UNIVERSIDADE FEDERAL DO CEARÁ

CENTRO DE CIÊNCIAS

DEPARTAMENTO DE COMPUTAÇÃO

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

MESTRADO ACADÊMICO EM CIÊNCIA DA COMPUTAÇÃO

DÊMORA BRUNA CUNHA DE SOUSA

UM ESTUDO DE CASO DO TRATAMENTO DE EXCEÇÃO NO DOMÍNIO DE SISTEMAS WEB CORPORATIVOS

FORTALEZA

2018

DÊMORA BRUNA CUNHA DE SOUSA

UM ESTUDO DE CASO DO TRATAMENTO DE EXCEÇÃO NO DOMÍNIO DE SISTEMAS WEB CORPORATIVOS

Dissertação apresentada ao Curso de Mestrado Acadêmico em Ciência da Computação do Programa de Pós-Graduação em Ciência da Computação do Centro de Ciências da Universidade Federal do Ceará, como requisito parcial à obtenção do título de mestre em Ciência da Computação. Área de Concentração: Engenharia de Software

Orientador: Prof. Dr. Windson Viana de Carvalho

Coorientador: Prof. Dr. Lincoln Souza Rocha

FORTALEZA

2018

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca Universitária
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

S696e Sousa, Dêmora Bruna Cunha de.
Um Estudo de Caso do Tratamento de Exceção no Domínio de Sistemas Web Corporativos / Dêmora Bruna Cunha de Sousa. – 2018.
129 f. : il. color.

Dissertação (mestrado) – Universidade Federal do Ceará, Centro de Ciências, Programa de Pós-Graduação em Ciência da Computação, Fortaleza, 2018.
Orientação: Prof. Dr. Windson Viana de Carvalho.
Coorientação: Prof. Dr. Lincoln Souza Rocha.

1. Tratamento de Exceção. 2. Anti-Padrões do Tratamento de Exceção. 3. Estudo de Caso. I. Título.
CDD 005

DÊMORA BRUNA CUNHA DE SOUSA

UM ESTUDO DE CASO DO TRATAMENTO DE EXCEÇÃO NO DOMÍNIO DE SISTEMAS WEB CORPORATIVOS

Dissertação apresentada ao Curso de Mestrado Acadêmico em Ciência da Computação do Programa de Pós-Graduação em Ciência da Computação do Centro de Ciências da Universidade Federal do Ceará, como requisito parcial à obtenção do título de mestre em Ciência da Computação. Área de Concentração: Engenharia de Software

Aprovada em:

BANCA EXAMINADORA

Prof. Dr. Windson Viana de Carvalho (Orientador)
Universidade Federal do Ceará (UFC)

Prof. Dr. Lincoln Souza Rocha (Coorientador)
Universidade Coorientador (UFC)

Prof. Dr. Joaquim Bento Cavalcante Neto
Universidade Federal do Ceará (UFC)

Prof. Dr. Paulo Henrique Mendes Maia
Universidade Estadual do Ceará (UECE)

Prof. Dr. Marco Túlio de Oliveira Valente
Universidade Federal de Minas Gerais (UFMG)

A todos que me ajudaram nessa caminhada, à minha família, amigos e professores. Obrigada por acreditaram em mim. Sou imensamente grata pela vida de vocês.

AGRADECIMENTOS

A Deus, pela oportunidade que me foi dada. Por me ajudar e me ouvir ao longo desses meses. Sou muito grata pela esperança, amor e força.

À minha família, que sempre me encorajou a progredir, oferecendo suporte, compreendendo minha ausência em vários momentos por causa das ocupações do mestrado, e pelas orações. Especialmente aos meus pais, Maria das Graças e Flávio, aos meus irmãos, Débora e Brendo, e ao meu cunhado, Kássio.

Aos amigos e colegas de mestrado, especialmente aos que caminharam ao meu lado desde o início do mestrado e partilharam a vivência acadêmica comigo, Rodrigo Lucas e Priscylla Tavares; e ao Paulo Artur por estar sempre disposto a ajudar, a sanar dúvidas, e pela amizade.

Ao meu orientador, Windson Viana, por ter me acolhido no seu grupo de orientandos, pelas várias ideias que deu e que foram essenciais para desenvolver este trabalho, pelo tempo cedido para reuniões e revisão de artigos, pela paciência que sempre demonstrou, pela dedicação e apoio para concluir esta pesquisa.

Ao meu coorientador, Lincoln Rocha, por ter me aceitado como orientanda, por fornecer sua expertise na temática de tratamento de exceções para desenvolver este trabalho, pelos direcionamentos, e todo o tempo dedicado.

Ao professor Paulo Henrique, que ajudou na divulgação e melhoria deste trabalho, contribuindo com a escrita dos artigos acadêmicos produzidos.

Aos colegas e chefes de trabalho, pelo apoio e incentivo à progressão acadêmica. Especialmente, agradeço ao Luís Vasconcelos, ao Jander Soares, ao Joaquim Bento, ao Fabiano Gadelha, e ao Sílvio Martins.

Agradeço à Universidade Federal do Ceará e ao Programa de Pós-Graduação em Ciência da Computação pelo curso de mestrado, pela estrutura que usufrui e pelos docentes que participam do programa, pois contribuíram imensamente para o meu aprendizado.

Muito obrigada!

"A luz resplandece nas trevas, e as trevas não prevaleceram contra ela."

(João 1:5)

RESUMO

Estudos da literatura mostraram que más práticas de design e codificação do tratamento de exceção podem afetar a qualidade geral de um software. A qualidade do código do tratamento de exceção é diretamente afetada por (i) uma ausência ou falta de conhecimento de uma política explícita de tratamento de exceção; e (ii) um aumento silencioso da disseminação de anti-padrões do tratamento de exceção. Para investigar tal fenômeno, foi conduzido um estudo de caso em um sistema Java Web em larga escala, tentando entender melhor a relação entre (i) e (ii). O estudo leva em consideração aspectos técnicos e humanos. Um survey foi conduzido com 21 desenvolvedores sobre suas percepção do tratamento de exceção na instituição mantenedora do sistema. Em seguida, foi analisada a evolução dos anti-padrões do tratamento de exceção em 15 versões do sistema alvo da pesquisa. A pesquisa também envolveu uma entrevista semiestruturada com três profissionais experientes, representantes da equipe de desenvolvimento, para apresentar resultados parciais do caso e levantar possíveis causas dos problemas encontrados. Após uma segunda análise do código em busca de comprovações sobre a rotatividade, o caso foi apresentado em um relato final para os responsáveis pelo sistema. As descobertas da pesquisa sugerem que a ausência de uma política explícita de tratamento de exceção tem um impacto negativo na percepção dos desenvolvedores e na sua implementação. Além disso, a ausência de tal política tem levado os desenvolvedores a replicar os anti-padrões já existentes no sistema e difundi-los através de novos recursos adicionados durante a evolução do sistema. Os entrevistados apontaram a alta rotatividade da equipe como a fonte desse fenômeno, uma vez que o processo de licitação pública para a contratação de novos desenvolvedores atraiu principalmente profissionais menos experientes em desenvolvimento Web. Esse fenômeno foi investigado e se mostrou uma das possíveis causas do aumento da presença de anti-padrões. Os achados da pesquisa beneficiaram a instituição mantenedora do sistema, conscientizando-os sobre os problemas encontrados e permitindo-lhes melhor projetar as ações para combatê-los.

Palavras-chave: Estudo de Caso. Tratamento de Exceção. Anti-Padrões do Tratamento de Exceção

ABSTRACT

Previous studies have shown that exception handling bad practices may impact the overall software quality. We believe that quality of exception handling code is directly affected by (i) an absence, or lack of awareness, of an explicit exception handling policy; and (ii) a silent rising and spreading of exception handling anti-patterns. To investigate such this phenomenon, we conducted a case study in a large-scale Java Web system, trying to better understand the relationship between (i) and (ii). The study takes into account technical and human aspects. We surveyed 21 developers regarding their perception of exception handling in the system's institution. Next, we analyse the evolution of exception handling anti-patterns across 15 releases of the target system. The research also involved a semi-structured interview with with three experienced professionals, representatives of the development team, to present partial results of the case and raise possible causes for the problems found. After a second analysis of the code for searching pieces of evidence concerning the turnover, the case was presented in a final report to those responsible for the system. Our findings suggest that the absence of an explicit exception handling policy impacts negatively in the developers' perception and implementation of exception handling. Furthermore, the absence of such policy has been leading developers to replicate anti-patterns already existing in the system and spread them through new features added during system evolution. The interviewed professionals point the high team turnover as the source of this phenomenon, once the public tender process for hiring new developers has mostly attracted novices ones. A second code analysis shows some pieces of evidence of this phenomenon in the system code. Our finds beneficiated the system's institution by making it aware of these problems and enabling it to take actions towards to combat them.

Keywords: Case Study. Exception Handling. Exception Handling Anti-Patterns

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Esta dissertação apresenta a análise do tratamento de exceção de um sistema Web de larga escala. A Seção 1.1 contextualiza a problemática sobre o desenvolvimento do tratamento de exceção. A Seção 1.2 detalha a motivação e as questões de pesquisa que nortearam a condução deste trabalho de mestrado. Os objetivos gerais e específicos são listados na Seção 1.3. Na Seção 1.4, são expostas as contribuições desta pesquisa. Por fim, a organização deste documento é abordada na Seção 1.5.

1.1 Contextualização

O tratamento de exceção (*Exception Handling*) é uma técnica de recuperação de erros usada para melhorar a robustez de um software, fornecendo meios para estruturar atividades de tolerância a falhas no código-fonte (GARCIA *et al.*, 2001; SHAHROKNI; FELDT, 2013). As exceções modelam situações anormais, detectadas no tempo de execução, que interrompem o fluxo de controle normal de um programa. Quando uma exceção é detectada, em tempo de execução, o sistema desvia o fluxo de controle normal do programa para o fluxo de controle excepcional. Neste ponto, o mecanismo de tratamento de exceção assume o controle da execução do programa e inicia uma busca por um tratamento apropriado para lidar com tal situação anormal (BUHR; MOK, 2000). Linguagens de programação amplamente adotadas, como Java, C# e Python, fornecem construções dedicadas para estruturar o fluxo de controle excepcional, especificando no código-fonte onde e como exceções podem ser levantadas, propagadas e tratadas (CACHO *et al.*, 2014).

No entanto, apesar de sua importância, o tratamento de exceção é comumente negligenciado pelos desenvolvedores. É reivindicado como a parte menos compreendida, documentada e testada de um sistema de software (SHAH *et al.*, 2010; KECHAGIA; SPINELLIS, 2014; CHANG; CHOI, 2016; OLIVEIRA *et al.*, 2018a). Estudos recentes foram realizados buscando entender as perspectivas dos desenvolvedores e das organizações com relação à temática. Shah *et al.* (2010) abordam os divergentes pontos de vista relatados por novos desenvolvedores e por desenvolvedores experientes. Ebert e Castor (2013) conduzem um estudo acerca das percepções de desenvolvedores sobre *bugs* do tratamento de exceção. Outros estudos sugerem novos métodos e técnicas para facilitar o desenvolvimento das atividades de tolerância a falhas. Huang *et al.* (2012) apresentam um *framework* do tratamento de exceção para a composição de

serviços Web. Filho *et al.* (2017) desenvolvem uma solução para verificação da conformidade arquitetural do tratamento de exceção. Sawadpong e Allen (2016) propõem um conjunto de métricas, obtidas com a análise estática de um grafo de chamadas do tratamento de exceção, para auxiliar na predição de defeitos de exceção.

Esses estudos fornecem evidências de que a qualidade do código do tratamento de exceção pode afetar a qualidade geral do software. Além disso, os pesquisadores apontaram os *anti-patterns* (por exemplo, Catch Generic, Catch e Do Nothing e Destructive Wrapping) como fonte de falhas de software, denominados *bugs* de tratamento de exceção (EBERT *et al.*, 2015; SENA *et al.*, 2016; PÁDUA; SHANG, 2017).

1.2 Motivação e Questões de Pesquisa

Com base em estudos anteriores (SHAH *et al.*, 2010; BARBOSA *et al.*, 2016; FILHO *et al.*, 2017; BARBOSA; GARCIA, 2017; OSMAN *et al.*, 2017a; PÁDUA; SHANG, 2017), entendeu-se que a qualidade do código do tratamento de exceção é diretamente afetada por dois aspectos: (i) A ausência ou a falta de conhecimento de uma política explícita de tratamento de exceção; e (ii) O aumento silencioso da disseminação de *anti-patterns* do tratamento de exceção. O primeiro aspecto pode levar os desenvolvedores a decidirem por conta própria como projetar e implementar o código para o comportamento excepcional. Portanto, isso pode causar, por exemplo, um problema de erosão arquitetural, em relação ao tratamento de exceção (BARBOSA *et al.*, 2016; FILHO *et al.*, 2017). Sem um processo sistemático de revisão de código, a aparência dos *anti-patterns* somente será contida pelas habilidades, conhecimento e consciência dos desenvolvedores sobre a importância do código do tratamento de exceção para a qualidade geral do software (SHAH *et al.*, 2010).

Para investigar tal fenômeno, foi conduzido um estudo de caso em um sistema Java Web de larga escala. Por meio desse estudo, buscou-se melhor entender a relação entre os dois aspectos apresentados. Um estudo de caso é um método científico destinado a investigar fenômenos no seu ambiente de origem, sendo caracterizado por sua flexibilidade para lidar com a dinamicidade existente em ambientes reais, além de se basear em uma clara cadeia de evidências e proporcionar fortes e relevantes conclusões (WOHLIN *et al.*, 2012). Tais características foram necessárias para alcançar os objetivos desta pesquisa, além de contribuírem para a condução de uma investigação qualitativa e quantitativa do tratamento de exceção de um sistema.

O estudo considera vertentes técnicas e humanas, buscando responder as seguintes

questões de pesquisa:

- **QP1**: Quais percepções o time de desenvolvimento de um sistema de larga escala tem sobre o tratamento de exceção?
- **QP2**: Quais *anti-patterns* do tratamento de exceção podem ser encontrados em um sistema de larga escala e como eles evoluem ao longo do tempo?
- **QP3**: Quais fatores motivaram a inserção de *anti-patterns* no sistema?

Um sistema de larga escala, definido por Dikert *et al.* (2016), é produzido em uma organização com 50 ou mais pessoas, ou pelo menos 6 times. Os profissionais envolvidos não precisam atuar apenas na codificação de funcionalidades, mas cooperar com o desenvolvimento do produto. O sistema selecionado para este estudo é mantido diretamente por uma equipe de 46 profissionais e diversos colaboradores, que fornecem sua perícia da área negocial apoiada pelo sistema e contribuem com diretrizes para o desenvolvimento dele. Esse sistema não possui uma documentação extensa do tratamento de exceção, o que o tornou um bom caso a ser investigado, já que essa ausência de diretrizes podem contribuir para o surgimento de problemas no funcionamento do tratamento de exceção.

Este estudo foi projetado para analisar exclusivamente aspectos diretamente relacionados ao tratamento de exceção. Portanto, não são incluídas verificações de outros elementos que compõem um software ou processos de desenvolvimento. Consequentemente, a qualidade integral do software analisado não pode ser inferida somente pelos resultados apresentados ao longo deste documento.

