bugs registrados pelo cliente), KPI2 (Quantidade de bugs por cliente), KPI3 (Percentual de eficácia na detecção de defeitos)

A Tabela 9 apresenta a quantidade de itens de Dívida Técnica por versão do produto. Dentro do período de 4 versões houve uma diminuição de aproximadamente 38% do total de Dívida Técnica, sendo: aproximadamente 64% com criticidade *Grave*; 34% com criticidade *Alta*; 70% com criticidade *Média*, e; 11% com criticidade *Baixa*.

Tabela 9. Total de itens de Dívida Técnica por criticidade e versão. Fonte: o Autor.

Versão	SonarQube				
	Blocker	*Critical*	*Major*	*Minor*	Todos
11	3.189	24.574	139.057	56.696	223.516
10	5.835	28.826	158.768	56.997	250.426
9	7.165	33.061	258.151	59.699	358.076
8	8.993	37.441	476.572	64.341	587.347

Nenhuma das equipes cumpriu todas as metas para o pagamento da Dívida Técnica estabelecidas na “Ação 1”. O principal fator que influenciou para não cumprir com as metas foi a definição da priorização ser por prioridade, desconsiderando a relevância do código-fonte para o projeto. Os desenvolvedores tiveram dificuldade em priorizar a refatoração de código-fonte que tem baixa importância para o projeto. Como essa descoberta aconteceu por meio dos relatos dos desenvolvedores durante o monitoramento das metas, a diretoria resolveu manter as ações conforme planejadas e incluir esta orientação para o próximo ciclo da pesquisa.

As regras de qualidade classificadas com prioridade *Média* tiveram uma grande diminuição na quantidade, mesmo não fazendo parte da meta das equipes, pois foi possível utilizar ferramentas automatizadas de editor de código-fonte para ajudar na tarefa de refatoração destas regras.

Devido à análise de grande parte do código-fonte PHP, um resultado inesperado deste projeto foi o descobrimento de código morto, ou seja, código-fonte que não é executado por nenhuma rotina do produto. Porém, não foi possível medir a quantidade de código morto eliminado do software.

A comunicação ocorreu em todas as etapas do projeto de maneira que todos os responsáveis pela implementação do projeto e as partes interessadas compreendessem os fundamentos e as razões sobre os quais as decisões foram tomadas. A comunicação foi realizada principalmente por reuniões da guilda, reuniões de acompanhamento com os líderes das equipes e a sua formalização e repasse ocorreu por e-mail ou documentação na base de conhecimento.

Os resultados apresentados motivaram a revisão do modelo. Na reunião de acompanhamento dos resultados deste projeto foram destacados quatro resultados percebidos pelos responsáveis:

1. Mudança na cultura dos desenvolvedores: Mudança de comportamento dos desenvolvedores, apresentando maior preocupação em produzir os artefatos com mais qualidade e dentro dos padrões;

2. Melhoria no monitoramento do código-fonte: O monitoramento da Dívida Técnica permitiu aos gestores o acompanhamento mais próximo do desenvolvimento;
3. Aumento da motivação dos desenvolvedores: A importância dada pelo Diretor da área em patrocinar o projeto, motivou as equipes nas mudanças de comportamento em relação à qualidade do produto;
4. A definição das metas deve estar alinhada com a importância do código-fonte para o produto: As informações das regras de qualidade como sendo o único critério para a priorização causou muito desperdício de recursos e desconfiança da efetividade das ações de pagamento da Dívida Técnica.

Após a avaliação deste 1º ciclo, foi possível descrever um Processo de Gestão da Dívida Técnica, descrito no capítulo 8, que foi utilizado e evoluído no próximo ciclo da pesquisa.

CAPÍTULO 6 - 2º CICLO

No segundo ciclo da pesquisa seguiu-se as orientações e boas práticas do ciclo anterior, dando continuidade na Gestão da Dívida Técnica no contexto de estudo. Neste ciclo da pesquisa os desafios foram implementar padrões e boas práticas de desenvolvimento e priorizar a Dívida Técnica mais aderente às necessidades das equipes. Os resultados deste ciclo contribuíram para a evolução do Processo de Gestão da Dívida Técnica e para a criação de um método de priorização da Dívida Técnica.

6.1 Coleta e análise de dados

A maior parte das atividades desta etapa seguiu as orientações do ciclo anterior, porém com o mesmo objetivo de obter e analisar os dados para apoiar a tomada de decisão no pagamento da Dívida Técnica.

As principais mudanças nesta etapa foram nas atividades de Identificação e Medição da Dívida Técnica, onde buscou-se entender melhor as regras de qualidade e o impacto nos objetivos de negócio.

A execução desta etapa iniciou com a revisão da guilda de Dívida Técnica e com a revisão das ferramentas necessárias para o projeto. Sequencialmente foram executadas as tarefas de Identificação, Análise e Medição da Dívida Técnica.

A conclusão desta etapa forneceu os dados necessários para o planejamento das ações de pagamento da Dívida Técnica, que visam melhorar a qualidade e produtividade no Desenvolvimento de Software.

6.1.1 Guilda de Dívida Técnica

A revisão da guilda, para este ciclo, seguiu as mesmas orientações e objetivos do ciclo anterior. Neste grupo se manteve dois membros do 1º Ciclo.

A guilda foi formada por cinco desenvolvedores experientes, dois profissionais da inspeção e qualidade e um profissional da área de negócios, sendo um total de oito profissionais com diferentes funções. O qual foi mantido o mesmo coordenador da guilda do ciclo anterior e sete membros ativos.

A mudança dos membros da guilda teve como objetivo melhorar o comprometimento e representatividade dos desenvolvedores, que são os mais afetados pelo projeto.

A primeira tarefa da guilda foi revisar os objetivos e as ações realizadas no 1º Ciclo e compreender os impactos no produto. Cada membro da guilda apresentou as propostas para o novo ciclo, e foi decido que uma das ações deveria ser a definição de padrões de codificação e documentação do código-fonte em PHP. Essa padronização foi evoluída em relação à ocorrida no 1º Ciclo.

A guilda avaliou e aprovou o processo de Gestão de Dívida Técnica proposto pelo pesquisador para orientar as atividades, a partir da experiência adquirida no 1º Ciclo

As reuniões da guilda de Dívida Técnica para o planejamento do ciclo iniciaram em setembro de 2018 com período de implementação e pagamento da Dívida Técnica de janeiro a dezembro de 2019. A guilda realizou 4 reuniões para orientar as ações, porém durante o período, a guilda manteve a comunicação ativa via e-mails ou *chats*.

6.1.2 Ferramentas

O uso das ferramentas neste projeto foi fundamental para apoiar a Gestão da Dívida Técnica. Além de identificar a necessidade de novas ferramentas deve-se conhecer as ferramentas existentes, e compreender como estas ferramentas podem ser utilizadas para apoiar a tomada de decisão.

Foi realizado um levantamento das ferramentas utilizadas no desenvolvimento de software e quais informações seriam possíveis de serem extraídas para ajudar na Gestão da Dívida Técnica.

A empresa utiliza o Git e o Jira integrados, e no 1º Ciclo da pesquisa foi instalado o SonarQube. Na análise das ferramentas necessárias, foram sugeridas e implementadas duas ferramentas para avaliar a dependência do arquivo-fonte e o esforço para pagar todos os itens de Dívida Técnica do arquivo-fonte.

As ferramentas possuem as seguintes características e funções:

- **Jira:** software comercial utilizado para o monitoramento de tarefas e acompanhamento de projetos. Todas as alterações realizadas no código-fonte da empresa possuem uma tarefa registrada no Jira. Desde 2015 a 2018 foram registradas 78.859 tarefas. As tarefas possuem diversas

classificações e informações e todas as tarefas que geram alteração no código-fonte possuem um identificador que se relaciona com o Git;

- **Git**: software comercial utilizado na empresa para o controle de versões dos arquivos-fonte, o qual possui o histórico de edições dos arquivos. De 2015 a 2018 foram registrados 32.648 *merges request*, que estão referenciando 309.393 alterações nos arquivos. Este número de *merge requests* se refere somente aos aprovados e cuja alteração foi efetivada no pacote de liberação;
- **SonarQube versão 7.5**: é uma plataforma de código aberto desenvolvida pela SonarSource utilizada para análise da qualidade do código-fonte. Diariamente o SonarQube é executado no código-fonte da empresa. A leitura de 913 regras de qualidade das linguagens PHP, Java Script, Java e XML geraram 792.800 itens de Dívida Técnica;
- **Esforço por artefato**: foi desenvolvido internamente um sistema que calcula o esforço necessário para eliminar todos os itens de Dívida Técnica de cada arquivo-fonte. Esse cálculo foi desenvolvido conforme descrito no capítulo 5.1.4;
- **Dependência**: foi desenvolvido internamente um sistema para analisar o código-fonte em PHP e identificar todas as dependências de cada arquivo-fonte ou classe e a quantidade de referências no sistema que chamam o arquivo-fonte ou uma classe. Este programa foi necessário devido à arquitetura interna do código-fonte, e principalmente para analisar o impacto da alteração no código-fonte.

Além das ferramentas, foi personalizado o SonarQube para incluir uma regra de qualidade que avalia a documentação do código-fonte. A regra verifica se as declarações de funções e métodos na pasta "*/include/*" estão documentadas adequadamente conforme o padrão estabelecido. A regra foi nomeada como "*Functions declarations in /include/* should be properly documented in docblock format"*.

6.1.3 Identificação da Dívida Técnica

O propósito da identificação da Dívida Técnica é reconhecer o tipo e a Dívida Técnica de código-fonte que são importantes para atingir os objetivos.

Nesta etapa o objetivo da guilda foi de conhecer detalhadamente as novas regras de qualidade e os impactos que podem gerar nos objetivos da empresa. Para isso, buscou-se classificar as regras de qualidade nos tipos de Dívida Técnica conhecidos. A Dívida Técnica não identificada nesta fase, obviamente não foi incluída nas etapas posteriores.

A partir do conhecimento adquirido no 1º Ciclo da pesquisa, a guilda decidiu revisar as regras de qualidade sugeridas pelo SonarQube. Desta maneira, 4 membros da guilda avaliaram separadamente todas as regras e apresentaram os resultados para todos os membros da guilda em uma reunião de consenso. Nesta reunião foi decidido o uso de 125 regras de qualidade.

O coeficiente de concordância de Kappa dos avaliadores das regras de qualidade foi de 71,44%. Desta maneira, não foi necessário fazer uma reunião extra de consenso, as regras com maiores divergências foram avaliadas pontualmente, por meio de uma ferramenta de mensagens interna, entre o grupo de avaliadores.

Cada regra de qualidade foi classificada por Tipo de Dívida Técnica e por Dívida Técnica. A classificação por Tipo de Dívida Técnica seguiu as orientações da Tabela 1, desta maneira as regras de qualidade foram classificadas dentro das seguintes orientações:

- **Dívida de Código:** refere-se aos problemas encontrados no código-fonte, código mal escrito que viola as melhores práticas de codificação ou regras de codificação, que podem afetar negativamente a legibilidade do código, dificultando sua manutenção.
  - Duplicação de código: sequência de código-fonte que ocorre mais de uma vez dentro de uma rotina ou de um programa;
  - Função depreciada ou incompatível: recursos da linguagem de programação que foram retirados, incompatíveis ou substituídos nas versões superiores. Também é considerado funções que serão depreciadas nas versões futuras;
  - Legibilidade do código: corresponde a práticas de codificação que dificultam a leitura e o entendimento do código-fonte;
  - Violação do padrão de formatação: as regras correspondem a falhas no cumprimento do padrão de formatação estabelecido pela empresa;

- Violação do padrão de nomenclatura: Os nomes de arquivos, funções, classes ou métodos devem estar em conformidade com um padrão definido.

- **Dívida de Defeito:** refere-se a defeitos conhecidos, que podem ser identificados por atividades de teste, pelo usuário ou por análise de código-fonte:
  - Código inadequado: prática de programação que não deve ser utilizada, podendo ser código desnecessário ou com desperdício que tem possibilidade de simplificação;
  - Defeito encontrado no código: código escrito de maneira errada que o compilador ou interpretador pode identificar;
  - Possibilidade de resultado inesperado: regras que reduzem a compreensibilidade e a capacidade de manutenção e pode ser propenso a erros;
  - Vulnerabilidade de segurança: defeitos e vulnerabilidades no software que pode ter consequências indesejáveis devido a violação da segurança de software.
- **Dívida de *Design*:** refere-se a dívidas que violam os princípios e padrões de projeto de software.
  - Código muito complexo: trechos de código difíceis ou complicados de entender devido à grande quantidade de caminhos diferentes até chegar ao resultado;
  - Violação de padrão de codificação: regras que violam padrão de codificação sugerido e podem gerar dificuldades na manutenção do software.
- **Dívida de Documentação:** refere-se a problemas encontrados na documentação do código-fonte.
  - Documentação inexistente ou inadequada: regras que identificam que não existe a documentação no código fonte ou a documentação não está dentro do padrão.

O resultado após classificação das regras de qualidade nos tipos de Dívida e Dívida Técnica é apresentado na Tabela 10. No Apêndice C está detalhada a classificação de cada regra de qualidade. Para este 2º Ciclo foram selecionadas 125

regras de qualidade, divididas nos tipos de Dívida Técnica de Código, Defeito, *Design* e Documentação.

A Dívida de Documentação teve uma importância maior que no ciclo anterior. Neste ciclo os membros da guilda decidiram aumentar a importância das Dívidas Técnicas de Documentação e de violação de padrões da empresa, pois foi observado no ciclo anterior que estes tipos de Dívida Técnica ajudam na mudança de cultura dos desenvolvedores.

Tabela 10. Quantidade de regras por Tipo e Dívida Técnica. Fonte: o Autor.

Tipo de dívida	Regras
Dívida de Código	**44**
Duplicação de código	2
Função depreciada ou incompatível	9
Legibilidade do código	16
Violação do padrão de formatação	14
Violação do padrão de nomenclatura	3
Dívida de Defeito	**41**
Código inadequado	15
Defeito encontrado no código	3
Possibilidade de resultado inesperado	13
Vulnerabilidade de segurança	10
Dívida de *Design*	**37**
Código muito complexo	4
Violação de padrão de codificação	33
Dívida de Documentação	**3**
Documentação inexistente ou inadequada	3
Total de regras	**125**

6.1.4 Análise da Dívida Técnica

Esta etapa seguiu as orientações do 1º Ciclo da pesquisa de classificar as regras por critérios de prioridade, complexidade e impacto. A partir destas classificações foi possível calcular uma estimativa do valor principal da Dívida Técnica.

A prioridade foi atribuída a cada uma das regras de qualidade da Dívida Técnica. Para isso foram levados em consideração o cenário da empresa, as limitações técnicas e as limitações dos recursos. As prioridades foram classificadas conforme as possibilidades de consequências aos objetivos que empresa deseja atingir dentro dos recursos disponíveis.

O Quadro 4 apresenta uma lista de opções de prioridade e uma descrição para orientar a classificação das regras de qualidade selecionadas na etapa anterior. As prioridades foram classificadas entre Baixa, Média, Alta e Grave. Porém, somente as prioridades Alta e Grave tiveram ações de pagamento da Dívida Técnica.

Quadro 4. Lista de prioridade do segundo ciclo. Fonte: o Autor.

Prioridade	
Baixa	Regras de boas práticas que devem ser monitoradas, porém têm pouco ou nenhum impacto
Média	Regras de menor importância e impacto na melhoria na qualidade do produto. Não será permitido o desenvolvimento de novas funcionalidades com essas características.
Alta	A quantidade de itens destas regras deve ser eliminada ou diminuída. Regras que a empresa entende que são muito importantes e têm alto impacto na melhoria da qualidade do produto e na padronização do código fonte.
Grave	Itens destas regras devem ser eliminadas totalmente e não será permitido fazer *commit* (efetivar a alteração) com itens destas regras. Regras consideradas como *bugs*, vulnerabilidades do sistema ou comandos que não devem ser utilizados.

A análise das prioridades das regras foi realizada por 4 membros da guilda isoladamente, seguindo as orientações descritas no Quadro 4. Após as análises dos profissionais foi realizada uma reunião de consenso com todos os participantes da guilda. Apesar das análises isoladas terem tidos poucas divergências, esta atividade foi importante para que o conhecimento sobre as regras fosse disseminado para todos os envolvidos.

Para entender a viabilidade das ações é necessário calcular o esforço necessário para resolver cada Dívida Técnica. A análise da complexidade, do impacto, do esforço e do esforço ajustado seguiram as orientações do 1º Ciclo, porém neste ciclo foram revisados os critérios da complexidade e do impacto.

A complexidade é entendida como a dificuldade técnica para solucionar uma Dívida Técnica. No Quadro 5 estão descridos os critérios para avaliar a complexidade da alteração de cada regra de qualidade. As complexidades foram classificadas entre “Muito baixa”, alterações simples que podem ser realizadas automaticamente, até “Muito Alta”, alteração complexa na qual é necessário revisar os objetos ou arquivos dependentes.

Quadro 5. Complexidade da alteração do segundo ciclo. Fonte: o Autor.

Complexidade	
Muito baixa	alteração simples e realizada automaticamente por ferramentas já homologadas e validadas
Baixa	alteração simples, porém, deve ser executada manualmente, alterando somente uma linha de código e não afetando outras rotinas.
Média	alteração simples, porém, é necessário revisar e/ou alterar mais de uma linha de código do mesmo bloco.
Alta	alteração complexa, que é necessário revisar o resultado de toda a função ou classe.
Muito alta	alteração complexa, que é necessário revisar o código-fonte dependente da classe ou arquivo-fonte.

O impacto da alteração foi classificado pela extensão da mudança e a probabilidade de afetar outros módulos ou classes. No Quadro 6 estão descritos os critérios para avaliar o impacto da alteração das regras. Cada um dos critérios possui uma orientação para ajudar na classificação da regra, que pode ser “Muito baixo”, “Baixo”, “Médio”, “Alto” e “Muito alto”.

Quadro 6. Impacto da alteração do segundo ciclo. Fonte: o Autor.

Impacto	
Muito baixo	alteração simples que não afeta o resultado do bloco de código
Baixo	alteração simples que afeta somente a função ou método da alteração
Médio	alteração moderada que pode afetar o resultado da função ou método, porém não afeta outras classes ou arquivos
Alto	alteração moderada que pode afetar outras classes e arquivos, com possibilidade de afetar os resultados
Muito alto	alteração complexa que pode causar mudanças em outras classes e arquivos, alto risco de gerar instabilidade no produto, podendo afetar o resultado de mais de uma função

No Apêndice C está detalhada a classificação de cada regra de qualidade por prioridade, complexidade, impacto, esforço e esforço ajustados. A partir destes valores é possível calcular uma estimativa para o valor principal da Dívida Técnica

que foi utilizado como base para a definição de ações de pagamento da Dívida Técnica.

6.1.5 Medição da Dívida Técnica

Nesta etapa de medição da Dívida Técnica, é realizada a execução das regras de qualidade no código-fonte e são identificados todos os itens de Dívida Técnica. Cada Item de Dívida Técnica representa um desvio do padrão estabelecido pela empresa.

A Tabela 11 apresenta a quantidade de itens de Dívida Técnica e o esforço em minutos agrupado por tipo de Dívida Técnica e por Dívida Técnica. Notou-se que as Dívidas Técnicas de formatação e documentação representam 80,74% (Violação do padrão de formatação + Dívida de Documentação) de todos os itens de Dívida Técnica identificados. Porém, proporcionalmente, as Dívidas Técnicas de *Design* necessitam de mais esforço para serem resolvidas.

Tabela 11. Quantidade de Itens por Dívida Técnica. Fonte: o Autor.

Dívida Técnica	Item da Dívida	% Item da Dívida	Esforço	% Esforço
Dívida de Código	**510.365**	**67,13%**	**101.339**	**8,87%**
Duplicação de código	219	0,03%	438	0,04%
Função depreciada ou incompatível	18.270	2,40%	13.646	1,19%
Legibilidade do código	26.130	3,44%	25.139	2,20%
Violação do padrão de formatação	465.198	61,19%	51.156	4,48%
Violação do padrão de nomenclatura	548	0,07%	10.960	0,96%
Dívida de Defeito	**6.211**	**0,82%**	**11.083**	**0,97%**
Código inadequado	3.729	0,49%	5.682	0,50%
Defeito encontrado no código	8	0,00%	66	0,01%
Possibilidade de resultado inesperado	2.307	0,30%	4.533	0,40%
Vulnerabilidade de segurança	167	0,02%	802	0,07%
Dívida de *Design*	**95.085**	**12,51%**	**482.532**	**42,22%**
Código muito complexo	2.078	0,27%	58.371	5,11%
Violação de padrão de codificação	93.007	12,23%	424.161	37,11%
Dívida de Documentação	**148.628**	**19,55%**	**547.940**	**47,94%**
Documentação inexistente ou inadequada	148.628	19,55%	547.940	47,94%
Total Geral	**760.289**	**100,00%**	**1.142.894**	**100,00%**

A Dívida Técnica precisa ser analisada sob diversos aspectos, tais como: criticidade; impacto da dívida na evolução e manutenção; entre outras informações pertinentes para a organização. A Tabela 11 mostra a importância de analisar a Dívida Técnica conhecendo a quantidade de itens de Dívida Técnica e o esforço necessário

para pagar a Dívida Técnica, pois a quantidade de itens não representa proporcionalmente o mesmo esforço para pagar a Dívida Técnica. Para exemplificar, a Dívida Técnica “Violação do padrão de formatação” possui 61,19% do total de itens de Dívida Técnica, porém corresponde a 4,48% do total de esforço necessário.

A empresa separa as ações por equipes, desta maneira para cada Item de Dívida Técnica deve-se identificar a equipe responsável.

A Tabela 12 apresenta a quantidade de itens de Dívida Técnica por equipe e por prioridade. Essa informação está detalhada por Dívida Técnica no Apêndice D. As colunas de Esforço, da Tabela 12, representam a quantidade de minutos necessários para eliminar todos os itens de Dívida Técnica por criticidade.

Tabela 12. Quantidade de itens de Dívida Técnica por equipe e prioridade. Fonte: o Autor.

Equipe	Baixa		Média		Alta		Grave	
	Item da DT	Esforço	Item da DT	Esforço	Item da DT	Esforço	Item da DT	Esforço
T01	14.608	47.159	42.754	139.838	1.348	6.511	714	883
T02	13.323	41.010	220.373	183.470	2.434	8.167	363	566
T03	16.339	28.342	65.851	86.293	1.524	5.147	336	500
T04	14.638	29.554	72.270	89.913	1.063	3.906	239	295
T05	6.931	16.911	112.723	79.164	513	2.604	29	73
T06	10.577	24.841	54.467	77.852	651	2.584	52	63
T07	3.366	13.155	10.890	33.559	305	1.489	136	136
T08	6.173	15.643	21.748	50.283	340	2.695	4	12
T09	2.551	11.612	8.529	46.330	141	685	12	12
T10	4.465	7.746	18.350	31.904	314	1.282	92	136
T11	4.041	7.501	15.264	24.813	297	942	80	127
T12	1.652	2.030	7.314	14.720	96	426	9	9
Total	**98.664**	**245.504**	**650.533**	**858.140**	**9.026**	**36.438**	**2.066**	**2.812**

A Figura 19 apresenta a quantidade linhas de código-fonte por Item de Dívida Técnica de cada equipe. A partir da análise desta informação pode-se concluir que a Dívida Técnica está presente em todas as funcionalidades do produto e que, proporcionalmente, as equipes possuem diferenças significativas. Por exemplo, as equipes T11 e T12, apesar de terem uma quantidade muito menor que as primeiras equipes, têm um alto índice de Dívida Técnica por código-fonte.

Não foi possível comparar os gráficos da Figura 18 com o gráfico da Figura 19 devido às mudanças de quantidade e prioridade das regras de qualidade e a quantidade de equipes.

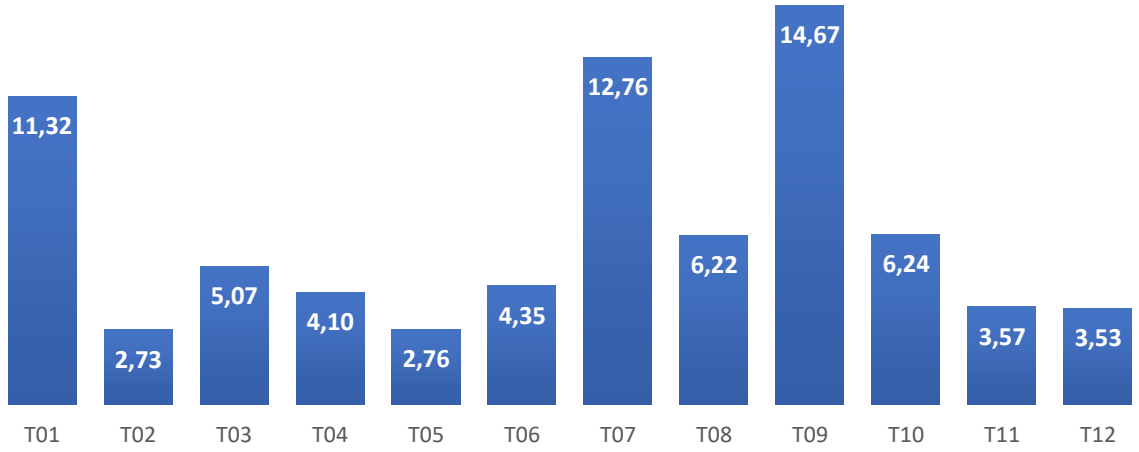


Figura 19. Quantidade de linhas de código fonte por Item de dívida técnica. Fonte: o Autor.

6.2 Planejamento

Na etapa de planejamento a guilda definiu as ações e o período de monitoramento. Neste 2º Ciclo, o período para a implementação e execução das ações foi entre janeiro e dezembro de 2019.

A partir da avaliação das ações do ciclo anterior e da coleta e a análise de dados deste ciclo, a guilda planejou 6 ações:

1. Elaborar as metas por equipe para o pagamento da Dívida Técnica. Nesta ação deverão ser selecionados os código-fonte que tiverem maior importância para o projeto ou equipe;
2. Definir KPIs para acompanhamento dos resultados das ações;
3. Definir e divulgar um padrão de codificação em PHP (evoluído);
4. Definir e divulgar um padrão de documentação do código-fonte em PHP;
5. Divulgar o projeto e as metas de cada equipe pela diretoria;
6. Elaborar um questionário interno para avaliar os resultados do projeto.

Todas as ações tiveram um representante da guilda como responsável. O responsável pela ação tinha o objetivo de garantir a execução da ação e apresentar os resultados para os interessados.

Para implementar a primeira ação, o pagamento da Dívida Técnica deve contar com argumentos que justifiquem a alocação dos recursos na alteração ou refatoração do código-fonte. Os argumentos utilizados para priorizar o pagamento da Dívida Técnica no 1º Ciclo da pesquisa não atendem a esta necessidade e foi necessário

elaborar um modelo de priorização do código-fonte que se aproximasse das decisões de negócio de cada equipe.

6.2.1 Priorização

A priorização é uma atividade de planejamento que fornece argumentos para executar o pagamento da Dívida Técnica. Nesta etapa foram apresentados os artefatos candidatos para o pagamento da Dívida Técnica de cada equipe.

Ao analisar a quantidade de Dívida Técnica por equipe, considerando o esforço e a criticidade fornecidos nas Tabelas 13 e 14, concluiu-se que a maioria das equipes não tinham recursos para pagar todos os itens de Dívidas Técnicas. Desta maneira, os gestores das equipes tiveram que priorizar o pagamento dos itens de Dívida Técnica que gerassem mais resultados para a empresa.

Inicialmente foi realizado um levantamento das ferramentas que apoiam o desenvolvimento de software e quais informações são possíveis de serem extraídas para ajudar na priorização e Gestão da Dívida Técnica. Antes de iniciar esta pesquisa a empresa já utilizava o Git e o Jira integrados e no 1º Ciclo da pesquisa foi instalado o SonarQube. Foi constatado que é possível centralizar as informações no arquivo-fonte, e o uso das informações fornecidas por estas ferramentas pode apoiar na decisão de pagar os itens de Dívida Técnica.

Nesta análise foram sugeridas e implementadas duas ferramentas para avaliar a dependência do arquivo-fonte e o esforço para pagar todos os itens de Dívida Técnica do arquivo-fonte.

Foi possível integrar as 5 ferramentas, conforme demonstrado na Figura 20, para obter o máximo de informação do arquivo-fonte. Como se pode observar, as tarefas registradas no Jira possuem uma integração com o Git. O Git, por sua vez, possui as informações do código-fonte alterado para atender a uma determinada tarefa. Na outra ponta da integração o SonarQube possui a quantidade de itens de Dívida Técnica de cada arquivo-fonte. Com esta informação é possível avaliar o esforço para refatorar o arquivo-fonte, conforme o cálculo sugerido no 1º Ciclo da pesquisa. Para analisar o impacto da alteração foi desenvolvida uma ferramenta que calcula a dependência do arquivo-fonte com outros arquivos em PHP.

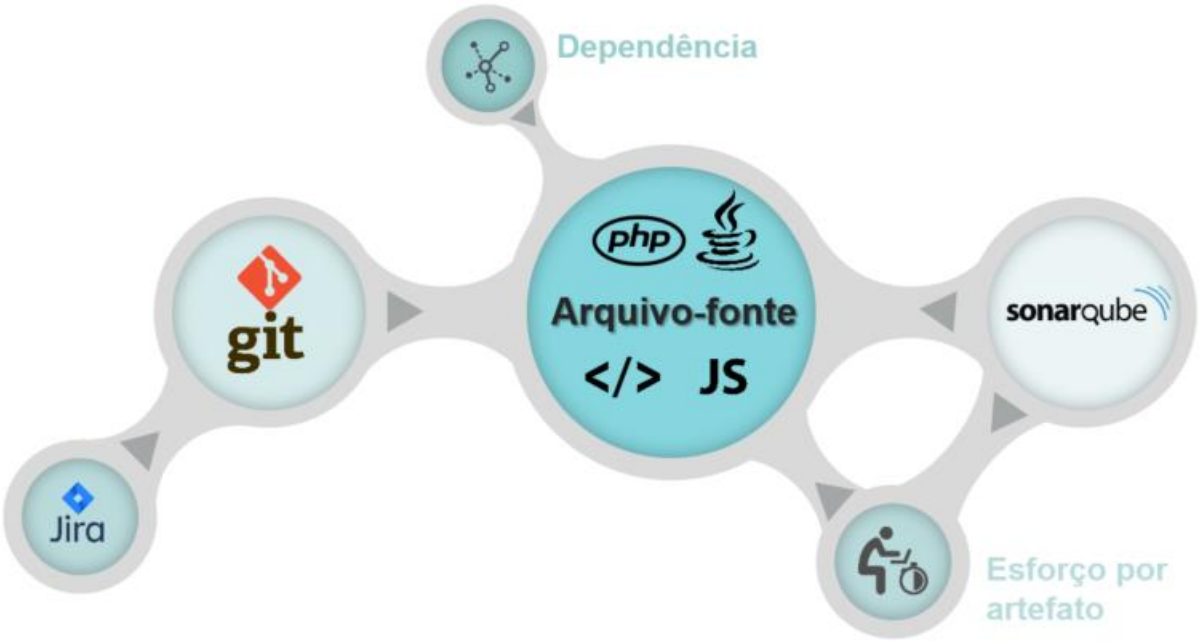


Figura 20. Estrutura básica da integração entre as ferramentas no 2º Ciclo. Fonte: o Autor.

A partir do entendimento da integração entre as ferramentas utilizadas na gestão do desenvolvimento de software, foi possível identificar as informações que poderiam ser extraídas para ajudar na priorização da Dívida Técnica. Os dados foram classificados em 4 características:

- **Histórico de alteração**: a partir das informações extraídas do Jira com o Git foi possível identificar os históricos das alterações realizadas no arquivo-fonte. O histórico das alterações é classificado por tipo, origem e data da alteração. O tipo do histórico da alteração foi dividido em “defeito” (*bug*) ou “evolução” do software, a origem foi classificada em “cliente” ou demanda “interna”. Para as alterações do tipo igual a “Defeito” foi analisado se a origem é de cliente (quando o defeito foi identificado pelo cliente) ou se a origem é interna (para os defeitos identificados nos procedimentos da empresa). Para as alterações motivadas pela evolução do produto, não houve separação por origem. Essa a decisão de não separar as evoluções por origem foi uma decisão interna;
- **Análise de impacto**: para a análise de impacto foram utilizadas duas métricas: o tamanho do arquivo-fonte (representado pela quantidade de linhas de código-fonte) e a quantidade de dependência de cada arquivo-

fonte (a quantidade de pontos no sistema que tem uma referência ao arquivo-fonte de origem);

- **Projeto**: cada equipe e cada projeto possuem responsabilidades e prioridades diferentes. Desta maneira, a equipe responsável e o nome do projeto que o arquivo-fonte faz parte são as informações mais relevantes e que atendem à necessidade da empresa;
- **Dívida técnica**: as informações da Dívida Técnica que o arquivo-fonte possui apresentam a distância que o código-fonte está do ideal para a empresa. Os atributos utilizados para analisar esta característica foram a quantidade de itens de Dívida Técnica por prioridade, o total de esforço necessário para eliminar todos os itens de Dívida Técnica do arquivo-fonte e uma classificação para representar os juros baseado na quantidade de alteração que o código fonte teve de alteração.

O Quadro 7 apresenta um resumo das características e dos atributos cujos dados foram possíveis de serem coletados para ajudar na priorização do arquivo-fonte.

Quadro 7. Características e atributos utilizados na priorização. Fonte: o Autor.

Arquivo-fonte	
Característica	**Atributo**
Projeto	Equipe
	Projeto
Análise de impacto	Tamanho
	Dependência
Histórico de alteração	Defeito (Cliente)
	Defeito (Interno)
	Todas as alterações
Dívida Técnica	Grave
	Alta
	Média
	Baixa
	Quantidade de IDT
	Esforço

As informações foram centralizadas no arquivo-fonte, conforme apresenta a Figura 21. No Apêndice E está descrito o modelo de dados utilizado para organizar as informações.

Para a avaliação e priorização dos arquivos-fonte, a empresa optou por dividir os arquivos-fonte por equipe. Essa foi a opção que a empresa encontrou para nomear um responsável para cada arquivo-fonte. A separação dos arquivos-fonte entre as equipes deixou explícito o universo de código-fonte que cada equipe é responsável.

As equipes têm a liberdade para a tomada de decisão, porém entre as equipes os níveis de maturidade e prioridades são diferentes. Desta maneira, os gestores ajudaram os responsáveis das equipes na definição dos objetivos para o pagamento da Dívida Técnica para o próximo período. Após realizar esta análise junto com as equipes, foi possível classificar as equipes em quatro cenários de priorização e identificar as características que poderiam ajudar na priorização dos arquivos-fonte:

- Cenário 1:
  - Contexto: projetos mais antigos e que possuem muita manutenção motivada por defeitos encontrados pelos clientes;
  - Objetivo: priorizar os arquivos-fonte que mais tiveram alterações motivadas por defeitos registrados pelos clientes;
  - Características: Projeto e Histórico de alteração;
- Cenário 2:
  - Contexto: projetos onde foram encontrados muitos problemas pela equipe de inspeção e testes;
  - Objetivo: priorizar os arquivos-fonte que mais tiveram alterações motivadas por defeitos registrados pela equipe de testes da empresa;
  - Características: Projeto e Histórico de alteração;
- Cenário 3:
  - Contexto: projetos de novos produtos ou produtos recentemente liberados para o mercado;
  - Objetivo: manter o ritmo de evolução do produto, priorizar os arquivos fontes com maior quantidade de itens de Dívida Técnica, priorizando os arquivos com dívidas de maior prioridade;
  - Características: Projeto e Dívida Técnica;
- Cenário 4:
  - Contexto: identificar possíveis arquivos-fonte que não são mais chamados pelas funcionalidades do produto;

- Objetivo: avaliar se o arquivo-fonte está sendo utilizado, usando como evidência se houve alguma alteração no arquivo-fonte ou se possui dependência de outros arquivos-fonte;
- Características: Projeto, Análise de impacto e Histórico de alteração;

O próximo passo da priorização foi criar um funil de priorização para cada um dos cenários, onde foram classificados em ordem de prioridade as características e os atributos. Para elaborar o funil de priorização, inicialmente foram ordenados os atributos do menos relevante para o mais relevante, e para cada um dos atributos foi realizada uma operação que pode ser de filtro ou ordenação das informações. Ao final das operações são apresentados os arquivos-fontes candidatos para o pagamento da Dívida Técnica.

Na Figura 21 é exemplificado o uso do funil de priorização para cada um dos 4 cenários de priorização. Em todos os cenários a primeira operação é o filtro por equipe, pois cada arquivo-fonte possui uma equipe responsável.

Os cenários de priorização 1 e 2 possuem as mesmas características e operações, porém devido aos objetivos de cada um dos cenários os atributos são diferentes, desta maneira os resultados são diferentes. O resultado do cenário 1 são os arquivos-fonte com maior quantidade de defeitos encontrados por clientes, no cenário 2, o resultado são os arquivos-fonte com maior quantidade de defeitos encontrados pela equipe de inspeção. Nestes dois cenários, a operação mais relevante é a ordenação decrescente da quantidade de defeitos. Ao final desta ordenação são apresentados os arquivos-fontes candidatos para a priorização em ordem crescente. No cenário 3 são apresentados os arquivos-fonte com a maior quantidade de itens de Dívida Técnica ordenados por criticidade. O resultado do cenário 4 são os possíveis arquivos-fontes não utilizados.

Para ajudar na visualização dos arquivos-fontes candidatos para pagar a Dívida Técnica, foi desenvolvido um “Portal de priorização” com os cenários e os filtros por atributos das características. O portal é divido em seis visões que fornecem as informações conforme a definição dos cenários.

Cenário 1 - Arquivos-fonte com mais defeitos de cliente			
#	**Característica**	**Operação**	**Atributo**
1	Projeto	Filter	Equipe = "nome da equipe"
2	Histórico de alteração	Filter	Defeito (Cliente) > 0
3	Histórico de alteração	Decrescent sort	Defeito (Cliente)

Cenário 2 - Arquivos-fonte com mais defeitos interno			
#	**Característica**	**Operação**	**Atributo**
1	Projeto	Filter	Equipe = "nome da equipe"
2	Histórico de alteração	Filter	Defeito (Interno) > 0
3	Histórico de alteração	Decrescent sort	Defeito (Interno)

Cenário 3 - Arquivo-fonte por severidade da Dívida Técnica			
#	**Característica**	**Operação**	**Atributo**
1	Projeto	Filter	Equipe = "nome da equipe"
2	Dívida Técnica	Decrescent sort	Baixa
3	Dívida Técnica	Decrescent sort	Media
4	Dívida Técnica	Decrescent sort	Alta
5	Dívida Técnica	Decrescent sort	Grave

Cenário 4 - Possível arquivo-fonte não utilizado			
#	**Característica**	**Operação**	**Atributo**
1	Projeto	Filter	Equipe = "nome da equipe"
2	Análise de impacto	Filter	Dependência = 0
3	Histórico de alteração	Filter	Todas as alterações = 0

Figura 21. Funil de priorização dos cenários. Fonte: o Autor.

A Figura 22 apresenta o Portal de priorização que foi disponibilizado para todas as equipes. Nesta imagem os arquivos-fonte estão filtrados para a equipe T02. Os resultados destas visões ajudam os responsáveis na elaboração das ações de pagamento da Dívida Técnica, conforme as necessidades da equipe.

A Figura 22 apresenta uma divisão numerada em seis partes, cada uma das partes representa uma visão, com as seguintes definições:

- Visão 1: apresenta o nome da equipe e um resumo da quantidade de código-fonte e os itens de Dívida Técnica por criticidade. Esta visão pode ser utilizada para realizar o filtro nas outras visões, ou seja, ao selecionar uma equipe nesta visão as outras visões são atualizadas para mostrar os resultados somente da equipe selecionada;

- Visão 2: esta visão apresenta os arquivos-fonte com a maior quantidade de defeitos internos, conforme sugerido no Cenário 2. A visão apresenta o nome e quantidade de defeitos encontrados pela equipe de testes no arquivo-fonte;
- Visão 3: esta visão apresenta os arquivos-fonte com a maior quantidade de defeitos encontrados pelos clientes, conforme identificado no Cenário 1. A visão apresenta o nome e quantidade de defeitos encontrados pelos clientes no arquivo-fonte;
- Visão 4: conforme identificado no Cenário 3, o resultado desta visão são os arquivos-fonte com maior quantidade de itens de Dívida Técnica por prioridade;
- Visão 5: esta visão apresenta os possíveis arquivos-fontes não utilizados. Esta é uma recomendação do Cenário 4, nesta visão mostrar os arquivos-fonte que não tiveram alteração e não possui dependências;
- Visão 6: esta visão apresenta um resumo das informações disponíveis do arquivo-fonte, o objetivo desta visão é fornecer informações para apoiar as equipes nos casos de dúvidas e falso-positivos;
- Todas as visões possuem o nome da equipe responsável, com base nesta informação as equipes podem filtrar somente os arquivos-fontes de sua responsabilidade;
- Nas visões 2, 3, 4 e 5 possuem a informação do Projeto que a equipe é responsável, pois os projetos mudam de prioridade.

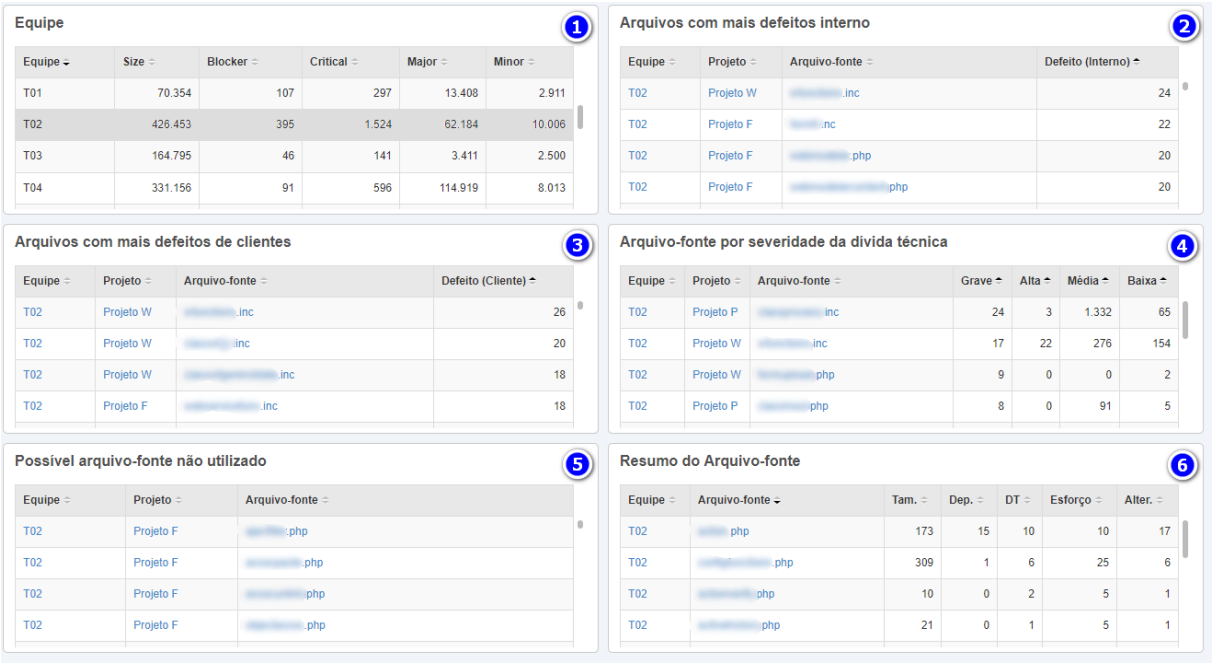


Figura 22. Portal de priorização dos arquivos-fonte. Fonte: o Autor.

As orientações e informações fornecidas nesta etapa, formam a base para a execução da primeira ação.

6.3 Implementação

Nesta etapa as 6 ações planejadas foram detalhadas e implementadas entre o período de janeiro a dezembro de 2019.

6.3.1 Ação 1: Elaborar e acompanhar as metas para pagamento da Dívida Técnica

As atividades anteriores de medição e priorização forneceram subsídios para as equipes definirem as ações de pagamento da Dívida Técnica. Assim, para elaborar as metas foram sugeridos três critérios:

- Critério 1: Cada equipe deveria definir um percentual de redução do total de Dívida Técnica, conforme sugerido no Cenário 3;
- Critério 2: Cada equipe deveria propor iniciativas para os arquivos-fonte que se destacaram nos Cenários 1 e 2 propostos no Portal de Priorização. Para os arquivos-fonte com a maior quantidade de defeitos encontrados, sugeriu-se das seguintes tarefas:
    - Refatorar o arquivo-fonte;

- Eliminar os itens de Dívida Técnica;
- Aumentar a cobertura de teste automatizado;
- Aumentar a cobertura de teste unitário;
- Documentar o código-fonte.

- Critério 3: As equipes deveriam analisar os arquivos-fontes sugeridos no Cenário 4 e propor uma redução na quantidade de código morto.

A partir dos critérios e das sugestões, cada equipe enviou para a aprovação da diretoria as ações de pagamento da Dívida Técnica para o período de 2019 (2º Ciclo). Os compromissos firmados entre a equipe e a diretoria foram atrelados aos indicadores de desempenho da equipe, para avaliação no programa de participação dos lucros da empresa.

Para exemplificar utilizou-se as metas definidas pela equipe T08, seguindo os critérios estabelecidos acima:

- Critério 1: para atender a este critério, a equipe T08 definiu o % de redução por prioridade da Dívida Técnica, conforme apresentado na Tabela 13. Para a prioridade Grave, a meta era pagar 60% de todos os itens de Dívida Técnica identificados. Nesse caso, dos 395 itens de Dívida Técnica identificados, a equipe pretendia eliminar 237. Da mesma maneira foi definido um percentual de redução para cada uma das prioridades.

Tabela 13. Percentual de redução da Dívida Técnica da equipe T08. Fonte: o Autor.

Prioridade	Atual	Pagar	% de redução
Grave	395	237	60,00%
Alta	1.524	610	40,00%
Média	62.184	15.546	25,00%
Baixa	10.006	2002	20,00%

- Critério 2: conforme os resultados dos Cenários 1 e 2 foram selecionados os principais arquivos-fonte para as ações de refatoração, documentação, cobertura de testes e eliminação dos principais itens de Dívida Técnica. A Figura 23 ilustra os arquivos-fonte selecionados e as ações propostas para os arquivos para os Cenários 1 e 2 da equipe T08

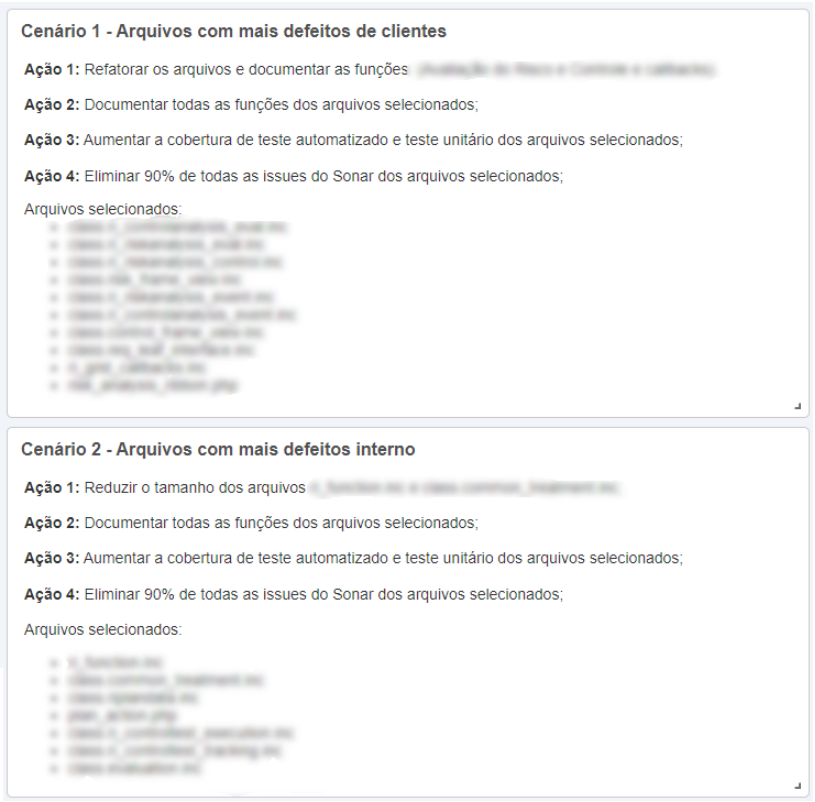


Figura 23. Priorização da equipe T08. Fonte: o Autor.

- Critério 3: a visão que representa o Cenário 4 apresentou 136 arquivos-fonte, da equipe T08, que poderiam não estar sendo utilizados pelo produto. A equipe entendeu que 90 arquivos-fonte não são mais utilizados ou foram substituídos por outros arquivos e não foram excluídos. Estes arquivos foram eliminados durante o período.

O monitoramento das metas foi quinzenal com as equipes. Essa ação foi realizada junto com a reunião de acompanhamento dos projetos com o Diretor da área.

6.3.2 Ação 2: KPIs para o acompanhamento dos resultados

Esta ação manteve os 3 KPIs selecionados no 1º Ciclo da pesquisa e utilizados para monitorar os resultados. Desta maneira, foi possível comparar os resultados obtidos nos dois ciclos da pesquisa.

6.3.3 Ação 3: Definir e divulgar o padrão de codificação em PHP

No 1º Ciclo, as ações de padronização foram direcionadas para separar os comandos SQL e o uso da global $_REQUEST do PHP. Neste 2º Ciclo, a ação foi

ampliada para definir um padrão de desenvolvimento e documentação do código-fonte.

A principal função de definir normas e padrão de desenvolvimento é melhorar a capacidade de comunicação entre os desenvolvedores, de maneira que possa existir mudanças de autor do código-fonte com menor transtorno, e manter os arquivos mais organizados e simples de ler e entender.

Para isso foi decidido o uso do padrão PSR (PHP *Standards Recommendation*), que são especificações de projetos propostos pelo PHP-FIG (PHP *Framework Interop Group*), que é atualmente o principal padrão utilizado no desenvolvimento em PHP e possui ferramentas de verificação do código-fonte que ajudam na adequação automática e no monitoramento do código-fonte. Para este projeto serão seguidas as recomendações dos padrões PSR-1 e PSR-2 disponíveis em https://www.php-fig.org/psr/.

Na empresa não é obrigatório o uso de editores de código-fonte, deixando para o desenvolvedor decidir qual editor utilizar. Porém, foi avaliado e sugerido o uso dos editores Sublime e VSCode, que possuem as funcionalidades de formatação automática do código-fonte conforme os padrões PSR-1 e PSR-2.

Para o monitoramento é utilizado o SonarQube com o plugin SonarPHP, que possui as regras de formatação conforme o padrão PSR. Para esta iniciativa não foi necessária a customização ou desenvolvimento de rotinas internas para atender aos padrões de codificação.

O repasse de conhecimento foi realizado por meio da documentação do padrão e de dois treinamentos internos para os desenvolvedores. Esta ação foi executada em janeiro de 2019.

6.3.4 Ação 4: Definir e divulgar o padrão de documentação do código-fonte em PHP

Após a implantação de práticas ágeis na empresa, a documentação do código-fonte vem sendo questionada, especialmente sobre o seu valor para o produto e para a equipe. Conforme os relatos dos desenvolvedores, principalmente dos desenvolvedores recém contratados, foram evidenciados que o código-fonte atual não possui um padrão de nomenclatura que possibilita uma compreensão rápida. Outra situação identificada pela equipe é a dificuldade de encontrar as rotinas já implementadas no sistema.

A proposta para essa iniciativa é a utilização do *DocBlock* (disponível em https://docs.phpdoc.org/) como referência para a documentação das funções, elementos do código-fonte, classes e métodos.

A partir da referência do *DocBlock* foi elaborada uma proposta mínima para documentar as funções e métodos no PHP, que deve ser composta por uma descrição e a *tag* (metadados para as ferramentas externas interpretarem o elemento) *@param* que é usado para documentar os argumentos da função ou método. Na Figura 24 é apresentado um exemplo de como deve ser a documentação das funções e métodos no PHP.

```
1  / **
2    * Este é o resumo de um DocBlock.
3    *
4    * Descrição para a função ou método. Este texto pode conter
5    * várias linhas.
6    *
7    * @param int $ exemplo Este é um exemplo de descrição de parâmetro de função
8    * @param string $ example2 Este é um segundo exemplo.
9    * /
```

Figura 24. Exemplo para documentar as funções e métodos. Fonte: o Autor.

Para a geração da documentação foi utilizado o *phpDocumentor*, ferramenta que gera documentação a partir do código-fonte PHP.

Após a definição do padrão de documentação, foi desenvolvida uma regra personalizada no SonarQube. A Figura 25 apresenta a regra implementada no SonarQube. Esta regra ajuda no monitoramento dos itens de Dívida Técnica de documentação, mantendo o padrão estabelecido pela empresa.

Figura 25. Regra de documentação personalizada. Fonte: o Autor.

Semelhante à ação anterior, o repasse de conhecimento foi realizado por meio da documentação do padrão e treinamentos internos para os desenvolvedores. Esta ação foi executada em janeiro de 2019.

6.3.5 Ação 5: Divulgação do projeto pela diretoria

A ação de divulgação pela diretoria teve um resultado positivo no 1º Ciclo da pesquisa e assim foi mantida esta ação neste ciclo. A divulgação foi realizada em janeiro de 2019 pelo Diretor da área, onde foram explicadas a importância do projeto, as metas das equipes para 2019 e as novas regras utilizadas para priorização da Dívida Técnica.

6.3.6 Ação 6: Questionário para avaliação do projeto

A guilda optou por elaborar um questionário para avaliar o projeto, que capturasse a opinião dos desenvolvedores de software que foram impactados por esta pesquisa. O questionário foi aplicado entre os dias 09 a 12/12/2019, onde foram coletadas 83 respostas dos profissionais da área de Desenvolvimento. Todos os respondentes do questionário foram de alguma maneira impactados pelo projeto durante este 2º Ciclo.

Na Figura 26 é apresentado o formulário enviado para 89 desenvolvedores. O questionário buscou avaliar os benefícios do projeto, o engajamento dos profissionais e o entendimento sobre a Dívida Técnica da empresa. O Apêndice F apresenta os resultados consolidados das 83 respostas.

O questionário possui 4 questões que utilizam o formato da escala Likert, na qual o respondente da pesquisa avalia o seu grau de concordância com a alternativa.

Para avaliar a questão aberta sobre os benefícios do projeto, os dados foram codificados para facilitar a análise. A categorização dos dados procurou seguir as características de qualidade sugeridas pela ISO/IEC 25010 (2011), também conhecida como SQuaRE (*Software Product Quality Requirements and Evaluation*). A lista utilizada para as características da qualidade foi:

- Compreensibilidade: legibilidade e compreensibilidade é parte da característica Usabilidade da SQuaRE. A usabilidade interna é utilizada para avaliar a extensão com que um software consegue ser entendido, aprendido, operado e atrativo para o usuário do código-fonte;

- Conformidade com padrões: está de acordo com padrões previamente estabelecidos pela equipe de desenvolvimento;
- Manutenibilidade: representa o grau de eficácia e eficiência na modificação do código-fonte;
- Redução de código morto: código-fonte que não é executado por nenhuma rotina do produto, baseado na sugestão do PriorTD;
- Redução de defeitos: representa a identificação de defeitos a partir dos resultados das análises sugeridas pelo PriorTD;
- Refatoração: processo de alteração do código-fonte que não altera as funcionalidades, porém melhora a estrutura interna. As sugestões de refatoração foram baseadas nas sugestões do PriorTD;
- Reusabilidade: representa a utilização de um código-fonte previamente desenvolvido em outras situações;
- Vulnerabilidade: permite o acesso não autorizado ou a modificação de programas ou dados.

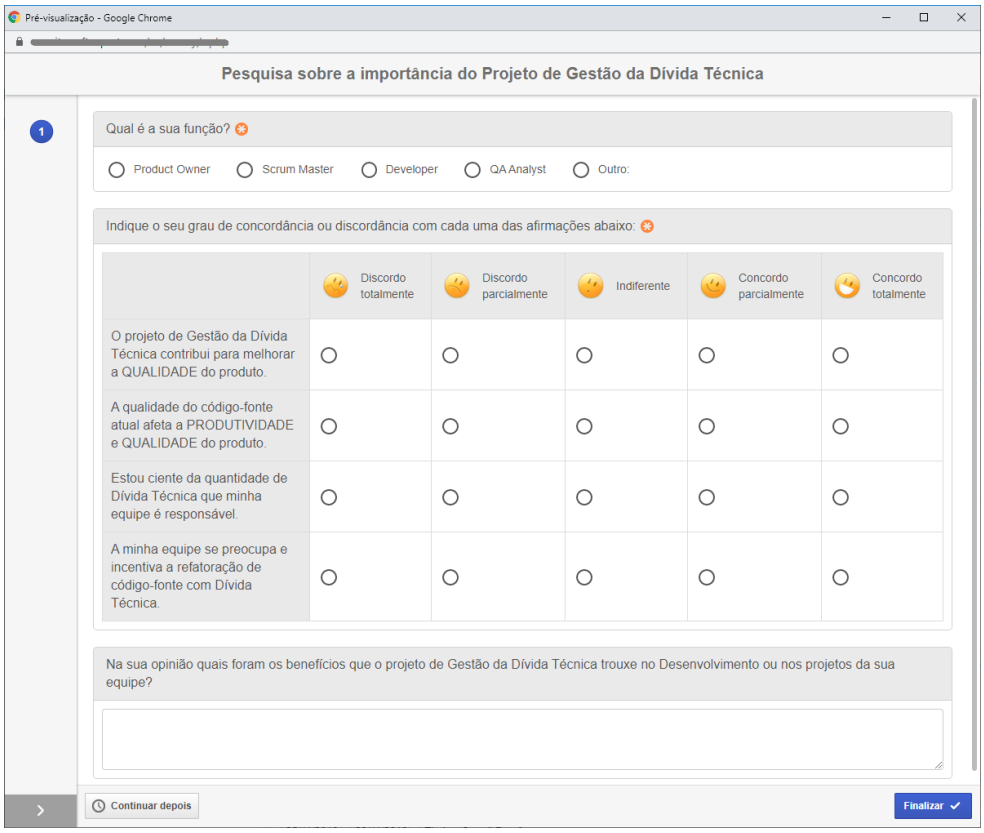


Figura 26. Formulário do questionário para avaliar o projeto de Gestão da Dívida Técnica. Fonte: o Autor.

6.4 Avaliação

As orientações de Rios; Mendonça e Spínola (2018) ajudaram na definição inicial do processo de desenvolvimento deste ciclo da pesquisa. Foram utilizadas as atividades de Identificação, Medição, Comunicação, Monitoramento, Priorização e Pagamento.

A avaliação deste ciclo foi dividida em três fases. Na primeira fase foram avaliadas a Coleta e análise de dados, o Planejamento e as ações implementadas e executadas até o início do período de monitoramento (de setembro de 2018 até janeiro de 2019). A segunda fase da avaliação apresenta os resultados coletados durante o período de monitoramento e no final da execução de todas as ações, que corresponde de fevereiro de 2019 até janeiro de 2020. Na terceira fase são avaliados os resultados como um todo e elaboradas as sugestões para continuidade do projeto na empresa.

Os resultados da avaliação contribuíram para a evolução do Processo de Gestão da Dívida Técnica e a criação do PriorTD, descritos nos capítulos 8 e 9.

6.4.1 Primeira fase da avaliação

Nesta fase foi realizada uma análise dos resultados extraídos antes de iniciar o período de monitoramento das ações de pagamento da Dívida Técnica. A maioria das atividades executadas seguiram os procedimentos do ciclo anterior. Porém, pode-se destacar a evolução do processo de Gestão da Dívida Técnica e o início da criação do PriorTD, que estão detalhados nos capítulos 7 e 8.

Muitas das práticas e procedimentos executados no ciclo anterior se repetiram neste ciclo, tais como:

- Guilda de Dívida Técnica: a guilda teve um papel fundamental para a Gestão da Dívida Técnica nos dois ciclos da pesquisa. A guilda esteve presente em todas as atividades de gestão e foi responsável pela criação de padrões e nas orientações repassadas as equipes;
- Levantamento das ferramentas: esta pesquisa utilizou diversas ferramentas para apoiar a Gestão da Dívida Técnica. Nos dois ciclos da pesquisa as ferramentas permitiram conhecer uma parte da Dívida Técnica da empresa e neste ciclo ajudaram na priorização. Apesar de não considerar uma atividade de gestão, o conhecimento das ferramentas pode ser fundamental para o sucesso da Gestão da Dívida Técnica;

- Medição da Dívida Técnica: nesta atividade é dada visibilidade aos itens de Dívida Técnica e qual o montante da Dívida Técnica. A medição da Dívida Técnica foi essencial para orientar na tomada de decisão e na definição das ações para o controle, eliminação ou diminuição da Dívida Técnica;
- Ação para elaborar e acompanhar as metas: nos dois ciclos, as equipes apresentaram as metas para a diretoria antes do período de monitoramento;
- KPIs para acompanhamentos das metas: os mesmos KPIs foram utilizados para avaliar os resultados de todas as ações para melhorar a qualidade do produto. Porém, não foi possível identificar um KPI que pudesse isolar os resultados específicos desta pesquisa;
- Divulgação do projeto pela diretoria: a divulgação do projeto para todos os envolvidos pela diretoria foi uma prática que obteve resultados no engajamento das equipes nos dois ciclos da pesquisa.

As atividades Identificação e Análise da Dívida Técnica foram uma sugestão a partir dos resultados do ciclo anterior. Na Identificação, buscou-se conhecer os tipos e as Dívidas Técnicas que faziam parte do projeto. Essa atividade foi considerada como crítica para o projeto, pois uma Dívida Técnica não identificada nesta fase não iria fazer parte das outras etapas do projeto. Na Análise, um dos resultados é a compreensão da Dívida Técnica a partir da definição da prioridade, complexidade, impacto e esforços. Esta fase se destacou por ajudar na estimativa dos possíveis resultados e dos impactos no sistema.

No Planejamento foram definidas 6 ações e nesta etapa foi possível utilizar os resultados da etapa de priorização para prever os recursos necessários.

A etapa de Priorização apresentou os arquivos-fonte candidatos para elaborar as ações de pagamento da Dívida Técnica, baseadas nas decisões das equipes. A priorização foi validada pelas lideranças das equipes e mostrou no uso prático a eficácia na escolha dos itens de Dívida Técnica mais relevantes para serem pagos.

Os resultados da priorização foram analisados por 5 SM e por 4 PO. A comunicação sobre as escolhas dos itens de Dívida Técnica para o pagamento foi facilitada utilizando o Portal de Priorização. A maneira que foi utilizada para referenciar as características com o arquivo-fonte, permitiu que houvesse uma melhora na comunicação entre a equipe técnica e os responsáveis pelo negócio, principalmente no momento de escolher quais artefatos seriam mais importantes para o projeto.

O método de priorização conseguiu se adequar aos objetivos de cada equipe, mostrando onde está ocorrendo o pagamento da Dívida Técnica.

Durante a avaliação dos resultados da priorização, o grupo responsável pela análise, identificou resultados falso-positivos. No ambiente do estudo os falso-positivos ocorreram pelos seguintes motivos:

- Arquitetura do software defasada: na arquitetura permite gerar chamadas dinâmicas de outros arquivos, desta maneira não foi possível identificar na análise de impacto dos arquivos-fonte;
- Procedimentos internos de atualização dos arquivos: alguns arquivos-fonte de rotinas de integração não possuem impactos, não são chamados por rotinas do produto e as últimas alterações foram há mais de 2 anos, porém são utilizados nas atualizações das versões;
- Informações inconsistentes nas bases de dados: a mudança de servidor e a reformulação para utilizar submódulos no Git geraram informações na base de dados do Git de alteração do código-fonte sem relevância.

Apesar da presença dos falso-positivos, a análise e seleção dos arquivos-fontes não foi prejudicada.

Na demonstração de uso do Painel de Priorização, a divisão por cenários aproximou a visão do negócio e a equipe técnica. Os gastos para pagar a Dívida Técnica tiveram uma ligação clara e de fácil compreensão para os *stakeholders*. Desta maneira, o procedimento utilizado na priorização apresentou visibilidade no pagamento dos itens de Dívida Técnica.

As 4 características utilizadas foram definidas com base nas informações disponíveis pelas ferramentas, porém durante o período de monitoramento foram identificadas outras caraterísticas como: *Backlog* e Cobertura de testes. Desta maneira, foi possível formar a base para a construção do PriorTD, que está descrito no Capítulo 8.

Na elaboração das ações para o pagamento da Dívida Técnica foram sugeridos alguns critérios para orientar as equipes na definição das metas. Cada equipe escolheu a quantidade e selecionou os arquivos-fontes que consideraram mais relevantes para a ações de melhoria, e as decisões foram alinhadas com os responsáveis da empresa. No momento do planejamento, as equipes se sentiram confortáveis e confiantes em atingir as metas para o pagamento da Dívida Técnica no período de janeiro a dezembro de 2019.

6.4.2 Segunda fase da avaliação

Na segunda fase da avaliação foram analisados os dados coletados durante o período de fevereiro de 2019 até janeiro de 2020. Neste período foram realizadas reuniões periódicas de acompanhamento das metas e dos KPIs. Em dezembro de 2019 foi executada a Ação 6, por meio da qual foram coletadas as opiniões dos envolvidos sobre o projeto.

As equipes foram responsáveis pelas atividades de Pagamento e Prevenção da Dívida Técnica. A Prevenção da Dívida Técnica foi realizada principalmente na atividade de revisão de código-fonte antes que a tarefa fosse concluída. O líder da equipe era o principal responsável por priorizar, monitorar a Dívida Técnica e acompanhar as metas estabelecidas para a equipe.

Cada equipe propôs ações de qualidade conforme a análise das informações fornecidas pelo PriorTD e das orientações para a implementação da Ação 1. No Apêndice G estão detalhadas cada uma das ações e o percentual de execução da ação no período. Nenhuma das equipes conseguiu finalizar todas as ações sugeridas no início do período; Ao todo 68,7% das ações foram finalizadas.

A segunda tabela do Apêndice G apresenta as metas e os resultados do pagamento dos itens de Dívida Técnica por equipe e por prioridade, propostos na implementação da Ação 1. Nenhuma das equipes conseguiu atingir 100% da meta de pagamento dos itens de Dívida Técnica. Porém, ao analisarmos resultados agrupados por prioridade, apresentados na Tabela 14, os itens de Dívida Técnica com prioridade “Grave” e “Média” atingiram a meta e os “Médio” e “Alta” tiveram uma diminuição de 15,36% e 13,13%, respectivamente. Estes resultados mostram que as equipes tiveram a preocupação em pagar os itens de Dívida Técnica, mas devido aos compromissos com entregas de novas funcionalidades e problemas internos, não foi possível atingir todas as metas estabelecidas.

Tabela 14. Resultados por prioridade do pagamento dos itens de Dívida Técnica. Fonte: o Autor.

Prioridade	Referência	Meta	Realizado	% Diminuição	Atingiu a meta?
Grave	2.066	985	666	67,76	SIM
Alta	9.026	6.274	7.640	15,36	NÃO
Média	650.533	349.002	299.642	53,94	SIM
Baixa	98.664	56.527	85.712	13,13	NÃO

Os resultados dos KPIs foram extraídos em janeiro de 2020. Conforme apresentado na Tabela 15, a versão 8 é o início do monitoramento do 1º Ciclo e a versão 12 é o início do 2º Ciclo. Estes KPIs consideram todas as iniciativas de qualidade da empresa, desta maneira não foi possível isolar os valores considerando somente este projeto de pesquisa.

Os resultados dos KPIs não apresentam melhora significativa, apesar da versão 15 ter valores positivos em comparação com as versões anteriores.

Tabela 15. Resultado dos KPIs do 2º Ciclo. Fonte: o Autor.

Início	Fim	Versão	KPI1	Bug Interno	Qtd. Cliente	KPI2	KPI3
mar/16	mai/16	1	571	2.310	687	0,83	80,18%
jun/16	ago/16	2	537	2.477	655	0,82	82,18%
set/16	nov/16	3	438	2.700	662	0,66	86,04%
dez/16	fev/17	4	614	2.605	678	0,91	80,93%
mar/17	mai/17	5	543	2.302	698	0,78	80,91%
jun/17	ago/17	6	602	1.966	696	0,86	76,56%
set/17	nov/17	7	557	2.017	717	0,78	78,36%
dez/17	fev/18	8	673	2.140	747	0,90	76,08%
mar/18	mai/18	9	616	2.027	781	0,79	76,69%
jun/18	ago/18	10	577	2.356	798	0,72	80,33%
set/18	nov/18	11	635	2.188	818	0,78	77,51%
dez/18	fev/19	12	847	2.332	834	1,02	73,36%
mar/19	mai/19	13	873	2.220	858	1,02	71,77%
jun/19	ago/19	14	710	2.269	887	0,80	76,17%
set/19	nov/19	15	324	2.175	917	0,35	87,03%

KPI1 (Quantidade de bugs registrados pelo cliente), KPI2 (Quantidade de bugs por cliente), KPI3 (Percentual de eficácia na detecção de defeitos)

Um dos fatores identificados pela estabilização dos KPIs, é que de janeiro de 2017 a dezembro de 2019 a empresa teve um aumento de aproximadamente 22% na carteira de clientes e 60% no faturamento total. Isso representa que neste período a empresa adquiriu contratos de valores mais expressivos e clientes com uso mais intenso do produto, ou seja, a empresa teve uso mais intenso do produto com mais usuários, porém a quantidade de clientes não aumentou significativamente. Isso levou o Diretor a refletir sobre os resultados destes indicadores como medição para os objetivos estratégicos.

A ação 6 foi outra maneira de avaliar os resultados da Gestão da Dívida Técnica sob a ótica dos envolvidos no projeto. Os resultados do questionário estão disponíveis

no Apêndice F. As questões fechadas tiveram altos valores de concordância, devido a isso não foi necessário avaliar os resultados por função.

A pesquisa aponta que aproximadamente 94% concordam que o projeto ajudou na melhoria da qualidade e que o código-fonte atual afeta a produtividade e a qualidade do produto.

Atualmente, existe muito código-fonte legado desenvolvido sob uma arquitetura com diversas anomalias tais como: dificuldades de reuso, forte acoplamento e falta de reponsabilidades entre as camadas da arquitetura. Os desenvolvedores entendem que é necessário evoluir o código-fonte, utilizando *frameworks* mais aderentes, melhorando as ferramentas de desenvolvimento e atualizando o código-fonte defasado. A Gestão da Dívida Técnica deve evoluir para questões de arquitetura e procurar trazer visibilidade de problemas estruturais do produto.

Percebeu-se que os desenvolvedores entenderam os impactos da Dívida Técnica e têm preocupação para refatorar o código-fonte, porém as cobranças para entregar novas funcionalidades, a falta de recursos e a arquitetura do código-fonte prejudicaram o pagamento da Dívida Técnica.

A Figura 27 apresenta o resultado após a codificação das respostas da questão sobre os benefícios do projeto, conforme as características da qualidade identificadas na Ação 6.

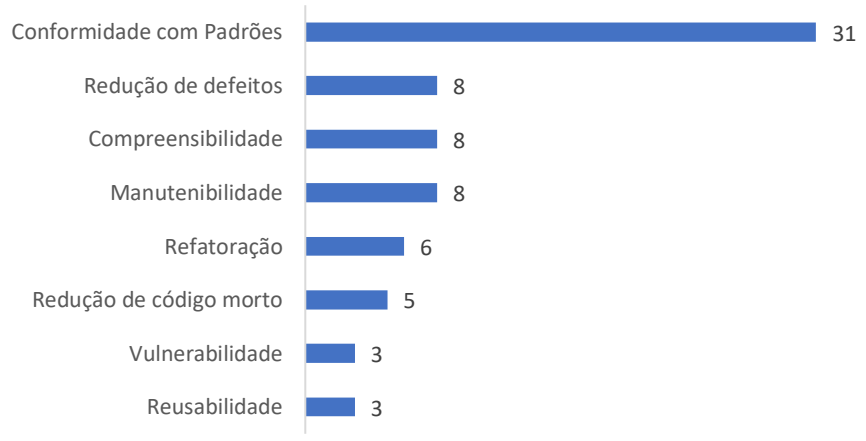


Figura 27. Resultado da codificação por característica de qualidade. Fonte: o Autor.

A característica mais citada pelos respondentes foi que o projeto trouxe a padronização do código-fonte e uso de melhores práticas, isso aconteceu devido às ações 3 e 4 de padronização e à visibilidade da Dívida Técnica de violação de padrão.

Estas ações ajudaram nas melhorias de compreensão e de manutenção do código-fonte, fato evidenciado em 16 respostas.

As características de Redução de defeitos, Refatoração e Redução de código morto foram resultados das ações a partir das sugestões do PriorTD.

A identificação de vulnerabilidade no sistema é resultado dos itens de Dívida Técnica identificados como Dívida Técnica de Vulnerabilidade de segurança. Os relatos de reuso do código podem ser um efeito sentido pelos desenvolvedores devido às mudanças provocadas pelo projeto, pois não foi planejada e implementada qualquer ação que tivesse impacto direto no reuso do código.

6.4.3 Terceira fase da avaliação

Nesta etapa da avaliação, os resultados e sugestões são decorrentes de uma análise holística do segundo ciclo.

Todas as decisões foram discutidas nas reuniões com a guilda, onde foram apresentados as propostas e os resultados alcançados. Essa abordagem ajudou no engajamento dos colaboradores, pois as decisões foram compartilhadas e aprovadas após obter o consenso dos envolvidos.

A partir dos resultados da Ação 6 conseguiu-se observar que os padrões de formatação e documentação do código-fonte ajudaram na comunicação entre os desenvolvedores e na busca por funcionalidades já implementadas no sistema.

Até este ciclo, não foi possível validar os resultados deste projeto na qualidade e produtividade do produto a partir dos KPIs definidos no início do projeto. Porém, foi identificado que os métodos de medição dos KPIs não estão aderentes com o crescimento da empresa. A diretoria entende que os resultados foram significativos e devido a isso foi aprovado a continuidade e evolução do projeto.

Neste ciclo as atividades e as operações realizadas na Gestão da Dívida Técnica ficaram mais evidentes. A partir deste ciclo foi possível descrever um processo de Gestão da Dívida Técnica, descrito no capítulo 8, para os próximos anos.

Neste ciclo foi possível montar a base do PriorTD, descrito no capítulo 9. A refatoração do código-fonte apontado pelo PriorTD teve resultado satisfatório nas equipes que conseguiram executar as ações de qualidade.

CAPÍTULO 7 - 3º CICLO

No terceiro ciclo da pesquisa seguiu-se as orientações e boas práticas do ciclo anterior, dando continuidade na Gestão da Dívida Técnica no contexto de estudo. Neste ciclo, manteve as ações de melhoria do código fonte PHP e criou medidas para gerenciar as Dívidas Técnicas de Teste, Teste Automatizado e *Build*. Os resultados deste ciclo contribuíram para a evolução do Processo de Gestão da Dívida Técnica considerando outros tipos de Dívida Técnica. Neste ciclo não teve alteração no método de priorização da Dívida Técnica definido no segundo ciclo da pesquisa, porém foi possível coletar mais dados para analisar os resultados do PriorTD.

7.1 Coleta e análise de dados

A maior parte das atividades desta etapa seguiu as orientações dos ciclos anteriores, com o mesmo objetivo de obter e analisar os dados para apoiar a tomada de decisão no pagamento da Dívida Técnica.

7.1.1 Guilda de Dívida Técnica

Seguindo as características do ciclo anterior, a guilda manteve as mesmas orientações e objetivos, e o grupo teve alterações na composição dos membros ativos.

Neste ciclo foi mantido o mesmo coordenador da guilda e dois membros do ciclo anterior, e teve mais 4 novos membros. Sendo, um total de sete membros ativos da guilda, divido em: um PO, três Analistas de testes, e três desenvolvedores com mais de 5 anos de experiência em desenvolvimento.

Os assuntos mais discutidos pela guilda de Dívida Técnica para este ciclo giraram em torno de duas questões principais:

i) O estado atual do planejamento, documentação e execução dos testes funcionais é ideal para o contexto?

ii) Os problemas de *build* estão afetando a produtividade das equipes?

As reuniões da guilda de Dívida Técnica para o planejamento do ciclo iniciaram em novembro de 2019 com período de implementação e pagamento da Dívida Técnica de janeiro a dezembro de 2020.

7.1.2 Ferramentas

Neste ciclo se manteve o uso das ferramentas do ciclo anterior e foram acrescentas as ferramentas utilizadas para a documentação de casos de teste e automatização dos testes funcionais, tais como:

- **TestLink**: software utilizado para documentar os casos de teste e a execução dos testes funcionais. A equipe de desenvolvimento de software possui 2.314 casos de testes documentados divididos em 6.392 passos, sendo 2.097 passos automatizados com execução diária e 4.295 passos manual executados a cada liberação de uma nova versão do produto;
- **Selenium WebDrive**: O Selenium WebDriver é um framework web utilizada para automatizar os testes de aplicativos baseados na Web. Os 2.097 passos documentados no TestLink foram automatizados a sua execução utilizando esta ferramenta.

7.1.3 Identificação de Dívida Técnica

Os membros da guilda entendem que anualmente deve-se atualizar a versão do SonarQube e rever a prioridade das regras de qualidade para o código-fonte. As regras foram revisadas conforme as orientações do ciclo anterior, desta maneira não houve discordância na seleção das regras.

Neste ciclo, 189 regras de qualidade foram revisadas, divididas em 181 regras fornecidas pelo SonarQube e oito regras adaptadas pelos membros da guilda. Destas, 149 regras foram aprovadas e classificadas por Prioridade e por Tipo de Dívida Técnica, conforme detalhado na Tabela 16.

Tabela 16. Quantidade de regras por Prioridade e Tipo de dívida. Fonte: o Autor.

		Regras
Prioridade	Não usado	40
	Baixa	19
	Média	32
	Alta	29
	Grave	69
Tipo de dívida	Dívida de Código	46
	Dívida de Defeito	26
	Dívida de *Design*	45
	Dívida de Documentação	3
	Dívida de Vulnerabilidade	29

7.1.4 Medição de Dívida Técnica

Esta etapa segue as orientações do ciclo anterior, onde foi realizado a medição inicial da Dívida Técnica identificada no código-fonte. A Tabela 17 apresenta a quantidade de itens de Dívida Técnica por equipe e por Prioridade. Da mesma maneira que nos dois ciclos anteriores, estas informações são utilizadas para gerar as metas das equipes para o pagamento da Dívida Técnica.

Tabela 17. Quantidade de itens de Dívida Técnica por Prioridade. Fonte: o Autor.

Equipe	Baixa	Média	Alta	Grave
T01	5.969	29.515	2.597	1.189
T02	10.126	65.815	5.976	3.516
T03	6.048	29.735	3.123	2.247
T04	5.464	21.462	1.966	1.358
T05	4.852	17.053	1.611	924
T06	5.278	15.235	2.690	888
T07	2.055	10.903	1.045	423
T08	2.632	5.240	557	359
T09	2.092	1.291	22	216
T10	1.577	7.565	1.939	622
T11	1.078	5.922	662	211
T12	866	1.704	329	136
Total	**48.037**	**211.440**	**22.517**	**12.089**

Neste ciclo não foi apresentado o esforço total por Prioridade e por Equipe, pois no ciclo anterior foi realizado a medição do esforço por arquivo-fonte. Devido aos dados coletados para utilizar o PriorTD foi possível substituir a informação de esforço por Prioridade com informações mais detalhadas do arquivo-fonte.

7.2 Planejamento

Na etapa de planejamento a guilda definiu as ações e o período de monitoramento. Neste 3º Ciclo, o período para a implementação e execução das ações foi entre janeiro e dezembro de 2020.

O objetivo principal da Guilda de Dívida Técnica é elaborar ações para a Dívida de *Build* e Testes. A partir da avaliação das ações do ciclo anterior e da coleta e a análise de dados deste ciclo, a guilda planejou 7 ações:

1. Elaborar as metas por equipe para o pagamento da Dívida Técnica;
2. Definir KPIs para acompanhamento dos resultados das ações;
3. Divulgar o projeto e as metas de cada equipe pela diretoria

4. Revisar a documentação dos casos de testes;
5. Definir um padrão para o desenvolvimento de Teste Automatizado;
6. Monitorar a execução dos Testes Automatizados;
7. Identificar a Dívida de *Build*.

Todas as ações tiveram um representante da guilda como responsável. Das 7 ações definidas para este ciclo, as ações 1, 2 e 3 se mantiveram em todos os ciclos da pesquisa.

7.3 Implementação

Nesta etapa as 7 ações planejadas foram detalhadas e implementadas entre o período de janeiro a dezembro de 2020.

7.3.1 Ação 1: Elaborar e acompanhar as metas para pagamento da Dívida Técnica

Conforme aconteceu nos ciclos anteriores, a meta para o pagamento da Dívida Técnica foi definida pelas equipes.

Na

Tabela 18 os valores são a quantidade de itens de Dívida Técnica agrupados por Equipe e por Prioridade. A coluna Ref. representa o valor de referência coletado na etapa 7.1.4 e os valores da coluna Meta foram definidos pelas equipes. Cada meta representam a quantidade de itens de Dívida Técnica que as equipes desejam ter no final do ciclo. Por exemplo: a equipe T01 possui 5.969 itens de Dívida Técnica com prioridade Baixa e deseja ter 2.000 itens de Dívida Técnica no final do ciclo.

Equipe	Baixa		Média		Alta		Grave	
	Ref.	Meta	Ref.	Meta	Ref.	Meta	Ref.	Meta
T01	5.969	2.000	29.515	25.000	2.597	1.000	1.189	1.000
T02	10.126	9.261	65.815	51.375	5.976	3.435	3.516	2.692
T03	6.048	4.850	29.735	26.700	3.123	2.800	2.247	1.800
T04	5.464	5.000	21.462	21.000	1.966	1.500	1.358	1.000
T05	4.852	3.400	17.053	14.500	1.611	1.125	924	650
T06	5.278	5.000	15.235	13.000	2.690	2.000	888	600
T07	2.055	2.055	10.903	9.812	1.045	940	423	380
T08	2.632	1.500	5.240	4.000	557	100	359	300
T09	2.092	2.092	1.291	904	22	0	216	172
T10	1.577	1.200	7.565	6.500	1.939	1.600	622	550
T11	1.078	1.078	5.922	5.000	662	100	211	145
T12	866	690	1.704	1.500	329	165	136	81
Total	**48.037**	**38.126**	**211.440**	**179.291**	**22.517**	**14.765**	**12.089**	**9.370**

As equipes definiram as suas metas baseadas nos arquivo-fontes sugeridos pelo PriorTD na etapa anterior.

Equipe	Baixa		Média		Alta		Grave	
	Ref.	Meta	Ref.	Meta	Ref.	Meta	Ref.	Meta
T01	5.969	2.000	29.515	25.000	2.597	1.000	1.189	1.000
T02	10.126	9.261	65.815	51.375	5.976	3.435	3.516	2.692
T03	6.048	4.850	29.735	26.700	3.123	2.800	2.247	1.800
T04	5.464	5.000	21.462	21.000	1.966	1.500	1.358	1.000
T05	4.852	3.400	17.053	14.500	1.611	1.125	924	650
T06	5.278	5.000	15.235	13.000	2.690	2.000	888	600
T07	2.055	2.055	10.903	9.812	1.045	940	423	380
T08	2.632	1.500	5.240	4.000	557	100	359	300
T09	2.092	2.092	1.291	904	22	0	216	172
T10	1.577	1.200	7.565	6.500	1.939	1.600	622	550
T11	1.078	1.078	5.922	5.000	662	100	211	145
T12	866	690	1.704	1.500	329	165	136	81
Total	**48.037**	**38.126**	**211.440**	**179.291**	**22.517**	**14.765**	**12.089**	**9.370**

Tabela 18. Metas por equipe e por Prioridade da Dívida Técnica. Fonte: o Autor.

7.3.2 Ação 2: KPIs para o acompanhamento dos resultados

Esta ação manteve os 3 KPIs selecionados no 1º Ciclo e no 2º Ciclo da pesquisa para monitorar os resultados. Desta maneira, é possível comparar os resultados obtidos nos três ciclos da pesquisa.

7.3.3 Ação 3: Divulgação do projeto pela diretoria

A ação de divulgação pela diretoria teve um resultado positivo nos dois ciclos da pesquisa. A divulgação foi realizada em janeiro de 2020 pelo Diretor da área, onde

foram explicadas as ações da Guilda de Dívida Técnica para a Dívida de *Build* e Testes. Na reunião foi divulgado as metas das equipes para 2020.

7.3.4 Ação 4: Revisar a documentação dos casos de testes

Esta ação visa revisar a descrição dos casos de teste executados manualmente ou automaticamente. Essa ação foi realizada por 12 PO e 13 *Testers*, onde foram revisados 2.314 casos de teste, dos quais 2.097 casos de testes são executados automaticamente e 4.295 casos de testes são executadas manualmente. A execução automática é realizada diariamente e a execução manual é realizada trimestralmente, a cada lançamento de uma nova versão do produto.

A ferramenta TestLink foi utilizada para registrar e revisar os casos de teste. De acordo com a orientação da Guilda de Dívida Técnica, os revisores deveriam responder às seguintes perguntas sobre os casos de teste de seu projeto:

- Os casos de teste documentados são adequados para o projeto?
- Os requisitos mais críticos do projeto têm casos de teste planejados?
- Os casos de teste são atualizados na ferramenta de gerenciamento de teste de software (TestLink)?
- Todos os casos de teste têm título, objetivo, ação, passos e os resultados esperados?
- Os resultados dos casos de teste podem ser validados?
- Os casos de teste têm um objetivo bem definido?

7.3.5 Ação 5: Definir um padrão para o desenvolvimento de Teste Automatizado

A necessidade de definir um padrão para desenvolvimento de scripts de testes automatizados foi identificada pela guilda a partir da desmotivação dos desenvolvedores em criar testes automatizados. A guilda discutiu esse problema junto com alguns desenvolvedores e líderes de equipe, e eles identificaram as seguintes causas:

- Os scripts de teste automatizados possuem muitas linhas de código;
- Existe muito código desatualizado e redundante;
- Muitas falhas relatadas pelos testes automatizados são devido a problemas nos ambientes de execução dos testes, bancos de dados e scripts de teste desatualizado.
- Falta de visibilidade dos resultados da automação de teste.

A guilda identificou que as ferramentas utilizadas para desenvolver e executar os testes automatizados estão de acordo com as necessidades da empresa.

A partir das causas identificadas, a guilda TD desenvolveu um projeto piloto com uma equipe de desenvolvimento. Este projeto desenvolveu 96 scripts de testes automatizados para ser utilizado como referência para o desenvolvimento de novos testes automatizados.

A guilda propôs e implementou a prática de *Code Review* para os testes automatizados. A guilda desenvolveu uma lista de verificação para dar suporte ao *Code Review* dos testes automatizados, com as seguintes perguntas:

- O teste automatizado está documentado e possui um caso de teste e os passos de referência?
- O script do teste automatizado contém código desatualizado? (por exemplo: *Xpath*, *Sleep*, seletores não padrão)
- Nos testes de comparação de imagens, possui uma referência correta da imagem do modelo?
- O script teste automatizado é independente, pode ser executado separadamente?
- Os dados são mantidos na base de dados de testes como evidência em caso de falha?
- É gerada uma imagem de evidência do teste executado corretamente no final da execução do teste?

7.3.6 Ação 6: Monitorar a execução dos Testes Automatizados

As equipes destacaram a falta de uma ferramenta para monitorar a execução dos testes automatizados. A proposta implementada pela guilda foi o desenvolvimento de um *Dashboard* para monitorar o status dos testes automatizados pelas equipes. O monitoramento de testes automatizados fornece a todos os desenvolvedores e interessados uma visão do progresso dos ciclos de teste, resultados alcançados, falhas identificadas e métricas de execução de teste.

A guilda disponibilizou duas métricas para monitorar a execução do teste. A primeira métrica apresenta o status de execução do teste, agrupando os dados por dia. A partir dessa métrica, as equipes podem acompanhar a execução dos testes automatizados. A segunda métrica representa o número de casos de teste documentados com execução automática ou manual. Essa métrica ajuda as equipes

a planejar a disponibilização de recursos para desenvolver novos testes automatizados.

7.3.7 Ação 7: Identificar a Dívida de *Build*

A empresa tem dezenas de milhões de linhas de código-fonte com uma taxa de alteração de 1.850 *commits* por mês. Destacamos que as principais vantagens são que a empresa possui um guia para padronizar o desenvolvimento, um único repositório de código fonte, um único sistema de compilação para todos os projetos e uma única infraestrutura de testes.

Nesta ação, a guilda buscou identificar os tempos de *build*, a taxa de sucesso dos *builds* e quais serviços mais falham na execução do *build*.

Os dados foram extraídos do sistema de controle de versão, onde destacamos as seguintes informações, agrupadas por mês:

- Média de 1.010 *merge requests*;
- Média de 3.100 solicitação de *build*;
- A taxa de sucesso dos *builds*, que é a porcentagem de compilações executadas com sucesso, diminuiu de ~60% para ~30% (mais detalhes na Figura 7);
- O tempo médio de *build* aumentou de ~10 para ~35 minutos (mais detalhes na Figura 8);
- Dos 43 serviços realizados no *build*, foram identificados os cinco serviços que mais falham;
- No último mês do ciclo, as falhas de compilação consumiram 107.036 minutos de processamento.

Nesta ação a guilda procurou identificar a Dívida de *build* e apresentar os resultados as equipes responsáveis.

7.4 Avaliação

Neste ciclo, a Gestão da Dívida Técnica foi além da fronteira do código-fonte para buscar soluções para melhorar a qualidade interna do produto e reduzir os custos de manutenção em outros artefatos do produto, como casos de teste, scripts de teste automatizados e *Build* (Pipeline de

A avaliação deste ciclo foi dividida em quatro temas, tais como:

1. Avaliação dos resultados do pagamento da Dívida Técnica identificada no código-fonte, que corresponde as metas definidas na ação 1.
2. Avaliação dos resultados das ações 4 e 5, que é referente a Dívida de Teste e a Dívida Teste Automatizado.
3. Resultados obtidos após a identificação da Dívida de *Build.*
4. Avaliação consolidada dos 3 ciclos da pesquisa, a partir da análise dos KPI´s definidos para este estudo.

Os resultados deste ciclo contribuíram para a evolução do Processo de Gestão da Dívida Técnica, a consolidação da guilda de Dívida Técnica como instrumento para a Gestão da Dívida Técnica e análise mais detalhada dos resultados do PriorTD.

7.4.1 Meta da Dívida Técnica identificada no código-fonte

O Apêndice H apresenta as metas e os resultados do pagamento dos itens de Dívida Técnica por equipe e por prioridade, propostos na implementação da Ação 1.

Nenhuma das equipes conseguiu atingir 100% das metas de pagamento dos itens de Dívida Técnica. Porém, ao analisarmos resultados agrupados por prioridade, apresentados na Tabela 19, os itens de Dívida Técnica com prioridade “Alta” e “Média” atingiram a meta e os “Grave” e “Baixa” tiveram uma diminuição de 21,39% e 11,84%, respectivamente. Os resultados demostram que as equipes mantêm a preocupação em pagar os itens de Dívida Técnica.

Tabela 19. Resultados por prioridade do pagamento dos itens de Dívida Técnica. Fonte: o Autor.

Prioridade	Referência	Meta	Realizado	% Diminuição	Atingiu a meta?
Grave	12.089	9.370	9.503	21,39	NÃO
Alta	22.517	14.765	11.530	48,79	SIM
Média	211.440	179.291	149.206	29,43	SIM
Baixa	48.037	38.126	42.349	11,84	NÃO

As equipes seguiram as orientações do PriorTD para selecionar os arquivos-fontes para o pagamento da Dívida Técnica. Que fortaleceu o PriorTD como método de priorização do código-fonte mais relevante para o pagamento da Dívida Técnica.

7.4.2 Dívida de Teste

Os casos de teste foram revisados de acordo com as diretrizes repassadas pela guilda. Essa ação envolveu 12 POs e 13 testadores que revisaram 2.314 casos de teste e 6.342 etapas de teste. Dez profissionais definiram um padrão para desenvolver os testes automatizados, reescrevendo 1.657 scripts de teste compatíveis com o novo padrão.

Um dashboard com métricas automatizadas de execução de testes foi desenvolvido para auxiliar as equipes no monitoramento dos resultados. A Figura 28 mostra o status de execução de teste de todas as equipes, mas o painel também apresenta os dados por equipe. Este painel reflete o momento após a definição do padrão para desenvolvimento de testes automatizados (Ação 5). O gráfico ilustra os scripts de teste executados diariamente durante doze dias. Por exemplo, no dia 1, 1.659 casos de teste foram aprovados e 33 falharam.