1.3 Objetivos

1.3.1 Objetivo Geral

Esta pesquisa de mestrado tem como principal objetivo analisar o tratamento de exceção em um sistema Web real, por meio de aspectos humanos e técnicos, buscando entender o estado atual do tratamento implementado e como ele evoluiu.

1.3.2 Objetivos Específicos

- Entender as percepções dos desenvolvedores responsáveis pelo sistema alvo sobre o tratamento de exceção. Principalmente sobre a opinião deles sobre a existência de políticas, a importância e a utilização de ferramentas para auxiliar na manutenção da qualidade do

tratamento de exceção;

- Analisar o código-fonte do sistema alvo a fim de verificar a existência de estruturas prejudiciais ao tratamento de exceção (*anti-patterns*); e
- Verificar se a falta de políticas, a rotatividade da equipe de desenvolvimento, a inexperiência de desenvolvedores contribuíram para inserção de *anti-patterns* e para a visão que os desenvolvedores têm sobre o tratamento implementado.

1.4 Contribuições

Nesta pesquisa, diversos métodos de coleta foram utilizados com o intuito de obter informações diversificadas e responder às questões de pesquisas elencadas, tais como formulários *online*, ferramentas de análise de código e entrevistas. As principais descobertas alcançadas durante o processo de análise são destacadas a seguir:

- **QP1** - A maioria dos desenvolvedores da instituição alvo acredita na importância do tratamento de exceção. Mas deparam-se com obstáculos durante as atividades de desenvolvimento, como a falta de documentações, a inexistência de políticas que definam como o tratamento de exceção deve ser implementado ao longo do sistema, e a ausência de uma cultura de incentivo ao uso ferramentas que detectem más práticas do tratamento de exceção;
- **QP2** - Diversos problemas foram detectados durante a análise do código do sistema alvo. Os *anti-patterns* Catch Generic, Throws Generic, Destructive Wrapping e Catch and Do Nothing são responsáveis pelo maior número de violações. Os tratamentos genéricos correspondem a mais de 70% de todas as violações nas versões consideradas do sistema. A análise das *releases* do sistema alvo comprovam um crescimento na quantidade de *anti-patterns*; e
- **QP3** - As descobertas da pesquisa sugerem que a ausência de uma política explícita de tratamento de exceção tem um impacto negativo na percepção dos desenvolvedores e na sua implementação. Além disso, a ausência de tal política tem levado os desenvolvedores a replicar os anti-padrões já existentes no sistema e difundi-los através de novos recursos adicionados durante a evolução do sistema. Por meio de entrevista, participantes da equipe de desenvolvimento ressaltaram a alta rotatividade da equipe como a fonte desse fenômeno, uma vez que o processo de licitação pública para a contratação de novos desenvolvedores atraiu principalmente profissionais menos experientes em desenvolvimento Web. Esse

fenômeno foi investigado e se mostrou uma das causas do aumento da presença de anti-padrões.

1.5 Organização da Dissertação

Este capítulo apresentou o contexto que motivou a condução desta pesquisa de mestrado, além das questões de pesquisa, objetivos e contribuições alcançadas. O restante deste documento possui a seguinte organização:

- **Capítulo 2 - Fundamentação Teórica:** Aborda os conceitos fundamentais utilizados neste trabalho, como exceções, políticas do tratamento de exceção e *anti-patterns*;
- **Capítulo 3 - Trabalhos Relacionados:** Apresenta trabalhos que tratam da temática do tratamento de exceção em sistemas em meio a aspectos reais de ambientes de desenvolvimento;
- **Capítulo 4 - Planejamento do Estudo de Caso:** Mostra como o estudo de caso foi estruturado, as unidades de análise que ele contém, os métodos de análise, os métodos de coleta e a estratégia de seleção do caso;
- **Capítulo 5 - Resultados:** Este capítulo descreve a condução do estudo de caso e os resultados encontrados. Por isso, destaca os procedimentos adotados ao longo do estudo e os materiais utilizados na coleta e análise de dados;
- **Capítulo 6 - Discussão:** Discute os principais achados da pesquisa, relacionando-os entre si e com a literatura;
- **Capítulo 7 - Conclusão:** Apresenta as considerações finais deste trabalho de mestrado, apontando as limitações, as ameaças à validade e as sugestões para trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo, são apresentados os conceitos que serviram como fundamento para o desenvolvimento deste trabalho. A definição de exceção é o primeiro tema abordado. Ele é visto praticamente por meio da implementação de exceções na linguagem Java na Seção 2.1. A Seção 2.2 mostra elementos prejudiciais ao bom funcionamento do mecanismo, como a erosão de software e os *anti-patterns* do tratamento de exceção. A Seção 2.3 expõe ferramentas que auxiliam na definição e na verificação da conformidade do tratamento de exceção durante a evolução de um sistema. As considerações finais deste capítulo podem ser vistas na Seção 2.4.

2.1 Tratamento de Exceção

Durante a execução de um sistema, nem sempre é possível alcançar um comportamento satisfatório. Às vezes, a chamada de uma rotina deixa a aplicação em um estado anormal ou ela mesma viola o procedimento determinado na especificação. Esse fenômeno é denominado falha. As falhas têm sua origem na manifestação de erros, que são o desvio do estado correto do serviço requisitado para o estado incorreto (AVIZIENIS *et al.*, 2004). Uma exceção é um resultado anormal que interrompe o fluxo normal de execução de um sistema durante a chamada de um procedimento e demanda um tratamento especial (SIEDERSLEBEN, 2006). Por meio desse tratamento, é possível que o serviço requisitado transite do estado incorreto para o correto novamente (*service restoration*) (AVIZIENIS *et al.*, 2004).

Mesmo que não seja de maneira padrão, as linguagens de programação de alto nível apresentam um sistema de tratamento de exceção. Por meio desse sistema, ocorre a comunicação entre o procedimento que detecta a exceção (i.e., o sinalizador da exceção) e a entidade que requisitou a operação. Além disso, um sistema de tratamento de exceção permite a definição de tratadores para exceções específicas. Segundo Dony (1990), para sinalizar uma exceção são necessários quatro passos:

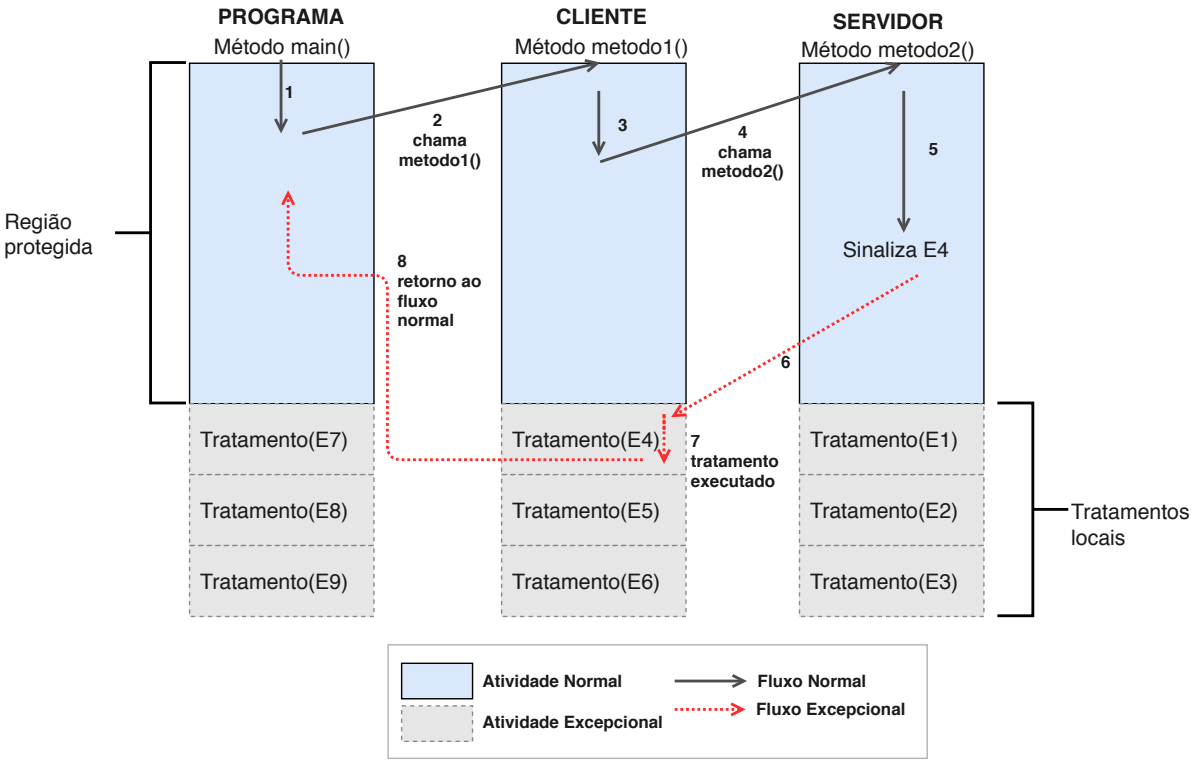
1. Identificar a situação causadora da exceção;
2. Interromper a sequência de execução;
3. Procurar um tratamento para exceção; e
4. Passar as informações necessárias para que o tratamento ocorra.

Os tratadores de exceção (*handlers*) são parte do comportamento excepcional desse sistema, sendo responsáveis por alternar entre o fluxo de controle normal e o fluxo de controle

excepcional (GARCIA *et al.*, 2001). Normalmente, componentes tolerantes a falha apresentam essa divisão bem definida.

Garcia *et al.* (2001) também ilustram o funcionamento das chamadas regiões protegidas ou contexto de tratamento. Elas constituem a parte do código cujos tratadores de exceção estão anexados. Quando uma exceção é lançada em uma região protegida, o fluxo normal é alternado para o fluxo excepcional. Uma região protegida pode ter mais de um tratador, sendo ele escolhido pelo sistema de tratamento de exceção de acordo com a exceção lançada. A Figura 1 apresenta três regiões protegidas com os seus respectivos tratamentos. Durante a sequência de chamadas entre os componentes, uma exceção é sinalizada durante o passo cinco do fluxo normal, sendo sinalizada pelo Servidor, que não possui um tratamento para ela. A exceção é então propagada para o componente Cliente, no qual um tratamento é encontrado e executado, permitindo que fluxo excepcional dê lugar ao fluxo normal de atividades do sistema.

Figura 1 – Cenário de uma possível operação do mecanismo de tratamento de exceção



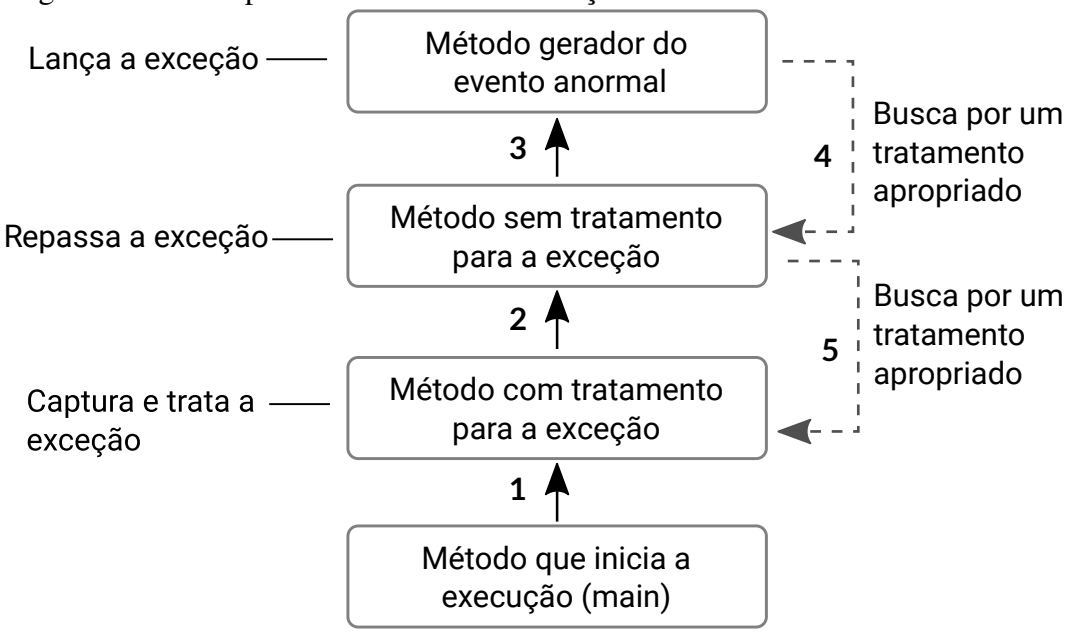
Fonte: Adaptado de (GARCIA *et al.*, 2001).

2.1.1 Linguagem Java

Na linguagem Java, quando exceções ocorrem durante a execução de um método, este cria uma entidade chamada objeto de exceção. Esse objeto tem informações referentes ao erro ocorrido e ao estado do programa no momento em que foi acometido pelo erro. O processo de criação do objeto e sua entrega ao sistema de execução é chamado de lançamento de exceção (CAMPIONE *et al.*, 2001).

O sistema de execução procura uma forma de tratar a exceção, verificando na pilha de execução se existe um método que contenha o trecho de código necessário ao tratamento dela. A busca é iniciada no método que lançou a exceção, prosseguindo na ordem inversa de métodos chamados, como apresentado na Figura 2. Caso nenhum tratamento seja encontrado para a exceção, o programa termina sua execução, sendo impossibilitado de retornar ao fluxo normal de execução.

Figura 2 – Busca por um tratamento de exceção



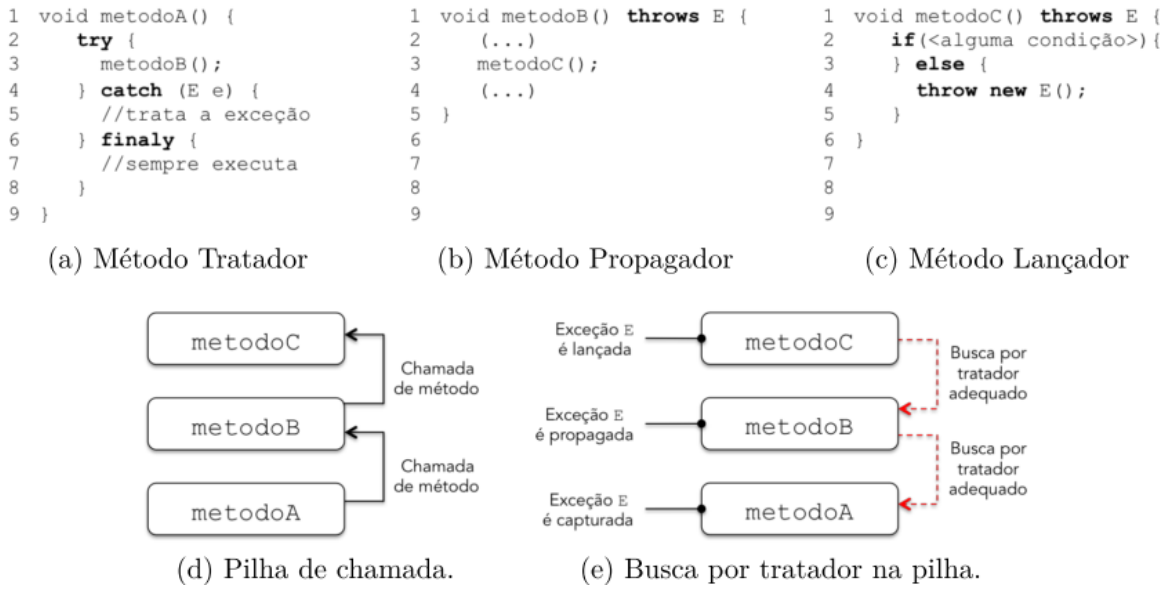
Fonte: Adaptado de Campione *et al.* (2001).