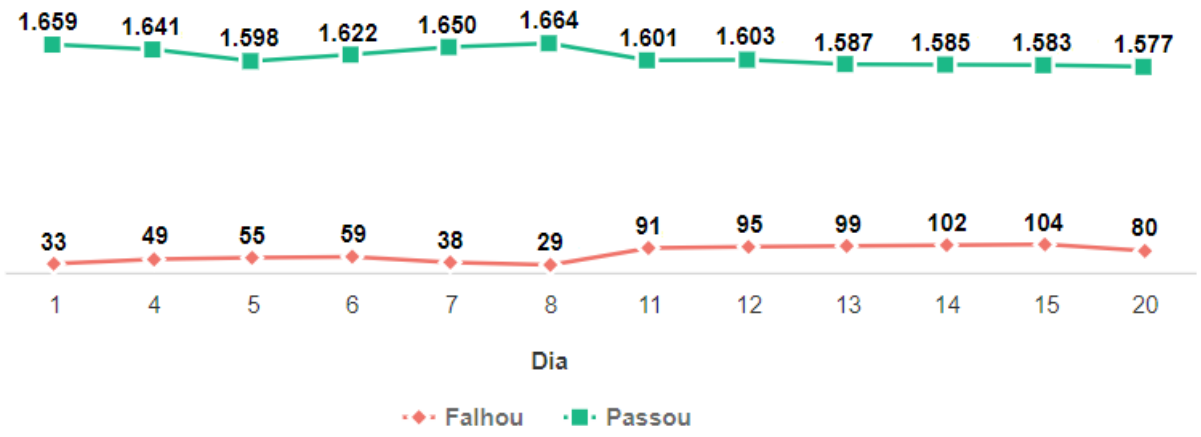


Figura 28. Execução dos testes por status e por dia. Fonte: o Autor.

A Figura 29 mostra o percentual do teste automatizado de todas as equipes, mas o painel também apresenta os dados por equipe. Após a revisão dos casos de teste, esses dados foram obtidos para rastrear o número de passos de teste que possuem execução automatizada e manual. Durante o 3º ciclo, os scripts de teste automatizados corresponderam a 1.735 (32,09%) dos passos de teste, restando ainda 3.671 (67,91%) passos de teste para serem automatizados.

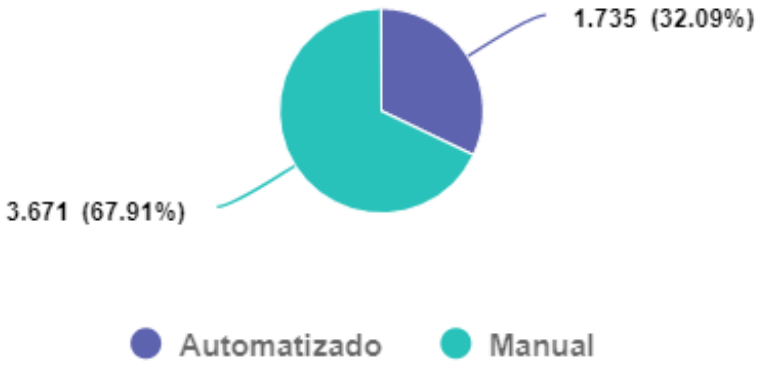


Figura 29. Quantidade de passos de testes automatizado e manual. Fonte: o Autor.

7.4.3 Dívida de *Build*

No procedimento de *Build*, a guilda de Dívida Técnica identificou a existência de Dívida de *Build*, devido ao tempo de compilação, a taxa de sucesso das compilações, e as falhas nos sistemas de *Build* que não estavam atendendo as necessidades da organização e estavam causando retrabalho, tanto para os *Testers* como para os Desenvolvedores. Nesta ação, o objetivo foi apresentar evidências quantitativas da existência de Dívida de *Build*.

Os dados dos gráficos apresentados na Figura 30 e Figura 31 foram extraídos dos últimos 22 meses e agrupados por mês. Esse período foi escolhido porque o número de *builds* solicitados por mês não foi inferior a 10% da média dos seis meses anteriores. Os meses foram selecionados até que a quantidade mensal de *builds* fosse superior a 933 solicitações de *build*. Esse procedimento foi escolhido para mitigar o risco de a quantidade de *builds* influenciar os resultados.

Cada vez que um *build* não é executado com sucesso, o *Tester* ou Desenvolvedor precisa solicitar o *build* novamente, causando desperdício de recursos. A Figura 30 apresenta a porcentagem de builds solicitados que foram concluídos com sucesso (por exemplo, sendo 200 *builds* solicitados, 125 builds foram executados com sucesso, o resultado é que 62,5% dos *builds* foram executados com sucesso).

Pode-se observar na Figura 30 que o percentual de *builds* executados com sucesso teve oscilações até o mês 17, com altos e baixos. Porém, nos últimos cinco meses a taxa de sucesso caíram muito abaixo dos períodos anteriores, e a taxa permanece em ~30% do total de builds executados com sucesso.

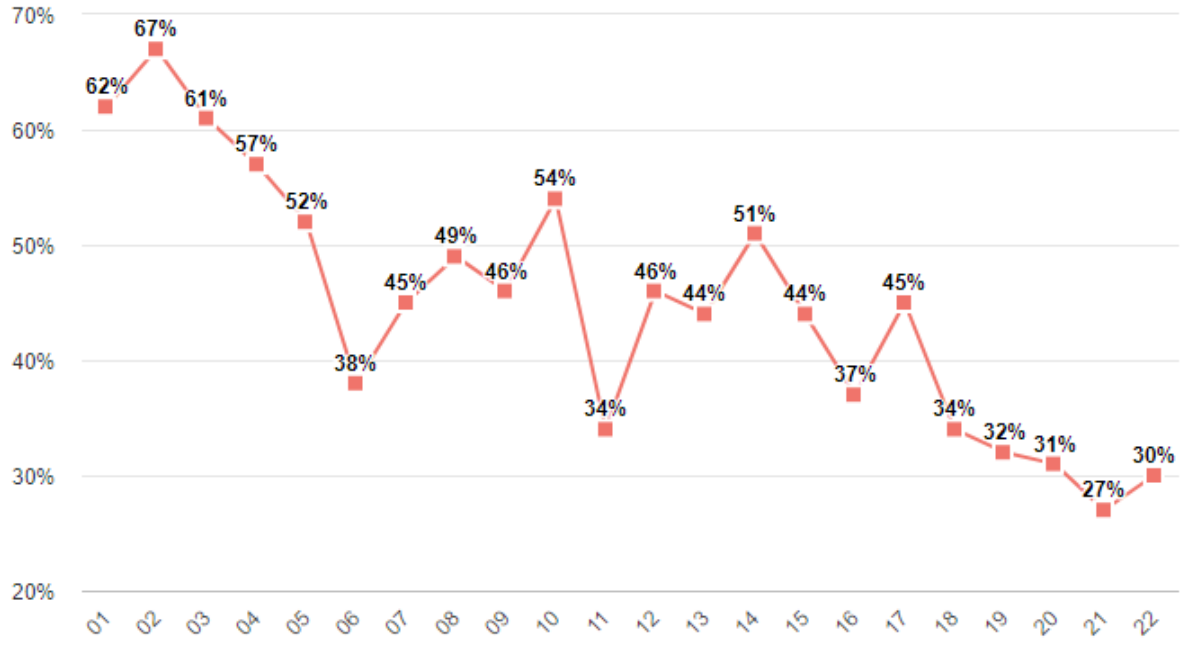


Figura 30. Percentual de *build*s executados com sucesso. Fonte: o Autor

A Figura 31 apresenta os tempos médios de processamento dos *builds* por mês (em minutos) ao longo de 22 meses. Visualmente é possível ver que o tempo médio de execução de um *build* aumentou. Nos primeiros cinco meses, a média foi de ~10 minutos. Nos últimos cinco meses, a média foi superior a 35 minutos.

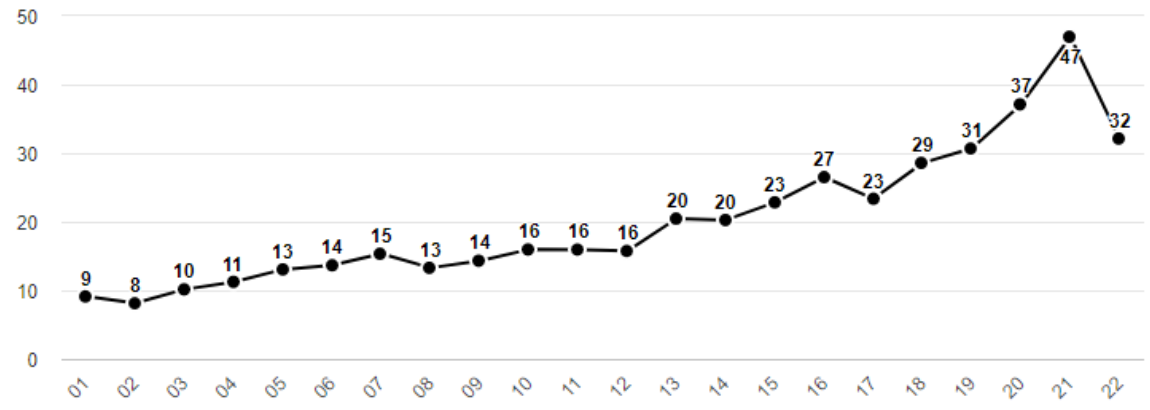


Figura 31. Tempo médio de execução de um *Build* por mês. Fonte: o Autor

Ao analisar os gráficos da Figura 30 e Figura 31, é possível observar a presença de Dívida de *Build*, pois apresentam evidências de que o retrabalho do Desenvolvedor e do *Tester* está aumentando, mesmo sem aumentar a demanda de *builds*. Os resultados de *builds* estão abaixo da meta desejada pela empresa. Os responsáveis concordaram que o valor ideal para a empresa deveria ser uma taxa de sucesso de

build superior a 60% e o tempo médio de *build* por mês inferior a 15 minutos. A equipe de DevOps será responsável por implementar medidas para o pagamento da Dívida de *Build*.

7.4.4 KPI´s

A Tabela 20 apresenta os resultados dos KPIs durante toda a pesquisa. O 1º ciclo foi da versão 8 à versão 11, o 2º ciclo foi da versão 12 à versão 15 e o 3º ciclo foi da versão 16 a versão 19. Outras versões são apresentadas para fins de uma visão longitudinal. Embora esses KPIs considerem todas as iniciativas de qualidade da empresa, durante todo o período de estudo não houve mudanças significativas no processo de desenvolvimento e gestão de projetos, mas sim a aplicação da abordagem de Gestão da Dívida Técnica e do PriorTD sugeridos por esta pesquisa.

A análise dos KPIs é por versão do produto, portanto todos os clientes devem estar utilizando a mesma versão em estudo. Este cenário só é possível após nove meses da versão lançada, pois este é o tempo necessário para que todos os clientes tenham atualizado o produto para as versões mais recentes

Tabela 20. Resultado dos KPIs do 3º Ciclo. Fonte: o Autor.

Início	Fim	Versão	KPI1	Bug Interno	Qtd. Cliente	KPI2	KPI3
mar/16	mai/16	1	571	2.310	687	0,83	80,18%
jun/16	ago/16	2	537	2.477	655	0,82	82,18%
set/16	nov/16	3	438	2.700	662	0,66	86,04%
dez/16	fev/17	4	614	2.605	678	0,91	80,93%
mar/17	mai/17	5	543	2.302	698	0,78	80,91%
jun/17	ago/17	6	602	1.966	696	0,86	76,56%
set/17	nov/17	7	557	2.017	717	0,78	78,36%
dez/17	fev/18	8	673	2.140	747	0,90	76,08%
mar/18	mai/18	9	616	2.027	781	0,79	76,69%
jun/18	ago/18	10	577	2.356	798	0,72	80,33%
set/18	nov/18	11	635	2.188	818	0,78	77,51%
dez/18	fev/19	12	847	2.332	834	1,02	73,36%
mar/19	mai/19	13	873	2.220	858	1,02	71,77%
jun/19	ago/19	14	710	2.269	887	0,80	76,17%
set/19	nov/19	15	636	2.175	917	0,69	77,37%
dez/19	fev/20	16	618	2.145	943	0,66	77,63%
mar/20	mai/20	17	558	3.040	945	0,59	84,49%
jun/20	ago/20	18	478	1.877	976	0,49	79,70%
set/20	nov/20	19	582	2.051	1.005	0,58	77,90%
dez/20	fev/21	20	600	2.415	1.024	0,59	80,10%

KPI1 (Quantidade de bugs registrados pelo cliente), KPI2 (Quantidade de bugs por cliente), KPI3 (Percentual de eficácia na detecção de defeitos)

Conforme mostrado na Tabela 20, parece que o projeto pode ter tido um impacto negativo nas versões 12 e 13 devido ao grande número de alterações feitas pelas equipes. No entanto, após estabilizar a versão, a empresa obteve resultados positivos significativos a partir da versão 15.

A partir do objetivo desta pesquisa de priorizar o pagamento da Dívida Técnica, a pesquisa buscava ajudar a organização a atingir o objetivo estratégico de *"Elevar o nível de produtividade e qualidade no desenvolvimento e lançamento dos produtos"*.

As evidências apresentadas na Tabela 20, mostram que as abordagens sugeridas por este estudo melhoraram a qualidade interna e a qualidade externa do produto. Assim, podemos concluir que esta pesquisa contribuiu para melhorar os KPIs utilizados para medir o objetivo estratégico.

O KPI2 apresentou resultados positivos não alcançados anteriormente pela empresa desde o início da medição deste KPI em 2016. Ao fazer uma análise de tendência linear (utilizando o método dos mínimos quadráticos) do KPI2, conforme apresentado na Figura 32, a linha de tendência Linear está em queda, ou seja, apresenta uma tendência decrescente.

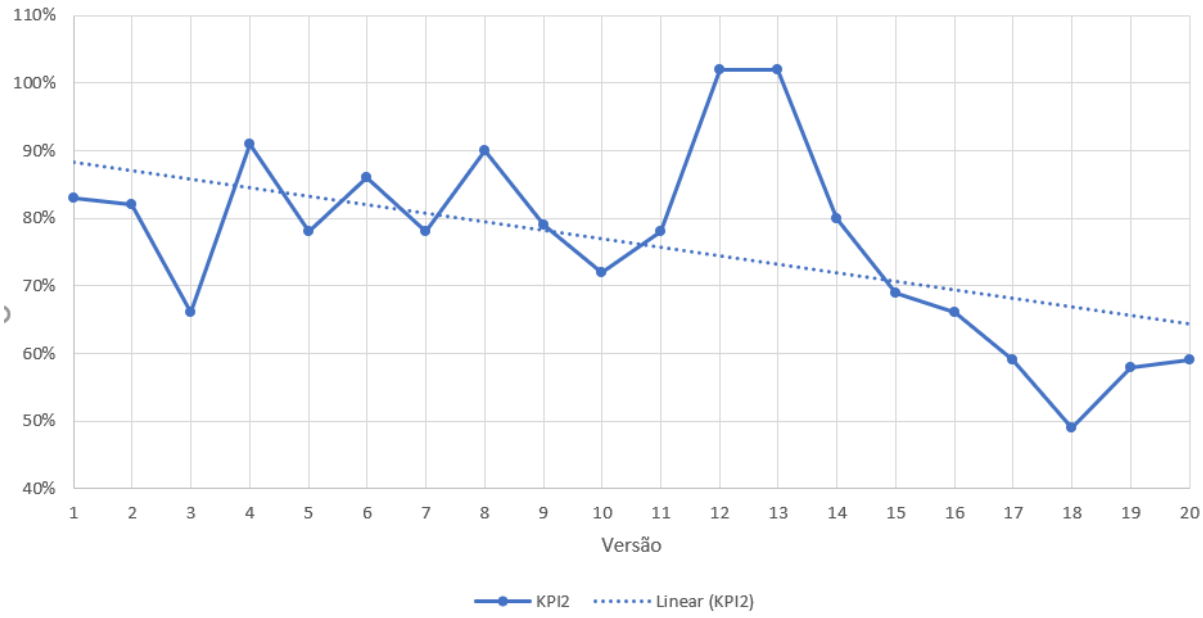


Figura 32. Análise de tendência do KPI2. Fonte: o Autor

Um dos fatores identificados no 2º ciclo da pesquisa que se manteve neste ciclo é que a quantidade de cliente e faturamento não estão aumentando na mesma proporção. Proporcionalmente, a empresa teve o faturamento acima do número de clientes, que caracteriza que os contratos têm valor mais expressivos e o uso mais intenso do produto, o qual afeta a análise do KPI2. Mesmo tendo conhecimento dessa informação os resultados são positivos.

Para o próximo ano, esta pesquisa apresenta as seguintes recomendações:

- Realizar uma revisão anual da Gestão da Dívida Técnica para evoluir e manter as melhores práticas implementadas;
- Mapear os impactos nas funcionalidades do produto devido às alterações no código-fonte, pois os resultados dos KPIs nas versões 12 e 13 evidenciam que os testes internos não contemplaram todas as funcionalidades impactadas pelas alterações no código-fonte;
- Utilizar o PriorTD para recomendar as refatorações do código-fonte;
- Tratar a refatoração como parte integrante do processo de desenvolvimento de software;
- Promover a refatoração contínua e incremental. Evitar de uma só vez grandes esforço em refatoração, pois as equipes possuem diversas responsabilidades e podem não atingir a expectativa;
- Desenvolver diretrizes de *Design*, código e teste para um software de alta qualidade;
- Implementar ferramentas de visualização da Dívida Técnica no momento da codificação;
- Elaborar programas internos e externos de treinamentos em arquitetura e *Design*, para que os desenvolvedores sejam treinados e ajudem na introdução da cultura de refatoração;
- Incorporar o pagamento da Dívida Técnica como parte do processo do de Desenvolvimento de Software.

CAPÍTULO 8 - RESULTADOS

Os resultados desta pesquisa podem ser divididos em três propostas principais, sendo o método de priorização denominado PriorTD, o processo de Gestão da Dívida Técnica e a Guilda de Dívida Técnica.

8.1 Guilda de Dívida Técnica

A guilda de Dívida Técnica esteve presente em todas as atividades na Gestão da Dívida Técnica identificadas no código-fonte e foi responsável pela prevenção da Dívida Técnica, criando padrões e diretrizes para as equipes. A guilda também contribuiu para determinar a prioridade da Dívida Técnica alinhando com os objetivos da empresa.

A Dívida Técnica muitas vezes é incorrida porque as pessoas não conhecem. A guilda disseminou conhecimento sobre a Dívida Técnica e orientou os desenvolvedores nas melhores práticas e padrões de desenvolvimento. Além disso, ajudou a implantar ferramentas para verificar e monitorar o código-fonte, dificultando o desenvolvimento incorreto

Nos dois primeiros ciclos, a guilda esteve focada na Dívida Técnica identificada no código-fonte do PHP, porém no terceiro ciclo, as ações promovidas pela guilda atingiram outros artefatos de software, como casos de teste, *scripts* de teste automatizados e a *build*. A guilda promoveu ações em diferentes Tipos de Dívida: Dívida de automação de teste, Dívida de *build*, Dívida de código, Dívida de defeito, Dívida de *Design*, Dívida de documentação e Dívida de teste.

A constituição de uma guilda com reuniões periódicas foi a proposta mais aderente ao contexto da empresa. À medida que a empresa evoluiu na Gestão da Dívida Técnica, aumentava também a necessidade de um profissional ou equipe alocada responsável a atividades de Gestão da Dívida Técnica. Este trabalho levanta a questão sobre a falta de um profissional capacitado e dedicado no gerenciamento da Dívida Técnica nas organizações: um Gestor de Dívida Técnica.

Essas experiências podem ser úteis para outros profissionais e fornecer conhecimento prático para ajudar na pesquisa de guildas, Comunidades de Prática e

Gestão da Dívida Técnica. Desta maneira, este estudo apresenta os fatores de sucesso, os principais desafios e orientações para criação de uma guilda de Dívida Técnica.

8.1.1 Fatores de sucesso

Esta seção destaca os principais elementos que contribuíram para a implementação bem-sucedida das ações e continuidade da guilda de Dívida Técnica. Eles foram analisados durante as reuniões de retrospectivas realizadas pela equipe da guilda:

- Patrocínio da alta direção: o patrocínio do diretor da área contribuiu positivamente para o engajamento dos integrantes e das equipes. Os membros sentiram-se motivados a participar, sabendo que as ações sugeridas tinham apoio da organização. As equipes aderiram às mudanças porque sabiam que as atividades estavam alinhadas com a visão da diretoria;
- Ferramentas de suporte: As ferramentas utilizadas foram fundamentais para dar visibilidade e transparência a Dívida Técnica. Por exemplo, usamos os dados fornecidos pela SonarQube em ferramentas de Data Analytics para monitorar e rastrear as ações de pagamentos da Dívida Técnica;
- Objetivos e orientações bem definidos: os objetivos e direcionamentos estabelecidos nas primeiras reuniões da guilda delimitaram o escopo dos assuntos e as tarefas estavam alinhadas às necessidades da organização;
- Equipe qualificada: a guilda de Dívida Técnica passava confiança para as equipes e os *stakeholders*, uma vez que a equipe era composta por técnicos experientes e profissionais de referência nas suas funções;
- Alinhamento com os objetivos estratégicos: todas as ações foram alinhadas com os objetivos estratégicos da organização;
- Resultados visíveis: o engajamento da guilda se deu principalmente por ver as ações sugeridas gerando valor para a organização e conhecimento para os membros.

8.1.2 Principais desafios

Durante o período deste estudo, a guilda de Dívida Técnica foi criada e obteve reconhecimento da organização, mas, ao mesmo tempo, enfrentou vários desafios. Destacam-se:

- Alinhar as necessidades dos membros com as necessidades da organização para gerar valor para ambos é um desafio constante na guilda. Esse desafio foi mitigado com o alinhamento antecipadamente dos objetivos e das diretrizes da guilda com o patrocinador e com os membros;
- Ao sugerir as ações de mudanças no código-fonte, os membros da guilda encontraram um ambiente complexo, com uma grande quantidade de código-fonte e uma taxa de alteração diária significativa. A guilda tinha habilidades técnicas para analisar detalhadamente o ambiente e propor soluções viáveis para superar esse desafio. A guilda também procurou comunicar os propósitos e os resultados esperados das mudanças de forma clara e permanente para conseguir o engajamento das equipes nas ações de mudança de código-fonte;
- Os padrões sugeridos pela guilda afetaram as características individuais relacionadas ao desenvolvimento de software. Os desenvolvedores tinham seus hábitos e padrões de codificação, mas com a guilda os padrões exigiram uma mudança na cultura no desenvolvimento de software;
- Em um ambiente organizacional onde os profissionais têm diversas atividades e compromissos, encontrar tempo para se dedicar às tarefas da guilda foi outro desafio. Este desafio foi mitigado por ser uma guilda patrocinada pelo diretor da área. Assim, as tarefas da guilda foram priorizadas e executadas em horário normal de trabalho;
- Para as ações que envolveram a análise de dados foi necessário obter permissão de visualização dos dados para outras áreas da organização. Esses acessos foram concedidos apenas a um membro da guilda responsável pela extração e divulgação dos dados;
- O patrocinador e as equipes reconheceram empiricamente que as ações promovidas pela guilda contribuíram para o desenvolvimento do software. No entanto, não foi possível avaliar quantitativamente os resultados na manutenção e evolução do software.

8.1.3 Orientações para criar uma guilda de Dívida Técnica

Neste capítulo são apresentas orientações para criar uma guilda de Dívida Técnica, dividias em três seções: Recomendações gerais, Reuniões de guilda e Ações de guilda.

Parte I – Recomendações gerais
Contexto: A guilda TD deve surgir dentro de um contexto organizacional alinhado aos objetivos estratégicos e às necessidades das equipes de desenvolvimento de software. Objetivo: Melhorar a qualidade interna e reduzir custos de manutenção e evolução de software. Desafio: Gerar valor para o produto e agregar conhecimento às equipes de desenvolvimento de software. Diretrizes: Desenvolver propósitos, objetivos e diretrizes para conduzir as reuniões e as ações alinhadas às expectativas da organização. Convite: O convite para a guilda de Dívida Técnica deve ser dirigido a todos os profissionais envolvidos nas atividades de manutenção e evolução do produto. Recomenda-se que o convite deve ser enviado pelo patrocinador da guilda. Patrocinador: Responsável por avaliar as ações propostas e aprovar e fornecer os recursos necessários para a execução das tarefas. Coordenador: O coordenador é responsável por organizar os assuntos e as reuniões, acompanhar a execução das tarefas, apoiar os membros da guilda e alinhar as necessidades com o patrocinador. Membro ativo: Um membro ativo é uma pessoa motivada que participa das reuniões e lidera as ações propostas pela guilda. Revisão: Os objetivos e as necessidades da guilda podem mudar ao longo do tempo. Assim, os membros da guilda devem revisar os objetivos e os procedimentos periodicamente. Recomendamos a revisão anual da guilda.

Parte II – Reuniões da guilda
Horário das reuniões: As reuniões da guilda podem ser mensais com duração de duas horas ou quinzenais com duração de uma hora. Discussão de ideias: Cada membro apresenta as ideias e os problemas aos quais a guilda deve ter atenção. Seleção das ações: As ações são selecionadas. Para cada ação, um membro da guilda é responsável por aprovar e implementar a ação. Definição do objetivo da ação: o objetivo da ação deve ser alinhado nas reuniões.

Parte III – Ações da guilda
Aprovação: O patrocinador aprova as ações para que o responsável possa priorizar essa tarefa com as demais demandas da equipe. Etapas/Execução da ação: lista os passos necessários para executar a ação e atingir as metas estabelecidas. Desjardins (2011) sugere que, para a execução da ação, considerar: a lista tarefas, os responsáveis, o apoio para os responsáveis, a comunicação com os interessados sobre a situação da ação, indicadores e orçamento, data de início e data de conclusão. Monitoramento: O progresso das ações em andamentos é apresentado e discutido durante as reuniões da guilda. Comunicação: Questões específicas sobre as ações podem ser discutidas em um canal de comunicação interno ou por e-mail após a reunião.

8.2 Processo de Gestão da Dívida Técnica

O processo de Gestão da Dívida Técnica, apresentado neste capítulo, é a sintetização e a formalização dos resultados obtidos nos três ciclos desta pesquisa.

Este processo fornece diretrizes e orientações para gerenciar a Dívida Técnica. A aplicação destas orientações e diretrizes podem ser personalizadas para outras organizações ou contextos.

A Gestão da Dívida Técnica baseia-se em propósitos, orientações e processos. Estes componentes podem ser implementados total ou parcialmente em uma organização, e podem ser adaptados ou melhorados, de forma que a Gestão da Dívida Técnicas seja eficiente, eficaz e consistente.

A Dívida Técnica é intrínseca à qualidade interna do software que sofre mudanças de manutenção e evolução a qualquer momento. Por isso necessita de monitoramento contínuo que independe da quantidade de Dívida Técnica existente.

Gerenciar a Dívida Técnica é um processo iterativo, que considera o contexto e as estratégicas para alcançar os objetivos. A cada nova iteração são avaliados os resultados, revisados os objetivos e a capacidade das equipes para então evoluir na Gestão da Dívida Técnica.

O propósito da Gestão da Dívida Técnica é melhorar a qualidade interna e diminuir os custos com manutenção e evolução do software. Esta pesquisa fornece algumas orientações para fazer uma Gestão da Dívida Técnica eficaz e eficiente:

- Comprometimento e apoio: a eficácia na Gestão da Dívida Técnica depende do comprometimento da equipe e do apoio das partes interessadas, em particular da alta direção;
- Integração ao processo de desenvolvimento de software: a Gestão da Dívida Técnica deve ser tratada como parte integrante no processo de desenvolvimento de software. Evite tratar a Gestão da Dívida Técnica como um projeto com prazo. O monitoramento deve ser contínuo;
- Competências da equipe: A Gestão da Dívida Técnica precisa de pessoas qualificadas, com conhecimento técnico e diversas habilidades para identificar a Dívida Técnica e orientar adequadamente as ações. Os melhores profissionais devem ser alocados para tais tarefas;
- Melhora na informação: a Gestão da Dívida Técnica se baseia em informações históricas, atuais e em expectativas do futuro. Convém que estas informações estejam disponíveis para todos e sejam extraídas de maneira clara, que todos os interessados entendam a origem e a transformação da informação;
- Personalizado: os objetivos para a Gestão da Dívida Técnica devem ser personalizados e aderentes ao contexto da organização;
- Alocação de recursos: convém que os responsáveis assegurem a alocação de recursos apropriados para o pagamento da Dívida Técnica. Isso significa pessoas habilitadas, ferramentas apropriadas, processos e procedimentos documentados;
- Treinamento e repasse de conhecimento: as mudanças de procedimentos e as novas orientações para o desenvolvimento de software, devem ser repassados a todos os envolvidos garantindo a capacidade técnica para a execução das atividades. Importante buscar treinamentos em padrões e boas práticas de desenvolvimento dentro do contexto da Gestão da Dívida Técnica.
- Dinâmico: a Dívida Técnica pode aparecer ou desaparecer à medida que o código-fonte ou o contexto sofram alterações. A Gestão da Dívida Técnica

deve estar atenta para detectar, reconhecer e responder a estas mudanças adequadamente;

- Mudança de cultura: a Gestão da Dívida Técnica pode provocar alterações na maneira de como o software é desenvolvido. Essas mudanças podem causar desconfortos inicial na equipe de desenvolvimento e precisam de apoio das partes interessadas;
- Melhoria contínua: o comportamento e a cultura dos desenvolvedores influenciam significativamente a melhoria contínua por meio da aprendizagem e experiências adquiridas no processo de Gestão da Dívida Técnica;

O processo de Gestão da Dívida Técnica envolve a aplicação sistemática de procedimentos e práticas para implementar uma Gestão da Dívida Técnica com sucesso. O processo deve ser visto como iterativo. Considerando o contexto e as estratégias para atingir os objetivos. A cada nova iteração, ou ciclo, os resultados são avaliados, os objetivos e a capacidade das equipes são revisados para evoluir na Gestão da Dívida Técnica. Este processo é ilustrado na Figura 33e detalhado nas próximas seções.

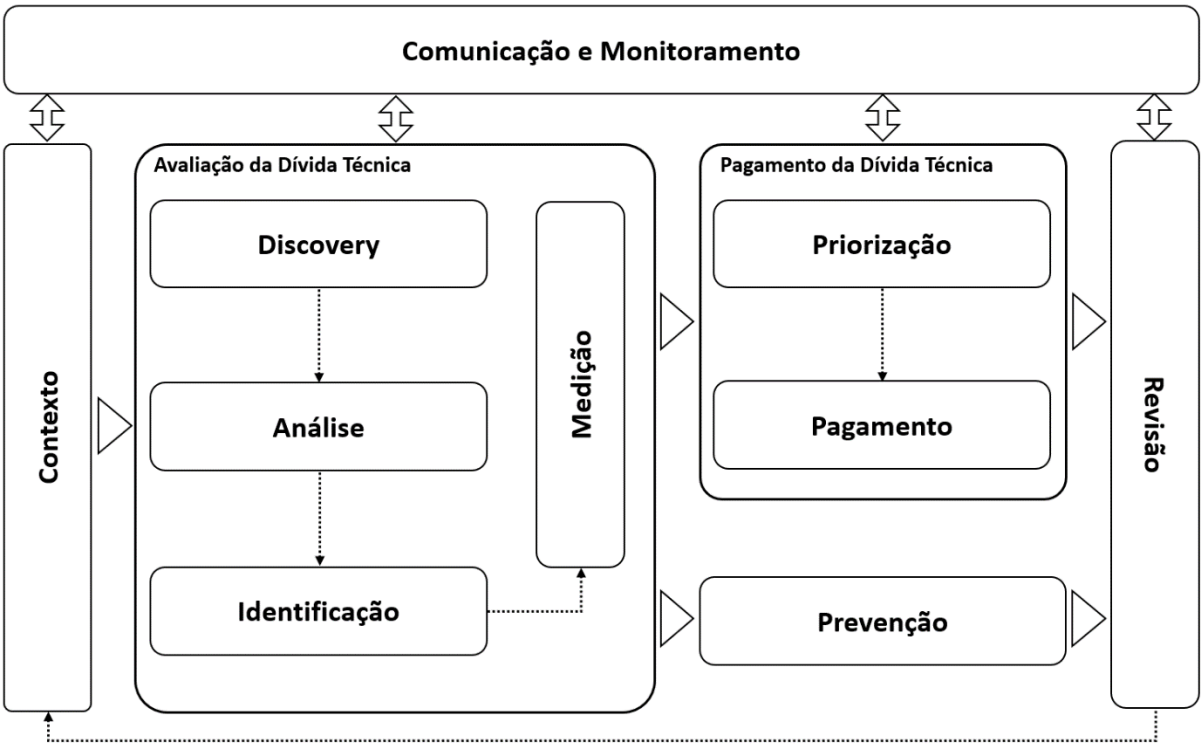


Figura 33. Processo de Gestão da Dívida Técnica. Fonte: o Autor.

8.2.1 Contexto

Atividade: Contexto

Descrição:

Na definição do contexto da Gestão da Dívida Técnica estão descritos os objetivos, o cenário utilizado, o tipo de Dívida Técnica que será gerenciada, as partes interessadas e os critérios para atingir os objetivos. Este entendimento é comunicado às partes interessadas para validar as informações. Convém que a Gestão da Dívida Técnica defina o escopo de suas atividades, pois o processo de gestão pode ser realizado sob diversos aspectos.

A Dívida Técnica pode ocorrer em vários artefatos ao longo do ciclo de vida do produto, tendo uma natureza distinta dependendo de quando é incorrida e das atividades às quais está associada (Rios, Mendonça e Spínola, 2018). Uma política de Gestão da Dívida Técnica deve ser baseada tanto no contexto de negócios de uma empresa quanto no ambiente tecnológico em que ela opera.

Tarefas:

- Definir os objetivos: os objetivos podem ter diferentes aspectos, como melhorar a qualidade interna ou externa do produto, diminuir o custo de desenvolvimento, diminuir o tempo de adaptação de profissionais recém-contratados, entre outros. Também pode ser aplicado a diferentes tecnologias;
- Definir uma estratégia: a organização deve elaborar sua estratégia para gerenciar a Dívida Técnica. Uma estratégia tem quatro pilares: missão, objetivo, estratégia e políticas (Ciancarini e Russo, 2020);
- Identificar as oportunidades e ameaças: para uma gestão eficaz da Dívida Técnica, é essencial que a tomada de decisão seja apoiada pelas oportunidades e ameaças externas e internas da Dívida Técnica, que estão relacionadas ao produto de software (Ciancarini e Russo, 2020);
- Identificar os resultados esperados: alinhar os resultados esperados com as expectativas das partes interessadas. É fundamental definir KPIs para monitorar os resultados durante o monitoramento da Gestão da Dívida Técnica;
- Selecionar a linguagem de programação: considerar os padrões e tecnologias inerentes à linguagem de programação que será gerenciada;

▪ Identificar o impacto nos projetos e processos existentes: identificar as equipes e projetos afetados e a necessidade de mudanças no processo de desenvolvimento de software; ▪ Definir ferramentas, recursos ou padrões: o contexto deve prever o estudo e implementação de novas ferramentas e padrões necessários para a Gestão da Dívida Técnica; ▪ Determine o período de cada ciclo: determine um período inicial e final para análise de desempenho e, em seguida, revise o processo para o próximo período; ▪ Identificar riscos: conhecer os riscos que podem afetar os objetivos e a tomada de decisões, prevendo estratégias para mitigá-los e gerenciá-los.
Saídas: Critérios para a Gestão da Dívida Técnica, Objetivos, Responsáveis, Lista dos Artefatos (e. g. código fonte PHP), Riscos, KPIs, Ferramentas, Metas, Estratégias e Planos de ação.
Papel responsável: Gestor da Dívida Técnica.
Outros papeis responsáveis: Patrocinadores, Gerentes e Líderes de equipes.

8.2.2 Avaliação da Dívida Técnica

O processo de avaliação da Dívida Técnica consiste nas atividades de *Discovery*, Análise, Identificação e Medição. Sugere-se que o processo de avaliação da Dívida Técnica ocorra de maneira sistemática e colaborativa com as partes interessadas.

8.2.2.1 *Discovery*

Atividade: *Discovery*
Descrição: O objetivo da atividade de *Discovery* é reconhecer o Tipo e a Dívida Técnica que são importantes para atingir os objetivos do contexto. A *Discovery* pode ocorrer em qualquer estágio do ciclo de vida do software. Esta é uma atividade crítica, uma vez que uma Dívida Técnica não é encontrada nesta atividade, não será incluído

nas atividades posteriores. Os resultados desta atividade devem estar alinhados com os objetivos e resultados esperados pelos *stakeholders*, conforme definido na atividade anterior. É fundamental nesta etapa contar com a experiência e habilidades da equipe de desenvolvimento para detectar as Dívidas Técnicas que fazem parte do contexto, independentemente se não forem identificadas automaticamente pelas ferramentas.
Tarefas: ▪ Escolha os tipos de Dívida Técnica: identificar quais os tipos de Dívida Técnica que podem estar relacionados aos artefatos do contexto; ▪ Descoberta da Dívida Técnica: a descoberta da Dívida Técnica é um procedimento para identificar as características dos artefatos que dificultam a evolução e manutenção do software e estão fora do padrão definido no contexto (Exemplo: *Design poor*, *performance*, documentação desatualizada). As ferramentas sugeridas para esta tarefa são reuniões, *brainstorming* e entrevistas com as partes interessadas e especialistas; ▪ Utilizar ferramentas automatizadas: é importante utilizar ferramentas automatizadas para coletar dados e informações que possibilitem a descoberta da Dívida Técnica e que sejam adequadas ao contexto. Por exemplo: o SonarQube possui diversas regras para identificar a Dívida Técnica, o uso deste tipo de ferramenta contribui para esta atividade.
Saídas: Lista das Dívidas Técnicas (Exemplo: Dívida de Código: Código duplicado, Violação do padrão de formatação; Dívida de *Design*: Código complexo, Violação do padrão de codificação).
Papel responsável: Gestor da Dívida Técnica.
Outros papeis responsáveis: Especialistas técnicos e de produtos.

8.2.2.2 Análise

Atividade: Análise
Descrição:

A atividade de Análise envolve o desenvolvimento de uma compreensão profunda de cada Dívida Técnica descoberta. A Análise da Dívida Técnica fornece as informações que servirão de base para a tomada de decisão e os níveis de detalhamento que serão utilizados na Identificação da Dívida Técnica.

Nesta etapa, para cada Dívida Técnica é registrado um conjunto de elementos e características que definem a Dívida Técnica.

Para entender as características da Dívida Técnica, considera-se o contexto de gestão para definir o objetivo da análise. A análise pode ser realizada com vários níveis de detalhamento e complexidade e pode ser qualitativa ou quantitativa, dependendo do objetivo da análise.

Na análise são compreendidas e documentadas as características comuns de cada Dívida Técnica, que serão utilizadas nas próximas atividades. As características comuns ajudam na gestão de grandes quantidades de itens de Dívida Técnica, pois nestes cenários requer muito esforço tornando-se até inviável a identificação das características particulares de cada Item de Dívida Técnica. As informações extraídas nesta atividade podem ajudar na criação de estratégias de prevenção da Dívida Técnica.

Nesta etapa é selecionado e validado as ferramentas que serão utilizadas na Gestão da Dívida Técnica.

Tarefas:

- Cadastro da Dívida Técnica: nesta tarefa deve-se complementar o cadastro da Dívida Técnica com informações relevantes que serão utilizadas nas próximas atividades. Destacam-se as seguintes informações: descrição detalhada da Dívida Técnica, exemplo de como identificar a Dívida Técnica; solução sugerida para a Dívida Técnica; artefatos ao qual a Dívida Técnica se aplica; parâmetros específicos (Exemplo: convenção, *suspicious, bad-practice*, limite = 20).
- Definir as características padrão:
    - Impacto: é uma classificação da importância da Dívida Técnica para o contexto;
    - Efeitos: identificar as implicações ou consequências negativas da Dívida Técnica. Estas informações podem auxiliar na priorização dos itens de Dívida Técnica;

○ Causa: identificar as razões ou motivos que levou a existência da Dívida Técnica; ○ Estimativa do valor principal: estimar o custo para eliminar um Item de Dívida Técnica. ▪ Aprovação de ferramentas: existem diversas ferramentas disponíveis para apoiar as atividades da Gestão da Dívida Técnica (Exemplo: Sonar Qube, Fixbugs, CAST, Jira). As ferramentas selecionadas devem passar por um processo de validação e aprovação, onde deve-se conhecer detalhadamente o propósito e os resultados que podem ser utilizados na Gestão da Dívida Técnica.
Saídas: Documentação da Dívida Técnica concluída. Lista de causas. Lista de efeitos ou consequências. Lista de impactos. Ferramentas aprovadas.
Papel responsável: Gestor da Dívida Técnica.
Outros papeis responsáveis: Especialistas técnicos e de produtos.

8.2.2.3 Identificação

Atividade: Identificação
Descrição: Os Itens de Dívida Técnica são identificados e documentados nesta etapa. As etapas anteriores fornecem uma compreensão da Dívida Técnica e do ambiente (por exemplo, onde os Itens do Dívida Técnica residem) estabelecidos quando o contexto foi considerado. Os resultados da atividade de Identificação da Dívida Técnica devem ser acessíveis, comunicados e validados com todos os *stakeholders*. Uma boa prática é usar ferramentas automatizadas de análise de código-fonte estático para encontrar os Itens de Dívida Técnica.
Tarefas: ▪ Identificar os Itens de Dívida Técnica: Nesta tarefa os Itens de Dívida Técnica são identificados e registrados. O Item de Dívida Técnica deve

ter uma referência do artefato onde foi encontrado. Os Itens de Dívida Técnica podem ser identificados manualmente ou automaticamente por ferramentas previamente aprovadas. ▪ Atualizar as características: incialmente o Item de Dívida Técnica recebe as características padrão definidas para a Dívida Técnica, estas características devem ser revisadas e atualizadas. ▪ Responsável: o Item de Dívida Técnica deve ter um responsável que tem o compromisso de monitorar ou eliminar o Item de Dívida Técnica, inicialmente é definido o responsável pelo Item da Dívida Técnica como sendo o mesmo responsável do artefato. Os responsáveis sugeridos para um Item de Dívida Técnica são: equipe, projeto, líder, desenvolvedor.
Saídas: Lista dos Itens de Dívida Técnica.
Papel responsável: Gestor da Dívida Técnica.
Outros papeis responsáveis: Responsável pelo artefato.

8.2.2.4 Medição

Atividade: Medição
Descrição: Na atividade de Medição da Dívida Técnica são determinados os valores da Dívida Técnica e os resultados planejados são transformados em metas de pagamento da Dívida Técnica. Nesta atividade os valores do principal e juros do Item da Dívida Técnica são revisados e estimados. A atividade de Medição busca gerar valores adequados às necessidades daqueles que tomam as decisões. As medições geradas nesta atividade podem ser utilizadas para avaliar o processo de Gestão da Dívida Técnica, pois é possível utilizar os valores do início do processo como base para avaliar a evolução no pagamento da Dívida Técnica.

As informações geradas nesta atividade são utilizadas na tomada de decisões, principalmente sobre quais Dívidas Técnica precisam de ações e iniciativas e como a Dívida Técnica será controlada e diminuída.
Tarefas: ▪ Estimativa do valor principal: a partir do valor principal sugerido na atividade de Análise, nesta tarefa é revisado o valor principal por item da Dívida Técnica. ▪ Estimativa dos juros: dada a dificuldade na obtenção de um valor preciso para os juros atuais ou juros previstos, a informação sobre os juros pode ser estimada. O valor estimado para os juros pode ser de maneira quantitativa, utilizando dados históricos do artefato para calcular um valor aproximado de manutenibilidade, ou qualitativa em forma de classificação (Exemplo: Muito alto, Alto, Médio, Baixo, Muito Baixo). ▪ Metas: nesta tarefa é definido as metas para a diminuição do montante da Dívida Técnica. Esta meta pode ser definida com base nos recursos disponibilizados para o pagamento da Dívida Técnica e o valor principal estimado.
Saídas: Atualização da lista dos Itens de Dívida Técnica, Metas.
Papel responsável: Gestor da Dívida Técnica.
Outros papeis responsáveis: Patrocinadores, Gerentes e Líderes de equipes.

8.2.3 Pagamento da Dívida Técnica

O processo de pagamento da Dívida Técnica consiste nas atividades de priorização e pagamento. Ele refere-se às atividades realizadas com o objetivo de apoiar a tomada de decisões sobre o momento mais adequado para eliminar os Itens da Dívida Técnica.

8.2.3.1 Priorização

Atividade: Priorização

Descrição: A priorização da Dívida Técnica trata de decidir quais Itens de Dívida Técnica devem ser reembolsados primeiro e quais Itens serão adiados até lançamentos posteriores (Li et al., 2015). A priorização deve fornecer informações para apoiar a tomada de decisão sobre qual Item Dívida Técnica deve ser pago. A saída desta atividade deve ser uma lista de Itens Dívida Técnica elegíveis para pagamento.
Tarefas: ▪ Priorização: organizar os Itens de Dívida Técnica em ordem de importância para o contexto. Importante nesta tarefa considerar os recursos disponíveis na seleção dos Itens de Dívida Técnica candidato para pagamento da Dívida Técnica.
Saídas: Lista dos Itens de Dívida Técnica para o pagamento.
Papel responsável: Gestor da Dívida Técnica.
Outros papeis responsáveis: Líderes de equipes e Responsável pelo artefato.

8.2.3.2 Pagamento

Atividade: Pagamento
Descrição: A atividade de Pagamento envolve a seleção e implementação de uma ou mais ações para alterar a situação atual da Dívida Técnica. A escolha das ações mais adequadas consiste em analisar os esforços, recursos e benefícios gerados para atingir os objetivos. As ações de pagamento da Dívida Técnica podem ser manuais ou automáticas, utilizando ferramentas aprovadas na atividade de Análise.
Tarefas: ▪ Eliminar Item de Dívida Técnica: para os Itens de Dívida Técnica priorizados são geradas tarefas onde é definido os recursos e o prazos para eliminá-los.

Saídas: Atualização da lista dos Itens de Dívida Técnica.
Papel responsável: Gestor da Dívida Técnica.
Outros papeis responsáveis: Patrocinadores, Gerentes e *Lìderes* de equipes.

8.2.4 Prevenção

Atividade: Prevenção
Descrição: A atividade de Prevenção da Dívida Técnica atua dentro do processo de desenvolvimento de software, com ações para evitar o aparecimento de Dívida Técnica. A prevenção refere-se a atividades cujo objetivo é prevenir a ocorrência de Dívida Técnica (Yli-Huumo et al., 2016). A Prevenção de Dívida Técnica pode ser vista como uma das atividades mais influentes do Gerenciamento da Dívida Técnica que uma equipe de desenvolvimento pode conduzir. Quando a equipe de desenvolvimento estabeleceu padrões de codificação obrigatórios, auxiliados, por exemplo, nas revisões de código e na prática de Definição de Pronto, a quantidade de Dívida Técnica que chega à base de código pode diminuir. Quando a Dívida Técnica é evitada o máximo possível, também ajuda outras atividades de Gestão de Dívida Técnica. Além disso, a criação de práticas de prevenção de Dívida Técnica ajuda a identificar as soluções “não tão boas” de desenvolvedores inexperientes (Yli-Huumo et al., 2016).
Tarefas: ▪ Identificar problemas em potenciais: revisar as atividades de desenvolvimento de Software, que fazem parte do contexto da Gestão da Dívida Técnica, para propor mudanças que evitem o aparecendo da Dívida Técnica; ▪ Criar ações preventivas: analisar as causas e criar ações para intervir o aparecimento de Dívida Técnica (Exemplo: Treinamento, ferramentas de codificação).