A linguagem Java fornece três componentes para escrever um tratamento de exceção, os blocos *try*, *catch* e *finally*. O bloco *try* envolve o trecho de código capaz de lançar uma exceção. Tratadores de exceção são adicionados com a inclusão de blocos *catch*, localizados ao final do bloco *try*. O bloco *finally* sempre é executado, mesmo durante a ocorrência de exceções não esperadas. Pode ser empregado na liberação de variáveis da memória ou para o fechamento

de conexões.

Filho *et al.* (2017) demonstra, por meio de exemplos, o tratamento, a propagação e o lançamento de exceções em Java (Figura 3). No item *a* da Figura 3, metodoA() realiza o tratamento da exceção do tipo *E*, sinalizada pelo método metodoB() por meio do construto *throws*. No item *c*, a exceção *E* é lançada com a utilização da construção *throw new*. O metodoB() não apresenta tratamentos ou lançamento de exceções, apenas propaga as vindas de métodos anteriores na pilha de execução (item *d* da Figura 3), especificamente, exceções do tipo *E*, como está sinalizado no método (*metodoB()* ***throws*** *E*).

Figura 3 – Tratamento de exceção em Java



Fonte: Filho *et al.* (2017).

Na linguagem Java, dois tipos de exceções estão disponíveis. Um deles são as exceções verificadas, aquelas que aplicação é capaz de antecipar e tratar apropriadamente. A elas são destinadas a construção dos blocos *try-catch-finally*. O outro tipo, as exceções não verificadas, são compostas pelas classes *Erro*, *RuntimeException* e suas subclasses, são reservadas para as situações excepcionais relacionadas a eventos externos e internos da aplicação, como problemas de hardware ou *bugs* de programação (CAMPIONE *et al.*, 2001).

2.2 Problemas do Tratamento de Exceção

Esta seção descreve alguns problemas que afetam o tratamento de exceção, e que estão relacionados às análises realizadas nesta pesquisa, sendo eles a erosão de software (2.2.1) e a ocorrência de *anti-patterns* (2.2.2).

2.2.1 Erosão

Perry e Wolf (1992) caracterizam, de forma geral, partes de um produto de software: requisitos, arquitetura, projeto e implementação. Dentre elas, a arquitetura se preocupa com o conjunto de elementos arquiteturais, as interações existentes entre eles e as regras para esses elementos e interações. As entidades que constituem a arquitetura são definidas para satisfazer os requisitos e utilizadas como base para as próximas etapas inerentes a construção de um software. Os autores também apresentam um modelo de arquitetura que é composto por três elementos arquiteturais, sendo eles: elementos, organização e decisões. A arquitetura é de suma importância para reger o bom funcionamento de um software, além de servir como guia para a adição de novas funcionalidades ou durante a alteração das existentes. No entanto, nem sempre os conceitos definidos pela arquitetura são obedecidos. Van Gurp e Bosch (GURP; BOSCH, 2002) destacam alguns fatores que facilitam a ocorrência dessas violações:

- **Rastreabilidade das decisões de projeto:** As notações utilizadas para descrever as decisões arquiteturais não são expressivas o suficiente para um entendimento posterior, além disso, o rastreamento dos conceitos a partir do sistema também é uma tarefa difícil;
- **Aumento do custo de produção:** Devido à evolução do sistema, que produz um aumento de sua complexidade, a realização de tarefas torna-se custosa. Isto facilita a tomada de decisões que ferem a integridade arquitetural motivadas por um não entendimento da arquitetura ou do grande esforço necessário para uma abordagem correta;
- **Acúmulo de decisões de projeto:** O acúmulo de decisões de *design* implica na não correspondência do sistema aos requisitos;
- **Metodologias iterativas:** Metodologias iterativas incorporam modificações que impactam na arquitetura durante o desenvolvimento, no entanto, tais decisões deveriam ser integradas durante a fase de projeto.

O acúmulo de violações causa um fenômeno conhecido como erosão arquitetural. Esse fenômeno ocorre quando a arquitetura implementada de um software, definida como o

modelo da arquitetura construída a partir de elementos de baixo nível ou código-fonte, diverge da arquitetura planejada (SILVA; BALASUBRAMANIAM, 2012).

O tratamento de exceção faz parte do *design* arquitetural do sistema. Por isso, também está sujeito a erosão de software caso o acúmulo de violações referentes a sua especificação aconteça. A verificação de conformidade é uma solução para evitar esse problema, pois permite a identificação do código violador, facilitando o providenciamento de uma correção. Dentre as técnicas utilizadas para o tratamento de erosão, a checagem de relações de dependência torna possível identificar relações que deveriam, ou não, existir entre exceções e os módulos de um sistema. Sendo possível derivar regras para definir essas relações.

2.2.2 Anti-Patterns

Diversos autores têm contribuído com a identificação e a caracterização de anomalias de código do tratamento de exceção. Durante a condução desta pesquisa, a utilização de nomenclaturas distintas foram observadas, tais como *bug patterns* (YUAN *et al.*, 2014), *inappropriate coding patterns* (SINHA *et al.*, 2004), *bad smells* (CHEN *et al.*, 2009) e *anti-patterns* (SENA *et al.*, 2016). Barbosa *et al.* (2014) identificam *bugs* causados por essas anomalias, tratando-os como falhas excepcionais ou falhas do tratamento de exceção, já que estão relacionadas à definição, ao lançamento, à propagação e à documentação de exceções.

Ebert *et al.* (2015) contribuem com a temática empregando o termo *bugs* do tratamento de exceção e o definem como *bugs* que ocorrem quando uma exceção é definida, lançada, propagada, tratada ou documentada; na ação de limpeza de uma região protegida onde é lançada; quando ela deveria ter sido lançada ou tratada, mas não foi. Além disso, categorizam *bugs* para o tratamento de exceção e suas causas baseando-se em informações extraídas de repositórios de *bug* e na opinião de desenvolvedores. Segundo os dados apresentados no estudo, dentre as categorias criadas, as exceções não tratadas e as não levantadas são as maiores geradoras de falha no contexto dos projetos analisados.

Yuan *et al.* (2014) informam que quase 92% das falhas catastróficas, obtidas dos projetos analisados em seu estudo, resultam de erros não fatais sinalizados pelo próprio software. Além disso, 35% dessas falhas eram decorrentes de tratamentos vazios, ações erradas em tratamentos de erros genéricos e tratamentos com comentários que informavam a necessidade de uma implementação ou correção posterior, como "*FIXME*" e "*TODO*".

No entanto, o uso da nomenclatura *anti-patterns* é observado quando essas estruturas

maliciosas não se manifestam necessariamente em falhas, mas podem resultar em consequências negativas para o projeto. Correa *et al.* (2000) descrevem como uma solução para um problema recorrente, mas que trazem mais malefícios do que benefícios. Sendo esses malefícios conhecidos, assim como os sintomas da sua presença no projeto, quando são documentados devidamente. Pádua e Shang (2017) fazem um estudo para identificar a prevalência de *anti-patterns* do tratamento de exceção em projetos *open source*. Segundo os autores, alguns desses problemas possuíram maior ocorrência, no entanto, todos foram encontrados em pelo menos 3 dos 9 projetos Java escolhidos por eles. Esses problemas foram considerados pontos de partida para a inspeção a ser realizada neste trabalho, contudo, apenas aqueles com ocorrência mais expressiva. Uma breve descrição desses *anti-patterns* é apresentada na Tabela 1. Exemplos práticos da aplicação dos *anti-patterns* em códigos Java podem ser vistos no Apêndice D.

2.3 Melhoria e Manutenção da Qualidade do Tratamento de Exceção

O tratamento de exceção é uma atividade diretamente relacionada à capacidade de um sistema de software se recuperar de situações anormais. Tal característica impacta em atributos gerais da qualidade de um software, como confiabilidade e segurança (CORPORATION, 2009), além da sua robustez (CHEN *et al.*, 2009). Políticas do tratamento de exceção e ferramentas de análise de código podem ajudar a manter um bom funcionamento do mecanismo ao longo da evolução de um software. Consequentemente, impactando na qualidade do sistema desenvolvido. As próximas subseções abordarão políticas e ferramentas do tratamento de exceção.

2.3.1 Políticas e Guidelines

A arquitetura de um sistema, como já mencionado neste documento, é constituída por diversos elementos arquiteturais e pelas conexões existentes entre eles. Esses elementos, por sua vez, são projetados para conter diferentes responsabilidade e níveis de abstração, assim como é definido por modelos arquiteturais amplamente conhecidos, como o modelo MVC (*Model-View-Controller*). O tratamento de exceção, pertencente ao *design* arquitetural do software, deve colaborar para que o baixo acoplamento entre os elementos arquiteturais seja mantido.

Uma ameaça a essa premissa ocorre quando exceções de camadas de baixo nível chegam até camadas de alto nível. Detalhes desnecessários da implementação são transportados entre as camadas e podem não colaborar com a atividade de recuperação de erros. Além de tornar

Tabela 1 – *Anti-pattern* do tratamento de exceção

Anti-patterns	Descrição	Consequência	Trabalho de Referência
Unhandled Exceptions	O tratamento não captura todas as possíveis exceções lançadas dentro de um bloco *try*	Exceções não são tratadas ao longo da cadeia de chamada de métodos. Se a exceção chegar ao ponto de entrada do programa, ela causará o fim da execução dele	Chen *et al.* (2009)
Catch Generic	O tratamento captura muitas exceções de baixo nível por meio da definição de uma exceção de alto nível	A exceção é capturada por um tratamento genérico. Isso evita que ela seja tratada apropriadamente	Pádua e Shang (2017)
Destructive Wrapping	Uma exceção é propagada como se fosse outra, sem a passagem dos devidos parâmetros e causa a perda de informação referente à exceção original	Perda de informação do erro original, dificulta a depuração do código	Pádua e Shang (2017)
Catch and Do Nothing	Bloco *catch* vazio	Apesar da exceção ser capturada, nada é feito para tratá-la. O programa continuará sua execução, mesmo após a ocorrência de um evento excepcional. Até mesmo as informações sobre a exceção capturada são perdidas	Pádua e Shang (2017)
Dummy Handler	O tratamento não possui ações para recuperação de erros	O tratamento realiza operações como registro de *log* ou impressão de certas informações no terminal, mas não contribui para a recuperação do erro. Essa abordagem dá a falsa impressão de que o problema é tratado, mas contribui para gerar uma eventual falha no sistema	Chen *et al.* (2009)
Ignoring Interrupted Exception	Uma exceção do tipo InterruptedException é ignorada	Uma exceção do tipo *InterruptedException* é ignorada. Ela indica que alguma medida deve ser tomada pois a operação realizada foi interrompida	Pádua e Shang (2017)
Throw Within Finally	Lançamento de uma exceção dentro de um bloco *finally*	Se o código presente no bloco *finally* lançar uma exceção, as exceções lançadas anteriormente serão perdidas	Pádua e Shang (2017)
Throws Generic	Propagação de uma exceção genérica	Permite que exceções de baixo nível, subclasses da exceção genérica sinalizada, sejam propagadas para camadas superiores da aplicação	Pádua e Shang (2017)
Wrong Exception Thrown	Lançamento de uma exceção genérica	Impede a implementação de um tratamento específico para o problema ocorrido	Ebert *et al.* (2015)

Fonte: o autor.

o processo de alteração mais complexo caso uma exceção baixo nível não seja mais utilizada (JOSHUA, 2012). Por exemplo, não é necessário que objetos da camada de negócio lidem com exceções de baixo nível, como uma *SQLException*, geradas por um problema com o banco de dados (DOSHI, 2003).

As políticas do tratamento de exceção regem o uso de exceções durante o desenvolvimento de software, e auxiliam os desenvolvedores nas fases de implementação e manutenção (BARBOSA *et al.*, 2016). Por definição, podem ser entendidas como as decisões de *design* que determinam como exceções devem ser usadas, tratadas e lançadas, nos diferentes escopos existentes na aplicação (MALAYERI; ALDRICH, 2006).

Além das políticas, existem as linhas-guia (*guidelines*) que fornecem recomendações gerais ou boas práticas para uso de exceções, algumas baseadas em práticas adotadas por programadores experientes. Joshua (2012) e Wirfs-Brock (2006) listam um conjunto de linhas-guia. Elas são sumarizadas a seguir:

- **Use exceções para apenas para situações excepcionais:** exceções não deve ser usadas para ações comuns como controle de fluxo;
- **Use exceções verificadas e não verificadas adequadamente:** Projete o seu software para usar exceções verificadas em situações que seja possível realizar ações de recuperação. Caso contrário, quando ocorre um erro de programação (e.g., divisões por zero, acesso a variáveis nulas), use exceções não verificadas, pois é melhor falhar rapidamente, já que não há como continuar a execução da funcionalidade, do que gastar recursos tentando se recuperar de tais erros;
- **Traduza uma exceção de baixo nível para uma exceção de nível mais alto ao propagá-la:** Se uma exceção não pode ser tratada pelo método de baixo nível, é necessário remapeá-la para o tipo significativo apropriado para a abstração. Essa ação evitará que detalhes desnecessários de implementação atravessem as camadas do sistema e diminuirá o acoplamento entre elementos de abstrações distintas;
- **Forneça as informações contextuais que contribuíram para a exceção:** É importante tornar a mensagem que detalha o problema significativa, de modo que auxilie o usuário (e.g., desenvolvedor, usuário final, equipe de suporte) a reconhecer o problema. Para isso, viabilize os valores dos parâmetros e a mensagem de detalhes da exceção;
- **Não silencie exceções:** Ignorar exceções com tratamentos vazios permite que a execução continue mesmo em estado de erro, o que pode causar uma arbitrária falha futura. Além

disso, impede que a equipe responsável pelo suporte tenha ciência do ocorrido; e

- **Documente as exceções que serão lançadas:** Documentar as exceções que serão lançadas por um método utilizando a anotação *@throws* permite que o desenvolvedor tenha ciência das condições contratuais desse método. Essa recomendação é importante principalmente para exceções não verificadas, pois não são sinalizadas na assinatura dos métodos. Portanto, o desenvolvedor não saberá quais serão lançadas pelo método, a menos que isso esteja documentado.

Algumas dessas recomendações previnem a inserção de *anti-patterns* como Catch and Do Nothing e Destructive Wrapping.

2.3.2 Ferramentas de Análise de Código

Diversas ferramentas foram propostas para checar a conformidade do tratamento de exceção com as políticas produzidas para reger o funcionamento do mecanismo. Algumas dessas ferramentas utilizam linguagens próprias para descrever os módulos, exceções e as relações de dependência entre esses elementos. A Figura 4 exibe um resumo das relações de dependência que tratam das possíveis situações inerentes ao tratamento de exceção, que são levantamento, re-levantamento, sinalização, tratamento, remapeamento e propagação.