Saídas: Plano de ação, mudanças no processo de desenvolvimento de software.
Papel responsável: Gestor da Dívida Técnica.
Outros papeis responsáveis: Gerentes e *Lìderes* de equipes.

8.2.5 Revisão

Atividade: Revisão
Descrição: A atividade de Revisão tem como objetivo evoluir o processo de Gestão da Dívida Técnica. O processo de Gestão da Dívida Técnica deve ser revisto periodicamente, porém a necessidade de revisão do processo de gestão pode ser identificada durante a atividade de monitoramento e análise crítica. Após a atividade de Revisão, deve-se retornar ao Contexto, pois os objetivos e necessidades de gestão podem mudar ao longo do tempo.
Tarefas: ▪ Identificar as lições aprendidas: extrair o conhecimento adquirido nas etapas anteriores que podem ser utilizados no próximo ciclo da Gestão da Dívida Técnica. ▪ Aprovar a revisão: alinhar com os stakeholders os objetivos da necessidade de revisão da Gestão da Dívida Técnica.
Saídas: Lições aprendidas, revisão aprovada.
Papel responsável: Gestor da Dívida Técnica.
Outros papeis responsáveis: Gerentes e *Lìderes* de equipes.

8.2.6 Comunicação e Monitoramento

Atividade: Comunicação e Monitoramento

Descrição:

A comunicação ocorre em todas as atividades da Gestão da Dívida Técnica, pois deve garantir que todos os responsáveis e as partes interessadas compreendam os fundamentos e as razões sobre os quais as decisões são tomadas.

A comunicação deve assegurar que os objetivos sejam compreendidos, informar e conscientizar os envolvidos, comunicar as decisões e critérios utilizados, facilitar a troca de informações, reunir diferentes pontos de vista, obter retorno e sugestões.

O monitoramento dos itens de Dívida Técnica deve ser planejado junto com o processo de Gestão da Dívida Técnica. A atividade de monitoramento ocorre durante a evolução ou manutenção do software. O monitoramento pode ser periódico ou acontecer como resposta no momento da atualização ou mudança do código-fonte.

Durante o monitoramento deve ocorrer a análise crítica e periódica do processo de gestão e do andamento das ações. O uso de ferramentas automatizadas e de portais, com informações atualizadas e disponíveis para as partes interessadas, ajudam na atividade de monitoramento e controle da Dívida Técnica.

Tarefas:

- Monitorar periodicamente a Dívida Técnica: a Dívida Técnica deve ser periodicamente analisada e monitorada junto com os responsáveis. Este monitoramento pode ser realizado com reuniões periódicas onde os responsáveis devem informar a situação das tarefas de pagamento da Dívida Técnica, enfatizando pontos de atenção e entregas que foram realizadas (ou deveriam ter sido);
- Gestão a vista: gerar informações relevantes para os gestores e *stakeholders*, permitindo o acompanhamento de indicadores, status e tendências da Dívida Técnica. Uma das práticas de gestão à vista pode ser a criação de dashboards online;
- Acompanhar as ações preventivas: nesta tarefa deve-se verificar e notificar aos responsáveis o andamento das ações de prevenção da Dívida Técnica.

Saídas: Reuniões, Relatórios, Portais, Feedback.
Papel responsável: Gestor da Dívida Técnica.
Outros papeis responsáveis: Partes interessadas (*stakeholders*).

8.3 PRIORTD: Método de priorização da Dívida Técnica

O método PriorTD recebeu esse nome porque está relacionado à atividade de Priorização da Gestão da Dívida Técnica. Este método busca analisar o código-fonte e a Dívida Técnica sob a ótica da importância do código-fonte para o projeto, equipe ou gestores, visando fornecer visões de análise que atendam às diversas necessidades do negócio. De acordo com o momento do projeto e as prioridades dos interessados, o método PriorTD apresenta o código-fonte mais relevante para o pagamento da Dívida Técnica ou para a possibilidade de refatoração.

Durante o ciclo de vida do software, os códigos-fonte são modificados, adquirem informações, participam de diversos eventos, compõe funcionalidades com diferentes valores para o software e possuem previsão de alteração. Todo código-fonte carrega características do passado, presente e futuro.

O PriorTD busca extrair essas características para compor um modelo de análise que apresente o código-fonte mais relevante para o projeto, dividido em duas etapas. Na primeira etapa, é necessário identificar as características do código-fonte. Na segunda etapa, as características seguem uma jornada pelo funil de priorização, cuja saída é o código-fonte mais relevante para priorização.

A saída de PriorTD são os códigos-fonte candidatos para serem priorizados para pagamento da Dívida Técnica. A seleção dos códigos-fonte eleitos para o pagamento da Dívida Técnica deve passar pela análise dos responsáveis. Esse procedimento é essencial para dar credibilidade ao modelo de priorização e eliminar resultados falso-positivos devido a procedimentos internos de atualização de código-fonte ou arquitetura de software. O PriorTD permite que os objetivos da equipe se relacionem diretamente com as ações realizadas no código-fonte.

O método PriorTD difere dos modelos de priorização de Dívida Técnica apresentados na revisão da literatura, fornecendo uma estratégia para coletar

informações passadas, presentes e estimadas sobre o que acontecerá com o código-fonte. A proposta é coletar diversas características do código-fonte que ajudem a priorizar o pagamento da Dívida Técnica e apresentar essas informações de forma clara para todos os envolvidos na tomada de decisão.

8.3.1 Características do PriorTD

As características no PriorTD são entendidas como um conjunto de atributos que ajudam a diferenciar as qualidades particulares de um código-fonte de outro. Um conjunto de propriedades do código-fonte corresponde a uma característica. Uma característica possui um significado, sobre o qual possui um conjunto de informações que a distingue das outras características.

Este estudo identificou sete características que são relevantes para serem utilizadas na atividade de priorização na Gestão da Dívida Técnica de código-fonte.

O trabalho de Ribeiro *et al.* (2016) contribuiu na validação das características que compõem o modelo de análise do PriorTD. O estudo apresenta uma lista de 14 critérios de decisão para facilitar a priorização no pagamento dos Itens de Dívida Técnica. Para cada um dos critérios de decisão, uma ou mais características fornecem informações relevantes para a tomada de decisão. A escolha das características do PriorTD converge com os critérios para tomada de decisão identificados no trabalho de Ribeiro et al. (2016).

Este estudo identificou seis características relevantes para a atividade de priorização na Gestão da Dívida Técnica identificada no código-fonte: Dívida Técnica, Histórico de alteração, Análise de Impacto, Cobertura de Testes, Backlog e Projeto, ilustrado na Figura 34.

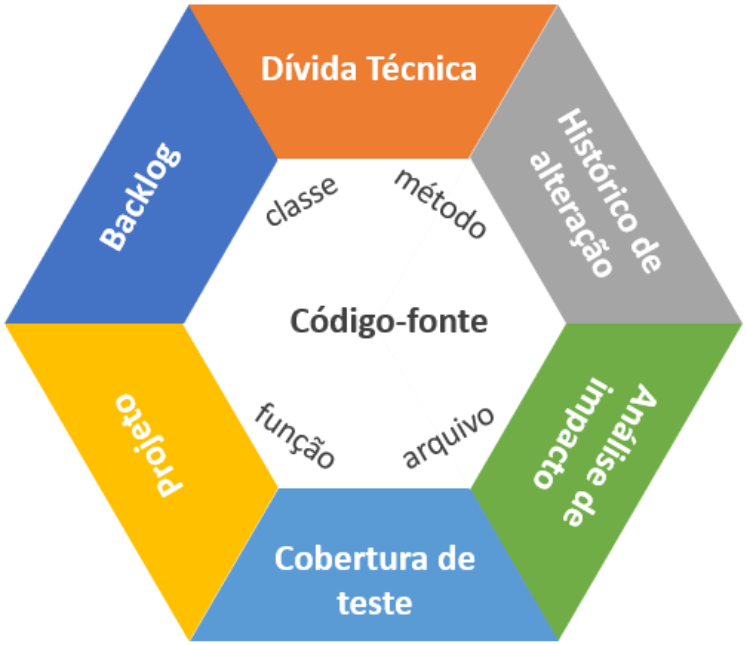


Figura 34. Características do PriorTD. Fonte: o Autor.

Cada característica pode ser representada por um ou mais atributos contendo valores quantitativos ou qualitativos, conforme descritos a seguir:

- **Dívida Técnica**: a característica mais crucial para o método PriorTD. Essa característica é composta pelas propriedades que fazem parte do Item de Dívida Técnica que estão presentes no código-fonte. Os Itens de Dívida Técnica possibilitam entender seu custo/benefício, o esforço necessário para eliminar um Item de Dívida Técnica, impacto e gravidade ao software. Os itens Dívida Técnica podem ser identificados fazendo análise de código-fonte. Existem diversas ferramentas que fazem análise estática de código-fonte para inspeção da qualidade como o SonarQube, Codeclimate, ndepend, Codacy, entre outras;
- **Histórico de alteração:** Nessa característica busca-se identificar o período, a frequência e a motivação das alterações no código-fonte. O código-fonte pode ser alterado por vários motivos. As mudanças são motivadas por defeitos ou evolução do produto. Esses históricos de alterações ajudam na identificação do código-fonte mais modificado e os mais instáveis. Um uso prático dessa característica é a priorização dos códigos-fonte que mais tiveram alterações motivadas por defeitos em um determinado período. As ferramentas de controle de versão (por exemplo GIT, CSV, SVN) possuem os históricos de alteração

do código-fonte. Analisando estes históricos é possível identificar e classificar os motivos que levaram à alteração do código-fonte. A prática de priorizar com base no histórico de alteração do código-fonte foi identificada em alguns estudos como de Sharma, Suryanarayana e Samarthyam et al. (2015) e Guo et al. (2016), e algumas ferramentas relatam que possuem a integração com sistemas de controle de versão;

- **Análise de impacto**: A análise de impacto avalia os diversos riscos associados a uma mudança no código-fonte (PFLEEGER; BOHNER, 1990). Os Itens de Dívida Técnica que afetam diversos recursos devem ter prioridade mais alta para o pagamento (GUO; SPÍNOLA; SEAMAN, 2016). Para o PriorTD, a análise de impacto se refere mais especificamente à análise dos efeitos que uma determinada mudança no código-fonte pode causar na aplicação. Uma métrica sugerida no PriorTD é descobrir a relação de dependência que pode existir entre os códigos-fonte. A quantidade de dependência pode sinalizar os níveis de acoplamento;
- **Cobertura de teste:** A cobertura de testes do código-fonte e a qualidade dos testes são tratados como um Tipo de Dívida. Essa característica destina-se a medir a cobertura de testes de código fonte, ou seja, a quantidade de código-fonte que está sendo testado, podendo ser testes unitários ou teste automatizado. As métricas de teste podem ajudar na priorização do pagamento da Dívida Técnica por sinalizar o nível de segurança e qualidade do código-fonte. Um código-fonte que impacta em diversos recursos do sistema e possui testes de baixa qualidade ou não tenha testes, é recomendada a priorização e a refatoração (SHARMA; SURYANARAYANA; SAMARTHYAM, 2015);
- **Backlog:** A priorização da Dívida Técnica deve considerar situações que dependem de fatores como a vida útil do produto e o impacto nas funcionalidades futuras (SHARMA; SURYANARAYANA; SAMARTHYAM, 2015). Não é necessário refatorar um software se nenhuma alteração adicional for solicitada no futuro (GUO; SPÍNOLA; SEAMAN, 2016). As informações para estas situações podem ser encontradas no Backlog do produto. O Backlog refere-se às funcionalidades desejadas para um software. No Backlog está acumulado o trabalho para um determinado intervalo de tempo. A partir do Backlog é possível identificar as funcionalidades estáveis, que não irão evoluir, e as funcionalidades que irão ter mais alteração dentro do período determinado.

A partir da priorização dos itens do Backlog é possível identificar as funcionalidades e os seus artefatos, e com isso é possível ter uma previsão de alteração do código-fonte. A previsão de alteração do código-fonte ajuda na análise de impacto da alteração para o projeto, e ajuda na priorização dos itens de Dívida Técnica que estão localizados em código-fonte que serão modificados devido a uma atividade de desenvolvimento ou manutenção;

- **Projeto:** Quanto maior e mais complexo o projeto, maior será o impacto da Dívida Técnica (MARTINI et al., 2017). Na análise do PriorTD, a característica Projeto é entendida como a importância (oportunidade do projeto, objetivo do projeto, resultado desejável, criticidade) da solução do problema para a organização. Como exemplo pode ser citada a criticidade do projeto para o negócio, pois é recomendável que os códigos-fonte que fazem parte dos projetos mais críticos devam ser priorizados.

8.3.2 Funil de priorização do PriorTD

O funil de priorização do PriorTD representa uma solução para o problema de priorização na Gestão da Dívida Técnica de código-fonte. O funil é dividido em diversas partes que possibilita a comunicação das instruções a serem passadas de um estágio para o outro.

O modelo de prioridade PriorTD pode usado para formular e organizar os artefatos e suas características relacionadas à qualidade do código-fonte.

Um funil de priorização pode representar o processo de priorização do PriorTD:

- a entrada do funil são os códigos-fonte;
- a execução da priorização ocorre no corpo do funil, que é composto pelas características ordenadas por prioridade;
- a saída do funil é o código-fonte para priorizar, ordenado do mais relevante para o menos relevante.

A Figura 35 representa graficamente o funil de priorização do PriorTD.

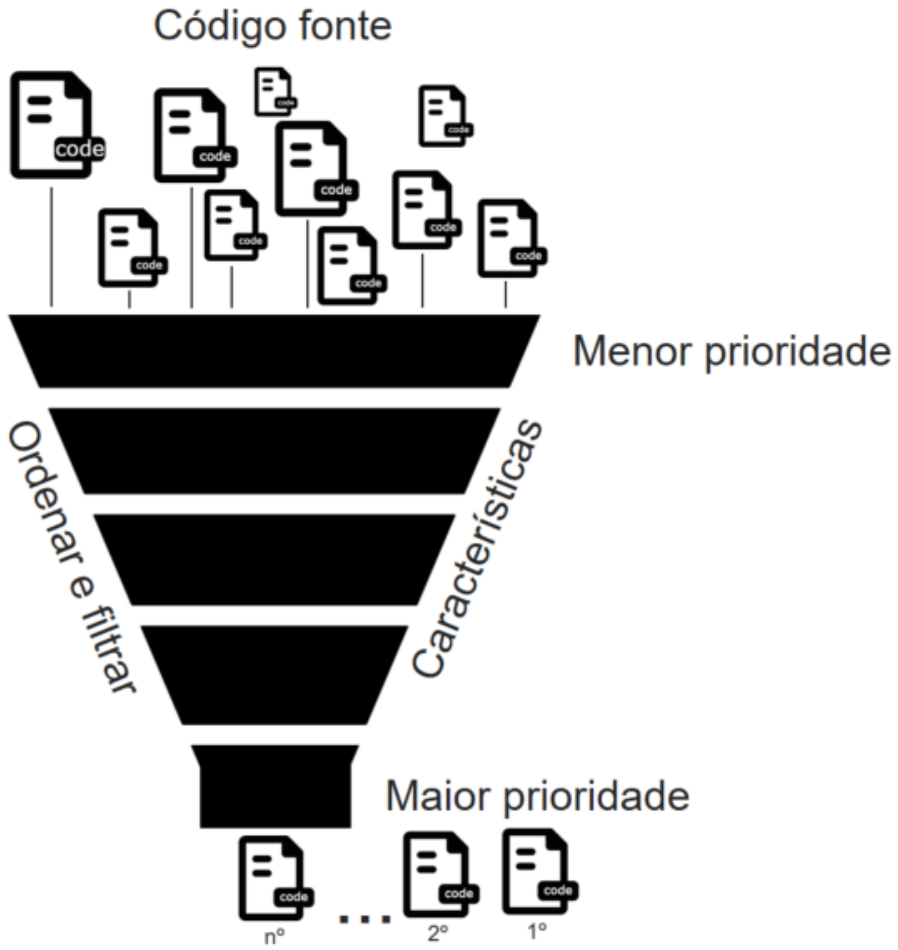


Figura 35. Funil de priorização do PriorTD. Fonte: o Autor.

O funil de priorização é abastecido com os códigos-fonte, que posteriormente serão analisados para gerar a priorização no pagamento da Dívida Técnica. Os códigos-fonte selecionados devem conter a mesmas características para a avaliação da priorização.

A execução da priorização ocorre dentro do funil de priorização, onde o código-fonte irá fazer uma jornada para obter a sua classificação como candidato para o pagamento da Dívida Técnica. Os estágios separam a jornada dentro do funil de priorização. Cada estágio compreende a uma característica, começando da característica de menor prioridade até a característica de maior prioridade. O número de estágios dentro do funil de priorização depende do número de características extraídas do código-fonte.

As prioridades das características são classificadas conforme as necessidades de negócio. A partir dos objetivos de negócio é determinada qual característica é mais

importante que as outras e assim determinar a sua ordem dentro do funil de priorização.

Dentro de cada etapa do funil são realizadas a operação de filtro e a ordenação dos códigos-fonte. Cada atributo da característica é analisado e é realizado o filtro dos códigos-fonte não desejados e a ordenação, na qual o mais importante é apresentado no topo.

Não é necessário que o código-fonte contenha todas as características sugeridas para utilizar o método PriorTD. Pode-se priorizar os códigos-fonte contando com as caraterísticas que são possíveis de serem extraídas pela organização. No segundo ciclo da pesquisa deste estudo, a priorização foi realizada utilizando 4 características para o código-fonte.

Após passar por todas as etapas do funil, ou seja, passar pelas operações de filtro e ordenação de cada característica, os resultados serão os códigos-fonte candidatos a serem priorizados para o pagamento da Dívida Técnica.

Dentro do universo dos códigos-fonte do software, o método PriorTD apresenta o código-fonte com maior importância para ser priorizado no pagamento da Dívida Técnica.

CAPÍTULO 9 - CONSIDERAÇÕES FINAIS

Este capítulo tem o objetivo de apresentar as considerações finais referentes aos resultados obtidos neste trabalho. A seguir são discutidas as limitações da pesquisa, os trabalhos relacionados, a relevância, as contribuições, a conclusão e os trabalhos futuros.

9.1 Limitação da pesquisa

No que se refere à Guilda de Dívida Técnica, observou-se que o patrocinador ou o coordenador da guilda foi quem convidou os participantes da guilda. Pode ser que ao convidar as pessoas para se juntarem à guilda possa ter criado algum constrangimento para os participantes que desejariam negar o convite, o que pode ter intimidado os membros periféricos a ter uma participação mais ativa na guilda. Isso também pode ser visto como um fator positivo (uma vez que o diretor patrocinou o projeto e endossou a autonomia da Guilda).

O processo de Gestão da Dívida Técnica deste estudo foi extraído a partir das observações da Dívida Técnica identificada nos artefatos de código-fonte, testes, *scripts* de testes e *build*. Não está no escopo desta pesquisa a utilização deste processo para os outros tipos de Dívida Técnica.

Não foi escopo desta pesquisa apresentar um estudo aprofundado de ferramentas de apoio à implementação de atividades de Gestão da Dívida Técnica, uma vez que dependem das tecnologias utilizadas pelas empresas. Algumas regras de qualidade sugeridas pela ferramenta SonarQube podem ser questionadas sobre o impacto na qualidade ou produtividade dos desenvolvedores. No entanto, acredita-se que neste estudo, todas as regras de qualidade selecionadas contribuíram para o desenvolvimento dentro das normas e melhores práticas propostas pela empresa.

O método PriorTD proposto busca atender a um grupo de questões para priorizar o pagamento dos itens de Dívida Técnica. As informações deste método não têm o objetivo de ser métricas para medir a qualidade do código-fonte.

O PriorTD se baseia nas Dívidas Técnicas que são extraídas a partir do código-fonte. A proposta de priorizar o pagamento da Dívida Técnica analisando a

importância do código-fonte apresenta características que possibilitam o seu uso para outros tipos de Dívida Técnica, porém não está no escopo desta pesquisa.

Para a característica do PriorTD de Dívida Técnica foram utilizados dados fornecidos pelo SonarQube. Algumas das regras utilizadas pela ferramenta podem não ser consideradas como Dívida Técnica, porém não foram encontradas evidências que esta prática prejudicou a análise da priorização.

Durante a avaliação do PriorTD, o grupo responsável pela análise dos resultados, identificou resultados falso-positivos. No contexto da pesquisa, os falso-positivos ocorreram pelos seguintes motivos: arquitetura do software proprietária, procedimentos internos de atualização dos arquivos e confiabilidade na extração dos valores das características. Apesar da presença dos falso-positivos, a análise e a seleção do código-fonte não foram prejudicadas.

Este estudo não procura descrever como o método PriorTD deve ser implementado em forma de ferramenta. Usuários e editores de ferramentas são livres para escolher ou construir uma solução implementando o método descrito neste documento.

Este estudo se limita à Dívida Técnica adquirida anteriormente e não leva em consideração as decisões para aquisição de nova Dívida Técnica.

Os esforços e custos associados à execução e implementação desta pesquisa não foram registrados. É necessário examinar o impacto do processo usando abordagens alternativas, como custos de qualidade, esforço e tempo de ciclo para a revisão da gerência.

Dado o contexto de execução da pesquisa e o fato de se tratar de um projeto de pesquisa-ação, seus resultados estão vinculados a um contexto organizacional específico. É possível que as atividades definidas possam ser aplicadas a outras organizações de desenvolvimento de software se forem consideradas suas variáveis de contexto.

9.2 Trabalhos relacionados

Da mesma forma que os resultados desta pesquisa foram divididos em três propostas principais, buscou-se analisar os trabalhos relacionados também sob a ótica de cada uma das propostas. Sendo, o método de priorização PriorTD, o processo de Gestão da Dívida Técnica e a Guilda de Dívida Técnica.

9.2.1 Guilda de Dívida Técnica

A guilda de Dívida Técnica possui os elementos essenciais de domínio, comunidade e prática que caracterizam uma Comunidade de Prática, conforme descrito por Wenger, McDermott e Snyder (2002). A guilda implementada nesta pesquisa possui os diferentes tipos de membros, conforme identificado por Smite et al. (2019), apresentado anteriormente na Figura 3.

A guilda de Dívida Técnica seguiu as recomendações de Smite et al. (2019), estabelecendo uma prática e um escopo bem definidos, tendo interações regulares com tarefas e responsabilidades que mostravam sinais de engajamento dos membros conforme os resultados apresentados. Neste estudo foi possível confirmar a afirmação de Smite et al. (2019) que a autoridade e a atenção do patrocinador contribuem para ajudar a alcançar os objetivos da guilda. Este apresenta evidências da relevância do papel do patrocinador. Como já apontado, o diretor teve um papel essencial no patrocínio da guilda.

A formação da guilda seguiu as diretrizes de diversos estudos na área. Ainda assim, a guilda de Dívida Técnica se diferencia por apoiar a implantação e continuidade da Gestão da Dívida Técnica no desenvolvimento de software.

Apesar da falta de estudos sobre estratégias para implementar e monitorar a Gestão da Dívida Técnica no contexto empresarial, vários estudos apresentam sugestões e desafios no qual uma guilda de Dívida Técnica pode contribuir, tais como:

- Considera o contexto na identificação e avaliação do DT, conforme Kruchten; Nord; Ozkaya (2012) sugerido.
- Ajudar as equipes a quantificar, priorizar e justificar o pagamento da Dívida Técnica. Os desafios citados nos estudos de Sharma et al. (2015), Fernández-Sánchez et al. (2017) e Cai e Kazman (2018) foram também observados em nosso estudo.
- Fornecer comunicação transparente sobre os retornos esperados no pagamento da Dívida Técnica (Fernández-Sánchez et al., 2017).
- Envolver todas as partes interessadas na tomada de decisões na Gestão da Dívida Técnica, conforme sugerido em Fernández-Sánchez et al., 2017; Rios, Mendonça e Spínola, 2018.

Os estudos que fizeram parte do estudo terciário de Rios, Mendonça e Spínola (2018) não apontaram estratégias que colaboram para a atividade de Prevenção da

Dívida Técnica. Por meio do apoio da guilda de Dívida Técnica na padronização de código-fonte, no treinamento das equipes, na padronização do desenvolvimento de *scripts* de teste e na revisão de código-fonte dos testes automatizados, uma guilda de Dívida Técnica pode ser usada como estratégia para Prevenir a Dívida Técnica.

9.2.2 Processo de Gestão de Dívida Técnica

A aplicação prática de Gestão da Dívida Técnica necessita de soluções confiáveis, que utilizem um conjunto de estratégias, ferramentas e soluções que abordem a Gestão da Dívida Técnica de maneira holística no desenvolvimento de software. Assim, buscou-se comparar o presente estudo com abordagens e estratégias que atendem a mais de uma atividade de Gestão da Dívida Técnica e possuem demonstração de uso em um ambiente real. Não foram consideradas aqui abordagens que tratam de apenas uma etapa ou atividade.

Oliveira, Goldman e Santos (2015) avaliam a aplicação de uma estratégia de Gestão da Dívida Técnica em um contexto real de projetos de software que adotam Scrum. O estudo não apresenta um alinhamento anterior com os *stakeholders* e não apresenta uma definição clara dos objetivos e propósitos da Gestão da Dívida Técnica nas empresas estudadas, gerando problemas de comprometimento com os prazos, identificação e priorização da Dívida Técnica. A presente abordagem definiu e implementou o alinhamento do contexto e dos objetivos da Gestão da Dívida Técnica com os *stakeholders* e buscou fazer uma ligação da Gestão da Dívida Técnica com a importância do projeto para as equipes.

No estudo de Guo, Spínola e Seaman (2016) é avaliada a abordagem de Gestão da Dívida Técnica de Seaman e Guo (2011), examinando os custos e benefícios do gerenciamento da Dívida Técnica em um projeto de software ativo que utiliza o processo de desenvolvimento semelhante ao Scrum. Para explorar os custos e benefícios foi necessário aplicar na prática a abordagem de Gestão da Dívida Técnica. Foi identificado que a abordagem proposta por Seaman e Guo (2011) e implementada no estudo Guo, Spínola e Seaman (2016) é centrada no Item da Dívida Técnica, afastando a importância do projeto (produto) na avaliação da Dívida Técnica, conforme relatado no estudo. Apesar do estudo de Guo, Spínola e Seaman (2016) citarem a etapa de controle de Gestão de Riscos, não foi identificada nenhuma atividade de controle preventivo da Dívida Técnica. No presente estudo se destaca a

prevenção da Dívida Técnica e a importância de alinhar os objetivos de negócio com a Gestão da Dívida Técnica.

Ramasubbu e Kemerer (2019) propõem um processo de Gestão da Dívida Técnica dividido em três etapas. O objetivo principal do estudo é integrar os processos de gerenciamento da Dívida Técnica com o gerenciamento de qualidade de software. O estudo foi implementado em três empresas de desenvolvimento de produtos de software, onde foram comprovados os benefícios econômicos com a adoção e uso do processo proposto. Ao comparar o presente estudo com a proposta de Ramasubbu e Kemerer (2019) entende-se que os estudos se complementam. No estudo de Ramasubbu e Kemerer (2019) são comprovados os benefícios econômicos da Gestão da Dívida Técnica. O presente estudo foca nas informações das soluções e desafios encontrados na execução de cada atividade de Gestão da Dívida Técnica.

Para adotar a proposta de Ramasubbu e Kemerer (2019) é necessário incluir ativos de processos organizacionais (Planos de Negócio, Requisito do produto, *Roadmap*, *Backlog* dos defeitos e políticas de gestão de risco), o que dificulta a adoção da proposta por empresas com níveis baixos de maturidade de processos, conforme destacado em uma das empresas estudadas. O nosso estudo foi validado em uma empresa que não possui certificação por modelo de maturidade e utiliza o método Scrum. Destaca-se a atividade de revisão da Gestão da Dívida Técnica permitindo que as empresas possam adotar a Gestão da Dívida Técnica independentemente do nível de maturidade dos processos e conforme adquiram mais controles, possam revisar a Gestão da Dívida Técnica e incorporar novas técnicas ou ferramentas.

Para gerenciar o risco de segurança no desenvolvimento de software, o estudo de Rindell e Holvitie (2019) propõe uma abordagem de Gestão da Dívida de segurança que é uma extensão da estrutura de Gestão da Dívida Técnica baseada em portfólio, inicialmente proposta por Guo e Seaman (2011). A abordagem é centralizada na avaliação do risco para cada Item da Dívida Técnica. Os autores apresentam uma sinergia da Gestão de Risco com a Gestão da Dívida Técnica, porém a abordagem carece de uso na prática e não apresenta alternativas para controlar ou prevenir a Dívida Técnica como foi apresentado no presente estudo.

Estudos como de Guo, Spínola e Seaman (2016), Fairley e Willshire (2017) e Rindell e Holvitie (2019), entre outros, apresentam técnicas de gerenciamento de risco para ajudar a detectar, monitorar e mitigar a Dívida Técnica. Compreender como os

projetos de software são afetados pela Dívida Técnica é importante para mitigar os riscos do projeto (por exemplo riscos de retrabalho, aumento de custos, atrasos de cronograma). Além do proposto nos trabalhos anteriores, o presente estudo se diferencia por incorporar as atividades de Contexto, Revisão e controle preventivo no processo de Gestão da Dívida Técnica, que foram inspirados na Gestão de Risco.

9.2.3 PRIORTD – Método de Priorização da Dívida Técnica

As principais abordagens sobre priorização e refatoração da Dívida Técnica de código-fonte podem ser obtidas nas revisões de literatura de Abid et al. (2020), Lenarduzzi et al. (2021) e Alfayez et al. (2020). Porém, algumas abordagens não foram avaliadas e outras não foram avaliadas na indústria. Essa falta de avaliações na indústria dificulta a capacidade de prever o desempenho das abordagens (Alfayez et al., 2020). Dado que o PriorTD foi aplicado na prática, nesta seção de trabalhos relacionados, as abordagens consideradas são as que foram avaliadas na indústria.

Além da ocorrência da Dívida Técnica, algumas abordagens utilizam o histórico de alterações do código fonte para priorizar o pagamento da Dívida Técnica, mas sem considerar a natureza da alteração. Dentre essas propostas, destacam-se as obras de Zazworka et al. (2011), Siverland et al. (2015), Choudhary e Singh (2016), Rani e Chhabra (2017), Sae-Lim et al. (2017), Charalampidou et al. (2017), Husien et al. (2017). No entanto, a eficácia dessas abordagens depende de se ter um histórico disponível suficiente para prever o nível de prioridade da Dívida Técnica. Além disso, utilizar alterações sem considerar sua natureza pode causar uma priorização de código-fonte que não possua defeitos ou problemas, pois pode ocorrer que as alterações sejam originadas da evolução do software.

No PriorTD, mesmo quando não há histórico de alterações disponível ou o histórico não é relevante para a tomada de decisão, dependendo do objetivo da equipe, o código fonte pode ser um candidato para o pagamento da Dívida Técnica. O PriorTD também difere das abordagens acima considerando a natureza das mudanças, motivadas por *bugs* do cliente, *bugs* internos ou evolução. Isso pode mitigar o risco de refatoração desnecessária.

Outras abordagens de priorização buscam identificar o código-fonte candidato para refatoração com base na avaliação da importância ou relevância do código-fonte no software. Entre essas abordagens destacam-se as pesquisas de Mkaouer et al. (2014), Ouni et al. (2015) e Singh et al. (2019). Mkaouer et al. (2014) utilizam um

modelo multiobjetivo baseado em NSGA-II, cuja importância da classe que possui *Code Smell* é representada pelo número de comentários, relacionamentos e métodos. Ouni et al. (2015) sugerem priorizar a refatoração utilizando a metaheurística *Chemical Reaction Optimization*, onde a importância do código-fonte corresponde a uma pontuação de *Code Smells* detectado após a aplicação de uma determinada refatoração. Singh et al. (2019) consideram que a importância do código fonte que possui *Code Smell* é obtida a partir do *feedback* dos desenvolvedores. O PriorTD difere dessas abordagens na medida em que, além de utilizar informações extraídas do código-fonte, considera as necessidades do projeto e da equipe na identificação do código-fonte mais relevante.

Almeida et al. (2018) propõem uma priorização da Dívida Técnica considerando os processos de negócio do sistema. Esta abordagem não considera as necessidades das equipes ou da organização conforme previsto no PriorTD. No entanto, pode contribuir para a implementação prática da característica *Backlog* proposta no PriorTD. A proposta do PriorTD em selecionar para o pagamento os itens de Dívida Técnica a partir da importância do código-fonte, apresenta uma visão diferente das propostas de priorização de Dívida Técnica apresentadas até o momento. O PriorTD se diferencia principalmente pela característica de fornecer diversas informações que, juntas, abordam diversos problemas para apoiar a tomada de decisão.

9.3 Contribuição da pesquisa

A motivação principal desta pesquisa foi fornecer subsídios para contribuir no pagamento dos itens de Dívida Técnica nos artefatos de código-fonte mais relevantes, fazendo uma ligação entre os objetivos de gestão com o código-fonte e com isso ajudando na comunicação dos interesses dos desenvolvedores de software e gerentes da empresa nas decisões de investir em aspectos internos de qualidade do software.

A partir dos resultados apresentados nesta pesquisa é possível identificar as seguintes contribuições:

- A Guilda de Dívida Técnica contribui para a Gestão da Dívida Técnica, e pode ser uma estratégia para implementação da Gestão da Dívida Técnica em uma organização;
- A pesquisa apresenta uma proposta para a Gestão da Dívida Técnica com evidências empíricas do uso em uma indústria de software.

- Cada uma das atividades implementadas conta com relatos dos procedimentos adotados e os resultados obtidos, com detalhes da implementação da Gestão da Dívida Técnica, destacando os benefícios e as decisões o que contribui para que outras organizações possam replicar este estudo;
- O processo de Gestão da Dívida Técnica proposto neste trabalho permite o direcionamento para a evolução contínua. A partir deste processo a organização tem um guia de trabalho para orientar a continuidade da Gestão da Dívida Técnica
- O PriorTD se mostrou eficiente na atividade de Priorização dos itens de Dívida Técnica, apresentando os artefatos de código-fonte mais relevantes para o projeto;
- A divisão por cenários aproximou as visões de negócio com a equipe técnica, possibilitando o entendimento claro sobre os investimentos gastos no pagamento da Dívida Técnica;
- O PriorTD contribuiu na análise dos resultados fornecidos por uma ferramenta de análise estática de código-fonte, pois as características do PriorTD complementam a análise do código-fonte com informações relevantes para os desenvolvedores;
- O uso do método PriorTD na Gestão da Dívida Técnica apresentou argumentos convincentes para o pagamento da Dívida Técnica, pois o modelo foca em várias informações para justificar a priorização e conforme apresentado neste estudo, foi justificada a necessidade de refatoração do código-fonte;
- As abordagens sugeridas neste estudo melhoram a qualidade interna e a qualidade externa do software. Conforme demostrado nos resultados dos KPIs na avaliação do 3º Ciclo da pesquisa.

A principal contribuição desta pesquisa é a criação do método PriorTD para a priorização do código-fonte no pagamento da Dívida Técnica. Conforme a identificação dos trabalhos relacionados à priorização da Dívida Técnica, a abordagem proposta pelo PriorTD não se assemelha com nenhum destes trabalhos.

9.4 Conclusão

Este estudo apresentou três propostas para melhorar a qualidade do produto de software com evidências empíricas de uso em uma empresa de software.

Na Gestão da Dívida Técnica, cada uma das atividades implementadas forneceu informações dos procedimentos adotados e dos resultados obtidos, com detalhes da implementação, destacando os benefícios, as decisões significativas e as lições aprendidas. O processo de Gestão da Dívida Técnica proposto neste estudo fornece um ponto de partida para uma organização começar a gerenciar a Dívida Técnica.

A Guilda de Dívida Técnica foi uma parte central no processo de Gestão da Dívida Técnica. Selecionar as pessoas certas para a guilda foi crucial para o sucesso na Gestão da Dívida Técnica, uma vez que as lições aprendidas no 1º ciclo levaram a alocar alguns dos desenvolvedores mais experientes para a guilda no 2º ciclo da pesquisa.

Conforme já descrito por Holvitie et al. (2018), este estudo também observou que práticas e processos ágeis, que verificam e mantêm a estrutura e a clareza dos artefatos (por exemplo, padrões de codificação e práticas de refatoração) de desenvolvimento de software, são percebidos como tendo um efeito positivo na Gestão da Dívida Técnica.

Assim como Falessi e Voegele (2015), foi identificado que o engajamento de especialistas constrói a confiança das equipes no projeto de Gestão da Dívida Técnica. Este estudo confirmou que o uso de uma categorização específica da Dívida Técnica para o contexto aumentou a confiança dos desenvolvedores no uso do SonarQube. Também foi identificado que os desenvolvedores carecem de um mecanismo de priorização para gerenciar as várias oportunidades de melhoria de qualidade fornecidas por ferramentas estáticas de análise de código-fonte.

O método PriorTD contribuiu para orientar o pagamento de itens TD no código-fonte mais relevantes por meio da relação entre os objetivos das equipes com o código-fonte e, consequentemente, ajudou as equipes nas decisões de investir em aspectos internos de qualidade de software.

A partir dos resultados, é possível concluir que PriorTD:

- mostrou-se eficiente na atividade de priorização de Itens de Dívida Técnica, apresentando o código-fonte mais relevante para o projeto;

- contribuiu para a análise dos resultados fornecidos por uma ferramenta estática de análise de código-fonte, pois as características complementam a análise do código-fonte com informações relevantes para os desenvolvedores;
- forneceu orientação e suporte para entender a necessidade de refatorar o código-fonte;
- ajudou na identificação de código morto;
- as características implementadas do PriorTD, ajudaram na comunicação entre a equipe de desenvolvimento e os gestores;
- a divisão por cenários aproximou as visões do negócio da equipe técnica, permitindo um entendimento claro dos investimentos gastos no pagamento da Dívida Técnica.

O método PriorTD junto com a Gestão da Dívida Técnica apresentaram argumentos convincentes para o pagamento da Dívida Técnica, pois o modelo foca em diversas informações para justificar a priorização.

Devido à demora dos clientes em atualizem a versão do produto, que podia levar até 9 meses para maioria dos clientes atualizarem para a versão que possuía o código-fonte com as alterações propostas nas ações, somente no 3º ciclo da pesquisa foi possível ter evidências que as abordagens sugeridas neste estudo melhoraram a qualidade interna e a qualidade externa do produto. Assim, podemos concluir que esta pesquisa contribuiu para melhorar os KPIs utilizados para medir o objetivo estratégico definido para o contexto.

9.5 Trabalhos futuros

A pesquisa tende a avançar sob a ótica de identificar a importância do código-fonte para o projeto. O PriorTD pode evoluir e considerar outras características, tais como: critérios de segurança e vulnerabilidade do código-fonte, a importância do código-fonte para o cliente e os riscos.

Os próximos passos desta pesquisa envolvem a realização de estudos experimentais em ambiente acadêmico e na indústria, e a construção de uma ferramenta com o objetivo de avaliar mais profundamente os resultados da proposta do PriorTD.

A construção de uma ferramenta pode facilitar o uso do PriorTD e ajudar no monitoramento contínuo e automático das informações, para então se adaptar com mais agilidade as mudanças no desenvolvimento do software.

Acredita-se que a união das informações e orientações do PriorTD alinhada com uma ferramenta, pode fornecer uma base empírica sólida para apoiar a melhoria contínua do software.

Olhando para o passado como um guia para o futuro, é possível estar alerta e começar a encontrar soluções para a Dívida Técnica futura antes de ficar sobrecarregado por ela.

REFERÊNCIAS BIBLIOGRÁFICAS

ABID, C.; ALIZADEH, V.; KESSENTINI, M.; FERREIRA, T.; DIG; D. **30 Years of Software Refactoring Research: A Systematic Literature Review.** 2020, ISELab, University of Michigan-Dearborn, arXiv:2007.02194.

AL MAMUN, M.; BERGER, C.; HANSSON, J. **Explicating, Understanding, and Managing Technical Debt from Self-Driving Miniature Car Projects.** 2014 Sixth International Workshop on Managing Technical Debt, Victoria, BC, Canada, 30 set. 2014.

AL-BARAK, M.; BAHSOON, R. **Database Design Debts through Examining Schema Evolution.** 2016 IEEE 8th International Workshop on Managing Technical Debt (MTD), Raleigh, NC, USA, 4 out. 2016.

ALFAYEZ, R.; BOEHM, B. **Technical Debt Prioritization: A Search-Based Approach.** 2019, IEEE 19th International Conference on Software Quality, Reliability and Security (QRS), Sofia, Bulgaria, pp. 434-445, DOI: 10.1109/QRS.2019.00060. 2019.

ALFAYEZ, R.; ALWEHAIBI, W.; WINN, R.; VENSON, E.; BOEHM, B. **A systematic literature review of technical debt prioritization.** 2020, 3rd International Conference on Technical Debt (TechDebt '20). Association for Computing Machinery, New York, NY, USA, 1–10. DOI: https://doi.org/10.1145/3387906.3388630.

ALI SHAH, S.; TORCHIANO, M.; VETRÒ, A.; MORISIO, M. **Exploratory Testing as a Source of Technical Debt.** 2014, IT Professional, 16, n. 3, 07/03/2014. 44-51.

ALLMAN, E**. Managing Technical Debt.** 2012, Queue, 10, mar. 2012. 10.

ALMEIDA, R.; KULESZA, U.; TREUDE, C.; FEITOSA, D.; LIMA, A. **Aligning Technical Debt Prioritization with Business Objectives: A Multiple-Case Study.** 2018 IEEE International Conference on Software Maintenance and Evolution (ICSME), ago. 2018.

ALMEIDA, R.; TREUDE, C.; KULESZA, U. **Tracy: A business-driven technical debt prioritization framework.** 2019, 2019 IEEE International Conference on Software Maintenance and Evolution (ICSME), pp. 181-185, DOI: 10.1109/ICSME.2019.00028.

ALMEIDA, R.; RIBEIRO, R.; TREUDE, C.; KULESZA, U. **Business-Driven Technical Debt Prioritization: An Industrial Case Study**. 2021, 2021 IEEE/ACM International Conference on Technical Debt (TechDebt), pp. 74-83, DOI: 10.1109/TechDebt52882.2021.00017.

ALVES, N.; MENDES, T.; DE MENDONÇA, M. G.; SPÍNOLA, R.; SHULL, F.; SEAMAN, C. **Identification and management of technical debt.** 2016, Information

and Software Technology, Butterworth-Heinemann Newton, MA, USA, 70, n. C, fev. 2016.

AMPATZOGLOU, A.; AMPATZOGLOU, A.; CHATZIGEORGIOU, A.; AVGERIOU, P. **The financial aspect of managing technical debt: A systematic literature review.** 2015, Information and Software Technology, 64, ago. 2015. 52-73.

AVGERIOU, P.; KRUCHTEN, P.; OZKAYA, I.; SEAMAN, C. **Managing Technical Debt in Software Engineering (Dagstuhl Seminar 16162)**. 2016, Dagstuhl Reports, Dagstuhl, Germany, 6, n. 4, 110-138.

AVGERIOU, P.; TAIBI, D.; AMPATZOGLOU, A.; ARCELLI FONTANA, F.; BESKER, T.; CHATZIGEORGIOU, A.; LENARDUZZI, V.; MARTINI, A.; MOSCHOU, A.; PIGAZZINI, I.; SAARIMAKI, N.; SAS, D. D.; DE TOLEDO, S. S.; TSINTZIRA, A. A. **An Overview and Comparison of Technical Debt Measurement Tools**. 2020, IEEE Software, DOI: 10.1109/MS.2020.3024958.

BAVANI, R. **Distributed Agile, Agile Testing, and Technical Debt.** 2012, IEEE Software, 29, n. 6, 22 out. 2012. 28-33.

BECKER, C.; CHITCHYAN, R.; BETZ, S.; MCCORD, C. **Trade-off decisions across time in technical debt management: a systematic literature review.** 2018, TechDebt '18 Proceedings of the 2018 International Conference on Technical Debt, Gothenburg, Sweden, maio 2018. 85-94.

BEHUTIYE, W.; RODRÍGUEZ, P.; OIVO, M.; TOSUN, A. **Analyzing the concept of technical debt in the context of agile software development: A systematic literature review.** 2017, Information and Software Technology, 82, fev. 2017. 139-158.

BESKER, T.; MARTINI, A.; BOSCH, J. **Managing architectural technical debt: A unified model and systematic literature review.** 2018, Journal of Systems and Software, 135, jan. 2018. 1-16.

BESKER, T.; MARTINI, A.; BOSCH, J. **Technical Debt Triage in Backlog Management.** 2019, 2019 IEEE/ACM International Conference on Technical Debt (TechDebt), Montreal, QC, Canada.

BROWN, N.; CAI, Y.; GUO, Y.; KAZMAN, R.; KIM, M.; KRUCHTEN, P.; LIM, E.; MACCORMACK, A.; NORD, R.; OZKAYA, I.; SANGWAN, R.; SEAMAN, C.; SULLIVAN, K.; ZAZWORKA, N. **Managing technical debt in software-reliant systems.** 2010, FoSER '10 Proceedings of the FSE/SDP workshop on Future of software engineering research, Santa Fe, New Mexico, USA, nov. 2010. 47-52.

BUSCHMANN, F. **To Pay or Not to Pay Technical Debt.** 2011, IEEE Software, 28, n. 6, 20 out. 2011. 29-31.

CAI, Y.; KAZMAN, R. **DV8: Automated Architecture Analysis Tool Suites.** 2019 IEEE/ACM International Conference on Technical Debt (TechDebt), Montreal, QC, Canada, 26 maio 2019.

CAST. **Technical Debt Estimation.** 2019, Cast Software. Disponível em: <https://www.castsoftware.com/research-labs/technical-debt-estimation>. Accesso em: 26 out. 2019.

CHARALAMPIDOU, S.; AMPATZOGLOU, A.; CHATZIGEORGIOU, A.; AVGERIOU, P. **Assessing Code Smell Interest Probability: A Case Study.** 2017, XP '17 Workshops, Cologne, Germany, 22 maio 2017.

CHOUDHARY, A.; SINGH, P. **Minimizing Refactoring Effort through Prioritization of Classes based on Historical, Architectural and Code Smell Information.** 2016, 1st International Workshop on Technical Debt Analytics (TDA 2016), Hamilton, New Zealand, 6 dez. 2016.

CODABUX, Z.; WILLIAMS, B.; BRADSHAW, G.; CANTOR, M. **An empirical assessment of technical debt practices in industry.** 2017, Journal of Software: Evolution and Process 2017, 2017.

CODABUX, Z.; WILLIAMS, B. **Managing technical debt: An industrial case study.** 2013, 2013 4th International Workshop on Managing Technical Debt (MTD), San Francisco, CA, USA, 26 nov. 2013.

COUGHLAN, P.; COGHLAN, D. **Action Research for Operations Management.** 2002, January 2002 International Journal of Operations & Production Management, 22, Jan. 2002. 220-240.

CUNNINGHAM, W. **The WyCash portfolio management system.** 1992, OOPSLA '92 Addendum to the proceedings on Object-oriented programming systems, languages, and applications (Addendum), Vancouver, British Columbia, Canada, 4, n. 2, out. 1992. 29-30.

CURTIS, B.; SAPPIDI, J.; SZYNKARSKI, A. **Estimating the Principal of an Application's Technical Debt.** 2012, IEEE Software, 29, n. 6, 34 - 42.

DAVIS, N. **Driving Quality Improvement and Reducing Technical Debt with the Definition of Done.** 2013, 2013 Agile Conference, Nashville, TN, USA, set. 2013.

DESJARDINS, M. **How to execute corporate action plans effectively**. 2011, Business in Vancouver. 2011, Archived from the original on 22 March 2014.

DICK, B. **A beginner's guide to action research**, 2000. Disponível em: <http://www.aral.com.au/resources/guide.html>. Accesso em: 03 set. 2019. Available at http://www.uq.net.au/action_research/arp/guide.html.

DRESCH, A.; LACERDA, D.; MIGUEL, P. **Uma Análise Distintiva entre o Estudo de Caso, A Pesquisa-Ação e a Design Science Research,** 17, n. 56, 2015. 1116-1133.

FAIRLEY, R., WILLSHIRE, M. **Better Now Than Later: Managing Technical Debt in Systems Development**, 2017, vol. 50, nº 5, 03 2017.

FALESSI, D.; VOEGELE, A. **Validating and Prioritizing Quality Rules for Managing Technical Debt: An Industrial Case Study.** 2015 IEEE 7th International Workshop on Managing Technical Debt (MTD), Bremen, Germany, 2 out. 2015.

FERNÁNDEZ-SÁNCHEZ, C.; DÍAZ, J.; PÉREZ, J.; GARBAJOSA, J. **Guiding Flexibility Investment in Agile Architecting.** 2014 47th Hawaii International Conference on System Sciences, Waikoloa, HI, USA, 2014.

FERNÁNDEZ-SÁNCHEZ, C.; GARBAJOSA, J.; YAGÜE, A.; PÉREZ, J. **Identification and analysis of the elements required to manage technical debt by means of a systematic mapping study.** Journal of Systems and Software, 124, fev. 2017. 22-38.

Fleiss, J. L. **Statistical methods for rates and proportions**. 1981, Publisher: John Wiley & Sons Inc; Subsequent edition (1 Mar. 1981)

FONTANA, F.; PIGAZZINI, I.; ROVEDA, R.; ZANONI, M. **Automatic Detection of Instability Architectural Smells.** 2016, In Proceedings of the 32nd International Conference on Software Maintenance and Evolution (Icsme 2016), Raleigh, North Carolina, USA, out. 2016.

FOWLER, M. **Refactoring: Improving the Design of Existing Code**. 1999, Publisher: Addison Wesley; 1st edition (28 Jun. 1999), ISBN-10: 0201485672, ISBN-13: 978-0201485677

FOWLER, M. **Technical debt quadrant.** 2009. Disponível em: <https://martinfowler.com/bliki/TechnicalDebt.html>. Acesso em: 16 abr. 2019.

GHANBARI, H.; BESKER, T.; MARTINI, A.; BOSCH, J. **Looking for Peace of Mind? Manage your (Technical) Debt. An Exploratory Field Study.** 2017, In: 2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), Toronto, ON, Canada, 9 nov. 2017.

GIL, A. **Como elaborar projetos de pesquisa.** 2002, São Paulo, n. 4, 2002.

GRIFFITH, I.; TAFFAHI, H.; IZURIETA, C.; CLAUDIO, D. **A simulation study of practical methods for technical debt management in agile software development.** 2015, Proceedings of the Winter Simulation Conference 2014, Savanah, GA, USA, 26 jan. 2015.

GUO, Y.; SEAMAN, C.; GOMES, R.; CAVALCANTI, A.; TONIN, G.; DA SILVA, F.; SANTOS, A.; SIEBRA, C. **Tracking technical debt - An exploratory case study.** 2011, 27th IEEE International Conference on Software Maintenance (ICSM), Williamsburg, VI, USA, set. 2011.

GUO, Y.; SEAMAN, C. **A portfolio approach to technical debt management.** 2011, MTD '11 Proceedings of the 2nd Workshop on Managing Technical Debt, Waikiki, Honolulu, HI, USA, 23 maio 2011. 31-34.

GUO, Y; SPÍNOLA, R.; SEAMAN, C. **Exploring the costs of technical debt management - a case study.** 2016, Empirical Software Engineering, DOI: https://doi.org/10.1007/s10664-014-9351-7

HOLVITIE, J.; LICORISH, S.; SPÍNOLA, R.; HYRYNSALMI, S.; MACDONELL, S.; MENDES, T.; BUCHAN, J.; LEPPÄNEN, V. **Technical debt and agile software development practices and processes: An industry practitioner survey.** Information and Software Technology, 96, abr. 2018. 141–160.

HOLVITIE, J.; LEPPÄNEN, V. **DebtFlag: technical debt management with a development environment integrated tool.** MTD '13 Proceedings of the 4th International Workshop on Managing Technical Debt, San Francisco, California, 20/03/2013. 20-27.

HUSIEN, H.; HARUN, M.; LICHTER, H. **Towards a Severity and Activity based Assessment of Code Smells.** 2017, Procedia Computer Science, vol. 116, 460-467.

ISO/IEC 25010. **Software Engineering - Software product Quality Requirements and Evaluation (SQuaRE),** 2011, Geneva, 43.

IZURIETA, C.; VETRÒ, A.; ZAZWORKA, N.; CAI, Y.; SEAMAN, C.; SHULL, F. **Organizing the technical debt landscape**. 2012, In: 2012 Third International Workshop on Managing Technical Debt (MTD), Zurich, Switzerland, jun. 2012.

IZURIETA, C.; OZKAYA, I.; SEAMAN, C.; KRUCHTEN, P.; NORD, R.; SNIPES, W.; AVGERIOU, P. **Perspectives on managing technical debt: A transition point and roadmap from Dagstuhl.** 2016, In CEUR Workshop Proceedings (Vol. 1771, pp. 84-87). (CEUR Workshop Proceedings). Hamilton.

KEMMIS, S.; MCTAGGART, R. **The action research planner,** 1988, Deakin University.

KHOMYAKOV, I.; MAKHMUTOV, Z.; MIRGALIMOVA, R.; SILLITTI, A. **Automated Measurement of Technical Debt: A Systematic Literature Review.** 2019, 21st International Conference on Enterprise Information Systems (ICEIS 2019), DOI: 10.5220/0007675900950106, ISBN: 978-989-758-372-8, 2019. 95-106.

KNIBERG, H. **Spotify engineering culture (part 1).** 2014, Spotify Labs, 27/03/2014. Disponível em: <https://labs.spotify.com/2014/03/27/spotify-engineering-culture-part-1/>. Accesso em: 05/09/2019.

KRISHNA, V.; BASU, A. **Minimizing Technical Debt: Developer's viewpoint.** 2012, International Conference on Software Engineering and Mobile Application Modelling and Development (ICSEMA 2012), Chennai, India, dez. 2012.

KRUCHTEN, P.; NORD, R.; OZKAYA, I.; VISSER, J. **Technical Debt in Software Development: From Metaphor to Theory.** 2012, Report on the Third International Workshop on Managing Technical Debt. SIGSOFT Softw. Eng. Notes, New York, NY, USA, 37, n. 5, set. 2012. 36-38.

KRUCHTEN, P.; NORD, R.; OZKAYA, I.; FALESSI, D. **Technical debt: Towards a crisper definition report on the 4th international workshop on managing technical debt.** 2013, ACM SIGSOFT Software Engineering, ago. 2013. 51-54.

KRUCHTEN, P.; PHILIPPE, R.; OZKAYA, I. **Technical Debt: From Metaphor to Theory and Practice.** 2012, IEEE Software, 29, n. 6, 22 out. 2012. 18-21.

LAM, A.; Li, V. **Chemical-reaction-inspired metaheuristic for optimization.** 2010, IEEE Transactions Evolutionary Computation, 14(3), 381–399.

LARMAN, C.; VODDE, B. **Practices for Scaling Lean & Agile Development: Large, Multisite, and Offshore Product Development with Large-Scale Scrum**. 2010, Addison-Wesley Professional

LAVE, J.; WENGER, E. **Situated Learning: Legitimate Peripheral Participation**. 1991, New York.

LENARDUZZI, V.; BESKER, T.; TAIBI, D.; MARTINI, A.; FONTANA, F. **Technical Debt Prioritization: State of the Art. A Systematic Literature Review.** 2019, arXiv preprint arXiv:1904.12538.

LENARDUZZI, V.; BESKER, T.; TAIBI, D.; MARTINI, A.; FONTANA, F. **A systematic literature review on Technical Debt prioritization: Strategies, processes, factors, and tools.** 2021, Journal of Systems and Software, ISSN 0164-1212, https://doi.org/10.1016/j.jss.2020.110827, Volume 171, 2021.

LEPPÄNEN, M.; MÄKINEN, S.; LAHTINEN, S.; SIEVI-KORTE, O.; TUOVINEN, A.; MÄNNISTÖ, T. **Refactoring-a Shot in the Dark?** 2015, IEEE Software, 32, n. 6, 28 out. 2015. 62-70.

LETOUZEY, J. **The SQALE method for evaluating Technical Debt.** 2012, 2012 Third International Workshop on Managing Technical Debt (MTD), Zurich, Switzerland, 5 jun. 2012.

LETOUZEY, J.; ILKIEWICZ, M. **Managing Technical Debt with the SQALE Method.** 2012, IEEE Software, 29, n. 6, 22 ago. 2012. 44-51.

LI, Z.; AVGERIOU, P.; LIANG, P. **A systematic mapping study on technical debt and its management.** 2015, Journal of Systems and Software, 101, mar. 2015. 193-220.

LIM, E.; TAKSANDE, N.; SEAMAN, C. **A Balancing Act: What Software Practitioners Have to Say about Technical Debt.** 2012, IEEE Software, 29, n. 6, 23 ago. 2012. 22-27.

LI, Z.; AVGERIOU, P.; LIANG, P. **A systematic mapping study on technical debt and its management.** 2015, Journal of Systems and Software, vol. 101, pp. 193-220, 3 2015.

MARINESCU, R. **Assessing technical debt by identifying design flaws in software systems.** 2012, IBM Journal of Research and Development, 56, n. 5, 7 ago. 2012. 9:1-9:13.

MARTINI, A.; BOSCH, J.; CHAUDRON, M. **Investigating Architectural Technical Debt accumulation and refactoring over time: A multiple-case study.** 2015, Information and Software Technology, p. 237-253, DOI: https://doi.org/10.1016/j.infsof.2015.07.005.

MARTINI, A.; VAJDA, S.; VASA, R.; JONES, A.; ABDELRAZEK, M.; GRUNDY, J.; BOSCH, J. **Technical debt interest assessment: from issues to project.** 2017, XP '17 Proceedings of the XP2017 Scientific Workshops, Cologne, Germany, 22/03/2017.

MARTINI, A.; BOSCH, J. **An Empirically Developed Method to Aid Decisions on Architectural Technical Debt Refactoring: AnaConDebt.** 2016, 2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C), Austin, TX, USA, mar. 2016.

MARTINI, A.; BESKER, T.; BOSCH, J. **The Introduction of Technical Debt Tracking in Large Companies. A Survey and Multiple Case-Study.** 2016, 2016 23rd Asia-Pacific Software Engineering Conference (APSEC), Hamilton, New Zealand, 6 dez. 2016.

MARTINI, A.; BOSCH, J.; CHAUDRON, M. **Architecture Technical Debt: Understanding Causes and a Qualitative Model.** 2014, 2014 40th EUROMICRO Conference on Software Engineering and Advanced Applications, Verona, Italy, 20 nov. 2014.

MARTINI, A.; BOSCH, J.; CHAUDRON, M. **Investigating Architectural Technical Debt accumulation and refactoring over time: A multiple-case study.** 2015, Information and Software Technology, 67, nov. 2015. 237-253.

MCCONNELL, S. **Technical Debt**, 2007. Disponível em: <http://blogs.construx.com/blogs/stevemcc/archive/2007/11/01/technical-debt-2.aspx>.

MKAOUER, M.; KESSENTINI, M.; BECHIKH, S.; CINNÉIDE, M. **A Robust Multi-Objective Approach for Software Refactoring under Uncertainty**. 2014, International Symposium on Search Based Software Engineering (SSBSE 2014), p. 68–183.

NORD, R.; OZKAYA, I.; KRUCHTEN, P.; GONZALEZ-ROJAS, M. **In Search of a Metric for Managing Architectural Technical Debt.** 2012 Joint Working IEEE/IFIP Conference on Software Architecture and European Conference on Software Architecture, 25 out. 2012. 20-24.

OLIVEIRA, F.; GOLDMAN, A.; SANTOS V. **Managing Technical Debt in Software Projects Using Scrum: An Action Research**, *2015 Agile Conference*, Washington, DC, 2015, pp. 50-59, DOI: 10.1109/Agile.2015.7.

OPDYKE, W; JOHNSON, R. **Refactoring: an aid in designing application frameworks and evolving object-oriented systems.** 1990, in Proc. SOOPPA'90: Symposium on Object-Oriented Programming Emphasizing Practical Applications

OPDYKE, W. **Refactoring object-oriented frameworks**. 1992.

OUNI, A.; KESSENTINI, M.; BECHIKH, S.; SAHRAOUI, H. **Prioritizing code-smells correction tasks using chemical reaction optimization**. 2015, Software Quality Journal, vol. 23, 323–361.

PAASIVAARA, M.; LASSENIUS, C. **Deepening Our Understanding of Communities of Practice in Large-Scale Agile Development.** 2014, 2014 Agile Conference, DOI: 10.1109/AGILE.2014.18.

PFLEEGER, S.; BOHNER, S. **A framework for software maintenance metrics.** 1990, Proceedings. Conference on Software Maintenance 1990, San Diego, CA, USA, nov. 1990.

PINA, D.; GOLDMAN, A.; TONIN, G. **Technical Debt Prioritization: Taxonomy, Methods Results, and Practical Characteristics**, 2021, *2021 47th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, 2021, pp. 206-213, DOI: 10.1109/SEAA53835.2021.00034.

RACHOW, P. **Refactoring Decision Support for Developers and Architects Based on Architectural Impact.** 2019, 2019 IEEE International Conference on Software Architecture Companion (ICSA-C), DOI: 10.1109/ICSA-C.2019.00054.

RAMASUBBU, N.; KEMERER, C. **Integrating Technical Debt Management and Software Quality Management Processes: A Normative Framework and Field Tests.** 2019, IEEE Transactions on Software Engineering, 45, n. 3, mar. 2019. 285 - 300.

RANI, A.; CHHABRA J. **Prioritization of smelly classes: A two phase approach (Reducing refactoring efforts).** 2017, 2017 3rd International Conference on Computational Intelligence & Communication Technology (CICT). DOI: https://doi.org/10.1109/CIACT.2017.7977311.

RIBEIRO, L.; FARIAS, M.; MENDONÇA, M.; SPÍNOLA, R. **Decision Criteria for the Payment of Technical Debt in Software Projects: A Systematic Mapping Study.** 2016, ICEIS 2016 Proceedings of the 18th International Conference on Enterprise Information Systems, Rome, Italy, 25 abr. 2016. 572-579.

RIBEIRO, L.; ALVES, N.; MENDONÇA, M.; SPÍNOLA, R. **A Strategy Based on Multiple Decision Criteria to Support Technical Debt Management.** 2017, 2017 43rd Euromicro Conference on Software Engineering and Advanced Applications (SEAA), Vienna, Austria, 30 ago. 2017.

RIBEIRO, L.; SPÍNOLA, R. **Um Survey sobre a Pertinência e Relevância de Critérios de Decisão para Apoiar o Gerenciamento de Itens de Dívida Técnica.** 2016, Simpósio Brasileiro De Qualidade De Software (SBQS 2016), p. 256-270.

RINDELL, K.; HOLVITIE, J. **Security Risk Assessment and Management as Technical Debt**. 2019, 2019 International Conference on Cyber Security and Protection of Digital Services (Cyber Security), 2019, pp. 1-8, DOI: 10.1109/CyberSecPODS.2019.8885100.

RIOS, N.; MENDONÇA, M.; SPÍNOLA, R. **A tertiary study on technical debt: Types, management strategies, research trends, and base information for practitioners.** 2018, Information and Software Technology, 102, out. 2018. 117-145.

RIOS, N.; SPÍNOLA, R.; MENDONÇA, M.; SEAMAN, C. **The practitioners' point of view on the concept of technical debt and its causes and consequences: a design for a global family of industrial surveys and its first results from Brazil.** 2020, Empirical Software Engineering (2020) 25:3216–3287, Published online: 13 June 2020, https://doi.org/10.1007/s10664-020-09832-9.

SAE-LIM, N.; HAYASHI, S.; SAEKI, M. **Context-based approach to prioritize code smells for prefactoring**. 2017, Journal of Software: Evolution and Process. DOI: https://doi.org/10.1109/ICPC.2016.7503705.

SCHMID, K. **A Formal Approach to Technical Debt Decision Making.** 2013, QoSA '13 Proceedings of the 9th international ACM Sigsoft conference on Quality of software architectures, Vancouver, British Columbia, Canada, 17 jun. 2013. 153-162.

SEAMAN, C.; GUO, Y.; ZAZWORKA, N.; SHULL, F.; IZURIETA, C.; CAI, Y.; VETRÒ, A. **Using technical debt data in decision making: Potential decision approaches.** 2012, 2012 Third International Workshop on Managing Technical Debt (MTD), Zurich, Switzerland, 5 maio 2012.

SEAMAN, C.; GUO, Y. **Measuring and Monitoring Technical Debt.** 2011, Advances in Computers, 82, p. 25-46.

SHARMA, T.; SURYANARAYANA, G.; SAMARTHYAM, G. **Challenges to and Solutions for Refactoring Adoption.** 2015, IEEE Software, 32, n. 6, Nov. 2015. 44-51.

SIEBRA, C.; TONIN, G.; DA SILVA, F.; OLIVEIRA, R.; ANTONIO, L.; MIRANDA, R.; SANTOS, A. **Managing technical debt in practice: An industrial report.** 2012, Proceedings of the 2012 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement, Lund, Sweden, set. 2012.

SINGH, R., BINDAL A.; KUMAR A. **Reducing Maintenance Efforts of Developers by Prioritizing Different Code Smells.** 2019, International Journal of Innovative Technology and Exploring Engineering (IJITEE), vol. 8, 139-143.

SIVERLAND, S.; WERNERSSON, R.; SENNERSTEN, C. **Optimal Refactoring.** 2015, International Conference on Agile Software Development, 224-229.

SMITE, D.; MOE, N.; LEVINTA, G.; FLORYAN, M. **Spotify Guilds: How to Succeed with Knowledge Sharing in Large-Scale Agile Organizations.** 2019, 32(2), pp. 51-57. DOI: 10.1109/MS.2018.2886178.

SMITE, D.; MOE, N.; FLORYAN, M.; LEVINTA, G.; CHATZIPETROU, P. **Spotify guilds.** 2020, 63(3), pp. 56–61. DOI: 10.1145/3343146.

SNIPES, W.; ROBINSON, B.; GUO, Y.; SEAMAN, C. **Defining the decision factors for managing defects: a technical debt perspective.** MTD '12 Proceedings of the Third International Workshop on Managing Technical Debt, Zurich, Switzerland, 5 jun. 2012. 54-60.

SPÍNOLA, R.; VETRÒ, A.; ZAZWORKA, N.; SEAMAN, C.; SHULL, F. **Investigating technical debt folklore: Shedding some light on technical debt opinion.** 2013 4th International Workshop on Managing Technical Debt (MTD), San Francisco, CA, USA, 20 maio 2013.

STOCHEL, M.; CHOŁDA, P.; WAWROWSKI, M. **Continuous Debt Valuation Approach (CoDVA) for Technical Debt Prioritization**, 2020 46th Euromicro

Conference on Software Engineering and Advanced Applications (SEAA), Portoroz, Slovenia, 2020, pp. 362-366, DOI: 10.1109/SEAA51224.2020.00066, 2020.

STOCHEL, M.; WAWROWSKI, M.; RABIEJ, M. **Value-Based Technical Debt Model and Its Application.** ICSEA 2012: The Seventh International Conference on Software Engineering Advances, Lisbon, Portugal, Nov. 2012, 205-212.

STRIPE. **The Developer Coefficient Software engineering efficiency and its $3 trillion impact on global GDP**. 2018. Disponível em: <https://stripe.com/files/reports/the-developer-coefficient.pdf>. Acesso em: 22 fev. 2021.

STROGGYLOS, K.; SPINELLIS, D. **Refactoring–Does It Improve Software Quality?** 2007, Fifth International Workshop on Software Quality (WoSQ'07: ICSE Workshops 2007), p. 3-8.

THIOLLENT, M. **Metodologia da pesquisa-ação**, São Paulo, 2011.

TOM, E.; AURUM, A.; VIDGEN, R. **An exploration of technical debt.** 2013, Journal of Systems and Software, 86, n. 6, jun. 2013. 1498-1516.

WEBBER, E. **Building Successful Communities of Practice: Discover How Connecting People Makes Better Organizations**, 2016. 94.

WENGER, E., WENGER-TRAYNER, B. **Introduction to communities of practice. A brief overview of the concept and its uses.** 2015, Accessed on: Oct/30/2020, Available: http://wenger-trayner.com/wp-content/uploads/2015/04/07-Brief-introduction-to-communities-of-practice.pdf

WENGER, E; MCDERMOTT, R.; SNYDER, W. **Cultivating Communities of Practice: A Guide to Managing Knowledge.** 2002, Harvard Business Press.

WOLEK, F. **The managerial principles behind guild craftsmanship.** 199, 5(7). doi:10.1108/13552529910297460

YIN, A.; FIGUEIREDO, S.; SILVA, M. **Scrum Maturity Model: Validation for IT organizations' roadmap to develop software centered on the client role.** ICSEA 2011, Barcelona, 2011. 20-29.

YLI-HUUMO, J.; MAGLYAS, A.; SMOLANDER, K.; HALLER, J.; TÖRNROOS, H. **Developing Processes to Increase Technical Debt Visibility and Manageability – An Action Research Study in Industry.** Product-Focused Software Process Improvement. PROFES 2016, 06 nov. 2016.

YLI-HUUMO, J.; MAGLYAS, A.; SMOLANDER, K. **How do software development teams manage technical debt? An empirical study.** Journal of Systems and Software archive, 120, out. 2016. 195-218.

ZAZWORKA, N.; SHAW, M.; SHULL, F.; SEAMAN, C. **Investigating the Impact of Design Debt on Software Quality.** MTD '11 Proceedings of the 2nd Workshop on Managing Technical Debt, Waikiki, Honolulu, HI, USA, 23 maio 2011. 17-23.

ZAZWORKA, N.; SEAMAN, C.; SHULL, F. **Prioritizing design debt investment opportunities.** MTD '11 Proceedings of the 2nd Workshop on Managing Technical Debt, Waikiki, Honolulu, HI, USA, maio 2011. 39-42.

APÊNDICE A. ANÁLISE DE PRIORIDADE

Tipo	Regra	Profissionais				
		PO1	PO2	DEV1	DEV2	DEVOPS
Bug	"$this" should not be used in a static context	Alta	Alta	Grave	Grave	Alta
Bug	"&&" and "||" should be used	Alta	Alta	Grave	Grave	Alta
Bug	"exit(...)" and "die(...)" statements should not be used	Não usado	Não usado	Média	Grave	Não usado
Bug	"php_sapi_name()" should not be used	Grave	Grave	Baixa	Média	Grave
Bug	"require_once" and "include_once" should be used instead of "require" and "include"	Grave	Grave	Alta	Alta	Grave
Bug	Closing tag "?>" should be omitted on files containing only PHP	Não usado	Não usado	Não usado	Não usado	Não usado
Bug	Empty statements should be removed	Grave	Grave	Baixa	Alta	Grave
Bug	Errors should not be silenced	Baixa	Alta	Média	Grave	Baixa
Bug	Failed unit tests should be fixed	Não usado	Não usado	Média	Não usado	Não usado
Bug	Files should not contain characters before "<?php"	Média	Alta	Média	Média	Média
Bug	Files that define symbols should not cause side-effects	Não usado	Não usado	Não usado	Alta	Não usado
Bug	Function argument names should be unique	Alta	Alta	Grave	Grave	Alta
Bug	Generic exceptions ErrorException, RuntimeException and Exception should not be thrown	Grave	Grave	Alta	Média	Grave
Bug	Identical expressions should not be used on both sides of a binary operator	Grave	Grave	Média	Alta	Grave
Bug	Jump statements should not be followed by other statements	Não usado	Não usado	Média	Alta	Não usado
Bug	Method visibility should be explicitly declared	Não usado	Não usado	Não usado	Média	Não usado

Bug	Multiline blocks should be enclosed in curly braces	Baixa	Não usado	Média	Alta	Baixa
Bug	Non-empty statements should change control flow or have at least one side-effect	Grave	Grave	Média	Média	Grave
Bug	Objects should not be created to be dropped immediately without being used	Alta	Alta	Média	Alta	Alta
Bug	Related "if/else if" statements and "cases" in a "switch" should not have the same condition	Alta	Alta	Grave	Alta	Alta
Bug	Short-circuit logic should be used to prevent null pointer dereferences in conditionals	Média	Média	Grave	Alta	Média
Bug	Static members should be referenced with "static::"	Média	Média	Baixa	Grave	Média
Bug	Switch cases should end with an unconditional "break" statement	Baixa	Grave	Média	Grave	Baixa
Bug	Two branches in the same conditional structure should not have exactly the same implementation	Alta	Alta	Alta	Alta	Alta
Bug	Useless "if(true) {...}" and "if(false){...}" blocks should be removed	Alta	Alta	Grave	Grave	Alta
Bug	Variables should not be self-assigned	Alta	Alta	Baixa	Grave	Alta
Vulnerability	"allow_url_fopen" and "allow_url_include" should be disabled	Não usado	Não usado	Não usado	Não usado	Não usado
Vulnerability	"cgi.force_redirect" should be enabled	Não usado	Não usado	Não usado	Não usado	Não usado
Vulnerability	"enable_dl" should be disabled	Não usado	Não usado	Não usado	Não usado	Não usado
Vulnerability	"file_uploads" should be disabled	Não usado	Não usado	Não usado	Não usado	Não usado
Vulnerability	"open_basedir" should limit file access	Não usado	Não usado	Não usado	Não usado	Não usado
Vulnerability	"session.use_trans_sid" should not be enabled	Não usado	Não usado	Não usado	Não usado	Não usado
Vulnerability	"sleep" should not be called	Não usado	Grave	Baixa	Grave	Não usado
Vulnerability	Code should not be dynamically injected and executed	Não usado	Baixa	Média	Grave	Não usado
Vulnerability	Credentials should not be hard-coded	Não usado	Não usado	Grave	Grave	Não usado
Vulnerability	Session-management cookies should not be persistent	Não usado	Não usado	Não usado	Não usado	Não usado
Code Smell	"<?php" and "<?=" tags should be used	Não usado	Não usado	Não usado	Não usado	Não usado
Code Smell	"__construct" functions should not make PHP 4-style calls to parent constructors	Média	Média	Baixa	Média	Média

Code Smell	"elseif" keyword should be used in place of "else if" keywords	Não usado	Não usado	Baixa	Não usado	Não usado
Code Smell	"final" should not be used redundantly	Baixa	Baixa	Baixa	Baixa	Baixa
Code Smell	"for" loop stop conditions should be invariant	Alta	Alta	Média	Média	Alta
Code Smell	"global" should not be used	Não usado	Não usado	Média	Alta	Não usado
Code Smell	"goto" statement should not be used	Não usado	Não usado	Média	Alta	Não usado
Code Smell	"if ... else if" constructs should end with "else" clauses	Não usado	Não usado	Não usado	Média	Não usado
Code Smell	"switch case" clauses should not have too many lines	Não usado	Não usado	Média	Não usado	Não usado
Code Smell	"switch" statements should have at least 3 "case" clauses	Não usado	Não usado	Média	Não usado	Não usado
Code Smell	"switch" statements should not have too many "case" clauses	Não usado	Não usado	Não usado	Não usado	Não usado
Code Smell	A "while" loop should be used instead of a "for" loop	Não usado	Não usado	Baixa	Baixa	Não usado
Code Smell	A close curly brace should be located at the beginning of a line	Não usado	Não usado	Baixa	Não usado	Não usado
Code Smell	Alias functions should not be used	Média	Média	Não usado	Baixa	Média
Code Smell	An open curly brace should be located at the beginning of a line	Não usado	Não usado	Não usado	Não usado	Não usado
Code Smell	An open curly brace should be located at the end of a line	Não usado	Não usado	Baixa	Não usado	Não usado
Code Smell	Boolean literals should not be redundant	Não usado	Não usado	Baixa	Média	Não usado
Code Smell	Branches should have sufficient coverage by tests	Não usado	Não usado	Não usado	Não usado	Não usado
Code Smell	Class constructors should not create other objects	Não usado	Não usado	Não usado	Baixa	Não usado
Code Smell	Class names should comply with a naming convention	Não usado	Não usado	Não usado	Não usado	Não usado
Code Smell	Classes should not be coupled to too many other classes (Single Responsibility Principle)	Não usado	Não usado	Alta	Média	Não usado

Code Smell	Classes should not be too complex	Não usado	Não usado	Alta	Alta	Não usado
Code Smell	Classes should not have too many fields	Não usado	Não usado	Alta	Média	Não usado
Code Smell	Classes should not have too many lines	Não usado	Não usado	Alta	Média	Não usado
Code Smell	Classes should not have too many methods	Não usado	Não usado	Não usado	Média	Não usado
Code Smell	Cognitive Complexity of functions should not be too high	Não usado	Não usado	Alta	Média	Não usado
Code Smell	Collapsible "if" statements should be merged	Não usado	Não usado	Média	Média	Não usado
Code Smell	Colors should be defined in upper case	Não usado	Não usado	Não usado	Não usado	Não usado
Code Smell	Comments should not be located at the end of lines of code	Não usado	Não usado	Não usado	Não usado	Não usado
Code Smell	Configuration should not be changed dynamically	Média	Média	Não usado	Baixa	Média
Code Smell	Constant names should comply with a naming convention	Não usado	Não usado	Não usado	Não usado	Não usado
Code Smell	Control flow statements "if", "for", "while", "switch" and "try" should not be nested too deeply	Não usado	Não usado	Alta	Alta	Não usado
Code Smell	Control structures should use curly braces	Não usado	Não usado	Baixa	Média	Não usado
Code Smell	Deprecated predefined variables should not be used	Média	Média	Não usado	Média	Média
Code Smell	Expressions should not be too complex	Não usado	Não usado	Alta	Média	Não usado
Code Smell	Field names should comply with a naming convention	Não usado	Não usado	Não usado	Não usado	Não usado
Code Smell	File names should comply with a naming convention	Não usado	Não usado	Não usado	Não usado	Não usado
Code Smell	Files should contain an empty new line at the end	Não usado	Não usado	Não usado	Não usado	Não usado
Code Smell	Files should contain only one top-level class or interface each	Alta	Alta	Não usado	Alta	Alta
Code Smell	Files should not contain inline HTML	Baixa	Baixa	Baixa	Alta	Baixa

Code Smell	Files should not have too many lines	Não usado	Não usado	Alta	Média	Não usado
Code Smell	Function names should comply with a naming convention	Não usado	Não usado	Não usado	Não usado	Não usado
Code Smell	Functions and variables should not be defined outside of classes	Não usado	Não usado	Não usado	Não usado	Não usado
Code Smell	Functions deprecated in PHP 5 should not be used	Grave	Grave	Baixa	Média	Grave
Code Smell	Functions should not be nested too deeply	Não usado	Não usado	Grave	Média	Não usado
Code Smell	Functions should not be too complex	Não usado	Não usado	Grave	Alta	Não usado
Code Smell	Functions should not contain too many return statements	Não usado	Não usado	Média	Média	Não usado
Code Smell	Functions should not have too many lines	Não usado	Não usado	Grave	Média	Não usado
Code Smell	Functions should not have too many parameters	Não usado	Não usado	Média	Média	Não usado
Code Smell	Increment (++) and decrement (--) operators should not be used in a method call or mixed with other operators in an expression	Não usado	Não usado	Baixa	Média	Não usado
Code Smell	Interface names should comply with a naming convention	Não usado	Não usado	Não usado	Não usado	Não usado
Code Smell	Lines should have sufficient coverage by tests	Não usado	Não usado	Não usado	Não usado	Não usado
Code Smell	Lines should not be too long	Não usado	Não usado	Média	Baixa	Não usado
Code Smell	Lines should not end with trailing whitespaces	Não usado	Não usado	Baixa	Não usado	Não usado
Code Smell	Local variable and function parameter names should comply with a naming convention	Não usado	Não usado	Não usado	Não usado	Não usado
Code Smell	Local Variables should not be declared and then immediately returned or thrown	Não usado	Não usado	Baixa	Baixa	Não usado
Code Smell	Local variables should not have the same name as class fields	Média	Média	Média	Média	Média
Code Smell	Method arguments with default values should be last	Média	Média	Alta	Baixa	Média
Code Smell	Modifiers should be declared in the correct order	Não usado	Não usado	Baixa	Não usado	Não usado
Code Smell	More than one property should not be declared per statement	Não usado	Não usado	Baixa	Não usado	Não usado

Code Smell	Nested blocks of code should not be left empty	Grave	Grave	Média	Média	Grave
Code Smell	Only LF character (Unix-like) should be used to end lines	Não usado	Não usado	Não usado	Não usado	Não usado
Code Smell	Overriding methods should do more than simply call the same method in the super class	Baixa	Baixa	Média	Baixa	Baixa
Code Smell	Parentheses should not be used for calls to "echo"	Alta	Alta	Não usado	Baixa	Alta
Code Smell	Perl-style comments should not be used	Baixa	Baixa	Não usado	Não usado	Baixa
Code Smell	PHP 4 constructor declarations should not be used	Não usado	Não usado	Não usado	Baixa	Não usado
Code Smell	PHP keywords and constants "true", "false", "null" should be lower case	Não usado	Não usado	Não usado	Baixa	Não usado
Code Smell	PHP parser failure	Não usado	Não usado	Não usado	Média	Não usado
Code Smell	References should not be passed to function calls	Alta	Alta	Baixa	Média	Alta
Code Smell	Return of boolean expressions should not be wrapped into an "if-then-else" statement	Não usado	Não usado	Baixa	Média	Não usado
Code Smell	Sections of code should not be "commented out"	Média	Média	Média	Não usado	Média
Code Smell	Skipped unit tests should be either removed or fixed	Não usado	Não usado	Não usado	Não usado	Não usado
Code Smell	Source code should comply with formatting standards	Não usado	Não usado	Não usado	Não usado	Não usado
Code Smell	Source files should have a sufficient density of comment lines	Não usado	Não usado	Não usado	Não usado	Não usado
Code Smell	Source files should not have any duplicated blocks	Não usado	Não usado	Média	Alta	Não usado
Code Smell	Statements should be on separate lines	Não usado	Não usado	Baixa	Baixa	Não usado
Code Smell	Statements should end with a "case default" clause	Não usado	Não usado	Não usado	Média	Não usado
Code Smell	String literals should not be concatenated	Não usado	Não usado	Não usado	Não usado	Não usado
Code Smell	String literals should not be duplicated	Não usado	Não usado	Baixa	Não usado	Não usado

Code Smell	Superglobals should not be accessed directly	Grave	Grave	Não usado	Alta	Grave
Code Smell	Tabulation characters should not be used	Não usado	Não usado	Não usado	Não usado	Não usado
Code Smell	The "var" keyword should not be used	Baixa	Baixa	Não usado	Baixa	Baixa
Code Smell	The names of methods with boolean return values should start with "is" or "has"	Não usado	Não usado	Baixa	Não usado	Não usado
Code Smell	Track lack of copyright and license headers	Não usado	Não usado	Não usado	Não usado	Não usado
Code Smell	Track uses of "FIXME" tags	Não usado	Não usado	Não usado	Baixa	Não usado
Code Smell	Track uses of "TODO" tags	Não usado	Não usado	Não usado	Baixa	Não usado
Code Smell	Unused "private" fields should be removed	Alta	Alta	Alta	Média	Alta
Code Smell	Unused "private" methods should be removed	Alta	Alta	Alta	Média	Alta
Code Smell	Unused function parameters should be removed	Alta	Alta	Média	Média	Alta
Code Smell	Unused local variables should be removed	Alta	Alta	Média	Média	Alta
Code Smell	Variable variables should not be used	Alta	Alta	Alta	Média	Alta
Code Smell	SQL commands should not be used in this folder	Alta	Alta	Alta	Alta	Alta
Code Smell	Check use $_REQUEST in folder include.	Alta	Alta	Alta	Alta	Alta

APÊNDICE B. PRIORIDADE, COMPLEXIDADE, IMPACTO E ESFORÇO DAS REGRAS DE QUALIDADE

Tipo	Regra	Prioridade	Complexidade	Impacto	Esforço	Esforço ajustado
Bug	"$this" should not be used in a static context	Grave	Low	Very low	5	1
Bug	"&&" and "||" should be used	Grave	Very low	Very low	3	0,1
Bug	Closing tag "?>" should be omitted on files containing only PHP	Não usado				
Bug	Empty statements should be removed	Grave	Low	Very low	5	1
Bug	Errors should not be silenced	Média	High	Very High	25	5
Bug	"exit(...)" and "die(...)" statements should not be used	Alta	Moderate	High	20	4
Bug	Failed unit tests should be fixed	Não usado				
Bug	Files should not contain characters before "<?php"	Alta	Low	Very low	5	1
Bug	Files that define symbols should not cause side-effects	Não usado				
Bug	Function argument names should be unique	Grave	Low	High	13	3
Bug	Generic exceptions ErrorException, RuntimeException and Exception should not be thrown	Grave	Low	Low	8	2
Bug	Identical expressions should not be used on both sides of a binary operator	Grave	Low	Very low	5	1
Bug	Jump statements should not be followed by other statements	Alta	Low	Very low	5	1
Bug	Method visibility should be explicitly declared	Não usado				
Bug	Multiline blocks should be enclosed in curly braces	Média	Low	Moderate	8	2
Bug	Non-empty statements should change control flow or have at least one side-effect	Grave	Moderate	Moderate	13	1
Bug	Objects should not be created to be dropped immediately without being used	Alta	Low	Moderate	8	2
Bug	"php_sapi_name()" should not be used	Grave	Low	Very low	5	1

Bug	Related "if/else if" statements and "cases" in a "switch" should not have the same condition	Alta	Moderate	Low	10	2
Bug	"require_once" and "include_once" should be used instead of "require" and "include"	Grave	Low	Very low	5	1
Bug	Short-circuit logic should be used to prevent null pointer dereferences in conditionals	Alta	Low	Very low	5	1
Bug	Static members should be referenced with "static::"	Alta	Low	Moderate	8	2
Bug	Switch cases should end with an unconditional "break" statement	Alta	Moderate	Moderate	13	3
Bug	Two branches in the same conditional structure should not have exactly the same implementation	Alta	Moderate	Low	10	2
Bug	Useless "if(true) {...}" and "if(false){...}" blocks should be removed	Grave	Low	Very low	5	1
Bug	Variables should not be self-assigned	Alta	Low	Very low	5	1
Vulnerability	"allow_url_fopen" and "allow_url_include" should be disabled	Grave	High	Moderate	15	4
Vulnerability	"cgi.force_redirect" should be enabled	Grave	Low	Very low	5	1
Vulnerability	Code should not be dynamically injected and executed	Alta	Very High	Very High	30	5
Vulnerability	Credentials should not be hard-coded	Grave	Very High	Very High	30	5
Vulnerability	"enable_dl" should be disabled	Grave	Low	Very low	5	1
Vulnerability	"file_uploads" should be disabled	Grave	Low	Very low	5	1
Vulnerability	"open_basedir" should limit file access	Grave	High	Moderate	15	4
Vulnerability	"session.use_trans_sid" should not be enabled	Grave	Low	Very low	5	1
Vulnerability	Session-management cookies should not be persistent	Grave	Low	Very low	5	1
Vulnerability	"sleep" should not be called	Grave	Moderate	High	20	4
Code Smell	"__construct" functions should not make PHP 4-style calls to parent constructors	Alta	Low	Very low	5	1
Code Smell	"<?php" and "<?=" tags should be used	Não usado				
Code Smell	A "while" loop should be used instead of a "for" loop	Alta	Low	Very low	5	1
Code Smell	A close curly brace should be located at the beginning of a line	Média	Very low	Very low	3	0,1
Code Smell	Alias functions should not be used	Média	Low	Very low	5	1
Code Smell	An open curly brace should be located at the beginning of a line	Não usado				
Code Smell	An open curly brace should be located at the end of a line	Não usado				
Code Smell	Boolean literals should not be redundant	Baixa				

Code Smell	Branches should have sufficient coverage by tests	Não usado				
Code Smell	Class constructors should not create other objects	Não usado				
Code Smell	Class names should comply with a naming convention	Não usado				
Code Smell	Classes should not be coupled to too many other classes (Single Responsibility Principle)	Baixa				
Code Smell	Classes should not be too complex	Não usado				
Code Smell	Classes should not have too many fields	Baixa				
Code Smell	Classes should not have too many lines	Baixa				
Code Smell	Classes should not have too many methods	Baixa				
Code Smell	Cognitive Complexity of functions should not be too high	Baixa				
Code Smell	Collapsible "if" statements should be merged	Baixa				
Code Smell	Colors should be defined in upper case	Baixa				
Code Smell	Comments should not be located at the end of lines of code	Não usado				
Code Smell	Configuration should not be changed dynamically	Alta	High	High	20	4
Code Smell	Constant names should comply with a naming convention	Não usado				
Code Smell	Control flow statements "if", "for", "while", "switch" and "try" should not be nested too deeply	Baixa				
Code Smell	Control structures should use curly braces	Média	Very low	Very low	3	0,1
Code Smell	Deprecated predefined variables should not be used	Grave	Moderate	Low	8	2
Code Smell	"elseif" keyword should be used in place of "else if" keywords	Não usado				
Code Smell	Expressions should not be too complex	Baixa				
Code Smell	Field names should comply with a naming convention	Não usado				
Code Smell	File names should comply with a naming convention	Não usado				
Code Smell	Files should contain an empty new line at the end	Não usado				
Code Smell	Files should contain only one top-level class or interface each	Alta	Moderate	High	20	4
Code Smell	Files should not contain inline HTML	Média	Moderate	Moderate	13	13
Code Smell	Files should not have too many lines	Baixa				
Code Smell	"final" should not be used redundantly	Alta	Low	Very low	5	1
Code Smell	"for" loop stop conditions should be invariant	Alta	High	Moderate	15	10
Code Smell	Function names should comply with a naming convention	Não usado				

Code Smell	Functions and variables should not be defined outside of classes	Não usado				
Code Smell	Functions deprecated in PHP 5 should not be used	Grave	Moderate	Moderate	13	3
Code Smell	Functions should not be nested too deeply	Grave	Very High	Very High	30	30
Code Smell	Functions should not be too complex	Baixa				
Code Smell	Functions should not contain too many return statements	Não usado				
Code Smell	Functions should not have too many lines	Baixa				
Code Smell	Functions should not have too many parameters	Baixa				
Code Smell	"global" should not be used	Não usado				
Code Smell	"goto" statement should not be used	Grave	Moderate	Low	10	2
Code Smell	"if ... else if" constructs should end with "else" clauses	Não usado				
Code Smell	Increment (++) and decrement (--) operators should not be used in a method call or mixed with other operators in an expression	Baixa				
Code Smell	Interface names should comply with a naming convention	Não usado				
Code Smell	Lines should have sufficient coverage by tests	Não usado				
Code Smell	Lines should not be too long	Não usado				
Code Smell	Lines should not end with trailing whitespaces	Não usado				
Code Smell	Local variable and function parameter names should comply with a naming convention	Não usado				
Code Smell	Local Variables should not be declared and then immediately returned or thrown	Baixa				
Code Smell	Local variables should not have the same name as class fields	Média	Moderate	Very low	8	2
Code Smell	Method arguments with default values should be last	Média	Moderate	High	20	10
Code Smell	Modifiers should be declared in the correct order	Baixa				
Code Smell	More than one property should not be declared per statement	Baixa				
Code Smell	Nested blocks of code should not be left empty	Grave	Low	Very low	5	1
Code Smell	Only LF character (Unix-like) should be used to end lines	Não usado				
Code Smell	Overriding methods should do more than simply call the same method in the super class	Média	Low	Moderate	8	2
Code Smell	Parentheses should not be used for calls to "echo"	Alta	Low	Very low	5	1
Code Smell	Perl-style comments should not be used	Baixa				
Code Smell	PHP 4 constructor declarations should not be used	Baixa				

Code Smell	PHP keywords and constants "true", "false", "null" should be lower case	Baixa				
Code Smell	PHP parser failure	Alta	Very High	Very High	30	30
Code Smell	References should not be passed to function calls	Alta	Low	Low	8	2
Code Smell	Return of boolean expressions should not be wrapped into an "if-then-else" statement	Não usado				
Code Smell	Sections of code should not be "commented out"	Média	Low	Very low	5	1
Code Smell	Skipped unit tests should be either removed or fixed	Não usado				
Code Smell	Source code should comply with formatting standards	Média	Very low	Very low	3	0,1
Code Smell	Source files should have a sufficient density of comment lines	Baixa				
Code Smell	Source files should not have any duplicated blocks	Não usado				
Code Smell	Statements should be on separate lines	Média	Very low	Very low	3	0,1
Code Smell	Statements should end with a "case default" clause	Não usado				
Code Smell	String literals should not be concatenated	Baixa				
Code Smell	String literals should not be duplicated	Não usado				
Code Smell	Superglobals should not be accessed directly	Grave	Low	Very low	5	1
Code Smell	"switch case" clauses should not have too many lines	Baixa				
Code Smell	"switch" statements should have at least 3 "case" clauses	Não usado				
Code Smell	"switch" statements should not have too many "case" clauses	Baixa				
Code Smell	Tabulation characters should not be used	Média	Very low	Very low	3	0,1
Code Smell	The "var" keyword should not be used	Média	Very low	Very low	3	0,1
Code Smell	The names of methods with boolean return values should start with "is" or "has"	Não usado				
Code Smell	Track lack of copyright and license headers	Não usado				
Code Smell	Track uses of "FIXME" tags	Baixa				
Code Smell	Track uses of "TODO" tags	Baixa				
Code Smell	Unused "private" fields should be removed	Alta	Low	Very low	5	1
Code Smell	Unused "private" methods should be removed	Alta	Low	Very low	5	1
Code Smell	Unused function parameters should be removed	Alta	Low	Very low	5	1
Code Smell	Unused local variables should be removed	Alta	Low	Very low	5	1
Code Smell	Variable variables should not be used	Alta	Low	Very low	5	1