Figura 4 – Relações de dependência entre exceções e módulos

Tipo	Descrição
`M` *raises* `E`	O módulo `M` levanta exceções do tipo `E`.
`M` *re-raises* `E`	O módulo `M` re-levanta exceções do tipo `E`.
`M` *handles* `E`	O módulo `M` trata exceções do tipo `E`.
`M` *signals* `E`	O módulo `M` sinaliza exceções do tipo `E`.
`M` *remaps* `E` to `F`	O módulo `M` remapeia exceções do tipo `E` para exceções do tipo `F`.
`E` *flows* $\mathtt{M}_1 \ldots \mathtt{M}_n$	As exceções do tipo `E` são levantadas e sinalizadas pelo módulo $\mathtt{M}_1$ e fluem através dos módulos $\mathtt{M}_2, \ldots, \mathtt{M}_{n-1}$ até serem tratadas pelo módulo $\mathtt{M}_n$.

Fonte: Filho *et al.* (2017).

As regras que restringem as relações de dependência podem ser usadas por arquitetos e desenvolvedores para expressar o comportamento excepcional de um sistema. Essas regras são usadas como parâmetro para as ferramentas que verificam a conformidade arquitetural do tratamento de exceção, mas descritas com de acordo com as especificidades das linguagens adotadas por elas. Exemplos de ferramentas propostas pela literatura com essa finalidade são: ArCatch (FILHO *et al.*, 2017), DCL Suite (TERRA; VALENTE, 2009) e TamDera (GURGEL *et*

al., 2014).

Outras ferramentas atuam na inspeção de código objetivando encontrar estruturas prejudiciais, como os *code-smells* e *bugs*. Tais ferramentas podem ser usadas por desenvolvedores diariamente, integradas a ambientes de desenvolvimento ou a servidores de integração contínua. Algumas ferramentas foram utilizadas neste trabalho e serão melhor descritas no Capítulo 4.

2.4 Conclusão

Neste capítulo, foram apresentados conceitos essenciais para a condução e o entendimento desta pesquisa de mestrado. Exceções, eventos pertencentes aos fluxos anormais dentro de um sistema, foram apresentadas, assim como os alguns desafios relacionados ao desenvolvimento de sistemas tolerantes a falhas. Esses desafios foram confirmados empiricamente ao longo desta pesquisa durante as análises realizadas, como a presença de *anti-patterns*, a violação de *guidelines* amplamente difundidos na literatura, ou a falta de políticas específicas para uma aplicação. Por fim, soluções sugeridas pela literatura e pelo mercado para esses desafios foram brevemente mostradas neste capítulo.

3 TRABALHOS RELACIONADOS

Neste capítulo, são apresentados os trabalhos correlatos a esta pesquisa. Esses trabalhos possuem ênfase na análise do tratamento de exceção. Eles utilizam, entre as suas fontes de dados, artefatos que pertencem ao ciclo de desenvolvimento de sistemas, como código-fonte, relatórios de *bugs* reportados, documentação ou partem da opinião de desenvolvedores profissionais de software.

Todos os trabalhos foram selecionados por meio da estratégia de pesquisa sistemática Snowballing (WOHLIN, 2014). O procedimento de pesquisa é descrito na Seção 3.1. Os trabalhos foram separados de acordo com a origem dos dados coletados pelos autores e classificados nas seguintes categorias: estudos centrados no desenvolvedor (aspecto humano), estudos centrados no software (aspecto técnico) e estudos híbridos (aspectos humanos e técnicos). Os trabalhos de cada categoria estão, respectivamente, nas Seções 3.2, 3.3 e 3.4. A síntese e a discussão dos estudos são apresentadas na Seção 3.5. Por fim, as considerações finais deste capítulo são destacadas na Seção 3.6.

3.1 Procedimento de Pesquisa

Nesta pesquisa, uma revisão da literatura foi conduzida com a abordagem de pesquisa Snowballing, que possibilita a identificação de trabalhos a partir da lista de referências (*backward*) e citações (*forward*) para um conjunto de trabalhos. O primeiro passo determinado pela abordagem é a seleção do conjunto inicial de trabalhos. Neste passo, foram selecionados estudos cujos títulos satisfazem a seguinte *string* de busca:

exception ("handling"OR "handler") ("case study"OR "empirical study"OR "exploratory study")

A *string* foi elaborada com objetivo de encontrar estudos cuja abordagem fosse semelhante à adotada nesta pesquisa de mestrado. No entanto, buscar apenas estudos de caso limitaria a quantidade de resultados e dificultaria a criação do conjunto inicial de trabalhos. Portanto, termos gerais que englobam estudos de caso foram incluídos na *string*, tais como "estudos empíricos"e "estudos exploratórios".

Os seguintes critérios de inclusão e exclusão foram considerados durante todo o procedimento:

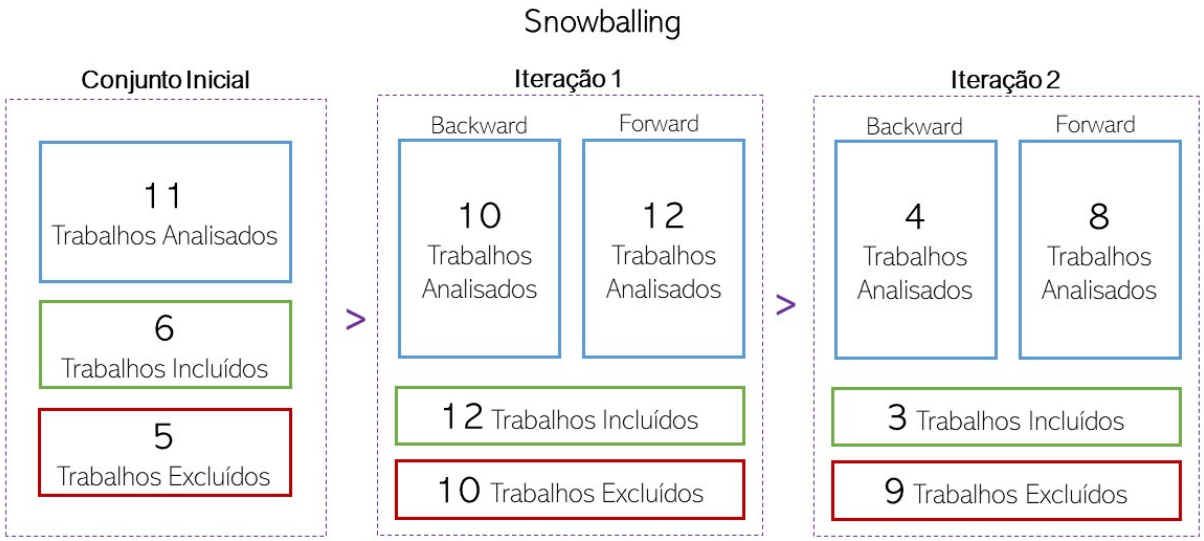
- Critérios de inclusão:

  - Os trabalhos deveriam analisar o tratamento de exceção baseando-se em elementos do ciclo de desenvolvimento de softwares reais; e
  - Trabalhos publicados entre 2010 e 2018.
- Critérios de exclusão:
  - Trabalhos cuja contribuição principal seja a proposição de ferramentas, técnicas ou métodos.

A ferramenta Google Scholar[1] foi utilizada para a busca de trabalhos. A ferramenta possui elementos que permitem navegar facilmente entre as citações e as referencias, o que a torna adequada para execução da estratégia de pesquisa adotada neste trabalho.

Após a seleção inicial, que resultou em 06 trabalhos selecionados, a primeira iteração foi conduzida. Ela consiste na verificação das listas de referências do conjunto inicial (*Backward Snowballing*) e dos trabalhos que citam o conjunto inicial (*Forward Snowballing*). Após a análise do título e a aplicação dos critérios de inclusão e exclusão, 11 trabalhos foram adicionados. Uma segunda iteração foi conduzida, resultando na adição de 03 trabalhos. Nenhum trabalho foi encontrado durante a terceira iteração, ocasionando a finalização do procedimento.

Figura 5 – Processo de pesquisa sistemática



Fonte: o autor.

A Figura 5 representa o processo completo e destaca a quantidade de trabalhos analisados, incluídos e excluídos em cada etapa. Não foi possível determinar a utilização de elementos pertencentes a ambientes reais de desenvolvimento durante a leitura de alguns estudos. Principalmente, quando tratava-se da mineração de repositórios públicos, onde qualquer

[1] https://scholar.google.com.br

Tabela 2 – Classificação de trabalhos por quantidade de citações

Trabalho	Conferência/*Jornal*	Citações	Categoria
Shah *et al.* (2010)	IEEE Transactions on Software Engineering	48	Centrado no Desenvolvedor
Sawadpong *et al.* (2012)	IEEE International Symposium on High-Assurance Systems Engineering	33	Centrado no Software
Kechagia e Spinellis (2014)	Working Conference on Mining Software	25	Centrado no Software
Ebert *et al.* (2015)	Journal of Systems and Software	20	Híbrido
Cacho *et al.* (2014)	IEEE International Conference on Software Maintenance and Evolution	18	Centrado no Software
Marinescu (2011)	International Workshop on Principles of Software Evolution and the 7th annual ERCIM Workshop on Software Evolution	15	Centrado no Software
Barbosa *et al.* (2014)	Brazilian Symposium on Software Engineering	13	Centrado no Software
Ebert e Castor (2013)	IEEE International Conference on Software Maintenance	12	Centrado no Desenvolvedor
Sena *et al.* (2016)	International Conference on Mining Software Repositories	9	Centrado no Software
Marinescu (2013)	International Symposium on Symbolic and Numeric Algorithms for Scientific Computing	9	Centrado no Software
Coelho *et al.* (2017)	Empirical Software Engineering	7	Híbrido
Bonifácio *et al.* (2015)	International Working Conference on Source Code Analysis and Manipulation	4	Híbrido
Oliveira *et al.* (2016)	Brazilian Symposium on Software Engineering	3	Centrado no Software
Oliveira *et al.* (2018b)	Journal of Systems and Software	3	Centrado no Software
Barbosa e Garcia (2011)	Latin-American Symposium on Dependable Computing Workshops	3	Centrado no Software
Cassee *et al.* (2018)	International Conference on Mining Software Repositori	1	Híbrido
Osman *et al.* (2017a)	International Conference on Software Analysis, Evolution and Reengineering	1	Centrado no Software
Pádua e Shang (2017)	IEEE/ACM International Conference on Program Comprehension	1	Centrado no Software
Kechagia *et al.* (2018)	Journal of Systems and Software	0	Centrado no Software
Queiroz e Coelho (2016)	Brazilian Symposium on Software Components, Architectures and Reuse	0	Híbrido
Nogueira *et al.* (2017)	IEEE International Conference on Software Analysis, Evolution and Reengineering	0	Centrado no Software

Fonte: o autor.

código pode ser incluído, desde atividades de disciplinas de programação até sistemas robustos e populares. Por esse motivo, alguns trabalhos foram excluídos mesmo sendo caracterizados como estudos empíricos que abordam o tratamento de exceção.

Nas próximas subseções, são detalhados os trabalhos mais citados, em cada uma das três categorias em que foram classificados. A classificação desses trabalhos por citação e categoria pode ser vista na Tabela 2.

3.2 Estudos Centrados no Desenvolvedor

Ebert e Castor (2013) abordam a percepção de desenvolvedores e organizações sobre *bugs* no tratamento de exceção. A coleta de dados ocorreu mediante questionários *online* respondidos por 154 desenvolvedores e pesquisadores com experiência na linguagem Java. A partir das respostas fornecidas, os autores apresentam algumas conclusões: os códigos relativos ao tratamento de exceção não são testados e documentados com frequência, profissionais mais experientes tendem a ser mais críticos com a qualidade do código destinado ao comportamento excepcional, desenvolvedores usam o tratamento de exceção para criar mecanismos de tolerância a falhas e para melhorar a qualidade do código desenvolvido.

Shah *et al.* (2010) realizaram dois estudos para captar as visões de desenvolvedores iniciantes e experientes sobre o tratamento de exceção. Os dados foram coletados por meio de entrevistas semiestruturadas com os participantes do estudo. Um total de 15 desenvolvedores contribuíram com estudo, todos trabalhavam com Java, e alguns, com outras linguagens como C++ e C#. De acordo com os resultados alcançados, desenvolvedores menos experientes tentam evitar o tratamento de exceção ou somente imitar práticas existentes no código, não dedicando tempo ao desenvolvimento desse tipo de tratamento. Já os desenvolvedores experientes veem o tratamento como parte integrante e inseparável do desenvolvimento de software, também o usam para fornecer melhores *feedbacks* sobre o erro corrido, evitar a corrupção de dados e controlar o fluxo de execução.

Ebert e Castor (2013) e Shah *et al.* (2010) produziram estudos cujo objeto central é a percepção dos desenvolvedores sobre o tratamento de exceção. Por isso, eles não abordam outros aspectos que fazem parte do desenvolvimento do tratamento de exceção, como o código-fonte dos sistemas, o processo de desenvolvimento, a documentação do sistema e as exceções que afetam o usuário final. No entanto, esses trabalhos proporcionaram um melhor entendimento sobre o vertente humana ligada ao tratamento de exceção. Além disso, subsidiaram a condução

de novos estudos, como este trabalho de mestrado.

3.3 Estudos Centrados no Software

Sena *et al.* (2016) conduziram um estudo para entender as abordagens empregadas no tratamento de exceção em bibliotecas Java. As bibliotecas selecionadas tiveram seus fluxos de exceção analisados. Além disso, também foram inspecionadas postagens de problemas relativos ao tratamento de exceção em sistemas de reportação de *bugs*, relacionando-os aos fluxos analisados. Os autores destacam entre suas descobertas que a maioria das bibliotecas averiguadas não documentam *RuntimeExceptions*. *Anti-patterns* do tratamento de exceção foram detectados em 25% dos fluxos analisados e mais de 20% dos problemas reportados das bibliotecas mais populares eram relacionados ao tratamento de exceção.

Sawadpong *et al.* (2012) tem como principal objetivo verificar se o tratamento de exceção pode ser considerado arriscado. Para isso, foi analisada a densidade de defeitos em classes do código da IDE (*Integrated Development Environment*) Eclipse e também um conjunto de métricas sobre o tratamento de exceção extraídas do seu código-fonte. Os resultados extraídos revelam um declínio na quantidade de defeitos durante a evolução do software. No entanto, os defeitos associados ao tratamento de exceção continuaram crescendo ao longo dos seis anos considerados na pesquisa. O contrário acontece com os defeitos não relacionados ao tratamento de exceção.

Cacho *et al.* (2014) conduziram um estudo para entender a relação existente entre a evolução de programas e sua robustez. A pesquisa apresenta foco na evolução do tratamento de exceção em sistemas desenvolvidos em Java e C#. A análise dos fluxos excepcionais é feita com a utilização de métricas que indicam as alterações efetuadas entre versões de um mesmo sistema e o seu impacto. Entre essas métricas, pode-se citar a quantidade de exceções não tratadas ao longo fluxo de execução, que oferece um indicativo de robustez do sistema. Os autores concluem que o tratamento de exceção de sistemas Java sofre mais modificações ao longo do tempo. No entanto, possuem menos cenários que impactam negativamente na sua robustez. De forma contrária, o tratamento de exceção em sistemas C# parece ser mais frágil, indicando uma maior quantidade de exceções não tratadas nos cenários verificados.