Code Smell	SQL commands should not be used in this folder	Alta	Very High	Very High	30	5
Code Smell	Check use $_REQUEST in folder include.	Alta	Very High	Very High	30	5

APÊNDICE C. LISTA DE DÍVIDA TÉCNICA DO 2º CICLO

Tipo de Dívida	Dívida Técnica	Regra	Prioridade	Complexidade	Impacto	Esforço	Esforço ajustado	IDT
Código	Duplicação de código	Methods should not have identical implementations	Baixa	Média	Muito baixo	8	2	219
		Source files should not have any duplicated blocks	Baixa	Baixa	Muito baixo	5	1	0
	Função depreciada ou incompatível	"php_sapi_name()" should not be used	Grave	Baixa	Muito baixo	5	1	1
		__construct functions should not make PHP 4-style calls to parent constructors	Alta	Baixa	Muito baixo	5	1	102
		Alias functions should not be used	Baixa	Baixa	Muito baixo	5	1	319
		declaração "goto" não deve ser usada	Grave	Média	Baixo	10	2	0
		Deprecated functions should not be used	Média	Baixa	Médio	8	2	47
		Deprecated predefined variables should not be used	Média	Média	Baixo	8	2	0
		Perl-style comments should not be used	Baixa	Baixa	Muito baixo	5	1	6.184
		PHP 4 constructor declarations should not be used	Baixa	Baixa	Muito baixo	5	1	6.427
		The "var" keyword should not be used	Média	Muito baixa	Muito baixo	3	0,1	5.190
	Legibilidade do código	"elseif" keyword should be used in place of "else if" keywords	Média	Muito baixa	Muito baixo	3	0,1	4.465
		"for" loop stop conditions should be invariant	Alta	Alta	Médio	15	10	40
		"switch" statements should have at least 3 "case" clauses	Baixa	Muito baixa	Muito baixo	3	0,1	710
		Assignments should not be made from within sub-expressions	Média	Baixa	Muito baixo	5	1	1.813
		Boolean literals should not be redundant	Baixa	Baixa	Muito baixo	5	1	3.841

		Collapsible "if" statements should be merged	Baixa	Baixa	Médio	8	2	1.266
		Duplicate values should not be passed as arguments	Média	Média	Médio	13	3	365
		Functions should use "return" consistently	Média	Baixa	Muito baixo	5	1	362
		Increment (++) and decrement (--) operators should not be used in a method call	Baixa	Baixa	Muito baixo	5	1	2.912
		Jump statements should not be followed by dead code	Alta	Baixa	Muito baixo	5	1	1.230
		Local variables should not have the same name as class fields	Baixa	Média	Muito baixo	8	2	1.209
		Redundant pairs of parentheses should be removed	Baixa	Baixa	Muito baixo	5	1	670
		References should not be passed to function calls	Alta	Baixa	Baixo	8	2	29
		Sections of code should not be "commented out"	Média	Baixa	Muito baixo	5	1	4.171
		String literals should not be concatenated	Baixa	Baixa	Muito baixo	5	1	3.011
		Switch cases should end with an unconditional "break" statement	Média	Média	Médio	13	3	36
	Violação do padrão de formatação	"switch case" clauses should not have too many lines of code	Baixa	Média	Médio	13	3	719
		A close curly brace should be located at the beginning of a line	Baixa	Muito baixa	Muito baixo	3	0,1	29
		Colors should be defined in upper case	Baixa	Baixa	Muito baixo	5	1	22
		Conditionally executed code should be denoted by either indentation or curly braces	Baixa	Baixa	Médio	8	2	286
		Conditionals should start on new lines	Média	Baixa	Muito baixo	5	1	80
		Control structures should use curly braces	Baixa	Muito baixa	Muito baixo	3	0,1	18.446
		Files should contain an empty newline at the end	Média	Muito baixa	Muito baixo	3	0,1	13.339
		Lines should not end with trailing whitespaces	Média	Muito baixa	Muito baixo	3	0,1	36.337

		Modifiers should be declared in the correct order	Baixa	Baixa	Muito baixo	5	1	114
		More than one property should not be declared per statement	Média	Baixa	Muito baixo	5	1	36
		Multiline blocks should be enclosed in curly braces	Alta	Baixa	Médio	8	2	470
		PHP keywords and constants "true", "false", "null" should be lower case	Média	Baixa	Muito baixo	5	1	987
		Source code should comply with formatting standards	Média	Muito baixa	Muito baixo	3	0,1	390.794
		Statements should be on separate lines	Média	Muito baixa	Muito baixo	3	0,1	3.539
	Violação do padrão de nomeclatura	[Customized] Classnames in /include/exp/* should comply with the naming convention	Alta	Média	Alto	20	20	82
		[Customized] Function names in /include/exp/* should comply with the naming convention	Alta	Média	Alto	20	20	3
		File names should comply with a naming convention	Média	Média	Alto	20	20	463
Defeito	Código inadequado	Array or Countable object count comparisons should make sense	Grave	Baixa	Muito baixo	5	1	0
		Empty statements should be removed	Grave	Baixa	Muito baixo	5	1	43
		Errors should not be silenced	Alta	Alta	Muito alto	25	5	467
		Identical expressions should not be used on both sides of a binary operator	Grave	Baixa	Muito baixo	5	1	14
		Local variables should not be declared and then immediately returned or thrown	Baixa	Baixa	Muito baixo	5	1	2.651
		Nested blocks of code should not be left empty	Alta	Baixa	Muito baixo	5	1	46
		Non-empty statements should change control flow or have at least one side-effect	Grave	Média	Médio	13	1	16
		Objects should not be created to be dropped immediately without being used	Grave	Baixa	Médio	8	2	70
		Parentheses should not be used for calls to "echo"	Média	Baixa	Muito baixo	5	1	211

		Related "if/else if" statements and "cases" in a "switch" should not have the same condition	Grave	Média	Baixo	10	2	15
		Return of boolean expressions should not be wrapped into an "if-then-else" statement	Baixa	Baixa	Muito baixo	5	1	159
		Return values from functions without side effects should not be ignored	Grave	Baixa	Muito baixo	5	1	31
		The output of functions that don't return anything should not be used	Alta	Baixa	Muito baixo	5	1	0
		Useless "if(true) {...}" and "if(false){...}" blocks should be removed	Grave	Baixa	Muito baixo	5	1	5
		Values should not be uselessly incremented	Grave	Baixa	Muito baixo	5	1	1
	Defeito encontrado no código	"$this" should not be used in a static context	Grave	Baixa	Muito baixo	5	1	6
		Exception should not be created without being thrown	Grave	Baixa	Muito baixo	5	1	0
		PHP parser failure	Grave	Muito alta	Muito alto	30	30	2
	Possibilidade de resultado inesperado	"&&" and "||" should be used	Média	Muito baixa	Muito baixo	3	0,1	278
		"=+" should not be used instead of "+="	Grave	Baixa	Muito baixo	5	1	3
		"exit(...)" and "die(...)" statements should not be used	Alta	Média	Alto	20	4	573
		"require_once" and "include_once" should be used instead of "require" and "include"	Grave	Baixa	Muito baixo	5	1	259
		A "for" loop update clause should move the counter in the right direction	Grave	Baixa	Médio	8	2	3
		All branches in a conditional structure should not have exactly the same implementation	Grave	Baixa	Médio	8	2	49
		Files should not contain characters before "<?php"	Grave	Baixa	Muito baixo	5	1	140
		Function argument names should be unique	Grave	Baixa	Alto	13	3	0
		Short-circuit logic should be used to prevent null pointer dereferences in conditionals	Alta	Baixa	Muito baixo	5	1	0

		Static members should be referenced with "static::"	Alta	Baixa	Médio	8	2	705
		Track uses of "FIXME" tags	Média	Baixa	Muito baixo	5	1	25
		Variables should be initialized before use	Grave	Baixa	Muito baixo	5	1	231
		Variables should not be self-assigned	Grave	Baixa	Muito baixo	5	1	41
	Vulnerabilidade de segurança	"sleep" should not be called	Grave	Média	Alto	20	4	5
		allow_url_fopen and "allow_url_include" should be disabled	Grave	Alta	Médio	15	4	0
		cgi.force_redirect should be enabled	Grave	Baixa	Muito baixo	5	1	0
		Code should not be dynamically injected and executed	Grave	Muito alta	Muito alto	30	5	132
		Configuration should not be changed dynamically	Alta	Alta	Alto	20	4	28
		Credentials should not be hard-coded	Grave	Muito alta	Muito alto	30	5	2
		enable_dl should be disabled	Grave	Baixa	Muito baixo	5	1	0
		open_basedir should limit file access	Grave	Alta	Médio	15	4	0
		session.use_trans_sid should not be enabled	Grave	Baixa	Muito baixo	5	1	0
		Session-management cookies should not be persistent	Grave	Baixa	Muito baixo	5	1	0
Design	Código muito complexo	Classes should not be too complex	Baixa	Muito alta	Muito alto	30	30	66
		Cognitive Complexity of functions should not be too high	Baixa	Muito alta	Muito alto	30	30	1.311
		Expressions should not be too complex	Baixa	Média	Médio	13	3	147
		Functions should not be too complex	Baixa	Muito alta	Muito alto	30	30	554
	Violação de padrão de codificação	"catch" clauses should do more than rethrow	Baixa	Baixa	Muito baixo	5	1	0
		"final" should not be used redundantly	Alta	Baixa	Muito baixo	5	1	73
		"switch" statements should not have too many "case" clauses	Baixa	Média	Alto	20	4	64
		[Customized] Check for $_REQUEST use in folder /include/*	Alta	Muito alta	Muito alto	30	5	5.079
		[Customized] Remove the extra characters before the opening tag.	Baixa	Baixa	Muito baixo	5	1	747
		[Customized] SQL commands should not be used in this folder	Média	Muito alta	Muito alto	30	30	7.061

		<?php and "<?=" tags should be used	Média	Baixa	Muito baixo	5	1	13.029
		A "while" loop should be used instead of a "for" loop	Média	Baixa	Muito baixo	5	1	0
		Class constructors should not create other objects	Baixa	Baixa	Muito baixo	5	1	3.489
		Classes should not be coupled to too many other classes (Single Responsibility Principle)	Baixa	Média	Médio	13	3	37
		Classes should not have too many fields	Baixa	Muito alta	Muito alto	30	30	100
		Classes should not have too many lines of code	Baixa	Muito alta	Muito alto	30	30	186
		Classes should not have too many methods	Baixa	Muito alta	Muito alto	30	30	290
		Closing tag "?>" should be omitted on files containing only PHP	Média	Baixa	Muito baixo	5	1	8.586
		Control flow statements "if", "for", "while", "switch" and "try" should not be nested too deeply	Baixa	Alta	Alto	20	4	225
		Files should contain only one top-level class or interface each	Alta	Média	Alto	20	4	82
		Files should not have too many lines of code	Baixa	Muito alta	Muito alto	30	30	56
		Functions should not be nested too deeply	Grave	Muito alta	Muito alto	30	30	0
		Functions should not have too many lines of code	Baixa	Muito alta	Muito alto	30	30	102
		Functions should not have too many parameters	Baixa	Média	Médio	13	13	76
		Generic exceptions ErrorException, RuntimeException and Exception should not be thrown	Média	Baixa	Baixo	8	2	46
		Lines should not be too long	Baixa	Média	Alto	20	4	26.527
		Method arguments with default values should be last	Média	Média	Alto	20	10	184
		Overriding methods should do more than simply call the same method in the super class	Baixa	Baixa	Médio	8	2	675

		Superglobals should not be accessed directly	Grave	Baixa	Muito baixo	5	1	997
		Track uses of "NOSONAR" comments	Alta	Baixa	Muito baixo	5	1	5
		Two branches in a conditional structure should not have exactly the same implementation	Média	Média	Baixo	10	2	726
		Unused "private" fields should be removed	Média	Baixa	Muito baixo	5	1	86
		Unused "private" methods should be removed	Média	Baixa	Muito baixo	5	1	69
		Unused function parameters should be removed	Média	Baixa	Muito baixo	5	1	1.041
		Unused local variables should be removed	Média	Baixa	Muito baixo	5	1	6.329
		Variable variables should not be used	Alta	Baixa	Muito baixo	5	1	12
		Method visibility should be explicitly declared	Média	Baixa	Muito baixo	5	1	17.028
Documentação	Documentação inexistente ou inadequada	[Customized] Functions declarations in /include/* should be properly documented in docblock format	Média	Baixa	Baixo	8	8	99.828
		Comments should not be located at the end of lines of code	Média	Baixa	Muito baixo	5	1	33.982
		Source files should have a sufficient density of comment lines	Baixa	Baixa	Muito baixo	5	1	14.818

APÊNDICE D. ITENS DE DÍVIDA TÉCNICA POR EQUIPE

Equipe	Tipo de Dívida	Dívida Técnica	Regra	Prioridade	IDT
T01	Código	Duplicação de código	Methods should not have identical implementations	Baixa	51
T01	Código	Função depreciada ou incompatível	__construct functions should not make PHP 4-style calls to parent constructors	Alta	8
T01	Código	Função depreciada ou incompatível	"php_sapi_name()" should not be used	Grave	1
T01	Código	Função depreciada ou incompatível	Alias functions should not be used	Baixa	60
T01	Código	Função depreciada ou incompatível	Deprecated functions should not be used	Média	11
T01	Código	Função depreciada ou incompatível	Perl-style comments should not be used	Baixa	664
T01	Código	Função depreciada ou incompatível	PHP 4 constructor declarations should not be used	Baixa	945
T01	Código	Função depreciada ou incompatível	The "var" keyword should not be used	Média	102
T01	Código	Legibilidade do código	"elseif" keyword should be used in place of "else if" keywords	Média	958
T01	Código	Legibilidade do código	"for" loop stop conditions should be invariant	Alta	19
T01	Código	Legibilidade do código	"switch" statements should have at least 3 "case" clauses	Baixa	119
T01	Código	Legibilidade do código	Assignments should not be made from within sub-expressions	Média	294
T01	Código	Legibilidade do código	Boolean literals should not be redundant	Baixa	1.255
T01	Código	Legibilidade do código	Collapsible "if" statements should be merged	Baixa	323
T01	Código	Legibilidade do código	Duplicate values should not be passed as arguments	Média	100
T01	Código	Legibilidade do código	Functions should use "return" consistently	Média	30
T01	Código	Legibilidade do código	Increment (++) and decrement (--) operators should not be used in a method call	Baixa	143
T01	Código	Legibilidade do código	Jump statements should not be followed by dead code	Alta	11
T01	Código	Legibilidade do código	Local variables should not have the same name as class fields	Baixa	321
T01	Código	Legibilidade do código	References should not be passed to function calls	Alta	1
T01	Código	Legibilidade do código	Sections of code should not be "commented out"	Média	831
T01	Código	Legibilidade do código	String literals should not be concatenated	Baixa	929

T01	Código	Legibilidade do código	Switch cases should end with an unconditional "break" statement	Média	6
T01	Código	Violação do padrão de formatação	"switch case" clauses should not have too many lines of code	Baixa	134
T01	Código	Violação do padrão de formatação	A close curly brace should be located at the beginning of a line	Baixa	12
T01	Código	Violação do padrão de formatação	Colors should be defined in upper case	Baixa	4
T01	Código	Violação do padrão de formatação	Conditionally executed code should be denoted by either indentation or curly braces	Baixa	5
T01	Código	Violação do padrão de formatação	Conditionals should start on new lines	Média	19
T01	Código	Violação do padrão de formatação	Control structures should use curly braces	Baixa	467
T01	Código	Violação do padrão de formatação	Files should contain an empty newline at the end	Média	965
T01	Código	Violação do padrão de formatação	Lines should not end with trailing whitespaces	Média	457
T01	Código	Violação do padrão de formatação	Modifiers should be declared in the correct order	Baixa	18
T01	Código	Violação do padrão de formatação	More than one property should not be declared per statement	Média	2
T01	Código	Violação do padrão de formatação	Multiline blocks should be enclosed in curly braces	Alta	11
T01	Código	Violação do padrão de formatação	PHP keywords and constants "true", "false", "null" should be lower case	Média	152
T01	Código	Violação do padrão de formatação	Source code should comply with formatting standards	Média	9.008
T01	Código	Violação do padrão de formatação	Statements should be on separate lines	Média	63
T01	Código	Violação do padrão de nomenclatura	[Customized] Classnames in /include/exp/* should comply with the naming convention	Alta	9
T01	Código	Violação do padrão de nomenclatura	[Customized] Function names in /include/exp/* should comply with the naming convention	Alta	1
T01	Código	Violação do padrão de nomenclatura	File names should comply with a naming convention	Média	62
T01	Defeito	Código inadequado	Errors should not be silenced	Alta	254
T01	Defeito	Código inadequado	Identical expressions should not be used on both sides of a binary operator	Grave	1
T01	Defeito	Código inadequado	Local variables should not be declared and then immediately returned or thrown	Baixa	479
T01	Defeito	Código inadequado	Nested blocks of code should not be left empty	Alta	3
T01	Defeito	Código inadequado	Non-empty statements should change control flow or have at least one side-effect	Grave	2
T01	Defeito	Código inadequado	Objects should not be created to be dropped immediately without being used	Grave	1

T01	Defeito	Código inadequado	Parentheses should not be used for calls to "echo"	Média	54
T01	Defeito	Código inadequado	Related "if/else if" statements and "cases" in a "switch" should not have the same condition	Grave	2
T01	Defeito	Código inadequado	Return of boolean expressions should not be wrapped into an "if-then-else" statement	Baixa	28
T01	Defeito	Código inadequado	Return values from functions without side effects should not be ignored	Grave	2
T01	Defeito	Código inadequado	Values should not be uselessly incremented	Grave	1
T01	Defeito	Defeito encontrado no código	"$this" should not be used in a static context	Grave	1
T01	Defeito	Defeito encontrado no código	PHP parser failure	Grave	1
T01	Defeito	Possibilidade de resultado inesperado	"&&" and "||" should be used	Média	1
T01	Defeito	Possibilidade de resultado inesperado	"exit(...)" and "die(...)" statements should not be used	Alta	205
T01	Defeito	Possibilidade de resultado inesperado	"require_once" and "include_once" should be used instead of "require" and "include"	Grave	87
T01	Defeito	Possibilidade de resultado inesperado	All branches in a conditional structure should not have exactly the same implementation	Grave	12
T01	Defeito	Possibilidade de resultado inesperado	Files should not contain characters before "<?php"	Grave	19
T01	Defeito	Possibilidade de resultado inesperado	Static members should be referenced with "static::"	Alta	5
T01	Defeito	Possibilidade de resultado inesperado	Track uses of "FIXME" tags	Média	3
T01	Defeito	Possibilidade de resultado inesperado	Variables should be initialized before use	Grave	60
T01	Defeito	Possibilidade de resultado inesperado	Variables should not be self-assigned	Grave	11
T01	Defeito	Vulnerabilidade de segurança	"sleep" should not be called	Grave	3
T01	Defeito	Vulnerabilidade de segurança	Code should not be dynamically injected and executed	Grave	27
T01	Defeito	Vulnerabilidade de segurança	Configuration should not be changed dynamically	Alta	5
T01	Defeito	Vulnerabilidade de segurança	Credentials should not be hard-coded	Grave	2
T01	*Design*	Código muito complexo	Classes should not be too complex	Baixa	20
T01	*Design*	Código muito complexo	Cognitive Complexity of functions should not be too high	Baixa	266
T01	*Design*	Código muito complexo	Expressions should not be too complex	Baixa	24
T01	*Design*	Código muito complexo	Functions should not be too complex	Baixa	97
T01	Defeito	Legibilidade do código	Redundant pairs of parentheses should be removed	Baixa	123
T01	*Design*	Violação de padrão de codificação	<?php and "<?=" tags should be used	Média	1.872
T01	*Design*	Violação de padrão de codificação	"final" should not be used redundantly	Alta	16
T01	*Design*	Violação de padrão de codificação	"switch" statements should not have too many "case" clauses	Baixa	5
T01	*Design*	Violação de padrão de codificação	[Customized] Check for $_REQUEST use in folder /include/*	Alta	772

T01	*Design*	Violação de padrão de codificação	[Customized] Remove the extra characters before the opening tag.	Baixa	16
T01	*Design*	Violação de padrão de codificação	[Customized] SQL commands should not be used in this folder	Média	826
T01	*Design*	Violação de padrão de codificação	Class constructors should not create other objects	Baixa	471
T01	*Design*	Violação de padrão de codificação	Classes should not be coupled to too many other classes (Single Responsibility Principle)	Baixa	2
T01	*Design*	Violação de padrão de codificação	Classes should not have too many fields	Baixa	37
T01	*Design*	Violação de padrão de codificação	Classes should not have too many lines of code	Baixa	44
T01	*Design*	Violação de padrão de codificação	Classes should not have too many methods	Baixa	121
T01	*Design*	Violação de padrão de codificação	Closing tag "?>" should be omitted on files containing only PHP	Média	158
T01	*Design*	Violação de padrão de codificação	Control flow statements "if", "for", "while", "switch" and "try" should not be nested too deeply	Baixa	66
T01	*Design*	Violação de padrão de codificação	Files should contain only one top-level class or interface each	Alta	17
T01	*Design*	Violação de padrão de codificação	Files should not have too many lines of code	Baixa	14
T01	*Design*	Violação de padrão de codificação	Functions should not have too many lines of code	Baixa	16
T01	*Design*	Violação de padrão de codificação	Functions should not have too many parameters	Baixa	21
T01	*Design*	Violação de padrão de codificação	Lines should not be too long	Baixa	4.568
T01	*Design*	Violação de padrão de codificação	Method arguments with default values should be last	Média	52
T01	*Design*	Violação de padrão de codificação	Method visibility should be explicitly declared	Média	324
T01	*Design*	Violação de padrão de codificação	Overriding methods should do more than simply call the same method in the super class	Baixa	65
T01	*Design*	Violação de padrão de codificação	Superglobals should not be accessed directly	Grave	481
T01	*Design*	Violação de padrão de codificação	[Customized] Remove the extra characters before the opening tag.	Baixa	310
T01	*Design*	Violação de padrão de codificação	Two branches in a conditional structure should not have exactly the same implementation	Média	108
T01	*Design*	Violação de padrão de codificação	Unused "private" fields should be removed	Média	9
T01	*Design*	Violação de padrão de codificação	Unused "private" methods should be removed	Média	7
T01	*Design*	Violação de padrão de codificação	Unused function parameters should be removed	Média	292
T01	*Design*	Violação de padrão de codificação	Unused local variables should be removed	Média	710
T01	*Design*	Violação de padrão de codificação	Variable variables should not be used	Alta	11
T01	Documentação	Documentação inexistente ou inadequada	[Customized] Functions declarations in /include/* should be properly documented in docblock format	Média	20.388

T01	Documentação	Documentação inexistente ou inadequada	Comments should not be located at the end of lines of code	Média	4.890
T01	Documentação	Documentação inexistente ou inadequada	Source files should have a sufficient density of comment lines	Baixa	2.365
T02	Código	Duplicação de código	Methods should not have identical implementations	Baixa	29
T02	Código	Função depreciada ou incompatível	__construct functions should not make PHP 4-style calls to parent constructors	Alta	14
T02	Código	Função depreciada ou incompatível	Alias functions should not be used	Baixa	10
T02	Código	Função depreciada ou incompatível	Deprecated functions should not be used	Média	26
T02	Código	Função depreciada ou incompatível	Perl-style comments should not be used	Baixa	324
T02	Código	Função depreciada ou incompatível	PHP 4 constructor declarations should not be used	Baixa	1.523
T02	Código	Função depreciada ou incompatível	The "var" keyword should not be used	Média	218
T02	Código	Legibilidade do código	"elseif" keyword should be used in place of "else if" keywords	Média	56
T02	Código	Legibilidade do código	"for" loop stop conditions should be invariant	Alta	6
T02	Código	Legibilidade do código	"switch" statements should have at least 3 "case" clauses	Baixa	204
T02	Código	Legibilidade do código	Assignments should not be made from within sub-expressions	Média	110
T02	Código	Legibilidade do código	Boolean literals should not be redundant	Baixa	460
T02	Código	Legibilidade do código	Collapsible "if" statements should be merged	Baixa	250
T02	Código	Legibilidade do código	Duplicate values should not be passed as arguments	Média	147
T02	Código	Legibilidade do código	Functions should use "return" consistently	Média	114
T02	Código	Legibilidade do código	Increment (++) and decrement (--) operators should not be used in a method call	Baixa	435
T02	Código	Legibilidade do código	Jump statements should not be followed by dead code	Alta	527
T02	Código	Legibilidade do código	Local variables should not have the same name as class fields	Baixa	222
T02	Código	Legibilidade do código	References should not be passed to function calls	Alta	1
T02	Código	Legibilidade do código	Sections of code should not be "commented out"	Média	765
T02	Código	Legibilidade do código	String literals should not be concatenated	Baixa	701
T02	Código	Legibilidade do código	Switch cases should end with an unconditional "break" statement	Média	2
T02	Código	Violação do padrão de formatação	"switch case" clauses should not have too many lines of code	Baixa	96
T02	Código	Violação do padrão de formatação	A close curly brace should be located at the beginning of a line	Baixa	3
T02	Código	Violação do padrão de formatação	Colors should be defined in upper case	Baixa	6

T02	Código	Violação do padrão de formatação	Conditionally executed code should be denoted by either indentation or curly braces	Baixa	5
T02	Código	Violação do padrão de formatação	Conditionals should start on new lines	Média	2
T02	Código	Violação do padrão de formatação	Control structures should use curly braces	Baixa	191
T02	Código	Violação do padrão de formatação	Files should contain an empty newline at the end	Média	3.063
T02	Código	Violação do padrão de formatação	Lines should not end with trailing whitespaces	Média	1.311
T02	Código	Violação do padrão de formatação	Modifiers should be declared in the correct order	Baixa	1
T02	Código	Violação do padrão de formatação	More than one property should not be declared per statement	Média	30
T02	Código	Violação do padrão de formatação	Multiline blocks should be enclosed in curly braces	Alta	6
T02	Código	Violação do padrão de formatação	PHP keywords and constants "true", "false", "null" should be lower case	Média	189
T02	Código	Violação do padrão de formatação	Source code should comply with formatting standards	Média	178.954
T02	Código	Violação do padrão de formatação	Statements should be on separate lines	Média	204
T02	Código	Violação do padrão de nomenclatura	[Customized] Classnames in /include/exp/* should comply with the naming convention	Alta	1
T02	Código	Violação do padrão de nomenclatura	File names should comply with a naming convention	Média	29
T02	Defeito	Código inadequado	Errors should not be silenced	Alta	31
T02	Defeito	Código inadequado	Identical expressions should not be used on both sides of a binary operator	Grave	8
T02	Defeito	Código inadequado	Local variables should not be declared and then immediately returned or thrown	Baixa	618
T02	Defeito	Código inadequado	Nested blocks of code should not be left empty	Alta	6
T02	Defeito	Código inadequado	Non-empty statements should change control flow or have at least one side-effect	Grave	10
T02	Defeito	Código inadequado	Objects should not be created to be dropped immediately without being used	Grave	57
T02	Defeito	Código inadequado	Parentheses should not be used for calls to "echo"	Média	36
T02	Defeito	Código inadequado	Related "if/else if" statements and "cases" in a "switch" should not have the same condition	Grave	2
T02	Defeito	Código inadequado	Return of boolean expressions should not be wrapped into an "if-then-else" statement	Baixa	35
T02	Defeito	Defeito encontrado no código	"$this" should not be used in a static context	Grave	3
T02	Defeito	Possibilidade de resultado inesperado	"&&" and "||" should be used	Média	58
T02	Defeito	Possibilidade de resultado inesperado	"exit(...)" and "die(...)" statements should not be used	Alta	64

T02	Defeito	Possibilidade de resultado inesperado	"require_once" and "include_once" should be used instead of "require" and "include"	Grave	35
T02	Defeito	Possibilidade de resultado inesperado	A "for" loop update clause should move the counter in the right direction	Grave	3
T02	Defeito	Possibilidade de resultado inesperado	All branches in a conditional structure should not have exactly the same implementation	Grave	9
T02	Defeito	Possibilidade de resultado inesperado	Files should not contain characters before "<?php"	Grave	7
T02	Defeito	Possibilidade de resultado inesperado	Static members should be referenced with "static::"	Alta	574
T02	Defeito	Possibilidade de resultado inesperado	Track uses of "FIXME" tags	Média	17
T02	Defeito	Possibilidade de resultado inesperado	Variables should be initialized before use	Grave	84
T02	Defeito	Possibilidade de resultado inesperado	Variables should not be self-assigned	Grave	21
T02	Defeito	Vulnerabilidade de segurança	Code should not be dynamically injected and executed	Grave	33
T02	Defeito	Vulnerabilidade de segurança	Configuration should not be changed dynamically	Alta	15
T02	*Design*	Código muito complexo	Classes should not be too complex	Baixa	3
T02	*Design*	Código muito complexo	Cognitive Complexity of functions should not be too high	Baixa	233
T02	*Design*	Código muito complexo	Expressions should not be too complex	Baixa	10
T02	*Design*	Código muito complexo	Functions should not be too complex	Baixa	86
T02	Defeito	Legibilidade do código	Redundant pairs of parentheses should be removed	Baixa	137
T02	*Design*	Violação de padrão de codificação	<?php and "<?=" tags should be used	Média	2.456
T02	*Design*	Violação de padrão de codificação	"switch" statements should not have too many "case" clauses	Baixa	5
T02	*Design*	Violação de padrão de codificação	[Customized] Check for $_REQUEST use in folder /include/*	Alta	1.151
T02	*Design*	Violação de padrão de codificação	[Customized] Remove the extra characters before the opening tag.	Baixa	5
T02	*Design*	Violação de padrão de codificação	[Customized] SQL commands should not be used in this folder	Média	1.643
T02	*Design*	Violação de padrão de codificação	Class constructors should not create other objects	Baixa	685
T02	*Design*	Violação de padrão de codificação	Classes should not be coupled to too many other classes (Single Responsibility Principle)	Baixa	29
T02	*Design*	Violação de padrão de codificação	Classes should not have too many fields	Baixa	25
T02	*Design*	Violação de padrão de codificação	Classes should not have too many lines of code	Baixa	25
T02	*Design*	Violação de padrão de codificação	Classes should not have too many methods	Baixa	33
T02	*Design*	Violação de padrão de codificação	Closing tag "?>" should be omitted on files containing only PHP	Média	2.139
T02	*Design*	Violação de padrão de codificação	Control flow statements "if", "for", "while", "switch" and "try" should not be nested too deeply	Baixa	26

T02	*Design*	Violação de padrão de codificação	Files should contain only one top-level class or interface each	Alta	38
T02	*Design*	Violação de padrão de codificação	Files should not have too many lines of code	Baixa	5
T02	*Design*	Violação de padrão de codificação	Functions should not have too many lines of code	Baixa	19
T02	*Design*	Violação de padrão de codificação	Functions should not have too many parameters	Baixa	29
T02	*Design*	Violação de padrão de codificação	Generic exceptions ErrorException, RuntimeException and Exception should not be thrown	Média	30
T02	*Design*	Violação de padrão de codificação	Lines should not be too long	Baixa	4.755
T02	*Design*	Violação de padrão de codificação	Method arguments with default values should be last	Média	53
T02	*Design*	Violação de padrão de codificação	Method visibility should be explicitly declared	Média	370
T02	*Design*	Violação de padrão de codificação	Overriding methods should do more than simply call the same method in the super class	Baixa	122
T02	*Design*	Violação de padrão de codificação	Superglobals should not be accessed directly	Grave	91
T02	*Design*	Violação de padrão de codificação	[Customized] Remove the extra characters before the opening tag.	Baixa	49
T02	*Design*	Violação de padrão de codificação	Two branches in a conditional structure should not have exactly the same implementation	Média	278
T02	*Design*	Violação de padrão de codificação	Unused "private" fields should be removed	Média	26
T02	*Design*	Violação de padrão de codificação	Unused "private" methods should be removed	Média	4
T02	*Design*	Violação de padrão de codificação	Unused function parameters should be removed	Média	210
T02	*Design*	Violação de padrão de codificação	Unused local variables should be removed	Média	1.409
T02	Documentação	Documentação inexistente ou inadequada	[Customized] Functions declarations in /include/* should be properly documented in docblock format	Média	19.817
T02	Documentação	Documentação inexistente ou inadequada	Comments should not be located at the end of lines of code	Média	6.607
T02	Documentação	Documentação inexistente ou inadequada	Source files should have a sufficient density of comment lines	Baixa	1.929
T03	Código	Duplicação de código	Methods should not have identical implementations	Baixa	29
T03	Código	Função depreciada ou incompatível	__construct functions should not make PHP 4-style calls to parent constructors	Alta	8
T03	Código	Função depreciada ou incompatível	Alias functions should not be used	Baixa	211
T03	Código	Função depreciada ou incompatível	Deprecated functions should not be used	Média	4
T03	Código	Função depreciada ou incompatível	Perl-style comments should not be used	Baixa	2.268
T03	Código	Função depreciada ou incompatível	PHP 4 constructor declarations should not be used	Baixa	631
T03	Código	Função depreciada ou incompatível	The "var" keyword should not be used	Média	1.709

T03	Código	Legibilidade do código	"elseif" keyword should be used in place of "else if" keywords	Média	517
T03	Código	Legibilidade do código	"for" loop stop conditions should be invariant	Alta	7
T03	Código	Legibilidade do código	"switch" statements should have at least 3 "case" clauses	Baixa	87
T03	Código	Legibilidade do código	Assignments should not be made from within sub-expressions	Média	209
T03	Código	Legibilidade do código	Boolean literals should not be redundant	Baixa	778
T03	Código	Legibilidade do código	Collapsible "if" statements should be merged	Baixa	128
T03	Código	Legibilidade do código	Duplicate values should not be passed as arguments	Média	19
T03	Código	Legibilidade do código	Functions should use "return" consistently	Média	45
T03	Código	Legibilidade do código	Increment (++) and decrement (--) operators should not be used in a method call	Baixa	285
T03	Código	Legibilidade do código	Jump statements should not be followed by dead code	Alta	320
T03	Código	Legibilidade do código	Local variables should not have the same name as class fields	Baixa	183
T03	Código	Legibilidade do código	Sections of code should not be "commented out"	Média	1.184
T03	Código	Legibilidade do código	String literals should not be concatenated	Baixa	336
T03	Código	Legibilidade do código	Switch cases should end with an unconditional "break" statement	Média	5
T03	Código	Violação do padrão de formatação	"switch case" clauses should not have too many lines of code	Baixa	112
T03	Código	Violação do padrão de formatação	A close curly brace should be located at the beginning of a line	Baixa	3
T03	Código	Violação do padrão de formatação	Conditionally executed code should be denoted by either indentation or curly braces	Baixa	170
T03	Código	Violação do padrão de formatação	Conditionals should start on new lines	Média	1
T03	Código	Violação do padrão de formatação	Control structures should use curly braces	Baixa	5.930
T03	Código	Violação do padrão de formatação	Files should contain an empty newline at the end	Média	1.496
T03	Código	Violação do padrão de formatação	Lines should not end with trailing whitespaces	Média	8.950
T03	Código	Violação do padrão de formatação	Modifiers should be declared in the correct order	Baixa	2
T03	Código	Violação do padrão de formatação	Multiline blocks should be enclosed in curly braces	Alta	257
T03	Código	Violação do padrão de formatação	PHP keywords and constants "true", "false", "null" should be lower case	Média	174
T03	Código	Violação do padrão de formatação	Source code should comply with formatting standards	Média	27.554
T03	Código	Violação do padrão de formatação	Statements should be on separate lines	Média	1.262
T03	Código	Violação do padrão de nomenclatura	File names should comply with a naming convention	Média	59
T03	Defeito	Código inadequado	Empty statements should be removed	Grave	18
T03	Defeito	Código inadequado	Errors should not be silenced	Alta	152

T03	Defeito	Código inadequado	Identical expressions should not be used on both sides of a binary operator	Grave	1
T03	Defeito	Código inadequado	Local variables should not be declared and then immediately returned or thrown	Baixa	379
T03	Defeito	Código inadequado	Nested blocks of code should not be left empty	Alta	21
T03	Defeito	Código inadequado	Non-empty statements should change control flow or have at least one side-effect	Grave	1
T03	Defeito	Código inadequado	Parentheses should not be used for calls to "echo"	Média	46
T03	Defeito	Código inadequado	Related "if/else if" statements and "cases" in a "switch" should not have the same condition	Grave	7
T03	Defeito	Código inadequado	Return of boolean expressions should not be wrapped into an "if-then-else" statement	Baixa	14
T03	Defeito	Código inadequado	Return values from functions without side effects should not be ignored	Grave	26
T03	Defeito	Código inadequado	Useless "if(true) {...}" and "if(false){...}" blocks should be removed	Grave	1
T03	Código	Legibilidade do código	Redundant pairs of parentheses should be removed	Baixa	83
T03	Defeito	Possibilidade de resultado inesperado	"&&" and "||" should be used	Média	207
T03	Defeito	Possibilidade de resultado inesperado	"=+" should not be used instead of "+="	Grave	1
T03	Defeito	Possibilidade de resultado inesperado	"exit(...)" and "die(...)" statements should not be used	Alta	162
T03	Defeito	Possibilidade de resultado inesperado	"require_once" and "include_once" should be used instead of "require" and "include"	Grave	76
T03	Defeito	Possibilidade de resultado inesperado	All branches in a conditional structure should not have exactly the same implementation	Grave	13
T03	Defeito	Possibilidade de resultado inesperado	Files should not contain characters before "<?php"	Grave	26
T03	Defeito	Possibilidade de resultado inesperado	Static members should be referenced with "static::"	Alta	57
T03	Defeito	Possibilidade de resultado inesperado	Track uses of "FIXME" tags	Média	4
T03	Defeito	Possibilidade de resultado inesperado	Variables should be initialized before use	Grave	23
T03	Defeito	Possibilidade de resultado inesperado	Variables should not be self-assigned	Grave	1
T03	Defeito	Vulnerabilidade de segurança	Code should not be dynamically injected and executed	Grave	36
T03	Defeito	Vulnerabilidade de segurança	Configuration should not be changed dynamically	Alta	1
T03	*Design*	Código muito complexo	Classes should not be too complex	Baixa	14
T03	*Design*	Código muito complexo	Cognitive Complexity of functions should not be too high	Baixa	166
T03	*Design*	Código muito complexo	Expressions should not be too complex	Baixa	11

T03	*Design*	Código muito complexo	Functions should not be too complex	Baixa	78
T03	*Design*	Violação de padrão de codificação	<?php and "<?=" tags should be used	Média	1.035
T03	*Design*	Violação de padrão de codificação	"switch" statements should not have too many "case" clauses	Baixa	17
T03	*Design*	Violação de padrão de codificação	[Customized] Check for $_REQUEST use in folder /include/*	Alta	535
T03	*Design*	Violação de padrão de codificação	[Customized] Remove the extra characters before the opening tag.	Baixa	22
T03	*Design*	Violação de padrão de codificação	[Customized] SQL commands should not be used in this folder	Média	747
T03	*Design*	Violação de padrão de codificação	Class constructors should not create other objects	Baixa	368
T03	*Design*	Violação de padrão de codificação	Classes should not be coupled to too many other classes (Single Responsibility Principle)	Baixa	1
T03	*Design*	Violação de padrão de codificação	Classes should not have too many fields	Baixa	7
T03	*Design*	Violação de padrão de codificação	Classes should not have too many lines of code	Baixa	27
T03	*Design*	Violação de padrão de codificação	Classes should not have too many methods	Baixa	23
T03	*Design*	Violação de padrão de codificação	Closing tag "?>" should be omitted on files containing only PHP	Média	917
T03	*Design*	Violação de padrão de codificação	Control flow statements "if", "for", "while", "switch" and "try" should not be nested too deeply	Baixa	7
T03	*Design*	Violação de padrão de codificação	Files should contain only one top-level class or interface each	Alta	3
T03	*Design*	Violação de padrão de codificação	Files should not have too many lines of code	Baixa	14
T03	*Design*	Violação de padrão de codificação	Functions should not have too many lines of code	Baixa	12
T03	*Design*	Violação de padrão de codificação	Functions should not have too many parameters	Baixa	2
T03	*Design*	Violação de padrão de codificação	Generic exceptions ErrorException, RuntimeException and Exception should not be thrown	Média	3
T03	*Design*	Violação de padrão de codificação	Lines should not be too long	Baixa	2.211
T03	*Design*	Violação de padrão de codificação	Method arguments with default values should be last	Média	14
T03	*Design*	Violação de padrão de codificação	Method visibility should be explicitly declared	Média	4.170
T03	*Design*	Violação de padrão de codificação	Overriding methods should do more than simply call the same method in the super class	Baixa	45
T03	*Design*	Violação de padrão de codificação	Superglobals should not be accessed directly	Grave	106
T03	*Design*	Violação de padrão de codificação	[Customized] Remove the extra characters before the opening tag.	Baixa	137
T03	*Design*	Violação de padrão de codificação	Two branches in a conditional structure should not have exactly the same implementation	Média	102
T03	*Design*	Violação de padrão de codificação	Unused "private" fields should be removed	Média	5
T03	*Design*	Violação de padrão de codificação	Unused function parameters should be removed	Média	142

T03	*Design*	Violação de padrão de codificação	Unused local variables should be removed	Média	1.025
T03	*Design*	Violação de padrão de codificação	Variable variables should not be used	Alta	1
T03	Documentação	Documentação inexistente ou inadequada	[Customized] Functions declarations in /include/* should be properly documented in docblock format	Média	8.725
T03	Documentação	Documentação inexistente ou inadequada	Comments should not be located at the end of lines of code	Média	5.521
T03	Documentação	Documentação inexistente ou inadequada	Source files should have a sufficient density of comment lines	Baixa	1.548
T04	Código	Duplicação de código	Methods should not have identical implementations	Baixa	51
T04	Código	Função depreciada ou incompatível	__construct functions should not make PHP 4-style calls to parent constructors	Alta	16
T04	Código	Função depreciada ou incompatível	Alias functions should not be used	Baixa	18
T04	Código	Função depreciada ou incompatível	Deprecated functions should not be used	Média	6
T04	Código	Função depreciada ou incompatível	Perl-style comments should not be used	Baixa	168
T04	Código	Função depreciada ou incompatível	PHP 4 constructor declarations should not be used	Baixa	614
T04	Código	Função depreciada ou incompatível	The "var" keyword should not be used	Média	1.271
T04	Código	Legibilidade do código	"elseif" keyword should be used in place of "else if" keywords	Média	847
T04	Código	Legibilidade do código	"for" loop stop conditions should be invariant	Alta	6
T04	Código	Legibilidade do código	"switch" statements should have at least 3 "case" clauses	Baixa	49
T04	Código	Legibilidade do código	Assignments should not be made from within sub-expressions	Média	338
T04	Código	Legibilidade do código	Boolean literals should not be redundant	Baixa	694
T04	Código	Legibilidade do código	Collapsible "if" statements should be merged	Baixa	254
T04	Código	Legibilidade do código	Duplicate values should not be passed as arguments	Média	16
T04	Código	Legibilidade do código	Functions should use "return" consistently	Média	59
T04	Código	Legibilidade do código	Increment (++) and decrement (--) operators should not be used in a method call	Baixa	933
T04	Código	Legibilidade do código	Jump statements should not be followed by dead code	Alta	197
T04	Código	Legibilidade do código	Local variables should not have the same name as class fields	Baixa	79
T04	Código	Legibilidade do código	References should not be passed to function calls	Alta	26
T04	Código	Legibilidade do código	Sections of code should not be "commented out"	Média	494
T04	Código	Legibilidade do código	String literals should not be concatenated	Baixa	563
T04	Código	Legibilidade do código	Switch cases should end with an unconditional "break" statement	Média	1
T04	Código	Violação do padrão de formatação	"switch case" clauses should not have too many lines of code	Baixa	62

T04	Código	Violação do padrão de formatação	A close curly brace should be located at the beginning of a line	Baixa	2
T04	Código	Violação do padrão de formatação	Colors should be defined in upper case	Baixa	7
T04	Código	Violação do padrão de formatação	Conditionally executed code should be denoted by either indentation or curly braces	Baixa	75
T04	Código	Violação do padrão de formatação	Conditionals should start on new lines	Média	4
T04	Código	Violação do padrão de formatação	Control structures should use curly braces	Baixa	5.215
T04	Código	Violação do padrão de formatação	Files should contain an empty newline at the end	Média	1.477
T04	Código	Violação do padrão de formatação	Lines should not end with trailing whitespaces	Média	10.781
T04	Código	Violação do padrão de formatação	Modifiers should be declared in the correct order	Baixa	14
T04	Código	Violação do padrão de formatação	More than one property should not be declared per statement	Média	2
T04	Código	Violação do padrão de formatação	Multiline blocks should be enclosed in curly braces	Alta	117
T04	Código	Violação do padrão de formatação	PHP keywords and constants "true", "false", "null" should be lower case	Média	226
T04	Código	Violação do padrão de formatação	Source code should comply with formatting standards	Média	34.186
T04	Código	Violação do padrão de formatação	Statements should be on separate lines	Média	538
T04	Código	Violação do padrão de nomenclatura	File names should comply with a naming convention	Média	43
T04	Defeito	Código inadequado	Empty statements should be removed	Grave	17
T04	Defeito	Código inadequado	Errors should not be silenced	Alta	12
T04	Defeito	Código inadequado	Identical expressions should not be used on both sides of a binary operator	Grave	2
T04	Defeito	Código inadequado	Local variables should not be declared and then immediately returned or thrown	Baixa	249
T04	Defeito	Código inadequado	Nested blocks of code should not be left empty	Alta	8
T04	Defeito	Código inadequado	Non-empty statements should change control flow or have at least one side-effect	Grave	3
T04	Defeito	Código inadequado	Objects should not be created to be dropped immediately without being used	Grave	7
T04	Defeito	Código inadequado	Parentheses should not be used for calls to "echo"	Média	24
T04	Defeito	Código inadequado	Related "if/else if" statements and "cases" in a "switch" should not have the same condition	Grave	2
T04	Defeito	Código inadequado	Return of boolean expressions should not be wrapped into an "if-then-else" statement	Baixa	46

T04	Defeito	Código inadequado	Return values from functions without side effects should not be ignored	Grave	1
T04	Defeito	Código inadequado	Useless "if(true) {...}" and "if(false){...}" blocks should be removed	Grave	4
T04	Defeito	Defeito encontrado no código	"$this" should not be used in a static context	Grave	2
T04	Defeito	Defeito encontrado no código	PHP parser failure	Grave	1
T04	Defeito	Possibilidade de resultado inesperado	"&&" and "||" should be used	Média	12
T04	Defeito	Possibilidade de resultado inesperado	"=+" should not be used instead of "+="	Grave	2
T04	Defeito	Possibilidade de resultado inesperado	"exit(...)" and "die(...)" statements should not be used	Alta	43
T04	Defeito	Possibilidade de resultado inesperado	"require_once" and "include_once" should be used instead of "require" and "include"	Grave	39
T04	Defeito	Possibilidade de resultado inesperado	All branches in a conditional structure should not have exactly the same implementation	Grave	8
T04	Defeito	Possibilidade de resultado inesperado	Files should not contain characters before "<?php"	Grave	8
T04	Defeito	Possibilidade de resultado inesperado	Static members should be referenced with "static::"	Alta	25
T04	Defeito	Possibilidade de resultado inesperado	Track uses of "FIXME" tags	Média	1
T04	Defeito	Possibilidade de resultado inesperado	Variables should be initialized before use	Grave	14
T04	Defeito	Possibilidade de resultado inesperado	Variables should not be self-assigned	Grave	6
T04	Defeito	Vulnerabilidade de segurança	"sleep" should not be called	Grave	2
T04	Defeito	Vulnerabilidade de segurança	Code should not be dynamically injected and executed	Grave	1
T04	Defeito	Vulnerabilidade de segurança	Configuration should not be changed dynamically	Alta	5
T04	*Design*	Código muito complexo	Classes should not be too complex	Baixa	4
T04	*Design*	Código muito complexo	Cognitive Complexity of functions should not be too high	Baixa	163
T04	*Design*	Código muito complexo	Expressions should not be too complex	Baixa	34
T04	*Design*	Código muito complexo	Functions should not be too complex	Baixa	80
T04	Defeito	Legibilidade do código	Redundant pairs of parentheses should be removed	Baixa	83
T04	*Design*	Violação de padrão de codificação	<?php and "<?=" tags should be used	Média	1.122
T04	*Design*	Violação de padrão de codificação	"switch" statements should not have too many "case" clauses	Baixa	6
T04	*Design*	Violação de padrão de codificação	[Customized] Check for $_REQUEST use in folder /include/*	Alta	605
T04	*Design*	Violação de padrão de codificação	[Customized] Remove the extra characters before the opening tag.	Baixa	7
T04	*Design*	Violação de padrão de codificação	[Customized] SQL commands should not be used in this folder	Média	685
T04	*Design*	Violação de padrão de codificação	Class constructors should not create other objects	Baixa	302

T04	*Design*	Violação de padrão de codificação	Classes should not be coupled to too many other classes (Single Responsibility Principle)	Baixa	1
T04	*Design*	Violação de padrão de codificação	Classes should not have too many fields	Baixa	9
T04	*Design*	Violação de padrão de codificação	Classes should not have too many lines of code	Baixa	19
T04	*Design*	Violação de padrão de codificação	Classes should not have too many methods	Baixa	31
T04	*Design*	Violação de padrão de codificação	Closing tag "?>" should be omitted on files containing only PHP	Média	969
T04	*Design*	Violação de padrão de codificação	Control flow statements "if", "for", "while", "switch" and "try" should not be nested too deeply	Baixa	91
T04	*Design*	Violação de padrão de codificação	Files should contain only one top-level class or interface each	Alta	3
T04	*Design*	Violação de padrão de codificação	Files should not have too many lines of code	Baixa	4
T04	*Design*	Violação de padrão de codificação	Functions should not have too many lines of code	Baixa	16
T04	*Design*	Violação de padrão de codificação	Functions should not have too many parameters	Baixa	4
T04	*Design*	Violação de padrão de codificação	Generic exceptions ErrorException, RuntimeException and Exception should not be thrown	Média	7
T04	*Design*	Violação de padrão de codificação	Lines should not be too long	Baixa	3.049
T04	*Design*	Violação de padrão de codificação	Method arguments with default values should be last	Média	24
T04	*Design*	Violação de padrão de codificação	Method visibility should be explicitly declared	Média	3.793
T04	*Design*	Violação de padrão de codificação	Overriding methods should do more than simply call the same method in the super class	Baixa	62
T04	*Design*	Violação de padrão de codificação	Superglobals should not be accessed directly	Grave	120
T04	*Design*	Violação de padrão de codificação	[Customized] Remove the extra characters before the opening tag.	Baixa	25
T04	*Design*	Violação de padrão de codificação	Two branches in a conditional structure should not have exactly the same implementation	Média	183
T04	*Design*	Violação de padrão de codificação	Unused "private" fields should be removed	Média	14
T04	*Design*	Violação de padrão de codificação	Unused "private" methods should be removed	Média	4
T04	*Design*	Violação de padrão de codificação	Unused function parameters should be removed	Média	263
T04	*Design*	Violação de padrão de codificação	Unused local variables should be removed	Média	976
T04	Documentação	Documentação inexistente ou inadequada	[Customized] Functions declarations in /include/* should be properly documented in docblock format	Média	10.179
T04	Documentação	Documentação inexistente ou inadequada	Comments should not be located at the end of lines of code	Média	3.725

T04	Documentação	Documentação inexistente ou inadequada	Source files should have a sufficient density of comment lines	Baixa	1.555
T05	Código	Duplicação de código	Methods should not have identical implementations	Baixa	13
T05	Código	Função depreciada ou incompatível	Alias functions should not be used	Baixa	1
T05	Código	Função depreciada ou incompatível	Perl-style comments should not be used	Baixa	140
T05	Código	Função depreciada ou incompatível	PHP 4 constructor declarations should not be used	Baixa	962
T05	Código	Legibilidade do código	"elseif" keyword should be used in place of "else if" keywords	Média	23
T05	Código	Legibilidade do código	"switch" statements should have at least 3 "case" clauses	Baixa	48
T05	Código	Legibilidade do código	Assignments should not be made from within sub-expressions	Média	4
T05	Código	Legibilidade do código	Boolean literals should not be redundant	Baixa	195
T05	Código	Legibilidade do código	Collapsible "if" statements should be merged	Baixa	71
T05	Código	Legibilidade do código	Duplicate values should not be passed as arguments	Média	33
T05	Código	Legibilidade do código	Functions should use "return" consistently	Média	55
T05	Código	Legibilidade do código	Increment (++) and decrement (--) operators should not be used in a method call	Baixa	391
T05	Código	Legibilidade do código	Local variables should not have the same name as class fields	Baixa	200
T05	Documentação	Legibilidade do código	Redundant pairs of parentheses should be removed	Baixa	23
T05	Código	Legibilidade do código	Sections of code should not be "commented out"	Média	233
T05	Código	Legibilidade do código	String literals should not be concatenated	Baixa	254
T05	Código	Violação do padrão de formatação	"switch case" clauses should not have too many lines of code	Baixa	18
T05	Código	Violação do padrão de formatação	A close curly brace should be located at the beginning of a line	Baixa	1
T05	Código	Violação do padrão de formatação	Control structures should use curly braces	Baixa	1
T05	Código	Violação do padrão de formatação	Files should contain an empty newline at the end	Média	2.463
T05	Código	Violação do padrão de formatação	Lines should not end with trailing whitespaces	Média	525
T05	Código	Violação do padrão de formatação	Modifiers should be declared in the correct order	Baixa	3
T05	Código	Violação do padrão de formatação	More than one property should not be declared per statement	Média	1
T05	Código	Violação do padrão de formatação	PHP keywords and constants "true", "false", "null" should be lower case	Média	49
T05	Código	Violação do padrão de formatação	Source code should comply with formatting standards	Média	93.857
T05	Código	Violação do padrão de formatação	Statements should be on separate lines	Média	14
T05	Código	Violação do padrão de nomenclatura	[Customized] Classnames in /include/exp/* should comply with the naming convention	Alta	2

T05	Código	Violação do padrão de nomenclatura	[Customized] Function names in /include/exp/* should comply with the naming convention	Alta	1
T05	Código	Violação do padrão de nomenclatura	File names should comply with a naming convention	Média	62
T05	Defeito	Código inadequado	Local variables should not be declared and then immediately returned or thrown	Baixa	296
T05	Defeito	Código inadequado	Return of boolean expressions should not be wrapped into an "if-then-else" statement	Baixa	11
T05	Defeito	Possibilidade de resultado inesperado	"exit(...)" and "die(...)" statements should not be used	Alta	6
T05	Defeito	Possibilidade de resultado inesperado	Files should not contain characters before "<?php"	Grave	1
T05	Defeito	Possibilidade de resultado inesperado	Variables should be initialized before use	Grave	9
T05	Defeito	Vulnerabilidade de segurança	Code should not be dynamically injected and executed	Grave	11
T05	*Design*	Código muito complexo	Classes should not be too complex	Baixa	3
T05	*Design*	Código muito complexo	Cognitive Complexity of functions should not be too high	Baixa	69
T05	*Design*	Código muito complexo	Expressions should not be too complex	Baixa	7
T05	*Design*	Código muito complexo	Functions should not be too complex	Baixa	27
T05	*Design*	Violação de padrão de codificação	<?php and "<?=" tags should be used	Média	1.994
T05	*Design*	Violação de padrão de codificação	"switch" statements should not have too many "case" clauses	Baixa	1
T05	*Design*	Violação de padrão de codificação	[Customized] Check for $_REQUEST use in folder /include/*	Alta	504
T05	*Design*	Violação de padrão de codificação	[Customized] SQL commands should not be used in this folder	Média	737
T05	*Design*	Violação de padrão de codificação	Class constructors should not create other objects	Baixa	453
T05	*Design*	Violação de padrão de codificação	Classes should not be coupled to too many other classes (Single Responsibility Principle)	Baixa	1
T05	*Design*	Violação de padrão de codificação	Classes should not have too many fields	Baixa	8
T05	*Design*	Violação de padrão de codificação	Classes should not have too many lines of code	Baixa	12
T05	*Design*	Violação de padrão de codificação	Classes should not have too many methods	Baixa	21
T05	*Design*	Violação de padrão de codificação	Closing tag "?>" should be omitted on files containing only PHP	Média	1.668
T05	*Design*	Violação de padrão de codificação	Control flow statements "if", "for", "while", "switch" and "try" should not be nested too deeply	Baixa	2
T05	*Design*	Violação de padrão de codificação	Files should not have too many lines of code	Baixa	2
T05	*Design*	Violação de padrão de codificação	Functions should not have too many lines of code	Baixa	5
T05	*Design*	Violação de padrão de codificação	Functions should not have too many parameters	Baixa	12
T05	*Design*	Violação de padrão de codificação	Lines should not be too long	Baixa	1.727
T05	*Design*	Violação de padrão de codificação	Method arguments with default values should be last	Média	10

T05	*Design*	Violação de padrão de codificação	Method visibility should be explicitly declared	Média	22
T05	*Design*	Violação de padrão de codificação	Overriding methods should do more than simply call the same method in the super class	Baixa	92
T05	*Design*	Violação de padrão de codificação	Superglobals should not be accessed directly	Grave	8
T05	*Design*	Violação de padrão de codificação	[Customized] Remove the extra characters before the opening tag.	Baixa	23
T05	*Design*	Violação de padrão de codificação	Unused "private" fields should be removed	Média	1
T05	*Design*	Violação de padrão de codificação	Unused "private" methods should be removed	Média	1
T05	*Design*	Violação de padrão de codificação	Unused local variables should be removed	Média	297
T05	Documentação	Documentação inexistente ou inadequada	[Customized] Functions declarations in /include/* should be properly documented in docblock format	Média	7.732
T05	Documentação	Documentação inexistente ou inadequada	Comments should not be located at the end of lines of code	Média	2.942
T05	Documentação	Documentação inexistente ou inadequada	Source files should have a sufficient density of comment lines	Baixa	1.838
T06	Código	Duplicação de código	Methods should not have identical implementations	Baixa	7
T06	Código	Função depreciada ou incompatível	__construct functions should not make PHP 4-style calls to parent constructors	Alta	35
T06	Código	Função depreciada ou incompatível	Alias functions should not be used	Baixa	2
T06	Código	Função depreciada ou incompatível	Perl-style comments should not be used	Baixa	33
T06	Código	Função depreciada ou incompatível	PHP 4 constructor declarations should not be used	Baixa	660
T06	Código	Função depreciada ou incompatível	The "var" keyword should not be used	Média	997
T06	Código	Legibilidade do código	"elseif" keyword should be used in place of "else if" keywords	Média	479
T06	Código	Legibilidade do código	"switch" statements should have at least 3 "case" clauses	Baixa	38
T06	Código	Legibilidade do código	Assignments should not be made from within sub-expressions	Média	12
T06	Código	Legibilidade do código	Boolean literals should not be redundant	Baixa	142
T06	Código	Legibilidade do código	Collapsible "if" statements should be merged	Baixa	63
T06	Código	Legibilidade do código	Duplicate values should not be passed as arguments	Média	15
T06	Código	Legibilidade do código	Functions should use "return" consistently	Média	20
T06	Código	Legibilidade do código	Increment (++) and decrement (--) operators should not be used in a method call	Baixa	313
T06	Código	Legibilidade do código	Jump statements should not be followed by dead code	Alta	95
T06	Código	Legibilidade do código	Local variables should not have the same name as class fields	Baixa	42
T06	Código	Legibilidade do código	Sections of code should not be "commented out"	Média	66

T06	Código	Legibilidade do código	String literals should not be concatenated	Baixa	30
T06	Código	Legibilidade do código	Switch cases should end with an unconditional "break" statement	Média	1
T06	Código	Violação do padrão de formatação	"switch case" clauses should not have too many lines of code	Baixa	32
T06	Código	Violação do padrão de formatação	Colors should be defined in upper case	Baixa	2
T06	Código	Violação do padrão de formatação	Conditionally executed code should be denoted by either indentation or curly braces	Baixa	6
T06	Código	Violação do padrão de formatação	Conditionals should start on new lines	Média	40
T06	Código	Violação do padrão de formatação	Control structures should use curly braces	Baixa	3.218
T06	Código	Violação do padrão de formatação	Files should contain an empty newline at the end	Média	1.660
T06	Código	Violação do padrão de formatação	Lines should not end with trailing whitespaces	Média	4.415
T06	Código	Violação do padrão de formatação	Modifiers should be declared in the correct order	Baixa	43
T06	Código	Violação do padrão de formatação	Multiline blocks should be enclosed in curly braces	Alta	22
T06	Código	Violação do padrão de formatação	PHP keywords and constants "true", "false", "null" should be lower case	Média	82
T06	Código	Violação do padrão de formatação	Source code should comply with formatting standards	Média	26.586
T06	Código	Violação do padrão de formatação	Statements should be on separate lines	Média	786
T06	Código	Violação do padrão de nomenclatura	[Customized] Classnames in /include/exp/* should comply with the naming convention	Alta	1
T06	Código	Violação do padrão de nomenclatura	File names should comply with a naming convention	Média	32
T06	Defeito	Código inadequado	Empty statements should be removed	Grave	3
T06	Defeito	Código inadequado	Errors should not be silenced	Alta	13
T06	Defeito	Código inadequado	Local variables should not be declared and then immediately returned or thrown	Baixa	247
T06	Defeito	Código inadequado	Nested blocks of code should not be left empty	Alta	4
T06	Defeito	Código inadequado	Objects should not be created to be dropped immediately without being used	Grave	2
T06	Defeito	Código inadequado	Parentheses should not be used for calls to "echo"	Média	29
T06	Código	Legibilidade do código	Redundant pairs of parentheses should be removed	Baixa	26
T06	Defeito	Possibilidade de resultado inesperado	"exit(...)" and "die(...)" statements should not be used	Alta	29
T06	Defeito	Possibilidade de resultado inesperado	"require_once" and "include_once" should be used instead of "require" and "include"	Grave	2
T06	Defeito	Possibilidade de resultado inesperado	All branches in a conditional structure should not have exactly the same implementation	Grave	1

T06	Defeito	Possibilidade de resultado inesperado	Files should not contain characters before "<?php"	Grave	4
T06	Defeito	Possibilidade de resultado inesperado	Static members should be referenced with "static::"	Alta	18
T06	Defeito	Possibilidade de resultado inesperado	Variables should be initialized before use	Grave	11
T06	Defeito	Vulnerabilidade de segurança	Code should not be dynamically injected and executed	Grave	2
T06	*Design*	Código muito complexo	Classes should not be too complex	Baixa	8
T06	*Design*	Código muito complexo	Cognitive Complexity of functions should not be too high	Baixa	95
T06	*Design*	Código muito complexo	Expressions should not be too complex	Baixa	25
T06	*Design*	Código muito complexo	Functions should not be too complex	Baixa	46
T06	*Design*	Violação de padrão de codificação	<?php and "<?=" tags should be used	Média	1.731
T06	*Design*	Violação de padrão de codificação	"switch" statements should not have too many "case" clauses	Baixa	5
T06	*Design*	Violação de padrão de codificação	[Customized] Check for $_REQUEST use in folder /include/*	Alta	433
T06	*Design*	Violação de padrão de codificação	[Customized] Remove the extra characters before the opening tag.	Baixa	4
T06	*Design*	Violação de padrão de codificação	[Customized] SQL commands should not be used in this folder	Média	639
T06	*Design*	Violação de padrão de codificação	Class constructors should not create other objects	Baixa	308
T06	*Design*	Violação de padrão de codificação	Classes should not have too many fields	Baixa	4
T06	*Design*	Violação de padrão de codificação	Classes should not have too many lines of code	Baixa	21
T06	*Design*	Violação de padrão de codificação	Classes should not have too many methods	Baixa	28
T06	*Design*	Violação de padrão de codificação	Closing tag "?>" should be omitted on files containing only PHP	Média	1.167
T06	*Design*	Violação de padrão de codificação	Files should contain only one top-level class or interface each	Alta	1
T06	*Design*	Violação de padrão de codificação	Files should not have too many lines of code	Baixa	4
T06	*Design*	Violação de padrão de codificação	Functions should not have too many lines of code	Baixa	5
T06	*Design*	Violação de padrão de codificação	Lines should not be too long	Baixa	3.600
T06	*Design*	Violação de padrão de codificação	Method arguments with default values should be last	Média	9
T06	*Design*	Violação de padrão de codificação	Method visibility should be explicitly declared	Média	4.403
T06	*Design*	Violação de padrão de codificação	Overriding methods should do more than simply call the same method in the super class	Baixa	28
T06	*Design*	Violação de padrão de codificação	Superglobals should not be accessed directly	Grave	27
T06	*Design*	Violação de padrão de codificação	[Customized] Remove the extra characters before the opening tag.	Baixa	28
T06	*Design*	Violação de padrão de codificação	Two branches in a conditional structure should not have exactly the same implementation	Média	28
T06	*Design*	Violação de padrão de codificação	Unused "private" fields should be removed	Média	25

T06	*Design*	Violação de padrão de codificação	Unused "private" methods should be removed	Média	1
T06	*Design*	Violação de padrão de codificação	Unused function parameters should be removed	Média	38
T06	*Design*	Violação de padrão de codificação	Unused local variables should be removed	Média	508
T06	Documentação	Documentação inexistente ou inadequada	[Customized] Functions declarations in /include/* should be properly documented in docblock format	Média	8.884
T06	Documentação	Documentação inexistente ou inadequada	Comments should not be located at the end of lines of code	Média	1.814
T06	Documentação	Documentação inexistente ou inadequada	Source files should have a sufficient density of comment lines	Baixa	1.464
T07	Código	Duplicação de código	Methods should not have identical implementations	Baixa	8
T07	Código	Função depreciada ou incompatível	PHP 4 constructor declarations should not be used	Baixa	191
T07	Código	Legibilidade do código	"elseif" keyword should be used in place of "else if" keywords	Média	713
T07	Código	Legibilidade do código	"switch" statements should have at least 3 "case" clauses	Baixa	38
T07	Código	Legibilidade do código	Assignments should not be made from within sub-expressions	Média	13
T07	Código	Legibilidade do código	Boolean literals should not be redundant	Baixa	25
T07	Código	Legibilidade do código	Collapsible "if" statements should be merged	Baixa	19
T07	Código	Legibilidade do código	Duplicate values should not be passed as arguments	Média	21
T07	Código	Legibilidade do código	Functions should use "return" consistently	Média	5
T07	Código	Legibilidade do código	Sections of code should not be "commented out"	Média	1
T07	Código	Legibilidade do código	String literals should not be concatenated	Baixa	3
T07	Código	Violação do padrão de formatação	"switch case" clauses should not have too many lines of code	Baixa	77
T07	Código	Violação do padrão de formatação	Conditionals should start on new lines	Média	1
T07	Código	Violação do padrão de formatação	Control structures should use curly braces	Baixa	1
T07	Código	Violação do padrão de formatação	Files should contain an empty newline at the end	Média	259
T07	Código	Violação do padrão de formatação	Lines should not end with trailing whitespaces	Média	742
T07	Código	Violação do padrão de formatação	Source code should comply with formatting standards	Média	305
T07	Código	Violação do padrão de nomenclatura	File names should comply with a naming convention	Média	34
T07	Defeito	Código inadequado	Errors should not be silenced	Alta	3
T07	Defeito	Código inadequado	Local variables should not be declared and then immediately returned or thrown	Baixa	2
T07	Defeito	Código inadequado	Parentheses should not be used for calls to "echo"	Média	1
T07	Defeito	Código inadequado	Return of boolean expressions should not be wrapped into an "if-then-else" statement	Baixa	12
T07	Código	Legibilidade do código	Redundant pairs of parentheses should be removed	Baixa	80

T07	Defeito	Possibilidade de resultado inesperado	"exit(...)" and "die(...)" statements should not be used	Alta	28
T07	Defeito	Possibilidade de resultado inesperado	Files should not contain characters before "<?php"	Grave	1
T07	Defeito	Possibilidade de resultado inesperado	Static members should be referenced with "static::"	Alta	1
T07	Defeito	Possibilidade de resultado inesperado	Variables should be initialized before use	Grave	2
T07	Defeito	Vulnerabilidade de segurança	Configuration should not be changed dynamically	Alta	1
T07	*Design*	Código muito complexo	Classes should not be too complex	Baixa	5
T07	*Design*	Código muito complexo	Cognitive Complexity of functions should not be too high	Baixa	83
T07	*Design*	Código muito complexo	Expressions should not be too complex	Baixa	9
T07	*Design*	Código muito complexo	Functions should not be too complex	Baixa	42
T07	*Design*	Violação de padrão de codificação	<?php and "<?=" tags should be used	Média	696
T07	*Design*	Violação de padrão de codificação	"switch" statements should not have too many "case" clauses	Baixa	7
T07	*Design*	Violação de padrão de codificação	[Customized] Check for $_REQUEST use in folder /include/*	Alta	271
T07	*Design*	Violação de padrão de codificação	[Customized] SQL commands should not be used in this folder	Média	178
T07	*Design*	Violação de padrão de codificação	Class constructors should not create other objects	Baixa	145
T07	*Design*	Violação de padrão de codificação	Classes should not be coupled to too many other classes (Single Responsibility Principle)	Baixa	2
T07	*Design*	Violação de padrão de codificação	Classes should not have too many fields	Baixa	2
T07	*Design*	Violação de padrão de codificação	Classes should not have too many lines of code	Baixa	10
T07	*Design*	Violação de padrão de codificação	Classes should not have too many methods	Baixa	7
T07	*Design*	Violação de padrão de codificação	Closing tag "?>" should be omitted on files containing only PHP	Média	69
T07	*Design*	Violação de padrão de codificação	Control flow statements "if", "for", "while", "switch" and "try" should not be nested too deeply	Baixa	27
T07	*Design*	Violação de padrão de codificação	Files should not have too many lines of code	Baixa	3
T07	*Design*	Violação de padrão de codificação	Functions should not have too many lines of code	Baixa	7
T07	*Design*	Violação de padrão de codificação	Functions should not have too many parameters	Baixa	4
T07	*Design*	Violação de padrão de codificação	Lines should not be too long	Baixa	1.620
T07	*Design*	Violação de padrão de codificação	Method arguments with default values should be last	Média	7
T07	*Design*	Violação de padrão de codificação	Method visibility should be explicitly declared	Média	75
T07	*Design*	Violação de padrão de codificação	Superglobals should not be accessed directly	Grave	133
T07	*Design*	Violação de padrão de codificação	Track uses of "NOSONAR" comments	Alta	1
T07	*Design*	Violação de padrão de codificação	[Customized] Remove the extra characters before the opening tag.	Baixa	11

T07	*Design*	Violação de padrão de codificação	Two branches in a conditional structure should not have exactly the same implementation	Média	1
T07	*Design*	Violação de padrão de codificação	Unused function parameters should be removed	Média	28
T07	*Design*	Violação de padrão de codificação	Unused local variables should be removed	Média	566
T07	Documentação	Documentação inexistente ou inadequada	[Customized] Functions declarations in /include/* should be properly documented in docblock format	Média	4.643
T07	Documentação	Documentação inexistente ou inadequada	Comments should not be located at the end of lines of code	Média	2.532
T07	Documentação	Documentação inexistente ou inadequada	Source files should have a sufficient density of comment lines	Baixa	926
T08	Código	Duplicação de código	Methods should not have identical implementations	Baixa	5
T08	Código	Função depreciada ou incompatível	Alias functions should not be used	Baixa	7
T08	Código	Função depreciada ou incompatível	Perl-style comments should not be used	Baixa	150
T08	Código	Função depreciada ou incompatível	PHP 4 constructor declarations should not be used	Baixa	145
T08	Código	Função depreciada ou incompatível	The "var" keyword should not be used	Média	29
T08	Código	Legibilidade do código	"elseif" keyword should be used in place of "else if" keywords	Média	230
T08	Código	Legibilidade do código	"switch" statements should have at least 3 "case" clauses	Baixa	51
T08	Código	Legibilidade do código	Assignments should not be made from within sub-expressions	Média	775
T08	Código	Legibilidade do código	Boolean literals should not be redundant	Baixa	100
T08	Código	Legibilidade do código	Collapsible "if" statements should be merged	Baixa	71
T08	Código	Legibilidade do código	Duplicate values should not be passed as arguments	Média	1
T08	Código	Legibilidade do código	Increment (++) and decrement (--) operators should not be used in a method call	Baixa	278
T08	Código	Legibilidade do código	Local variables should not have the same name as class fields	Baixa	54
T08	Documentação	Legibilidade do código	Redundant pairs of parentheses should be removed	Baixa	26
T08	Código	Legibilidade do código	Sections of code should not be "commented out"	Média	13
T08	Código	Legibilidade do código	String literals should not be concatenated	Baixa	20
T08	Código	Legibilidade do código	Switch cases should end with an unconditional "break" statement	Média	2
T08	Código	Violação do padrão de formatação	"switch case" clauses should not have too many lines of code	Baixa	86
T08	Código	Violação do padrão de formatação	Conditionally executed code should be denoted by either indentation or curly braces	Baixa	16
T08	Código	Violação do padrão de formatação	Control structures should use curly braces	Baixa	1.113

T08	Código	Violação do padrão de formatação	Files should contain an empty newline at the end	Média	628
T08	Código	Violação do padrão de formatação	Lines should not end with trailing whitespaces	Média	2.951
T08	Código	Violação do padrão de formatação	Modifiers should be declared in the correct order	Baixa	27
T08	Código	Violação do padrão de formatação	Multiline blocks should be enclosed in curly braces	Alta	12
T08	Código	Violação do padrão de formatação	PHP keywords and constants "true", "false", "null" should be lower case	Média	33
T08	Código	Violação do padrão de formatação	Source code should comply with formatting standards	Média	6.707
T08	Código	Violação do padrão de formatação	Statements should be on separate lines	Média	247
T08	Código	Violação do padrão de nomenclatura	[Customized] Classnames in /include/exp/* should comply with the naming convention	Alta	69
T08	Código	Violação do padrão de nomenclatura	File names should comply with a naming convention	Média	16
T08	Defeito	Código inadequado	Local variables should not be declared and then immediately returned or thrown	Baixa	117
T08	Defeito	Código inadequado	Return of boolean expressions should not be wrapped into an "if-then-else" statement	Baixa	6
T08	Defeito	Possibilidade de resultado inesperado	Variables should be initialized before use	Grave	2
T08	Defeito	Vulnerabilidade de segurança	Code should not be dynamically injected and executed	Grave	2
T08	*Design*	Código muito complexo	Classes should not be too complex	Baixa	2
T08	*Design*	Código muito complexo	Cognitive Complexity of functions should not be too high	Baixa	76
T08	*Design*	Código muito complexo	Expressions should not be too complex	Baixa	14
T08	*Design*	Código muito complexo	Functions should not be too complex	Baixa	32
T08	*Design*	Violação de padrão de codificação	<?php and "<?=" tags should be used	Média	986
T08	*Design*	Violação de padrão de codificação	"switch" statements should not have too many "case" clauses	Baixa	2
T08	*Design*	Violação de padrão de codificação	[Customized] Check for $_REQUEST use in folder /include/*	Alta	255
T08	*Design*	Violação de padrão de codificação	[Customized] SQL commands should not be used in this folder	Média	429
T08	*Design*	Violação de padrão de codificação	Class constructors should not create other objects	Baixa	351
T08	*Design*	Violação de padrão de codificação	Classes should not have too many fields	Baixa	2
T08	*Design*	Violação de padrão de codificação	Classes should not have too many lines of code	Baixa	7
T08	*Design*	Violação de padrão de codificação	Classes should not have too many methods	Baixa	3
T08	*Design*	Violação de padrão de codificação	Closing tag "?>" should be omitted on files containing only PHP	Média	521
T08	*Design*	Violação de padrão de codificação	Files should contain only one top-level class or interface each	Alta	4
T08	*Design*	Violação de padrão de codificação	Files should not have too many lines of code	Baixa	2
T08	*Design*	Violação de padrão de codificação	Functions should not have too many lines of code	Baixa	9

T08	*Design*	Violação de padrão de codificação	Functions should not have too many parameters	Baixa	3
T08	*Design*	Violação de padrão de codificação	Lines should not be too long	Baixa	2.040
T08	*Design*	Violação de padrão de codificação	Method arguments with default values should be last	Média	4
T08	*Design*	Violação de padrão de codificação	Method visibility should be explicitly declared	Média	247
T08	*Design*	Violação de padrão de codificação	Overriding methods should do more than simply call the same method in the super class	Baixa	153
T08	*Design*	Violação de padrão de codificação	[Customized] Remove the extra characters before the opening tag.	Baixa	7
T08	*Design*	Violação de padrão de codificação	Two branches in a conditional structure should not have exactly the same implementation	Média	1
T08	*Design*	Violação de padrão de codificação	Unused "private" fields should be removed	Média	1
T08	*Design*	Violação de padrão de codificação	Unused "private" methods should be removed	Média	1
T08	*Design*	Violação de padrão de codificação	Unused function parameters should be removed	Média	2
T08	*Design*	Violação de padrão de codificação	Unused local variables should be removed	Média	276
T08	Documentação	Documentação inexistente ou inadequada	[Customized] Functions declarations in /include/* should be properly documented in docblock format	Média	6.365
T08	Documentação	Documentação inexistente ou inadequada	Comments should not be located at the end of lines of code	Média	1.283
T08	Documentação	Documentação inexistente ou inadequada	Source files should have a sufficient density of comment lines	Baixa	1.198
T09	Código	Duplicação de código	Methods should not have identical implementations	Baixa	5
T09	Código	Legibilidade do código	"elseif" keyword should be used in place of "else if" keywords	Média	390
T09	Código	Legibilidade do código	"switch" statements should have at least 3 "case" clauses	Baixa	36
T09	Código	Legibilidade do código	Assignments should not be made from within sub-expressions	Média	19
T09	Código	Legibilidade do código	Collapsible "if" statements should be merged	Baixa	1
T09	Código	Legibilidade do código	Duplicate values should not be passed as arguments	Média	4
T09	Código	Legibilidade do código	Functions should use "return" consistently	Média	24
T09	Código	Legibilidade do código	Increment (++) and decrement (--) operators should not be used in a method call	Baixa	6
T09	Código	Legibilidade do código	Local variables should not have the same name as class fields	Baixa	2
T09	Código	Violação do padrão de formatação	"switch case" clauses should not have too many lines of code	Baixa	57
T09	Código	Violação do padrão de formatação	Conditionals should start on new lines	Média	2
T09	Código	Violação do padrão de formatação	Files should contain an empty newline at the end	Média	133
T09	Código	Violação do padrão de formatação	Lines should not end with trailing whitespaces	Média	595

T09	Código	Violação do padrão de formatação	Source code should comply with formatting standards	Média	145
T09	Código	Violação do padrão de nomenclatura	File names should comply with a naming convention	Média	34
T09	Defeito	Código inadequado	Local variables should not be declared and then immediately returned or thrown	Baixa	2
T09	Defeito	Código inadequado	Nested blocks of code should not be left empty	Alta	1
T09	Defeito	Código inadequado	Return of boolean expressions should not be wrapped into an "if-then-else" statement	Baixa	1
T09	Defeito	Código inadequado	Return values from functions without side effects should not be ignored	Grave	1
T09	Código	Legibilidade do código	Redundant pairs of parentheses should be removed	Baixa	27
T09	Defeito	Possibilidade de resultado inesperado	Files should not contain characters before "<?php"	Grave	1
T09	Defeito	Possibilidade de resultado inesperado	Variables should be initialized before use	Grave	10
T09	*Design*	Código muito complexo	Classes should not be too complex	Baixa	2
T09	*Design*	Código muito complexo	Cognitive Complexity of functions should not be too high	Baixa	81
T09	*Design*	Código muito complexo	Expressions should not be too complex	Baixa	1
T09	*Design*	Código muito complexo	Functions should not be too complex	Baixa	37
T09	*Design*	Violação de padrão de codificação	<?php and "<?=" tags should be used	Média	79
T09	*Design*	Violação de padrão de codificação	"switch" statements should not have too many "case" clauses	Baixa	15
T09	*Design*	Violação de padrão de codificação	[Customized] Check for $_REQUEST use in folder /include/*	Alta	136
T09	*Design*	Violação de padrão de codificação	[Customized] SQL commands should not be used in this folder	Média	624
T09	*Design*	Violação de padrão de codificação	Class constructors should not create other objects	Baixa	53
T09	*Design*	Violação de padrão de codificação	Classes should not have too many fields	Baixa	2
T09	*Design*	Violação de padrão de codificação	Classes should not have too many lines of code	Baixa	7
T09	*Design*	Violação de padrão de codificação	Classes should not have too many methods	Baixa	6
T09	*Design*	Violação de padrão de codificação	Closing tag "?>" should be omitted on files containing only PHP	Média	5
T09	*Design*	Violação de padrão de codificação	Files should not have too many lines of code	Baixa	5
T09	*Design*	Violação de padrão de codificação	Functions should not have too many lines of code	Baixa	9
T09	*Design*	Violação de padrão de codificação	Lines should not be too long	Baixa	1.534
T09	*Design*	Violação de padrão de codificação	Method visibility should be explicitly declared	Média	5
T09	*Design*	Violação de padrão de codificação	Overriding methods should do more than simply call the same method in the super class	Baixa	1
T09	*Design*	Violação de padrão de codificação	Track uses of "NOSONAR" comments	Alta	4

T09	*Design*	Violação de padrão de codificação	[Customized] Remove the extra characters before the opening tag.	Baixa	2
T09	*Design*	Violação de padrão de codificação	Unused local variables should be removed	Média	29
T09	Documentação	Documentação inexistente ou inadequada	[Customized] Functions declarations in /include/* should be properly documented in docblock format	Média	5.047
T09	Documentação	Documentação inexistente ou inadequada	Comments should not be located at the end of lines of code	Média	1.394
T09	Documentação	Documentação inexistente ou inadequada	Source files should have a sufficient density of comment lines	Baixa	659
T10	Código	Duplicação de código	Methods should not have identical implementations	Baixa	14
T10	Código	Função depreciada ou incompatível	__construct functions should not make PHP 4-style calls to parent constructors	Alta	10
T10	Código	Função depreciada ou incompatível	Alias functions should not be used	Baixa	6
T10	Código	Função depreciada ou incompatível	Perl-style comments should not be used	Baixa	991
T10	Código	Função depreciada ou incompatível	PHP 4 constructor declarations should not be used	Baixa	301
T10	Código	Função depreciada ou incompatível	The "var" keyword should not be used	Média	352
T10	Código	Legibilidade do código	"elseif" keyword should be used in place of "else if" keywords	Média	139
T10	Código	Legibilidade do código	"for" loop stop conditions should be invariant	Alta	2
T10	Código	Legibilidade do código	"switch" statements should have at least 3 "case" clauses	Baixa	23
T10	Código	Legibilidade do código	Assignments should not be made from within sub-expressions	Média	17
T10	Código	Legibilidade do código	Boolean literals should not be redundant	Baixa	101
T10	Código	Legibilidade do código	Collapsible "if" statements should be merged	Baixa	40
T10	Código	Legibilidade do código	Duplicate values should not be passed as arguments	Média	5
T10	Código	Legibilidade do código	Functions should use "return" consistently	Média	1
T10	Código	Legibilidade do código	Increment (++) and decrement (--) operators should not be used in a method call	Baixa	17
T10	Código	Legibilidade do código	Jump statements should not be followed by dead code	Alta	24
T10	Código	Legibilidade do código	Local variables should not have the same name as class fields	Baixa	56
T10	Código	Legibilidade do código	Redundant pairs of parentheses should be removed	Baixa	47
T10	Código	Legibilidade do código	Sections of code should not be "commented out"	Média	370
T10	Código	Legibilidade do código	String literals should not be concatenated	Baixa	140
T10	Código	Legibilidade do código	Switch cases should end with an unconditional "break" statement	Média	14
T10	Código	Violação do padrão de formatação	"switch case" clauses should not have too many lines of code	Baixa	10

T10	Código	Violação do padrão de formatação	A close curly brace should be located at the beginning of a line	Baixa	8
T10	Código	Violação do padrão de formatação	Colors should be defined in upper case	Baixa	3
T10	Código	Violação do padrão de formatação	Conditionally executed code should be denoted by either indentation or curly braces	Baixa	7
T10	Código	Violação do padrão de formatação	Conditionals should start on new lines	Média	11
T10	Código	Violação do padrão de formatação	Control structures should use curly braces	Baixa	1.062
T10	Código	Violação do padrão de formatação	Files should contain an empty newline at the end	Média	494
T10	Código	Violação do padrão de formatação	Lines should not end with trailing whitespaces	Média	2.454
T10	Código	Violação do padrão de formatação	Modifiers should be declared in the correct order	Baixa	1
T10	Código	Violação do padrão de formatação	More than one property should not be declared per statement	Média	1
T10	Código	Violação do padrão de formatação	Multiline blocks should be enclosed in curly braces	Alta	23
T10	Código	Violação do padrão de formatação	PHP keywords and constants "true", "false", "null" should be lower case	Média	10
T10	Código	Violação do padrão de formatação	Source code should comply with formatting standards	Média	6.134
T10	Código	Violação do padrão de formatação	Statements should be on separate lines	Média	134
T10	Código	Violação do padrão de nomenclatura	File names should comply with a naming convention	Média	49
T10	Defeito	Código inadequado	Empty statements should be removed	Grave	4
T10	Defeito	Código inadequado	Identical expressions should not be used on both sides of a binary operator	Grave	2
T10	Defeito	Código inadequado	Local variables should not be declared and then immediately returned or thrown	Baixa	119
T10	Defeito	Código inadequado	Nested blocks of code should not be left empty	Alta	1
T10	Defeito	Código inadequado	Objects should not be created to be dropped immediately without being used	Grave	1
T10	Defeito	Código inadequado	Parentheses should not be used for calls to "echo"	Média	9
T10	Defeito	Código inadequado	Related "if/else if" statements and "cases" in a "switch" should not have the same condition	Grave	2
T10	Defeito	Código inadequado	Return of boolean expressions should not be wrapped into an "if-then-else" statement	Baixa	3
T10	Defeito	Possibilidade de resultado inesperado	"exit(...)" and "die(...)" statements should not be used	Alta	16
T10	Defeito	Possibilidade de resultado inesperado	"require_once" and "include_once" should be used instead of "require" and "include"	Grave	12

T10	Defeito	Possibilidade de resultado inesperado	All branches in a conditional structure should not have exactly the same implementation	Grave	1
T10	Defeito	Possibilidade de resultado inesperado	Files should not contain characters before "<?php"	Grave	33
T10	Defeito	Possibilidade de resultado inesperado	Static members should be referenced with "static::"	Alta	20
T10	Defeito	Possibilidade de resultado inesperado	Variables should be initialized before use	Grave	6
T10	Defeito	Vulnerabilidade de segurança	Code should not be dynamically injected and executed	Grave	10
T10	*Design*	Código muito complexo	Classes should not be too complex	Baixa	2
T10	*Design*	Código muito complexo	Cognitive Complexity of functions should not be too high	Baixa	38
T10	*Design*	Código muito complexo	Expressions should not be too complex	Baixa	5
T10	*Design*	Código muito complexo	Functions should not be too complex	Baixa	11
T10	*Design*	Violação de padrão de codificação	<?php and "<?=" tags should be used	Média	415
T10	*Design*	Violação de padrão de codificação	"switch" statements should not have too many "case" clauses	Baixa	1
T10	*Design*	Violação de padrão de codificação	[Customized] Check for $_REQUEST use in folder /include/*	Alta	205
T10	*Design*	Violação de padrão de codificação	[Customized] Remove the extra characters before the opening tag.	Baixa	31
T10	*Design*	Violação de padrão de codificação	[Customized] SQL commands should not be used in this folder	Média	280
T10	*Design*	Violação de padrão de codificação	Class constructors should not create other objects	Baixa	126
T10	*Design*	Violação de padrão de codificação	Classes should not have too many fields	Baixa	3
T10	*Design*	Violação de padrão de codificação	Classes should not have too many lines of code	Baixa	8
T10	*Design*	Violação de padrão de codificação	Classes should not have too many methods	Baixa	8
T10	*Design*	Violação de padrão de codificação	Closing tag "?>" should be omitted on files containing only PHP	Média	363
T10	*Design*	Violação de padrão de codificação	Control flow statements "if", "for", "while", "switch" and "try" should not be nested too deeply	Baixa	5
T10	*Design*	Violação de padrão de codificação	Files should contain only one top-level class or interface each	Alta	13
T10	*Design*	Violação de padrão de codificação	Files should not have too many lines of code	Baixa	2
T10	*Design*	Violação de padrão de codificação	Functions should not have too many lines of code	Baixa	2
T10	*Design*	Violação de padrão de codificação	Functions should not have too many parameters	Baixa	1
T10	*Design*	Violação de padrão de codificação	Generic exceptions ErrorException, RuntimeException and Exception should not be thrown	Média	1
T10	*Design*	Violação de padrão de codificação	Lines should not be too long	Baixa	637
T10	*Design*	Violação de padrão de codificação	Method arguments with default values should be last	Média	1
T10	*Design*	Violação de padrão de codificação	Method visibility should be explicitly declared	Média	1.486

T10	*Design*	Violação de padrão de codificação	Overriding methods should do more than simply call the same method in the super class	Baixa	31
T10	*Design*	Violação de padrão de codificação	Superglobals should not be accessed directly	Grave	21
T10	*Design*	Violação de padrão de codificação	[Customized] Remove the extra characters before the opening tag.	Baixa	18
T10	*Design*	Violação de padrão de codificação	Two branches in a conditional structure should not have exactly the same implementation	Média	15
T10	*Design*	Violação de padrão de codificação	Unused "private" fields should be removed	Média	3
T10	*Design*	Violação de padrão de codificação	Unused "private" methods should be removed	Média	36
T10	*Design*	Violação de padrão de codificação	Unused function parameters should be removed	Média	43
T10	*Design*	Violação de padrão de codificação	Unused local variables should be removed	Média	328
T10	Documentação	Documentação inexistente ou inadequada	[Customized] Functions declarations in /include/* should be properly documented in docblock format	Média	3.294
T10	Documentação	Documentação inexistente ou inadequada	Comments should not be located at the end of lines of code	Média	1.891
T10	Documentação	Documentação inexistente ou inadequada	Source files should have a sufficient density of comment lines	Baixa	587
T11	Código	Duplicação de código	Methods should not have identical implementations	Baixa	7
T11	Código	Função depreciada ou incompatível	__construct functions should not make PHP 4-style calls to parent constructors	Alta	4
T11	Código	Função depreciada ou incompatível	Alias functions should not be used	Baixa	4
T11	Código	Função depreciada ou incompatível	Perl-style comments should not be used	Baixa	1.186
T11	Código	Função depreciada ou incompatível	PHP 4 constructor declarations should not be used	Baixa	149
T11	Código	Função depreciada ou incompatível	The "var" keyword should not be used	Média	335
T11	Código	Legibilidade do código	"elseif" keyword should be used in place of "else if" keywords	Média	103
T11	Código	Legibilidade do código	"switch" statements should have at least 3 "case" clauses	Baixa	12
T11	Código	Legibilidade do código	Assignments should not be made from within sub-expressions	Média	17
T11	Código	Legibilidade do código	Boolean literals should not be redundant	Baixa	78
T11	Código	Legibilidade do código	Collapsible "if" statements should be merged	Baixa	41
T11	Código	Legibilidade do código	Duplicate values should not be passed as arguments	Média	4
T11	Código	Legibilidade do código	Functions should use "return" consistently	Média	6
T11	Código	Legibilidade do código	Increment (++) and decrement (--) operators should not be used in a method call	Baixa	55
T11	Código	Legibilidade do código	Jump statements should not be followed by dead code	Alta	56

T11	Código	Legibilidade do código	Local variables should not have the same name as class fields	Baixa	47
T11	Código	Legibilidade do código	Sections of code should not be "commented out"	Média	130
T11	Código	Legibilidade do código	String literals should not be concatenated	Baixa	24
T11	Código	Legibilidade do código	Switch cases should end with an unconditional "break" statement	Média	5
T11	Código	Violação do padrão de formatação	"switch case" clauses should not have too many lines of code	Baixa	32
T11	Código	Violação do padrão de formatação	Conditionally executed code should be denoted by either indentation or curly braces	Baixa	2
T11	Código	Violação do padrão de formatação	Control structures should use curly braces	Baixa	1.080
T11	Código	Violação do padrão de formatação	Files should contain an empty newline at the end	Média	277
T11	Código	Violação do padrão de formatação	Lines should not end with trailing whitespaces	Média	2.387
T11	Código	Violação do padrão de formatação	Multiline blocks should be enclosed in curly braces	Alta	12
T11	Código	Violação do padrão de formatação	PHP keywords and constants "true", "false", "null" should be lower case	Média	69
T11	Código	Violação do padrão de formatação	Source code should comply with formatting standards	Média	5.738
T11	Código	Violação do padrão de formatação	Statements should be on separate lines	Média	271
T11	Código	Violação do padrão de nomenclatura	File names should comply with a naming convention	Média	27
T11	Defeito	Código inadequado	Empty statements should be removed	Grave	1
T11	Defeito	Código inadequado	Errors should not be silenced	Alta	2
T11	Defeito	Código inadequado	Local variables should not be declared and then immediately returned or thrown	Baixa	77
T11	Defeito	Código inadequado	Objects should not be created to be dropped immediately without being used	Grave	2
T11	Defeito	Código inadequado	Parentheses should not be used for calls to "echo"	Média	10
T11	Defeito	Código inadequado	Return of boolean expressions should not be wrapped into an "if-then-else" statement	Baixa	3
T11	Defeito	Possibilidade de resultado inesperado	"exit(...)" and "die(...)" statements should not be used	Alta	20
T11	Defeito	Possibilidade de resultado inesperado	"require_once" and "include_once" should be used instead of "require" and "include"	Grave	8
T11	Defeito	Possibilidade de resultado inesperado	All branches in a conditional structure should not have exactly the same implementation	Grave	5
T11	Defeito	Possibilidade de resultado inesperado	Files should not contain characters before "<?php"	Grave	37
T11	Defeito	Possibilidade de resultado inesperado	Static members should be referenced with "static::"	Alta	5
T11	Defeito	Possibilidade de resultado inesperado	Variables should be initialized before use	Grave	10