Kechagia e Spinellis (2014) tiveram por objetivo mostrar quais métodos com exceções não documentadas são responsáveis por falhas em aplicações. Os autores analisaram diversos *stack traces* de falhas em aplicações Android para identificar os métodos responsáveis

por elas. Eles também analisaram o código-fonte da API Android e catalogaram os métodos que possuem exceções documentadas. A partir dos dados colhidos, eles descobriram que 69% dos métodos envolvidos em falhas não tinham suas exceções documentadas. Além disso, apenas 18% dos métodos públicos ou protegidos possuíam esse tipo de documentação. 24% dos métodos encontrados nos *stack traces* lançavam exceções genéricas, como *RuntimeException* e *NullPointerException*, que não estavam documentadas nas interfaces.

Marinescu (2011) investigou se classes que lidam diretamente com exceções possuem maior propensão a defeitos. Ele analisou três versões do Eclipse. Foram extraídos um conjunto de métricas relacionadas às entidades (classes, métodos, exceções lançadas) e aos defeitos reportados na ferramenta Bugzilla. Testes estatísticos foram realizados para detectar a existência de correlações entre classes que lançam ou tratam exceções com os defeitos reportados. Por fim, concluiu-se que classes que lidam com exceções possuem maior probabilidade de apresentarem defeitos em comparação com classes que não desempenham essa tarefa.

Barbosa *et al.* (2014) buscam entender quais tipos de falhas excepcionais, falhas originadas por causa do uso inadequado ou pela falta do tratamento de exceção, ocorrem em sistemas. Os autores identificaram *releases* dos sistemas Hadoop e Tomcat Apache relacionadas a faltas do tratamento de exceção. Para isso, eles implementaram *scripts* que identificaram *releases* candidatos por meio das mensagens de *commit*. Após esse procedimento, eles manualmente confirmaram os resultados e categorizaram as faltas encontradas. Quase 41% das faltas encontradas foram classificadas na categoria *"Information Swallowed"*, que abriga faltas causadas pela ausência de informações adequadas de uma exceção, também englobando o *anti-pattern* conhecido como *Destructive Wapping*.

Os trabalhos apresentados nesta subseção são estudos que utilizam diversos elementos como código-fonte, *logs* do sistema e *bugs* reportados, para entender como os desenvolvedores utilizam praticamente o tratamento de exceção. Ademais, expõem as deficiências do tratamento de exceção e o impacto para nos sistemas desenvolvidos. Por manterem foco no software, não apresentam a percepção dos desenvolvedores e às dificuldades encontradas nos ambientes de desenvolvimento que interferem na qualidade do tratamento de exceção. Tais aspectos são investigados neste trabalho de mestrado.

3.4 Estudos Híbridos

Bonifácio *et al.* (2015) investigam o uso dos mecanismos do tratamento de exceção em C++ com a finalidade de entender como os desenvolvedores os utilizam para a recuperação de erros em meio às outras atividades necessárias na construção de um software. Eles investigam as práticas implementadas no código-fonte de 65 projetos *open-source* por intermédio de análises estáticas fornecidas por uma ferramenta própria. Os autores analisam a ocorrência de algumas métricas relacionadas à quantidade de *statements* do tratamento de exceção nos sistemas-alvo do estudo. A partir dessa etapa, eles destacaram que, em média, apenas 0.03% do código é destinado às ações de recuperação, que a exceção mais lançada é do tipo *RuntimeException*, e que 16,71% dos tratamentos são vazios. Além disso, conduziram uma pesquisa *online* para captar o entendimento e a percepção de desenvolvedores C++ sobre o tratamento de exceção. A maioria dos respondentes concordou que os desenvolvedores frequentemente evitam o uso dos construtos do tratamento de exceção. Entre as razões indicadas para essa atitude, foram destacadas questões de desempenho e da falta de conhecimento sobre como utilizar esses elementos.

Ebert *et al.* (2015) contribuem com a temática de *bugs* do tratamento de exceção analisando os erros reportados de dois sistemas, Tomcat e Eclipse. Os autores têm por objetivo melhor entender as causas, a severidade, a frequência e a dificuldade de solucionar esses *bugs*. Adicionalmente, por meio de um questionário *online*, buscaram entender as percepções que os desenvolvedores, dos sistemas analisados e de outras instituições, possuem sobre o tratamento de exceção e os *bugs* relacionados ele. Ao final do estudo, os autores concluem que as organizações frequentemente não institucionalizam políticas para o tratamento de exceção, assim como raramente possuem testes ou documentação específica para essa finalidade. Eles também constataram que os *bugs* do tratamento de exceção reportados são raros e as principais causas desses *bugs*, segundo a classificação produzida pelos autores, são a falta de tratamentos que deveriam existir, exceções que deveriam ser lançadas e erros de programação em blocos *catch*.

Queiroz e Coelho (2016) buscaram obter um melhor entendimento sobre características e problemas do tratamento de exceção em aplicativos Android. Os autores analisaram o código de 15 aplicações e relataram que as principais ações nos tratamentos observados eram de *log* (44%) e tratamentos vazios (12%). Além disso, 47% dos tratamentos eram genéricos, a maior parte deles, 41% correspondem ao uso do tipo *Exception*. Os autores também obtiveram informações sobre o tratamento de exceção em aplicações Android por meio da visão

de desenvolvedores, obtida a partir de um questionário *online*. Os desenvolvedores, em sua maioria, informaram que existem especificidades da plataforma que dificultam o tratamento de exceção. O ciclo de vida dos componentes foi a principal característica apontada como fator dificultante. Dentre as ações relatadas que são utilizadas pelos profissionais para sanar problemas do tratamento de exceção estão: o uso de ferramentas que reportam erros, notificar o usuário sobre a ocorrência do problema e falhar rapidamente para evitar estados inconsistentes.

Cassee *et al.* (2018) estudaram como (*guidelines*) produzidos e divulgados sobre o tratamento de exceção na linguagem Swift afetam a maneira que os desenvolvedores utilizam os mecanismos inerentes à implementação dos fluxos excepcionais. Os autores buscaram um conjunto de recomendações e *anti-patterns* e, posteriormente, analisaram se essas práticas são encontrados no código-fonte dos projetos desenvolvidos com a linguagem. Além disso, entrevistaram desenvolvedores para verificar se as percepções sobre o uso do tratamento de exceção são distintas entre experientes e iniciantes em programação. Os autores destacam entre suas descobertas que as recomendações analisadas são pouco adotadas, o que acaba resultando na inclusão de *anti-patterns* no código, como o silenciamento de exceções (*Catch and do Nothing*). Eles sugerem que uma melhor explicação de como os construtos do tratamento de exceção devem ser utilizados impactaria positivamente na qualidade dos fluxos excepcionais.

Coelho *et al.* (2017) investigaram quando *stack traces* reportados podem expressar circunstâncias em que há, em aplicações Android, uma maior probabilidade de *bugs* estarem presentes em códigos (*bug hazards*) do tratamento de exceção. Os autores desenvolveram uma ferramenta para a mineração de *stack traces* armazenados em repositórios públicos. Uma análise manual dos artefatos gerados pela ferramenta foi efetuada para caracterizar as exceções conforme a sua origem e o seu tipo. Entre as descobertas, destacou-se que:

- A causa original da maioria dos *stack traces* das tarefas reportadas eram de erros de programação (Ex: *NullPointerException*, *IllegalArgumentException*, *RuntimeException*);
- 65% das falhas de aplicação eram de exceções *runtime*, apenas uma estava documentada com a *tag* @*throws*; e
- Métodos nativos podem lançar exceções verificadas sem que precisem ser sinalizadas na assinatura, tal comportamento pode ocasionar falhas e dificulta a identificação do problema.

Os autores conduziram uma pesquisa *online* para obter a percepção de desenvolvedores sobre os *bug hazards* encontrados na primeira etapa da pesquisa. Os respondentes afirmaram

que lidam com tratamento de exceções na maior parte do tempo de desenvolvimento, mas poucas vezes ou raramente lançam alguma. A maioria menciona que conhece os *guidelines* apresentados pelos autores, mas poucos utilizam recomendações específicas para Android. Assim como não estavam cientes que métodos nativos podem lançar exceções verificadas sem sinalizações.

3.5 Síntese e Discussão

Os trabalhos apresentados neste capítulo expõem abordagens, finalidades ou métodos de análise distintos para compreender o uso do tratamento de exceção em sistemas e na visão dos profissionais de desenvolvimento. A Tabela 3 resume esses trabalhos, destacando os objetivos, a origem dos dados coletados e os métodos de análise de cada estudo.

Percebe-se, a partir das informações da Tabela 3, que os trabalhos concentram suas contribuições em quatro grandes tópicos: *anti-patterns* do tratamento de exceção (CACHO *et al.*, 2014; SENA *et al.*, 2016; OLIVEIRA *et al.*, 2018a; QUEIROZ; COELHO, 2016; NOGUEIRA *et al.*, 2017; PÁDUA; SHANG, 2017), defeitos do tratamento de exceção (SAWADPONG *et al.*, 2012; MARINESCU, 2011; BARBOSA *et al.*, 2014), percepção dos desenvolvedores (SHAH *et al.*, 2010; EBERT; CASTOR, 2013; CASSEE *et al.*, 2018) e a evolução do tratamento de exceção (OSMAN *et al.*, 2017b; BARBOSA; GARCIA, 2011). Alguns trabalhos contribuem em mais de uma temática, como são os casos de Ebert *et al.* (2015), Bonifácio *et al.* (2015) e Queiroz e Coelho (2016), para realizar o cruzamento das informações coletadas em fontes distintas.

A visão dos desenvolvedores foi coletada mediante formulários *online* (QUEIROZ; COELHO, 2016; EBERT; CASTOR, 2013; BONIFÁCIO *et al.*, 2015; COELHO *et al.*, 2017) e entrevistas semiestruturadas (SHAH *et al.*, 2010; CASSEE *et al.*, 2018)). De forma geral, os autores dessas pesquisas desejavam entender como os desenvolvedores usam o tratamento de exceção, quais abordagens são empregadas, a diferença entre desenvolvedores novatos e experientes, o que eles entendem sobre *bugs* do tratamento de exceção e o como utilizam essas construções em aplicações Android e Swift.

A ocorrência de *anti-patterns* foi percebida em sistemas de naturezas distintas (OLIVEIRA *et al.*, 2018a; PÁDUA; SHANG, 2017), como aplicações *desktop*, servidores e bibliotecas. Assim como em diferentes linguagens de programação (CACHO *et al.*, 2014; PÁDUA; SHANG, 2017). Os autores se utilizaram de inspeções manuais no código-fonte dos sistemas, produziram *scripts* para automatizar a detecção de *anti-patterns* e analisaram a evolução de *anti-patterns* em diversas versões dos sistemas analisados (NOGUEIRA *et al.*, 2017;

OLIVEIRA *et al.*, 2018a; CACHO *et al.*, 2014).

De forma semelhante, defeitos reportados em sistemas foram associados ao uso inadequado do tratamento de exceção. Os dados foram obtidos por meio de *bug reports* e dos sistemas de versionamento. Os autores dessas pesquisas produziram categorizações de *bugs* (EBERT *et al.*, 2015; BARBOSA *et al.*, 2014) e identificaram se classes que tratam exceções têm maior suscetibilidade a defeitos (MARINESCU, 2011; MARINESCU, 2013). Também investigaram quando utilizar o tratamento de exceção é arriscado por meio do cálculo de métricas de densidade de defeitos (SAWADPONG *et al.*, 2012) e descobriram circunstâncias que possuem maiores chances de manifestarem um *bug* (COELHO *et al.*, 2017). Além disso, a falta de documentação para exceções também foi investigada como possível causadora de falhas em aplicações (KECHAGIA; SPINELLIS, 2014; KECHAGIA *et al.*, 2018).

A evolução do tratamento de exceção foi percebida ao se observar o uso dos seus construtos (*try*, *catch*, *throws*), o uso de exceções customizadas e o de *anti-patterns* ao longo das versões dos sistemas analisados. Tais informações eram necessárias para entender o desenvolvimento da parcela do código destinada aos fluxos excepcionais (BONIFÁCIO *et al.*, 2015) e quais mudanças impactam tratamento de exceção (OLIVEIRA *et al.*, 2018a). Além disso, foram investigados como as interfaces excepcionais evoluem (BARBOSA; GARCIA, 2011), se os desenvolvedores fazem mais uso de exceções customizadas à medida que vão adquirindo conhecimento sobre o projeto (OSMAN *et al.*, 2017b) e como mudanças no código normal e excepcional estão relacionadas a falhas no tratamento de exceção em aplicações Java, C# e Android (CACHO *et al.*, 2014; OLIVEIRA *et al.*, 2016).

Tabela 3 – Síntese dos trabalhos relacionados

Trabalho	Objetivo	Origem dos Dados	Métodos de Coleta
Shah *et al.* (2010)	Entender como desenvolvedores novatos e experientes lidam com o tratamento de exceção	Desenvolvedores	Entrevista semiestruturada
Ebert e Castor (2013)	Entender a percepção de desenvolvedores sobre *bugs* cuja causa é associada a problemas do tratamento de exceção	Desenvolvedores com experiência na linguagem Java	Questionário *online*

(Continuação da Tabela 3)

Trabalho	Objetivo	Origem dos Dados	Métodos de Coleta
Sawadpong *et al.* (2012)	Responder se o tratamento de exceção é arriscado	Código-fonte de versões do Eclipse, *bugs* reportados	*Scripts* para a extração de informações do código-fonte e para encontrar os *bugs* reportados que estão associados ao tratamento de exceção
Cacho *et al.* (2014)	Entender a relação existente entre a evolução de programas e sua robustez. A pesquisa apresenta foco na evolução do tratamento de exceção em sistemas desenvolvidos em Java e C#	Código-fonte de um conjunto de bibliotecas, servidores e aplicações	Métricas extraídas do código-fonte por ferramentas de análise de código e por inspeção manual
Sena *et al.* (2016)	Caracterizar as estratégias utilizadas para o tratamento de exceção em bibliotecas Java e identificar os *anti-patterns* do tratamento de exceção presentes nessas bibliotecas	Código-fonte de bibliotecas Java, *bugs* reportados das bibliotecas mais populares	Utilização de uma ferramenta para coleta e análise de fluxos excepcionais, análise manual de *bugs* reportados por desenvolvedores ou usuários das bibliotecas
Oliveira *et al.* (2016)	Entender a relação entre mudanças em aplicativos Android a sua robustez. Além disso, comparam a evolução do tratamento de exceção nesses aplicativos e em programas Java	Código-fonte de um conjunto de bibliotecas, servidores, aplicações e aplicativos Android	Métricas extraídas do código-fonte por ferramentas de análise de código e por inspeção manual

(Continuação da Tabela 3)

Trabalho	Objetivo	Origem dos Dados	Métodos de Coleta
Marinescu (2011)	Identificar se classes que utilizam exceções (tratamentos e lançamentos) apresentam maior suscetibilidade a defeitos	Código-fonte de *releases* do Eclipse, sistema de versionamento do código, sistema de reportação de *bugs*	Extração de métricas de código com ferramentas de análise de código e extração de defeitos reportados/corrigidos por meio do sistema de versionamento e do sistema Bugzilla
Barbosa *et al.* (2014)	Entender as causas das falhas do tratamento de exceção reportadas em sistemas industriais	Código-fonte de *releases*, sistema de versionamento do código e *bugs* reportados dos sistemas Hadoop e Apache Tomcat	Coleta do histórico de revisões dos sistemas e identificação de *releases* relacionados à falhas do tratamento de exceção por meio de *scripts* e inspeções manuais
Nogueira *et al.* (2017)	Caracterizar a evolução do *anti-pattern* "*Empty Catch*" em bibliotecas Java	Código-fonte de projetos da Apache Software Foundation	Métricas extraídas do código-fonte por uma ferramenta e posteriormente processadas. Análise manual dos blocos de tratamento de exceção (*catch*)
Queiroz e Coelho (2016)	Levantar características e problemas do tratamento de exceção das aplicações avaliadas. Visão dos desenvolvedores sobre o tratamento de exceção	Código-fonte de aplicações Android *open-source* e desenvolvedores especialistas em Android	Análise manual do código-fonte e questionário *online*

(Continuação da Tabela 3)

Trabalho	Objetivo	Origem dos Dados	Métodos de Coleta
Ebert *et al.* (2015)	Buscar um melhor entendimento sobre *bugs* do tratamento de exceção, sua frequência, dificuldade de solucionar, severidade e a percepção que os desenvolvedores tem sobre eles	Repositórios de *bug reports* do Tomcat e Eclipse, desenvolvedores Java	Extração de erros reportados por meio da ferramenta Bugzilla, análise manual para a classificação de erros associados ao tratamento de exceção, questionário *online*
Cassee *et al.* (2018)	Analisar se *guidelines* do tratamento de exceção para Swift são seguidas, quais *anti-patterns* são recorrentes e como desenvolvedores iniciantes e veteranos lidam com o tratamento de exceção	Código-fonte de projetos Swift e desenvolvedores Swift	Mineração de repositórios Swift, inspeção manual de códigos-fonte, entrevistas
Oliveira *et al.* (2018a)	Análise da relação entre o uso de abstrações Android e exceções não capturadas	Códigos-fonte de projetos Android de diferentes domínios	Métricas extraídas por meio de ferramentas, análise manual de mudanças entre as versões dos projetos
Osman *et al.* (2017b)	Entender como o uso do tratamento de exceção evolui durante o desenvolvimento de um software, principalmente o uso de exceções customizadas	Códigos-fonte de projetos Java de diferentes domínios	Extração de métricas das releases analisadas por meio de *scripts*
Marinescu (2013)	Examinar classes que tratam e lançam exceções com relação à complexidade e à propensão a defeitos	Códigos-fonte de três releases do Eclipse	Extração de métricas das releases analisadas a partir de ferramentas de análise de código

(Continuação da Tabela 3)

Trabalho	Objetivo	Origem dos Dados	Métodos de Coleta
Kechagia e Spinellis (2014)	Verificar se exceções não documentadas na API podem ser responsáveis por falhas em aplicações	*Stack traces* de falhas em aplicações Android e código-fonte da API Android	Extração de métricas do código-fonte e dos relatórios de erro (*stack traces*)
Pádua e Shang (2017)	Investigar a prevalência de *anti-patterns* em aplicações Java e C#	Códigos-fonte de projetos Java e C#	Detecção de anti-patterns por meio de scripts
Coelho *et al.* (2017)	Investigar quando *stack traces* reportados podem expressar circunstâncias em que há uma maior probabilidade de *bugs* estarem presentes em códigos (*bug hazards*) do tratamento de exceção em aplicações Android	Código-fonte de projetos Android, *stack traces* de falhas em aplicações Android, desenvolvedores Android	Análise de *stack traces* por meio de uma ferramenta própria categorização manual de informações sobre as exceções presentes nos *stack traces*, questionário *online*
Barbosa e Garcia (2011)	Buscar informações sobre as abordagens utilizadas para o desenvolvimento do design excepcional de interfaces em *frameworks*	Código-fonte de *frameworks* Java em diferentes *releases*	Extração de um conjunto de métricas sobre as exceções sinalizadas nos métodos, entre elas, as exceções genéricas como *Throwable* e *Exception*
Bonifácio *et al.* (2015)	Investigar o uso dos mecanismos do tratamento de exceção em C++ com a finalidade de entender como os desenvolvedores os utilizam para a recuperação de erro	Códigos-fonte de 65 projetos *open-source*, desenvolvedores C++	Extração de métricas relacionadas à quantidade de "*statements*" do tratamento do exceção e a diversidade de exceções tratadas e lançadas, questionário *online*

(Continuação da Tabela 3)

Trabalho	Objetivo	Origem dos Dados	Métodos de Coleta
Kechagia *et al.* (2018)	Investigar o tratamento de exceção na API Android para entender como e onde os desenvolvedores utilizam o mecanismo	Documentação e código-fonte da API Android, aplicações clientes da API, *strack traces* de falhas	Mineração de informações sobre as exceções documentadas e não documentadas na API Android, das tratadas em aplicações clientes, das lançadas na API Android e as reportadas como causa original de falha em aplicações

Fonte: o autor.

3.6 Conclusão

Este capítulo abordou os trabalhos relacionados a esta pesquisa de mestrado. Os trabalhos foram separados de acordo com a origem dos dados coletados e resumidos na Tabela 3. A partir do estado da arte obtido por meio desses estudos, foi possível identificar como o tratamento de exceção está sendo analisado, implementado e compreendido. Além disso, os estudos revelaram que *bugs* e *anti-patterns* estão em diversos sistemas, demonstrando a dificuldade que desenvolvedores e instituições possuem em otimizar a utilização e explorar as vantagens do tratamento de exceção.

Os trabalhos relacionados citados neste capítulo foram fonte inspiradora e norteadora desta pesquisa de mestrado. Percebeu-se que as pesquisas híbridas, que investigam aspectos humanos e técnicos, produzem melhor compreensão dos fenômenos investigados e trazem resultados mais impactantes e melhor justificados que as pesquisas que adotam apenas uma única estratégia. Além disso, poucas pesquisas aqui relatadas tiveram acesso simultaneamente ao código e aos profissionais envolvidos no desenvolvimento de um mesmo sistema. Desta forma, optou-se por essa abordagem uma vez que um estudo conduzido próximo ao ambiente real que dá origem ao fenômeno investigado pode elucidar melhor as suas causas e também produzir efeitos positivos neste ambiente (e.g., melhorias na qualidade de versões futuras do sistema). Quanto aos aspectos investigados poucos trabalhos focaram na presença e evolução de *anti-patterns* no desenvolvimento de sistemas Web, sendo este o aspecto principal de investigação desta

pesquisa de mestrado. Outro ponto de inovação é o estudo e análise do impacto da rotatividade na produção e correção de *anti-patterns* do tratamento de exceção.

4 PLANEJAMENTO DO ESTUDO DE CASO

Este capítulo apresenta a descrição e o planejamento do estudo de caso seguido durante toda esta pesquisa de mestrado. O estudo de caso é descrito tendo como base os elementos de design propostos por Runeson *et al.* (2012). A Seção 4.1 lista as motivações para a execução deste estudo. A Seção 4.2 caracteriza o estudo quanto à estratégia de pesquisa, aos métodos de coleta e à origem dos dados. A Seção 4.3 expõe o contexto em que o estudo é conduzido, que é constituído pelo sistema investigado, a equipe desenvolvedora e a instituição mantedora do sistema. Na Seção 4.4, as unidades de análise são descritas . A Seção 4.5 contém a listagem e a descrição de todos os métodos de coleta utilizados. A Seção 4.6 lista todos os métodos de análise. A estratégia de seleção dos objetos de estudo é apresentada na Seção 4.7. Por fim, a Seção 4.9 apresenta as conclusões deste capítulo.

4.1 Razão e Propósito

Este estudo de caso foi conduzido com a intenção de medir a qualidade do tratamento de exceção em um sistema de larga escala utilizado por usuários reais. Sua realização tem foco na investigação e confirmação, de maneira exploratória e prática, da ocorrência dos problemas do tratamento de exceção retratados em pesquisas anteriores, tais como *bugs*, *anti-patterns* e a falta de entendimento desses elementos pelos desenvolvedores. Para isso, foi preciso perceber como os desenvolvedores entendem o tratamento de exceção dos sistemas que desenvolvem, mensurar o estado atual desse tratamento e investigar as motivações para os resultados encontrados. Essas informações foram adquiridas e analisadas durante a execução das unidades que compõem o estudo de caso descritas na Seção 4.4. A pesquisa contou com a colaboração da instituição mantedora do sistema e da sua equipe de desenvolvimento.

4.2 Caracterização

Diferentes aspectos definem como uma estratégia de pesquisa é utilizada para alcançar os objetivos definidos. Runeson *et al.* (2012) apresentam aspectos inerentes às estratégias empíricas. De acordo com as definições apresentadas por eles, pode-se caracterizar o propósito do estudo de caso conduzido nesta pesquisa como exploratório. De fato, buscou-se descobrir o que está acontecendo em um determinado contexto e, consequentemente, gerar conhecimento a respeito do fenômeno investigado.

Os dados buscados durante este estudo são classificados como quantitativos e qualitativos, chamados também de métodos mistos (CRESWELL, 2014), cujo objetivo é prover um melhor entendimento por meio de tipos distintos de informações. Já o processo de pesquisa naturalmente empregado em estudos de caso é definido como flexível. Diferentemente de experimentos, o estudo de caso é conduzido próximo ao ambiente em que fenômeno ocorre, por isso detém um menor controle dos resultados e permite a obtenção de informações inesperadas inerentes ao realismo do contexto. Por isso, ajustes no design original do estudo podem ser necessários a fim de que os objetivos sejam alcançados. Nesta pesquisa, adaptações foram realizadas ao longo do tempo, sendo informadas no decorrer deste documento.

A triangulação, como explanado por Runeson *et al.* (2012), é fundamental para que a compreensão gerada a partir de diferentes perspectivas tornem válidos os resultados encontrados durante o estudo, principalmente por causa da menor precisão característica dos dados qualitativos. Neste estudo, foram aplicados dois tipos de triangulação, que são: triangulação de dados e triangulação de métodos. A triangulação de dados foi alcançada pela utilização de diversas fontes de dados, como pessoas, métricas e documentos. Já a triangulação de métodos foi atendida devido ao uso de métodos mistos de coleta de dados.

4.3 Contexto

4.3.1 Sistema Alvo

O sistema selecionado como alvo deste estudo é Web, possui mais de 20 mil usuários ativos em uma instituição pública de educação do Brasil. É um sistema de gerenciamento implementado em JEE (*Java Enterprise Edition*) e utiliza um conjunto de *frameworks*, como JSF e Structs e Hibernate. Ele foi adquirido há 7 anos, sendo adaptado, expandido e mantido por uma divisão da instituição. 46 profissionais atuam direta ou indiretamente na codificação e análise desse sistema. O sistema estudado atende a diversos segmentos da organização que o mantém. Devido a sua importância, e por fins de segurança e confiabilidade, assegurados pelos envolvidos desta pesquisa, seu nome permanecerá anônimo, sendo referenciado no decorrer deste documento como Sistema XSA.

O sistema foi projetado para englobar um vasto escopo de necessidades para as instituições que iria atender, sendo possível a administração de recursos patrimoniais, pessoal e acadêmico. Ele é divido em quatro camadas: apresentação, aplicação, domínio/negócio e

infraestrutura/acesso a dados. Essa divisão é responsável pela separação de responsabilidades e o agrupamento de módulos com finalidades semelhantes. É composto por um total de 11 projetos, alguns dependentes entre si. Os desenvolvedores trabalham em times dedicados a manter e desenvolver módulos em cada projeto. Os analistas lideram as equipes de desenvolvimento e tratam da comunicação com o cliente. Essas equipes operam de forma independente, por isso podem adotar metodologias diferentes de trabalho, como às ágeis, às tradicionais ou mesmo optar por não utilizá-las.

Durante o estudo de caso, a evolução do sistema foi avaliada. Para isso, foram investigadas 15 *releases*. Como informado, o sistema XSA não foi criado na instituição alvo, ele foi adquirido. No entanto, era necessário que ele se adequasse às regras de negócio específicas da instituição que não eram contempladas em sua versão original. A primeira *release* analisada neste estudo data de 2010. Ela representa o sistema em seu estado inicial, sem qualquer customização inserida pela instituição. O início das customizações podem ser percebidas inicialmente nas *releases* de 2011. A partir desse ano, foram consideradas *releases* semestrais até o final de 2017. A Tabela 4 mostra métricas de alguns dessas *releases*.

Tabela 4 – Evolução do sistema XSA

Dado	2010	2011	2014	2017
Linhas de Código	455819	476645	532685	650261
Número de Pacotes	258	266	275	326
Pastas de Código	28	29	30	35
Número de Classes	3260	3368	3548	4622
Número de Métodos	47019	48711	53405	64307
Número de Interfaces	66	70	77	115
Linhas de Códigos em Blocos *Catch*	5874	6098	6943	8375
Número de Tratamentos	3407	3545	4024	4835

Fonte: o autor.

4.3.2 Instituição

A instituição alvo desta pesquisa é parte de uma corporação pública. Ela é responsável por gerir os sistemas e a infraestrutura da tecnologia da informação de toda a corporação. Seus funcionários são organizados em setores encarregados por cada serviço ou produto oferecido pela instituição, como sistemas, sites, redes e suporte.

Os funcionários são contratados por meio de processos seletivos que testam os conhecimentos do concorrente para o cargo ofertado. Portanto, são pessoas minimamente capacitadas em Tecnologia da Informação.

4.3.3 Equipe de Desenvolvimento

A equipe de desenvolvimento é responsável por todos os sistemas da corporação. É organizada internamente em subequipes: desenvolvimento, banco de dados e gestão de configuração. Cada subequipe é composta por analistas e técnicos. Até o momento da escrita desta dissertação, ela contava com 46 funcionários. A Tabela 5 mostra a evolução da equipe com relação à quantidade de funcionários. A partir da tabela, pode-se constatar um crescimento de 206% de 2011 à 2017. Esse dado é utilizado posteriormente nas análises da Unidade de Análise 3 (Seção 5.3).

Tabela 5 – Crescimento da equipe

Dado	2011	2014	2017
Quantidade de Funcionários	15	35	46

Fonte: o autor.

4.4 Unidades de Análise

As unidades de análise foram definidas de acordo com as questões de pesquisa descritas no Capítulo 1 (Introdução). Essas unidades são ilustradas na Figura 6, que também exibe a origem dos dados obtidos e alguns métodos de coleta.

4.4.1 Unidade de Análise 01: Percepção dos Desenvolvedores

A primeira unidade foi destinada ao aspecto humano do tratamento de exceção, que são as pessoas que lidam diretamente com o planejamento e desenvolvimento dos fluxos excepcionais do sistema. Durante a execução dessa unidade, desejou-se verificar a percepção dos desenvolvedores/analistas a respeito do:

- Grau de importância dado por eles ao tratamento de exceção;
- Estado atual percebido do sistema XSA com relação ao tratamento de exceção;
- Conhecimento adquirido sobre o tratamento de exceção empregado nos sistemas desenvolvidos; e
- Contentamento sobre suas próprias habilidades e conhecimento do tratamento de exceção.

Figura 6 – Estrutura do estudo de caso

Fonte: o autor.

4.4.2 Unidade de Análise 02: Análise Empírica do Sistema Java Web

A segunda unidade de análise foi destinada ao aspecto técnico do tratamento de exceção, que diz respeito ao Sistema XSA e ao ferramental que oferece suporte ao monitoramento e normatização do tratamento de exceção. Nesta unidade, um conjunto de ferramentas foi utilizado para determinar a qualidade do tratamento de exceção. A seleção dessas ferramentas foi apoiada pela condução de uma pesquisa bibliográfica de modo a utilizar contribuições acadêmicas existentes.