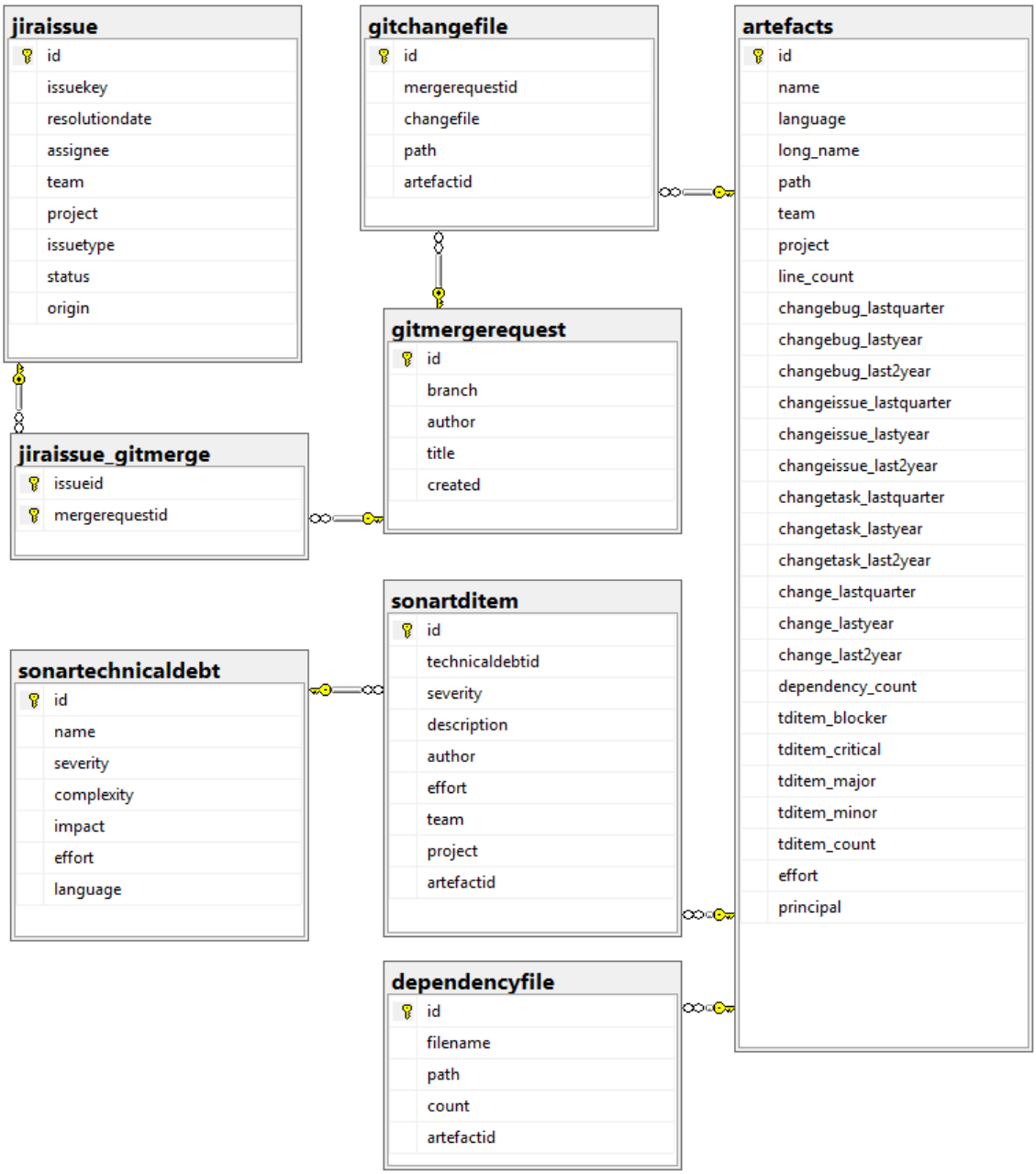
T11	Defeito	Possibilidade de resultado inesperado	Variables should not be self-assigned	Grave	1
T11	Defeito	Vulnerabilidade de segurança	Code should not be dynamically injected and executed	Grave	10
T11	Defeito	Vulnerabilidade de segurança	Configuration should not be changed dynamically	Alta	1
T11	*Design*	Código muito complexo	Classes should not be too complex	Baixa	3
T11	*Design*	Código muito complexo	Cognitive Complexity of functions should not be too high	Baixa	39
T11	*Design*	Código muito complexo	Expressions should not be too complex	Baixa	7
T11	*Design*	Código muito complexo	Functions should not be too complex	Baixa	18
T11	Defeito	Legibilidade do código	Redundant pairs of parentheses should be removed	Baixa	11
T11	*Design*	Violação de padrão de codificação	<?php and "<?=" tags should be used	Média	271
T11	*Design*	Violação de padrão de codificação	"final" should not be used redundantly	Alta	57
T11	*Design*	Violação de padrão de codificação	[Customized] Check for $_REQUEST use in folder /include/*	Alta	137
T11	*Design*	Violação de padrão de codificação	[Customized] Remove the extra characters before the opening tag.	Baixa	37
T11	*Design*	Violação de padrão de codificação	[Customized] SQL commands should not be used in this folder	Média	206
T11	*Design*	Violação de padrão de codificação	Class constructors should not create other objects	Baixa	99
T11	*Design*	Violação de padrão de codificação	Classes should not be coupled to too many other classes (Single Responsibility Principle)	Baixa	1
T11	*Design*	Violação de padrão de codificação	Classes should not have too many fields	Baixa	1
T11	*Design*	Violação de padrão de codificação	Classes should not have too many lines of code	Baixa	5
T11	*Design*	Violação de padrão de codificação	Classes should not have too many methods	Baixa	8
T11	*Design*	Violação de padrão de codificação	Closing tag "?>" should be omitted on files containing only PHP	Média	216
T11	*Design*	Violação de padrão de codificação	Control flow statements "if", "for", "while", "switch" and "try" should not be nested too deeply	Baixa	1
T11	*Design*	Violação de padrão de codificação	Files should contain only one top-level class or interface each	Alta	3
T11	*Design*	Violação de padrão de codificação	Files should not have too many lines of code	Baixa	1
T11	*Design*	Violação de padrão de codificação	Functions should not have too many lines of code	Baixa	2
T11	*Design*	Violação de padrão de codificação	Generic exceptions ErrorException, RuntimeException and Exception should not be thrown	Média	5
T11	*Design*	Violação de padrão de codificação	Lines should not be too long	Baixa	668
T11	*Design*	Violação de padrão de codificação	Method arguments with default values should be last	Média	2
T11	*Design*	Violação de padrão de codificação	Method visibility should be explicitly declared	Média	1.110
T11	*Design*	Violação de padrão de codificação	Overriding methods should do more than simply call the same method in the super class	Baixa	26

T11	*Design*	Violação de padrão de codificação	Superglobals should not be accessed directly	Grave	6
T11	*Design*	Violação de padrão de codificação	[Customized] Remove the extra characters before the opening tag.	Baixa	3
T11	*Design*	Violação de padrão de codificação	Two branches in a conditional structure should not have exactly the same implementation	Média	9
T11	*Design*	Violação de padrão de codificação	Unused "private" fields should be removed	Média	1
T11	*Design*	Violação de padrão de codificação	Unused "private" methods should be removed	Média	15
T11	*Design*	Violação de padrão de codificação	Unused function parameters should be removed	Média	15
T11	*Design*	Violação de padrão de codificação	Unused local variables should be removed	Média	180
T11	Documentação	Documentação inexistente ou inadequada	[Customized] Functions declarations in /include/* should be properly documented in docblock format	Média	2.803
T11	Documentação	Documentação inexistente ou inadequada	Comments should not be located at the end of lines of code	Média	1.052
T11	Documentação	Documentação inexistente ou inadequada	Source files should have a sufficient density of comment lines	Baixa	314
T12	Código	Função depreciada ou incompatível	__construct functions should not make PHP 4-style calls to parent constructors	Alta	7
T12	Código	Função depreciada ou incompatível	Perl-style comments should not be used	Baixa	260
T12	Código	Função depreciada ou incompatível	PHP 4 constructor declarations should not be used	Baixa	306
T12	Código	Função depreciada ou incompatível	The "var" keyword should not be used	Média	177
T12	Código	Legibilidade do código	"elseif" keyword should be used in place of "else if" keywords	Média	10
T12	Código	Legibilidade do código	"switch" statements should have at least 3 "case" clauses	Baixa	5
T12	Código	Legibilidade do código	Assignments should not be made from within sub-expressions	Média	5
T12	Código	Legibilidade do código	Boolean literals should not be redundant	Baixa	13
T12	Código	Legibilidade do código	Collapsible "if" statements should be merged	Baixa	5
T12	Código	Legibilidade do código	Functions should use "return" consistently	Média	3
T12	Código	Legibilidade do código	Increment (++) and decrement (--) operators should not be used in a method call	Baixa	56
T12	Código	Legibilidade do código	Local variables should not have the same name as class fields	Baixa	3
T12	Código	Legibilidade do código	References should not be passed to function calls	Alta	1
T12	Código	Legibilidade do código	Sections of code should not be "commented out"	Média	84
T12	Código	Legibilidade do código	String literals should not be concatenated	Baixa	11
T12	Código	Violação do padrão de formatação	"switch case" clauses should not have too many lines of code	Baixa	3
T12	Código	Violação do padrão de formatação	Control structures should use curly braces	Baixa	168

T12	Código	Violação do padrão de formatação	Files should contain an empty newline at the end	Média	424
T12	Código	Violação do padrão de formatação	Lines should not end with trailing whitespaces	Média	769
T12	Código	Violação do padrão de formatação	Modifiers should be declared in the correct order	Baixa	5
T12	Código	Violação do padrão de formatação	Multiline blocks should be enclosed in curly braces	Alta	10
T12	Código	Violação do padrão de formatação	PHP keywords and constants "true", "false", "null" should be lower case	Média	3
T12	Código	Violação do padrão de formatação	Source code should comply with formatting standards	Média	1.620
T12	Código	Violação do padrão de formatação	Statements should be on separate lines	Média	20
T12	Código	Violação do padrão de nomenclatura	[Customized] Function names in /include/exp/* should comply with the naming convention	Alta	1
T12	Código	Violação do padrão de nomenclatura	File names should comply with a naming convention	Média	16
T12	Defeito	Código inadequado	Local variables should not be declared and then immediately returned or thrown	Baixa	66
T12	Defeito	Código inadequado	Nested blocks of code should not be left empty	Alta	2
T12	Defeito	Código inadequado	Parentheses should not be used for calls to "echo"	Média	2
T12	Defeito	Código inadequado	Return values from functions without side effects should not be ignored	Grave	1
T12	Defeito	Possibilidade de resultado inesperado	Files should not contain characters before "<?php"	Grave	3
T12	Defeito	Possibilidade de resultado inesperado	Variables should not be self-assigned	Grave	1
T12	*Design*	Código muito complexo	Cognitive Complexity of functions should not be too high	Baixa	2
T12	Código	Legibilidade do código	Redundant pairs of parentheses should be removed	Baixa	4
T12	*Design*	Violação de padrão de codificação	<?php and "<?=" tags should be used	Média	372
T12	*Design*	Violação de padrão de codificação	[Customized] Check for $_REQUEST use in folder /include/*	Alta	75
T12	*Design*	Violação de padrão de codificação	[Customized] Remove the extra characters before the opening tag.	Baixa	3
T12	*Design*	Violação de padrão de codificação	[Customized] SQL commands should not be used in this folder	Média	67
T12	*Design*	Violação de padrão de codificação	Class constructors should not create other objects	Baixa	128
T12	*Design*	Violação de padrão de codificação	Classes should not have too many lines of code	Baixa	1
T12	*Design*	Violação de padrão de codificação	Classes should not have too many methods	Baixa	1
T12	*Design*	Violação de padrão de codificação	Closing tag "?>" should be omitted on files containing only PHP	Média	394
T12	*Design*	Violação de padrão de codificação	Lines should not be too long	Baixa	118
T12	*Design*	Violação de padrão de codificação	Method arguments with default values should be last	Média	8
T12	*Design*	Violação de padrão de codificação	Method visibility should be explicitly declared	Média	1.023

T12	*Design*	Violação de padrão de codificação	Overriding methods should do more than simply call the same method in the super class	Baixa	50
T12	*Design*	Violação de padrão de codificação	Superglobals should not be accessed directly	Grave	4
T12	*Design*	Violação de padrão de codificação	[Customized] Remove the extra characters before the opening tag.	Baixa	9
T12	*Design*	Violação de padrão de codificação	Two branches in a conditional structure should not have exactly the same implementation	Média	1
T12	*Design*	Violação de padrão de codificação	Unused "private" fields should be removed	Média	1
T12	*Design*	Violação de padrão de codificação	Unused function parameters should be removed	Média	8
T12	*Design*	Violação de padrão de codificação	Unused local variables should be removed	Média	25
T12	Documentação	Documentação inexistente ou inadequada	[Customized] Functions declarations in /include/* should be properly documented in docblock format	Média	1.951
T12	Documentação	Documentação inexistente ou inadequada	Comments should not be located at the end of lines of code	Média	331
T12	Documentação	Documentação inexistente ou inadequada	Source files should have a sufficient density of comment lines	Baixa	435

APÊNDICE E. MODELO DE DADOS DA PRIORIZAÇÃO DO ARQUIVO-FONTE



Descrição das tabelas e dos atributos:

Tabela: artefacts	
Atributo	**Descrição**
id	Código sequencial do arquivo
name	Nome do arquivo
language	Linguagem [Java, JS, PHP, XML]
long_name	Junção do path e o nome do arquivo
path	Path do arquivo
team	Equipe responsável pelo arquivo
project	Nome do Projeto
line_count	Quantidade de linha de código
changebug_lastquarter	Quantidade de alterações originadas por bug registrado por cliente nos últimos 3 meses
changebug_lastyear	Quantidade de alterações originadas por bug registrado por cliente no último ano
changebug_last2year	Quantidade de alterações originadas por bug registrado por cliente nos últimos 2 anos
changeissue_lastquarter	Quantidade de alterações originadas por bug registrado internamente cliente nos últimos 3 meses
changeissue_lastyear	Quantidade de alterações originadas por bug registrado internamente no último ano
changeissue_last2year	Quantidade de alterações originadas por bug registrado internamente nos últimos 2 anos
changetask_lastquarter	Quantidade de alterações originadas por evolução nos últimos 3 meses
changetask_lastyear	Quantidade de alterações originadas por evolução no último ano
changetask_last2year	Quantidade de alterações originadas por evolução nos últimos 2 anos
change_lastquarter	Quantidade total de alterações dos últimos 3 meses
change_lastyear	Quantidade total de alterações do último ano
change_last2year	Quantidade total de alterações dos últimos 2 anos
dependency_count	Quantidade de arquivos dependente deste arquivo
tditem_blocker	Quantidade de itens de Dívida Técnica classificado com prioridade igual a Blocker
tditem_critical	Quantidade de itens de Dívida Técnica classificado com prioridade igual a Critical
tditem_major	Quantidade de itens de Dívida Técnica classificado com prioridade igual a Major
tditem_minor	Quantidade de itens de Dívida Técnica classificado com prioridade igual a Minor
tditem_count	Quantidade de todos itens de Dívida Técnica do arquivo
effort	Quantidade de minutos necessários para resolver todos os itens de Dívida Técnica deste arquivo.
principal	Classificação do Juro da Dívida Técnica [Very High; High; Moderate; Low; Very low; Not apply]

Observação: Registro de todos os arquivos-fonte utilizados para a análise do PriorTD

Tabela: sonartechnicaldebt	
Atributo	**Descrição**
id	Código sequencial da Dívida Técnica
name	Nome da Dívida Técnica
severity	Prioridade [Blocker; Critical; Major; Minor]
complexity	Complexidade [Very High; High; Moderate; Low; Very low]
impact	Impacto [Very High; High; Moderate; Low; Very low]
effort	Esforço para eliminar a Dívida Técnica
language	Linguagem [Java, JS, PHP, XML]
Observação: Regras da Dívida Técnica utilizada pelo SonarQube para analisar o código fonte.	

Tabela: sonartditem	
Atributo	**Descrição**
id	Código sequencial do item da Dívida Técnica
technicaldebtid	Identificador da Dívida Técnica
severity	Prioridade [Blocker; Critical; Major; Minor]
description	Descrição do item da Dívida Técnica
author	O último que fez o commit da dívida, informação extraída do GIT
effort	Esforço para eliminar o item da Dívida Técnica
team	Nome da equipe responsável
project	Nome do projeto
artefactid	Identificador do artefato
Observação: Tabela com todos os itens de Dívida Técnica identificado pelo SonarQube conforme as regras configuradas.	

Tabela: jiraissue	
Atributo	**Descrição**
id	Código sequencial interno
issuekey	Identificador chave da tarefa, código alfanumérico único.
resolutiondate	Data que a tarefa foi resolvida (status igual a Done).
assignee	Responsável pela finalização da tarefa.
team	Nome da equipe responsável pela tarefa.
issuetype	Tipo da tarefa [Bug; Story; Simple Task].
status	Situação atual da tarefa [Open; Reject; To Do; In Progress; Done].
origin	Origem da identificação da tarefa [Cliente; Interna].
mergerequestid	Identificador do Merge Request criado para o Git.
project	Nome do projeto que esta tarefa faz parte.
Observação: Dados extraídos da base de dados do Jira. Representa todos as tarefas registradas e executadas pelas equipes.	

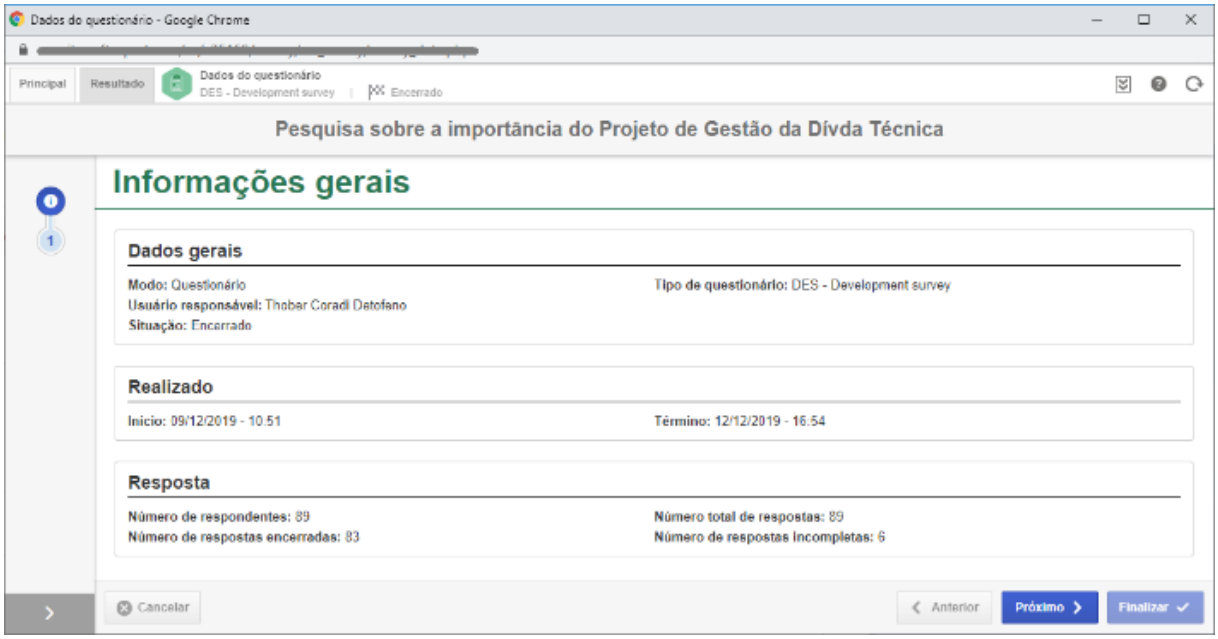
Tabela: jiraissue_gitmerge	
Atributo	**Descrição**
Issueid	Código da issue criado no Jira
mergerequestid	Código do Merge Request criado para o Git.
Observação: Tabela que relaciona os merges requests criado para resolver uma issue registrada no Jira. Pode ter mais de um merge por issue.	

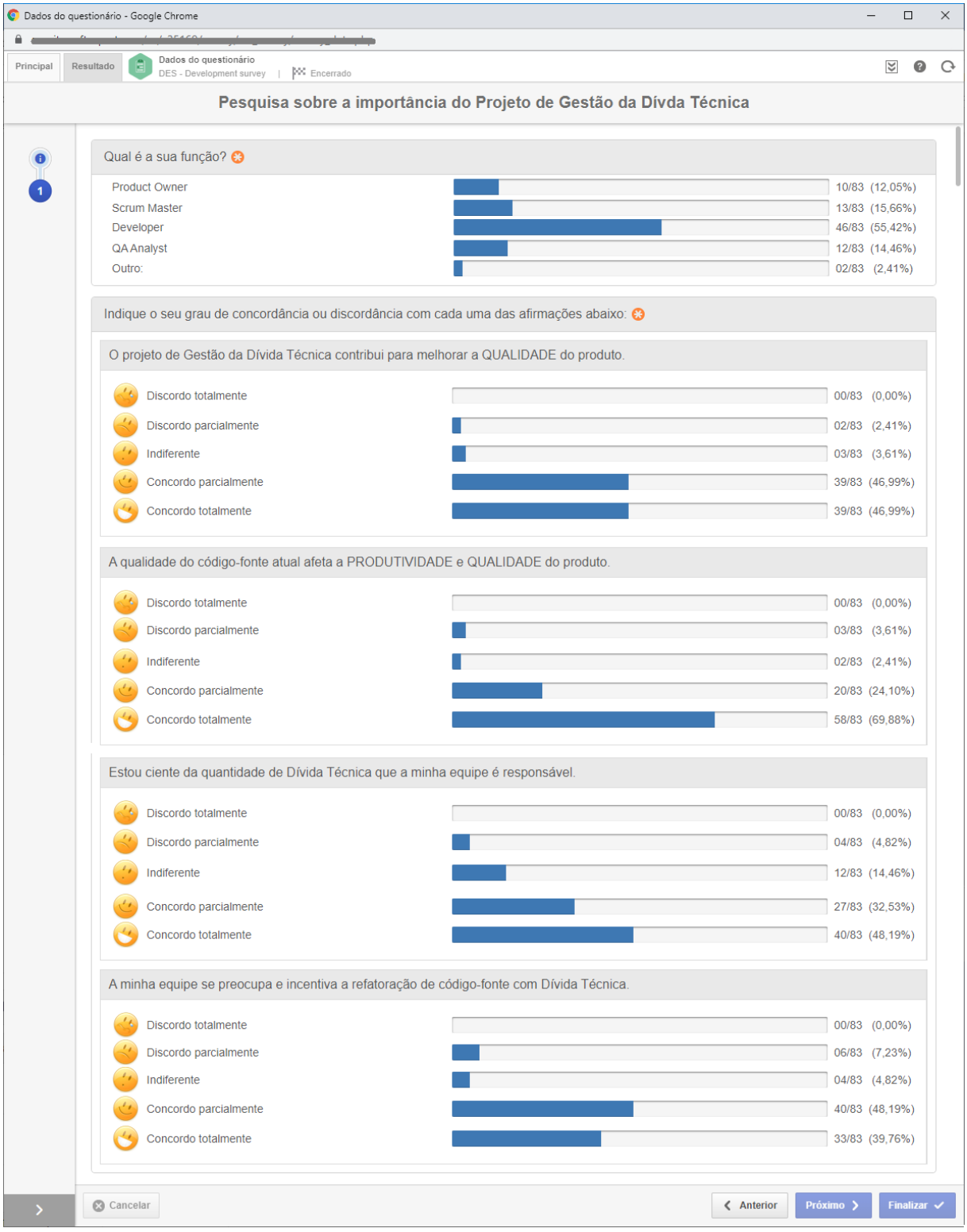
Tabela: gitmergerequest	
Atributo	**Descrição**
Id	Identificador do Merge Request
branch	Ponteiro de referência ao repositório [Master; Dev]
author	Nome do autor do Merge Request
title	Título dado ao merge request utilizado para facilitar a pesquisa e identificação de sua origem
created	Data que foi criado o Merge Request
Observação: Possui todos os merges requests criado no GIT. Cada merge request possui um ou mais arquivos alterados.	

Tabela: gitchangefile	
Atributo	**Descrição**
Id	Código sequencial interno
mergerequestid	Identificador do Merge Request
changefile	Nome do arquivo alterado
path	Caminho onde o arquivo está localizado
artefactid	Identificador do artefato
Observação: Todas as alterações nos arquivos estão registradas nesta tabela, a ligação com a tabela de artefato ajuda na descoberta da origem da alteração.	

Tabela: dependencyfile	
Atributo	**Descrição**
id	Código sequencial
filename	Nome do arquivo
path	Caminho onde o arquivo está localizado
count	Quantidade de ocorrência deste arquivo em outros arquivos
artefactid	Identificador do artefato
Observação: Resultados da análise de dependência de cada arquivo fonte.	

APÊNDICE F. RESULTADO DO QUESTIONÁRIO DE AVALIAÇÃO DO 2º CICLO




Dados do questionário - Google Chrome
Principal
Resultado
Dados do questionário
DES - Development survey
Encerrado
Pesquisa sobre a importância do Projeto de Gestão da Dívda Técnica
1
Qual é a sua função?
Product Owner 10/83 (12,05%)
Scrum Master 13/83 (15,66%)
Developer 46/83 (55,42%)
QA Analyst 12/83 (14,46%)
Outro: 02/83 (2,41%)
Indique o seu grau de concordância ou discordância com cada uma das afirmações abaixo:
O projeto de Gestão da Dívida Técnica contribui para melhorar a QUALIDADE do produto.
Discordo totalmente 00/83 (0,00%)
Discordo parcialmente 02/83 (2,41%)
Indiferente 03/83 (3,61%)
Concordo parcialmente 39/83 (46,99%)
Concordo totalmente 39/83 (46,99%)
A qualidade do código-fonte atual afeta a PRODUTIVIDADE e QUALIDADE do produto.
Discordo totalmente 00/83 (0,00%)
Discordo parcialmente 03/83 (3,61%)
Indiferente 02/83 (2,41%)
Concordo parcialmente 20/83 (24,10%)
Concordo totalmente 58/83 (69,88%)
Estou ciente da quantidade de Dívida Técnica que a minha equipe é responsável.
Discordo totalmente 00/83 (0,00%)
Discordo parcialmente 04/83 (4,82%)
Indiferente 12/83 (14,46%)
Concordo parcialmente 27/83 (32,53%)
Concordo totalmente 40/83 (48,19%)
A minha equipe se preocupa e incentiva a refatoração de código-fonte com Dívida Técnica.
Discordo totalmente 00/83 (0,00%)
Discordo parcialmente 06/83 (7,23%)
Indiferente 04/83 (4,82%)
Concordo parcialmente 40/83 (48,19%)
Concordo totalmente 33/83 (39,76%)
Cancelar
Anterior
Próximo
Finalizar


Dados do questionário - Google Chrome
Principal
Resultado
Dados do questionário
DES - Development survey
Encerrado
Pesquisa sobre a importância do Projeto de Gestão da Dívda Técnica
Na sua opinião quais foram os benefícios que o projeto de Gestão da Dívida Técnica trouxe no Desenvolvimento ou nos projetos da sua equipe?
Padronização de código
Trouxe um padrão para o código. Algo que já deveria existir mas não era aplicado.
Revisão de códigos possivelmente ruins e problemáticos.
Estabeleceu critérios, prazos e padronização do código fonte. Mas os critérios precisam ser revisados.
Levou a equipe a procurar identificar formas melhores de desenvolver.
O projeto do Sonar deixou bem claro que existe interesse da empresa na melhoria de código legado, não haviam metas nesse quesito. A manutenibilidade de um código escrito dentro da exp é muito melhor porque nos obriga a seguir alguns padrões, porém houve uma freada na velocidade de entrega por conta do tempo gasto em testes unitários. O projeto do Sonar nos fez excluir bastante código morto que recebia adequações desnecessárias.
Códigos mais enxutos e organizados, com redução de bugs.
Padrão de código. --- A falta de algo para consulta dos problemas atuais do arquivo durante a edição, faz com que sejam criados outras issues.
Padronização do código fonte
Padronização de código
Os benefícios são nítidos, como por exemplo: códigos mais simples, complexidade reduzida, melhor uso da linguagem de programação, detecção de práticas ruins e anti-padroes.. todos os pontos propostos pelo SONAR são bons para o processo de desenvolvimento e qualidade.
Trouxe um padrão de desenvolvimento. Código mais clean (limpo), fácil de entendimento. Melhor código, menor esforço para solucionar possíveis bugs
O Sonar trouxe algum padrão para onde não existia, isso melhorou a qualidade de códigos, principalmente os mais antigos. Porém, seria bom se ao abrir um MR fosse possível fazer uma verificação do SONAR dos arquivos alterados\incluídos.
Ajudam a identificar alterações no sistema que causaram impacto em lugares que não haviam sido mapeados.
Usando como referência apenas o trabalho realizado em nossa equipe, estamos sempre tentando "agradar" a entidade Sonar, apontando todas regras do Sonar durante a etapa de Code Review, evitando assim que código feio, sujo e mal formatado entre para a origin.
A remoção de arquivos não mais utilizados trouxe melhoria na segurança em alterar diversos arquivos.
A ideia de padronização de código já rondava a equipe porém antes da existência do sonar, acabava tendo muita discordância pois sempre tinha alguém achando que as regras são todas da cabeça de alguém
Melhoria da qualidade de código, o que impacta diretamente na legibilidade, entendimento e manutenção do mesmo.
Em nossa equipe, não notei benefícios.
Padronização de código, deixou o código mais legível.
COM O SONAR PODEMOS TOMAR AÇÕES MAIS RÁPIDAS NO QUE DIZEM RESPEITO A QUALIDADE
Foram pegos alguns erros de SQL e centralizado os SQLs em um único local. Por enquanto não vi benefício maior.
A percepção atual é de que o SONAR é uma das ferramentas que ajudou de certa forma a diminuir o número de bugs que temos em clientes. Outro ponto importante do SONAR, é que ele nos indicou os arquivos que possuem uma complexidade muito grande para refatorarmos, e isso contribui bastante para a qualidade do produto.
Boa prática no código fonte e reaproveitamento de código.
Melhorou a padronização dos fontes e forneceu insumos para reestruturação de alguns recursos.
Padronização na identação do código ajuda muito. Documentação da função, ajuda um pouco , mas se o nome da função for clara o suficiente, com nomes de parâmetros claros o suficiente,não haveria necessidade de documentá-las. Validar espaçamento, já acho um pouco de excesso.
Melhorar a qualidade do código-fonte para que não cause vulnerabilidades no sistema, muitas vezes por falta de conhecimento acaba sendo cometido. Com essas regras ajuda a ter um código mais limpo e evita fazer coisas desnecessárias.
Sem dúvidas a qualidade de código, pois o desenvolvedor é forçado a escrever um código pensado, com mais qualidade, do que simplesmente criar uma gambiarra que vai funcionar no momento, mas que vai dar dor de cabeça depois.
Incentivo à reutilização de SQLs que estão na pasta exp; Conscientização sobre o uso de $_REQUEST na pasta include; Incentivo à documentação de métodos do PHP; Padronização do código (espaçamento, indentação);
A contribuição do projeto Sonar foi a padronização minima entre os arquivos do sistema, respeitando as regras definidas pela Soft. Vejo que esse projeto deve ser continuo e as regras devem ser criadas gradativamente para termos um código mais homogêneo. Uma sugestão que eu deixo é a criação de regras que removam os métodos depreciados por métodos equivalentes, assim como problemas já sabidos no Encode de textos e conversão para salvar em diferentes formatos de bancos.
Acredito que o Sonar trouxe muitos beneficios na melhoria de código, mas acho que seria mais ágil e eficaz se as inconsistências fossem apresentadas no Sonar antes do merge ser aprovado, pois acredito que muitas pessoas não lembrem de ver essas inconsistências após a aprovação do merge
Padronização do código PHP. Identificação de pequenos bugs antes de cairem em produção. Identificação de vulnerabilidades.
Não participo muito na parte de desenvolvimento que envolve PHP, mas em algumas reuniões da equipe sei que existem iniciativas para diminuir índices de bug, duplicidade de código, arquivos não mais utilizados etc
Alterar o codigo fonte para adequar ao sonar não é o suficiente. Pouca mão de obra para fazer isso. O ideal seria refatorar para um framework mais moderno.
Abriu a cabeça para se pensar melhor no código produzido e em novas práticas de desenvolvimento que estimulam a simplicidade e manutenção do código fonte
Padrão de codigo
Está trazendo um melhor entendimento do código fonte.
Com o SONAR, passamos a não poder criar SQLs na pasta include, que acarretou solução de criarmos a exp.
Refatoração e exclusão de códigos antigos e defasados.
Código limpo, dentro dos padrões definidos
Alguns, pois ajuda a reescrevermos trechos de código em que há badcoding. Mas, o fator de análise humano, principalmente de toda a equipe e desenvolvedores é essencial.
Padrões foram definidos e ferramentas que melhoram o desenvolvimento foram criadas.
Facilidade de compreenção do código desenvolvido por outras equipes.
Ajudou no padrão a definir um padrão de código, e remover ineficiências e vulnerabilidades do código.
Um código mais limpo e padronizado.
Melhora na leitura e interpretação do codigo.
Alguns padrões de desenvolvimento, que algumas vezes podem ser bobos, mas fazem grande diferença quando temos uma codificação robusta.
Melhoria de código Diminuição de arquivos sem utilização
Melhoria na qualidade do código, quando falamos em seguir os padrões de escrita de códigos PHP. Refatoração de trechos de código que antes a manutenção era custosa, desta forma alguns bugs que eram frequentes passaram a não existir.
Os que só vem pra dar ctrl C em código pronto, pelo menos agora tem que dar uma ajeitada antes de commitar
Forçou a implementação de pelo menos alguma padronização no código e de algumas boas práticas
A boa prática e os padrões de desenvolvimento que ajudam com que façamos um código mais limpo e eficiente.
Cancelar
Anterior
Próximo
Finalizar

APÊNDICE G. RESULTADOS DA AÇÃO 1 DO 2º CICLO

Projeto com as ações propostas após a análise dos resultados do PriorTD:

Equipe	Situação	EAP	Ação	% Real
T01	Encerrada	7.1.1	Aumentar a coesão dos arquivos selecionados	100
T01	Encerrada	7.1.2	Diminuir o acoplamento dos arquivos selecionados	100
T01	Encerrada	7.1.3	Documentar funções	100
T01	Encerrada	7.1.4	Rever os modificadores de acesso dos arquivos selecionados	100
T01	Encerrada	7.1.5	Rastreamento dos recursos genéricos	100
T01	Encerrada	7.1.6	Automatizar os caminhos críticos do XXXX	100
T02	Execução	11.1.1	Refatoração dos arquivos sugeridos pelo PriorTD	25
T02	Execução	11.1.2	Documentar todas as funções dos arquivos sugeridos pelo PriorTD	25
T02	Execução	11.1.3	Aumentar cobertura de testes unitários dos arquivos listados	25
T02	Execução	11.1.4	Resolver 80% das issues do Sonar dos arquivos sugeridos pelo PriorTD	25
T03	Encerrada	2.1.1	Adoção de cross test em merges selecionados	100
T03	Execução	2.1.2	Cobrir com testes 80% do projeto do formulário responsivo no xxxxx	10
T03	Execução	2.1.3	Cobrir com testes 85% do projeto do xxxxx responsivo no react	30
T03	Iniciar	2.1.4	Automatizar testes de 100% dos métodos SOAP	0
T03	Iniciar	2.1.5	Incrementar validação dos retornos de falha nos testes de automação dos métodos web service SOAP	0
T03	Iniciar	2.1.6	Automatizar testes de 100% dos métodos REST	0
T03	Execução	2.1.7	Evoluir formulário responsivo e suas regras	10
T03	Execução	2.1.8	Construir nova engine de xxxxxx	1
T04	Encerrada	12.1.1	Refatorar arquivos	100
T04	Encerrada	12.1.2	Documentar funções	100
T04	Encerrada	12.1.3	Eliminar issues do Sonar	100
T05	Execução	4.1.1	Modularização das classes action e data de execução de atividades	5
T05	Execução	4.1.2	Refatorar movimentações do XXXXX	5
T05	Iniciar	4.1.3	Refatorar integração com XXXXX	0
T05	Encerrada	4.1.4	Refatorar interfaces do XXXXXX	100
T06	Execução	9.1.1	Teste automatizado	62
T06	Execução	9.1.2	Documentar todos os métodos e reduzir 90% dos erros do sonar. (Arquivos com mais erro de clientes sugeridos pelo PiorTD)	1
T06	Execução	9.1.3	Documentar todos os métodos e reduzir 90% erros do sonar dos arquivos com mais *bugs* internos sugeridos pelo PriorTD	1
T07	Iniciar	5.1.1	Refatorar os arquivos com mais *bugs* de clientes	0

T07	Iniciar	5.1.2	Refatorar os arquivos com mais *bugs* Interno	0
T07	Encerrada	5.1.3	Avaliar arquivos não utilizados	100
T07	Encerrada	5.1.4	Automatizar caminho crítico - Revisão	100
T07	Encerrada	5.1.5	Automatizar caminho crítico - Envio de cópias impressas	100
T08	Iniciar	8.1.1	Refatorar os arquivos com maior quantidade de *bugs* e documentar as funções	80
T08	Iniciar	8.1.2	Documentar todas as funções	80
T08	Encerrada	8.1.3	Aumentar a cobertura de teste automatizado e teste unitário dos arquivos selecionados	100
T08	Iniciar	8.1.4	Eliminar 90% de todas as issues do Sonar dos arquivos sugeridos pelo PriorTD	75
T09	Encerrada	3.1.1	Reduzir, refatorar e documentar as funções dos arquivos sugeridos pelo PriorTD	100
T09	Encerrada	3.1.2	Reduzir as issues do Sonar dos dos arquivos sugeridos pelo PriorTD; Aumentar a cobertura de teste automatizado e teste unitário dos arquivos sugeridos pelo PriorTD	100
T09	Encerrada	3.1.3	Atingir 30% de cobertura de teste automatizado	100
T09	Encerrada	3.1.4	Converter frames de configuração para o novo framework PHP	100
T09	Encerrada	3.1.5	Atingir 50% de cobertura de teste unitário no cálculo	100
T09	Encerrada	3.1.6	Realizar pelo menos quatro atividades de usabilidade por release	100
T10	Encerrada	6.1.1	Realizar ações nos arquivos com mais problemas de Clientes e Interno	100
T10	Encerrada	6.1.2	Refatoração de RPCs	100
T10	Encerrada	6.1.3	Refatoração de callbacks	100
T10	Encerrada	6.1.4	Refatoração de arquivos functions	100
T10	Encerrada	6.1.5	Tratamento de exceções e erros	100
T10	Encerrada	6.1.6	SONAR (Blocker e Critical)	100
T10	Encerrada	6.1.7	Migração de herança para composição quando possível	100
T10	Encerrada	6.1.8	PHP PSR-2 e UTF-8	100
T11	Encerrada	1.1.1	Refatorar arquivos	100
T11	Encerrada	1.1.2	Documentar todos os métodos dos arquivos sugeridos pelo PriorTD	100
T11	Execução	1.1.3	Refatorar o arquivo xxx.inc	30
T11	Execução	1.1.4	Refatorar o arquivo xxx.php	1
T11	Encerrada	1.1.5	Eliminar 70% de todas as issues dos arquivos sugeridos pelo PriorTD	100
T12	Encerrada	10.1.1	Documentar a maioria das funções dos arquivos sugeridos pelo PriorTD	100
T12	Encerrada	10.1.2	Testes	100
T12	Encerrada	10.1.3	Sonar	100
T12	Encerrada	10.1.4	Atuar na refatoração do arquivo xxxx.inc sugerido pelo PriorTD	100
T12	Encerrada	10.1.5	Atuar em conjunto com a equipe do XXXX	100
			% Médio realizado	**68,70**

Resultados por equipe e por prioridade do pagamento dos itens de Dívida Técnica:

Equipe	Prioridade	Referência	Meta	Realizado	% Diminuição	Atingiu a meta?
T01	Grave	714	178	219	69,33	NÃO
T01	Alta	1.348	1.011	1.329	1,41	NÃO
T01	Média	42.754	21.609	41.569	2,77	NÃO
T01	Baixa	14.608	7.071	16.678	-14,17	NÃO
T02	Grave	363	196	157	56,75	SIM
T02	Alta	2.434	1.703	2.090	14,13	NÃO
T02	Média	220.373	139.994	73.074	66,84	SIM
T02	Baixa	13.323	10.474	16.786	-25,99	NÃO
T03	Grave	336	260	194	42,26	SIM
T03	Alta	1.524	1.124	1.275	16,34	NÃO
T03	Média	65.851	29.184	71.001	-7,82	NÃO
T03	Baixa	16.339	7.456	12.110	25,88	NÃO
T04	Grave	239	180	14	94,14	SIM
T04	Alta	1.063	850	690	35,09	SIM
T04	Média	72.270	60.000	17.840	75,31	SIM
T04	Baixa	14.638	8.400	8.406	42,57	NÃO
T05	Grave	29	12	3	89,66	SIM
T05	Alta	513	385	418	18,52	NÃO
T05	Média	112.723	17.000	17.060	84,87	NÃO
T05	Baixa	6.931	3.600	7.508	-8,32	NÃO
T06	Grave	52	20	4	92,31	SIM
T06	Alta	651	325	324	50,23	SIM
T06	Média	54.467	28.842	30.306	44,36	NÃO
T06	Baixa	10.577	5.887	6.815	35,57	NÃO
T07	Grave	136	120	0	100,00	SIM
T07	Alta	305	275	279	8,52	NÃO
T07	Média	10.890	7.000	6.506	40,26	SIM
T07	Baixa	3.366	2.000	3.008	10,64	NÃO
T08	Grave	4	2	1	75,00	SIM
T08	Alta	340	170	306	10,00	NÃO
T08	Média	21.748	11.430	7.548	65,29	SIM
T08	Baixa	6.173	4.048	3.963	35,80	SIM
T09	Grave	12	0	0	100,00	SIM
T09	Alta	141	91	90	36,17	SIM
T09	Média	8.529	2.985	4.691	45,00	NÃO
T09	Baixa	2.551	1.275	3.136	-22,93	NÃO
T10	Grave	92	9	10	89,13	NÃO
T10	Alta	314	125	547	-74,20	NÃO
T10	Média	18.350	13.500	15.415	15,99	NÃO
T10	Baixa	4.465	2.700	3.413	23,56	NÃO
T11	Grave	80	4	63	21,25	NÃO
T11	Alta	297	148	238	19,87	NÃO
T11	Média	15.264	12.258	10.908	28,54	SIM
T11	Baixa	4.041	2.516	2.697	33,26	NÃO
T12	Grave	9	4	1	88,89	SIM
T12	Alta	96	67	54	43,75	SIM
T12	Média	7.314	5.200	3.724	49,08	SIM
T12	Baixa	1.652	1.100	1.192	27,85	NÃO

* O valor de Referência é o total de itens de Dívida Técnica no início do período

APÊNDICE H. RESULTADOS POR EQUIPE E PRIORIDADE DO 3º CICLO

Equipe	Prioridade	Referência	Meta	Realizado	% Diminuição	Atingiu a Meta?
T01	Blocker	1.189	1.000	1.019	14,30	NÃO
T01	Critical	2.597	1.000	934	64,04	SIM
T01	Major	29.515	25.000	23.981	18,75	SIM
T01	Minor	5.969	2.000	5.417	9,25	NÃO
T02	Blocker	3.516	2.692	2.579	26,65	SIM
T02	Critical	5.976	3.435	3.738	37,45	NÃO
T02	Major	65.815	51.375	40.359	38,68	SIM
T02	Minor	10.126	9.261	9.576	5,43	NÃO
T03	Blocker	2.247	1.800	1.718	23,54	SIM
T03	Critical	3.123	2.800	1.193	61,80	SIM
T03	Major	29.735	26.700	19.792	33,44	SIM
T03	Minor	6.048	4.850	5.174	14,45	NÃO
T04	Blocker	1.358	1.000	1.211	10,82	NÃO
T04	Critical	1.966	1.500	1.587	19,28	NÃO
T04	Major	21.462	21.000	19.144	10,80	SIM
T04	Minor	5.464	5.000	5.177	5,25	NÃO
T05	Blocker	924	650	630	31,82	SIM
T05	Critical	1.611	1.125	289	82,06	SIM
T05	Major	17.053	14.500	11.450	32,86	SIM
T05	Minor	4.852	3.400	3.555	26,73	NÃO
T06	Blocker	888	600	637	28,27	NÃO
T06	Critical	2.690	2.000	1.916	28,77	SIM
T06	Major	15.235	13.000	11.132	26,93	SIM
T06	Minor	5.278	5.000	5.026	4,77	NÃO
T07	Blocker	423	380	504	-19,15	NÃO
T07	Critical	1.045	940	884	15,41	SIM
T07	Major	10.903	9.812	9.590	12,04	SIM
T07	Minor	2.055	2.055	1.913	6,91	SIM
T08	Blocker	359	300	291	18,94	SIM
T08	Critical	557	100	84	84,92	SIM
T08	Major	5.240	4.000	3.379	35,52	SIM
T08	Minor	2.632	1.500	1.426	45,82	SIM
T09	Blocker	216	172	70	67,59	SIM
T09	Critical	22	0	18	18,18	NÃO
T09	Major	1.291	904	767	40,59	SIM
T09	Minor	2.092	2.092	1.820	13,00	SIM
T10	Blocker	622	550	613	1,45	NÃO

T10	Critical	1.939	1.600	765	60,55	SIM
T10	Major	7.565	6.500	4.532	40,09	SIM
T10	Minor	1.577	1.200	1.572	0,32	NÃO
T11	Blocker	211	145	194	8,06	NÃO
T11	Critical	662	100	73	88,97	SIM
T11	Major	5.922	5.000	4.501	24,00	SIM
T11	Minor	1.078	1.078	1.166	-8,16	NÃO
T12	Blocker	136	81	37	72,79	SIM
T12	Critical	329	165	49	85,11	SIM
T12	Major	1.704	1.500	579	66,02	SIM
T12	Minor	866	690	527	39,15	SIM