A avaliação do tratamento de exceção tem sido abordada considerando-se aspectos distintos da qualidade de software, sendo eles relevantes para o entendimento, o funcionamento, e a avaliação do tratamento de exceção utilizado em um sistema. Pesquisas, como as correlatas a esta e citadas no Capítulo 3, abordam alguns fatores, como documentação, *anti-patterns*, *bugs* e princípios gerais do tratamento de exceção. De maneira semelhante, alguns desses aspectos serviram como guia para análise do tratamento de exceção do sistema XSA.

4.4.3 Unidade de Análise 03: Investigação Sobre as Causas dos Problemas do Tratamento de Exceção Implementado

Em princípio, a terceira unidade era dedicada à normatização do tratamento de exceção, à divulgação e à checagem periódica dessas normas. A normatização seria obtida na própria organização, entendendo os requisitos desejáveis para o tratamento de exceção, e na literatura, por meio do atendimento às boas práticas, às recomendações comprovadamente estabelecidas, metodologias e técnicas para a definição de regras. Prendia-se englobar o tratamento de exceção internamente nas entidades do software (classes, componentes, etc), como também nos relacionamentos entre os componentes do sistema, criando uma regulamentação arquitetural.

No entanto, as informações obtidas na Unidade de Análise 02, que indicaram a ocorrência de *anti-patterns* desde os anos iniciais do sistema, fizeram surgir novos questionamentos e hipóteses:

- Questionamento 01: Quais eram as causas para a ocorrência dos *anti-patterns*?
  - Suposição 01: A alta rotatividade de desenvolvedores dificulta que os profissionais contratados se tornem experientes no código e, consequentemente, no correto tratamento de exceção exigido para o sistema;
  - Suposição 02: Desenvolvedores menos experientes são responsáveis pela inserção da maioria dos *anti-patterns*, pesquisas anteriores destacaram que eles não veem o tratamento de exceção como uma atividade prioritária;
  - Suposição 03: Os desenvolvedores não sabem que estão inserindo *anti-patterns* no código, pois não conhecem essas estruturas;
  - Suposição 04: Os desenvolvedores não sabem como devem lidar com o tratamento de exceção ao longo das camadas do sistema porque faltam políticas que descrevam esse uso, o que propicia a inserção de *anti-patterns*;
  - Suposição 05: O sistema XSA, assim como outros de mesma procedência, possuem diversos *anti-patterns* porque eles já existiam no sistema original.
- Questionamento 02: Quais são as exceções que mais afetam o usuário final do sistema alvo?

As respostas para esses questionamentos apresentados se mostraram mais importantes para que a instituição tomasse ciência do contexto que origina os *anti-patterns*. E assim determinar estratégias para que os corrigisse e impedisse que novos fossem inseridos. Por isso, o objetivo da Unidade 03 foi modificado, tornando-se dedicada a responder essas questões e

validar as suposições elaboradas.

4.5 Métodos de Coleta

Vários métodos de coleta foram utilizados neste estudo devido à natureza diversa da origem dos dados (pessoas, código e documentação). Eles são classificados na Tabela 6. Dentre os métodos utilizados, há aqueles que fornecem dados quantitativos, qualitativos ou ambos. Os métodos também podem ser categorizados de acordo com o grau de intervenção humana requerida para serem executados, assim como sugere a taxonomia de Lethbridge *et al.* (2005). Métodos de primeiro grau de contato necessitam de acesso direto aos participantes de estudo. Para a execução dos métodos de segundo grau, é preciso acessar o ambiente dos participantes, mas não é necessário que pesquisadores interajam com eles diretamente. Já métodos de terceiro grau requerem acesso apenas à artefatos, como documentos, códigos e diagramas. Os métodos usados são descritos em detalhes nas subseções seguintes.

Tabela 6 – Síntese e classificação dos métodos de coleta

Método	Natureza dos Dados	Categoria	Unidades de Análise
Questionário *Online*	Qualitativa e Quantitativa	Primeiro Grau	Unidade 1 e Unidade 3
Entrevista	Qualitativa	Primeiro Grau	Unidade 3
Métricas de Código	Quantitativa	Terceiro Grau	Unidade 2 e Unidade 3
Dados Arquivados	Qualitativa e Quantitativa	Terceiro Grau	Unidade 3

Fonte: o autor.

4.5.1 Questionário Online

Dois questionários *online* foram aplicados durante todo o estudo. O primeiro formulário *online* foi elaborado para obter a percepção dos desenvolvedores e analistas sobre o tratamento de exceção. Os trabalhos de Ebert et al. (EBERT *et al.*, 2015) e Shah et al. (SHAH *et al.*, 2010) foram utilizados como base para a construção do formulário, assim como alguns aspectos inerentes ao desenvolvimento de software e ao tratamento de exceção, sendo eles:

- Experiência no desenvolvimento de sistemas Web;
- Experiência com os sistemas desenvolvidos na corporação em que o profissional atua;
- Existência de uma documentação que guie o desenvolvimento do tratamento de exceção;
- Importância do tratamento de exceção;
- Conhecimento técnico sobre o tratamento de exceção;

- *Bugs* do tratamento de exceção;
- Utilização do tratamento de exceção durante as atividades desenvolvidas;
- Satisfação com o próprio conhecimento e com o tratamento de exceção praticado na corporação;

O segundo questionário tinha como propósito identificar quais estruturas prejudiciais do tratamento de exceção eram conhecidas e passíveis de reconhecimento pelos desenvolvedores da instituição. O instrumento foi elaborado com base no trabalho de Palomba *et al.* (2014) que, de maneira semelhante, buscava entender como *code smells* eram percebidos por desenvolvedores.

Os dois questionários podem ser encontrados integralmente nos Apêndices B e C.

4.5.2 Métricas de Código

Algumas das unidades de análise descritas neste capítulo demandam o conhecimento de como o tratamento de exceção é usado no sistema XSA. Essa informação foi obtida com a extração de métricas de código, como a quantidade de métodos, classes, blocos *catch* e *anti-patterns*. Para isso, diversas ferramentas de análise estática de código foram buscadas na academia e no mercado. Dentre elas, para a detectar presença de *anti-patterns*, optou-se por soluções livres e que possibilitassem a identificação de pelo menos um dos *anti-patterns* mostrados na Tabela 1. A seguir, a lista das ferramentas encontradas:

- PMD[1]: Solução Java para análise de código disponível para diversas linguagens. É capaz de detectar possíveis *bugs*, código não utilizado, duplicações e complexidades desnecessárias;
- Checkstyle[2]: Ferramenta que auxilia na identificação de implementações fora de um padrão estabelecido. Encontra problemas relativos ao *design* de elementos de código, formatação e *layout*;
- FindBugs[3]: Ferramenta que busca *bug patterns* em códigos Java, desde más práticas até implementações vulneráveis;
- Parychayana[4]: Ferramenta que detecta *code smells* e *anti-patterns* do tratamento de exceção em códigos Java.

As ferramentas foram utilizadas em uma análise inicial do XSA. A partir dos dados colhidos, vários *anti-patterns* foram detectados no código-fonte da versão mais atual do sistema.

[1] https://pmd.github.io/
[2] http://checkstyle.sourceforge.net/
[3] http://findbugs.sourceforge.net
[4] https://github.com/ashishsureka/parichayana

No entanto, para facilitar a condução desta pesquisa, foi preciso a adotar apenas uma ferramenta. Já que a comparação de ferramentas e a indicação de alguma para a detecção de problemas no tratamento de exceção não estão no escopo deste trabalho. Por apresentar diversas regras para o tratamento de exceção, por ser amplamente utilizada e aceita pela comunidade de desenvolvedores Java, a ferramenta PMD foi adotada. O procedimento seguido nessa análise inicial com todas as ferramentas é detalhado no Apêndice A.

Além dessas, as ferramentas Eclipse Metrics[5] e JMetrix foram aplicadas para a extração das demais métricas necessárias para as análises desenvolvidas no decorrer das unidades. A ferramenta JMetrix foi desenvolvida na Universidade Federal do Ceará, ela é baseada na ferramenta Spoon (PAWLAK *et al.*, 2015).

4.5.3 Entrevista

Uma entrevista semiestruturada foi conduzida com alguns funcionários da instituição mantedora do sistema XSA. A entrevista teve como objetivo proporcionar um melhor entendimento sobre os resultados obtidos no estudo e o contexto motivador desses resultados. Ela foi dividida em 4 fases:

- Fase 1: Informações sobre a pesquisa, seus objetivos e estrutura;
- Fase 2: Apresentação e discussão dos resultados obtidos com o preenchimento do questionário *online*, o primeiro método de coleta utilizado (Unidade de Análise 01);
- Fase 3: Apresentação e discussão dos resultados da coleta de dados referentes à segunda questão de pesquisa (Unidade de Análise 02);
- Fase 4: Sugestões e avaliação da entrevista.

Três pessoas participaram dessa etapa do estudo, um analisa de software, o gerente de projetos e o gestor da equipe de desenvolvimento. A entrevista foi gravada e depois transcrita para posterior análise.

4.5.4 Dados Arquivados

Informações sobre a equipe de desenvolvimento foram extraídas do banco de dados da instituição. As informações consistiam na listagem de todos os funcionários que haviam integrado a equipe de desenvolvimento entre os anos de 2010 e 2017. Além disso, a data de afastamento ou desligamento, os cargos e a data de ingresso na instituição também foram

[5] https://marketplace.eclipse.org/content/eclipse-metrics

necessários.

A documentação do tratamento de exceção também foi averiguada. O objetivo era entender como o tratamento de exceção estava estruturado dentro dos sistemas e quais informações estavam disponíveis para os desenvolvedores da instituição.

4.6 Métodos de Análise

A variabilidade de dados coletados tornou necessária a aplicação de diversos métodos de análise. Esses métodos são indicados na Tabela 7, sendo classificados de acordo com o tipo de dado coletado.

Tabela 7 – Síntese dos métodos de análise

Dado Coletado	Método de Coleta	Método de Análise
Percepção dos desenvolvedores sobre o tratamento de exceção	Questionário *online*	Estatística descritiva
Opinião dos profissionais da instituição sobre os resultados obtidos	Entrevista	Codificação
Informações sobre o tratamento de exceção presentes no código	Métricas de código	Estatística descritiva
Informações sobre a experiência dos desenvolvedores	Dados arquivados	Análise manual
Dados sobre as exceções que mais afetam os usuários finais do sistema XSA	Dados arquivados	Estatística descritiva
Conhecimento dos desenvolvedores sobre *anti-patterns* do tratamento de exceção	Questionário *online*	Estatística descritiva e Codificação

Fonte: o autor.

Além dos métodos apresentados, foram conduzidas análises comparativas com o estado da arte ao longo do estudo. Tais análises propiciaram um melhor entendimento do contexto da instituição com relação ao desenvolvimento nacional e internacional do tratamento de exceção.

4.7 Estratégia de Seleção do Caso

A instituição, contactada previamente antes do início desta pesquisa, tornou possível a condução do estudo de caso, se mostrando favorável e viabilizando as informações necessárias. Além disso, ela não apresenta uma documentação extensa sobre o tratamento de exceção, o que pode gerar dúvidas durante as atividades de desenvolvimento e, consequentemente, interferir na capacidade do sistema se recuperar de faltas, assim como é sugerido em pesquisas anteriores (CASSEE *et al.*, 2018; FILHO *et al.*, 2017). Esse contexto propicia a existência dos problemas

investigados neste estudo e contribuiu para a utilização da instituição nesta pesquisa de mestrado.

As unidades de análise foram selecionadas de acordo com as necessidades identificadas no início e no decorrer do estudo. Por esse motivo, sofreram alterações ao longo do tempo como já foi explicado. As duas primeiras unidades foram elaboradas logo no início da pesquisa, principalmente pela urgência em entender como o tratamento de exceção era visto e utilizado na instituição. A terceira unidade de análise era destinada à criação de uma regulamentação do tratamento de exceção para o uso da instituição. No entanto, achou-se proveitoso modificar a sua finalidade para que a motivação para os resultados obtidos nas unidades anteriores fosse encontrada.

4.8 Estratégia de Seleção dos Dados

A equipe de desenvolvimento foi escolhida como participante deste estudo por ser aquela que lida com o planejamento e a implementação dos fluxos excepcionais do sistema. As outras equipes, apesar de lidarem com atividades relacionadas ao sistema, como suporte a usuários e segurança, não teriam conhecimento sobre como o tratamento de exceção é usado no sistema. A equipe de desenvolvimento participou durante as entrevistas e respondendo aos questionários disponibilizados.

O sistema Java mais importante foi utilizado para a análise de código, sendo ele o que concentra as principais funcionalidades e a maior quantidade de usuários. Ele foi examinado de forma evolutiva a fim de que o desenvolvimento do tratamento de exceção fosse entendido desde o início da customização do sistema em 2010.

4.9 Conclusão

Este capítulo abordou o planejamento e os métodos de pesquisa que guiaram a condução do Estudo de Caso deste trabalho de mestrado. A pesquisa está centrada na análise do tratamento de exceção em um sistema utilizado por usuários reais. Ela é caracterizada como um estudo de caso exploratório, e faz uso de diferentes fontes de dados e métodos de análise, portanto, utiliza informações originárias de perspectivas distintas.

O estudo foi dividido em três unidades de análise que destinam-se ao aspecto humano do tratamento de exceção (desenvolvedores), ao aspecto técnico (análise do sistema) e à busca pelos elementos contextuais que originaram os resultados encontrados. Os dados foram colhidos

a partir de quatro métodos, sendo eles qualitativos ou quantitativos (Tabela 6). Os resultados são descritos no próximo capítulo desta dissertação.

5 EXECUÇÃO DO ESTUDO DE CASO

Neste capítulo, são detalhados os materiais, métodos, procedimentos planejados e implementados durante cada etapa do estudo de caso. Além disso, são detalhados os resultados produzidos durante as unidades de análise, descritas no Capítulo 4. Os resultados referentes à Unidade de Análise 1 podem ser vistos na Seção 5.1. Assim como os resultados das Unidades de Análise 1 e 2 são encontrados na Seção 5.2 e na Seção 5.3, respectivamente. Ao fim deste capítulo, encontra-se a Seção 5.4, que apresenta as considerações finais.

5.1 Unidade de Análise 1: Percepções Sobre o Tratamento de Exceção

A Unidade de Análise 01 mostra a perspectiva dos desenvolvedores sobre o tratamento de exceção, abordando questões gerais como importância dada ao mecanismo no desenvolvimento Web, a documentação, a necessidade de melhoria do tratamento de exceção nos sistemas desenvolvidos, e a existência de problemas que afetam o usuário final. As subseções a seguir explicam como os dados foram coletados e as conclusões geradas a partir deles.

5.1.1 Materiais e Métodos

O primeiro método de coleta foi um formulário *online* baseado nos trabalhos de Ebert *et al.* (2015) e Shah *et al.* (2010). O questionário é composto pelos seguintes questionamentos:

- 2 questões abordam a experiência do respondente no seu local de trabalho e no desenvolvimento de software;
- 26 afirmativas correspondentes aos aspectos de interesse deste trabalho relacionados ao tratamento de exceção;
- 2 questões sobre conceitos gerais do tratamento de exceção;
- 2 questões abertas a respeito do uso e das percepções do tratamento de exceção;
- e 1 questão para a avaliação do formulário.

Na etapa correspondente às afirmativas, era necessário que o respondente informasse o grau de conformidade com elas por meio das alternativas disponíveis que seguiam a escala Likert. Essas alternativas tinham conteúdo padronizado e representam cinco níveis de concordância, sendo eles: concordo totalmente, concordo, indiferente, discordo e discordo totalmente. Todas as questões do questionário podem ser vistas no Apêndice B.

5.1.2 Procedimento

Antes da aplicação, o formulário *online* foi avaliado por três especialistas em qualidade e arquitetura de software Web que encontraram erros e sugeriram modificações. Após a correção do instrumento, foi enviada uma mensagem para a lista de *e-mails* dos profissionais da corporação. Nessa mensagem, o propósito da pesquisa foi apresentado, um link para o instrumento foi disponibilizado, assim como também foi garantida a anonimidade das informações enviadas pelo formulário e o seu uso exclusivo para fins acadêmicos.

5.1.3 Perfil dos Participantes

O público-alvo do formulário *online* foi composto por desenvolvedores técnicos e analistas da instituição detentora do sistema do XSA. Esses profissionais são divididos em atribuições, como desenvolvimento de sistemas, análise de negócio, gerência de banco de dados e gerência de projetos. Podendo atuar em apenas uma ou em várias dessas atribuições.

Os profissionais lidam, direta ou diretamente, com o desenvolvimento de sistemas Java Web, sendo essa a tecnologia predominante na instituição. Utilizam banco de dados relacional e um sistema para o gerenciamento dos projetos e documentação.

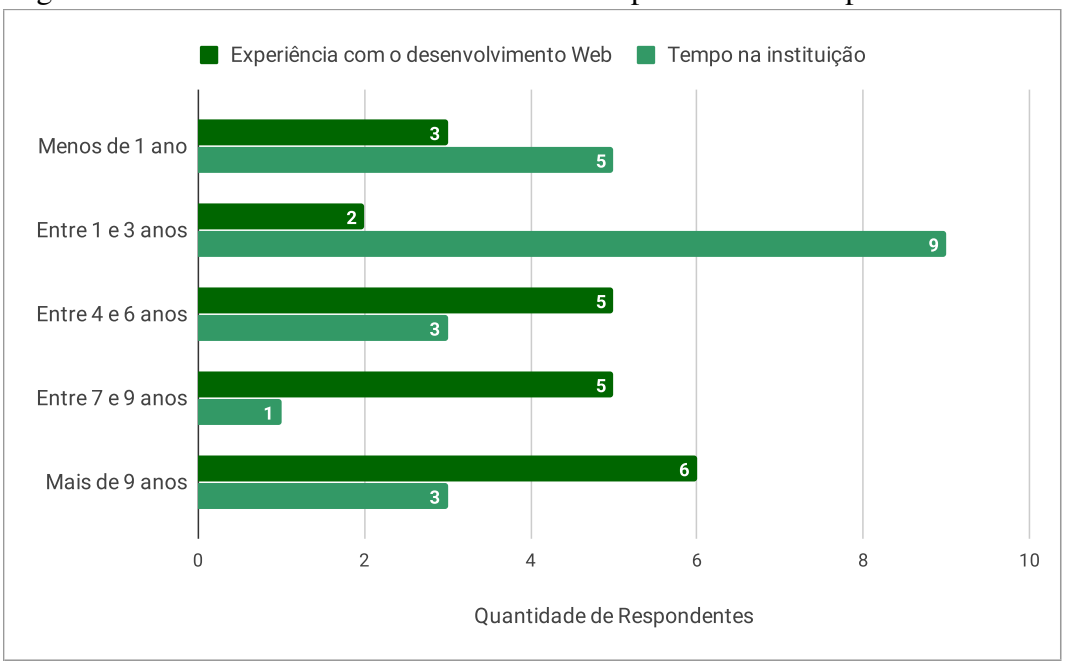
5.1.4 Resultados

As respostas recebidas para cada afirmativa nos itens "Concordo"e "Concordo totalmente"foram agregadas durante a análise dos dados aqui expostos. O mesmo processo de agregação foi seguido para as repostas "Discordo"e "Discordo totalmente". Essa abordagem foi empregada com a finalidade de facilitar a interpretação dos resultados e a sua discussão ao longo das seções deste trabalho.

O formulário *online* recebeu 21 respostas (53,84% dos profissionais envolvidos no desenvolvimento do sistema). Quase metade dos respondentes trabalha há menos de 6 anos com o desenvolvimento Web, cerca de 47,61% (Figura 7). O restante está nesse ramo há mais de 7 anos. Grande parte, 66,66%, trabalha na instituição em que atua há apenas 3 anos. 61,9% se consideram experientes quando se trata dos sistemas desenvolvidos (Figura 8 - Item A1). Contudo, 52,3% dos profissionais declaram ainda não possuir domínio completo do ferramental e das linguagens de programação utilizados na corporação (Figura 8 - Item A2).

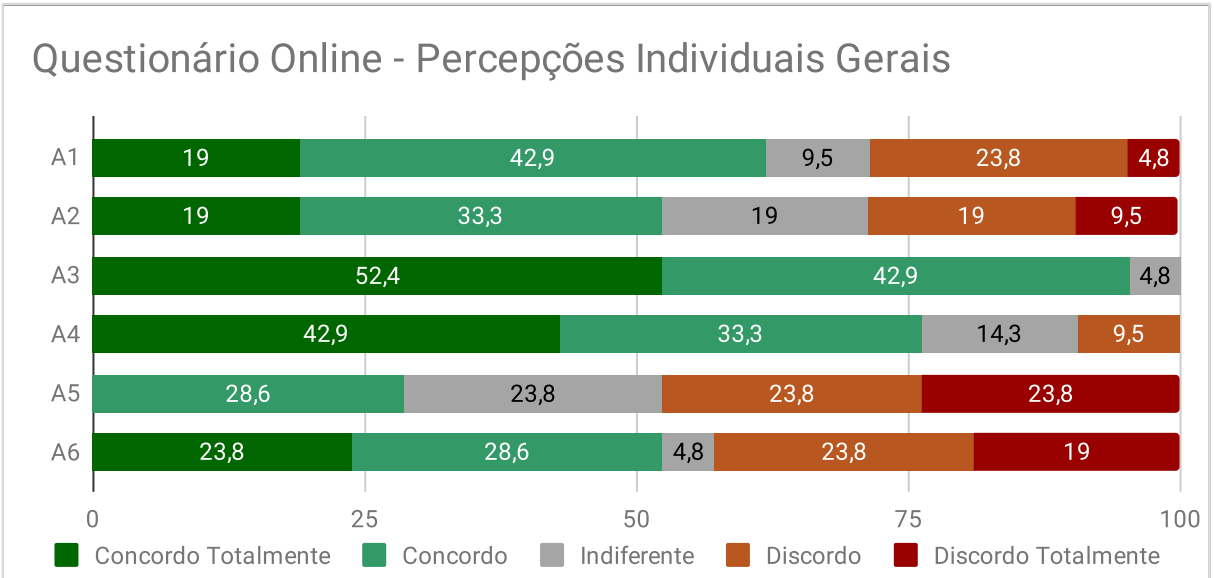
A maioria dos respondentes considera o tratamento de exceção crucial para o desen-

Figura 7 – Resultados do formulário *online* - Experiência dos respondentes



Fonte: o autor.

Figura 8 – Resultados do formulário *online* - Percepções gerais

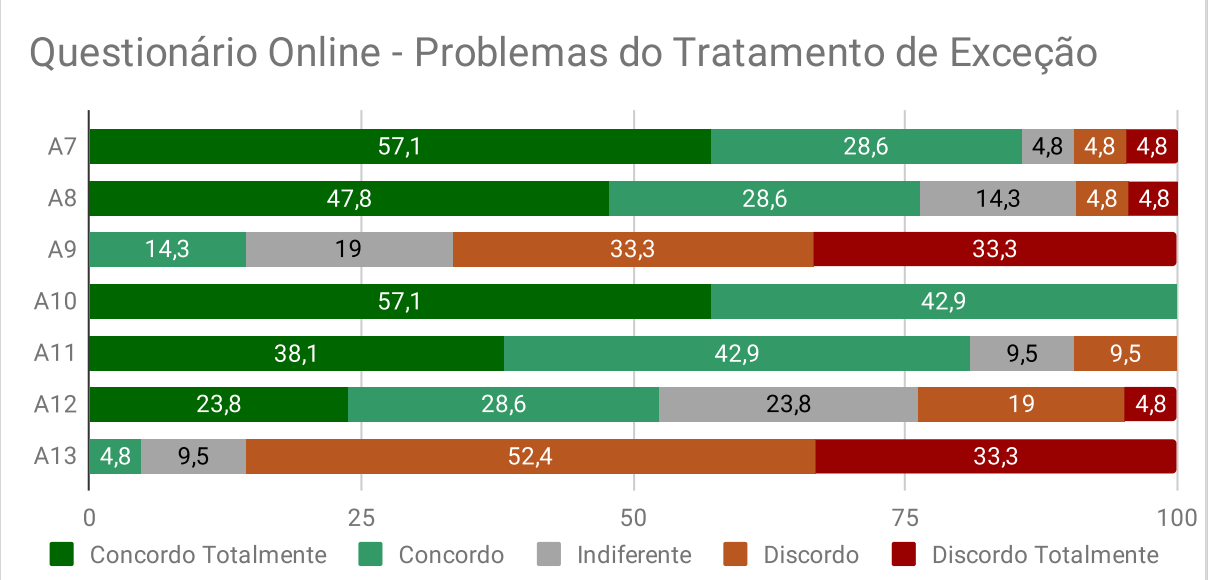


A1	Sou um desenvolvedor/analista experiente considerando o nível de conhecimento que tenho sobre os sistemas Web que desenvolvo
A2	Ainda não domino completamente a linguagem de programação (backend ou frontend) e o ferramental que utilizo no desenvolvimento dos sistemas Web que participo
A3	Considero o tratamento de exceção crucial no desenvolvimento de um sistema Web
A4	A obrigação imposta por algumas linguagens de programação de backend para que o tratamento de exceções seja implementado corrobora com a minha visão de uma boa linguagem de programação
A5	Estou satisfeito com a maneira que lido com o tratamento de exceção durante a execução de tarefas diárias em minha equipe de desenvolvimento
A6	Aprendi em cursos ou treinamentos como utilizar o tratamento de exceção adequadamente

Fonte: o autor.

volvimento de sistemas Web (Figura 8 - Item A3). 76.2% concordam com a obrigação imposta por algumas linguagens para que o tratamento de exceção seja realizado (Figura 8 - Item A4). Esses profissionais também se preocupam com a implementação das atividades excepcionais, apesar delas exigirem maior atenção por causa da sua complexidade (Figura 9 - Item A11). Essas visões corroboram com as respostas obtidas na questão aberta "Qual é a sua opinião sobre o tratamento de exceção?", exibidas na Tabela 8.

Figura 9 – Resultados do formulário *online* - Problemas do tratamento de exceção



A7	Acredito que usuários finais dos sistemas Web que desenvolvo são afetados por erros do tratamento de exceção
A8	Erros ocasionados por problemas do tratamento de exceção já foram reportados à equipe de desenvolvimento
A9	Utilizo rotineiramente ferramentas/técnicas para detectar bad smells do tratamento de exceção
A10	Acredito que o tratamento de exceção dos sistemas em que trabalho precisa ser melhorado
A11	O desenvolvimento de fluxos de exceção exigem maior atenção pois são mais complexos
A12	Geralmente, não documento elementos de código destinados ao tratamento de exceção
A13	A instituição em que trabalho possui especificações, políticas documentadas ou padrões relacionados à implementação do tratamento de exceção

Fonte: o autor.

Cerca de 52,4% dos respondentes revelam que não costumam documentar elementos de código relativos ao tratamento de exceção (Figura 9 - Item A12). Separando os desenvolvedores em dois grupos (iniciantes e experientes), descobriu-se que nenhum dos iniciantes (menos de três anos de experiência), se posicionou como um desenvolvedor que documenta códigos dos fluxos excepcionais. De forma contraria, 43,75% dos experientes confirmaram que fornecem documentação aos códigos do tratamento de exceção.

Tabela 8 – Opinião dos desenvolvedores sobre o tratamento de exceção

Experiência com desenvolvimento de software Web	Qual a sua opinião sobre o tratamento de exceção?
Menos de 1 ano	De muita importância para o desenvolvimento do sistema, uma vez que ele deverá atender as expectativas do usuário
	Tratar exceções é necessário para evitar comportamentos inesperados em qualquer sistema
	É muito complexo saber todos os casos que devem ser tratados
Entre 1 e 3 anos	Deve haver um tratamento de exceções, preferencialmente seguindo padrões especificados pela instituição. (...)
Entre 7 e 9 anos	Algo que deve ser atendado por desenvolvedores, tanto para evitar tratar erros que ocorram a nível de aplicação quanto para minimizar o impacto de erros oriundos de procedimentos errôneos dos usuários
	Necessário pra validar a robustez e confiabilidade de um software
Mais de 9 anos	É imprescindível
	É de fundamental importância
	Necessário e eficiente, se bem trabalhado
	Essencial
	Muito importante para a construção de sistemas confiáveis e grau elevado de disponibilidade
	Pessoalmente, tento não evitar tratamento de exceção. (...)

Fonte: o autor.

Quase todos os desenvolvedores da instituição concordam que não existem *guidelines* ou políticas para o tratamento de exceção para os seus sistemas (Figura 9 - Item A13). O impacto de erros relativos ao tratamento de exceção nos usuários finais do sistema é confirmado por 85,7% dos colaboradores (Figura 9 - Item A7). 76,2% concordam que esses erros, em algum momento, já foram reportados a equipe de desenvolvimento (Figura 9 - Item A8). Apenas 14,3% dos desenvolvedores afirmam usar ferramentas de manutenção da qualidade do tratamento de exceção, como para a checagem de *bad smells* (Figura 9 - Item A9). Particularmente, os desenvolvedores iniciantes não utilizam essas ferramentas. Assim, os erros reportados podem ser consequência da baixa adesão dessas ferramentas ao longo do ciclo de desenvolvimento e da manutenção dos sistemas desenvolvidos. Por fim, as respostas indicam um não contentamento dos profissionais com a forma que lidam com o tratamento de exceção na condução de suas atividades de desenvolvimento e análise (Figura 8 - Item A5).

5.2 Unidade de Análise 2: Análise Empírica do Projeto Java

A Unidade de Análise 02 apresenta a análise do código-fonte do tratamento de exceção contido no sistema XSA. A utilização de ferramentas de análise estática de código possibilitaram encontrar estruturas prejudiciais, os *anti-patterns* do tratamento de exceção, nas versões analisadas do sistema neste estudo, o que forneceu uma perspectiva evolutiva da problemática investigada. As próximas subseções apresentarão as ferramentas utilizadas, os procedimentos e os resultados alcançados.

5.2.1 Materiais e Procedimento

A análise evolutiva foi efetuada por meio da extração de métricas e *anti-patterns* de 15 versões do sistema. Para isso, um projeto na linguagem Java, nomeado ViolationsByRepository (VBR), foi desenvolvido. O projeto utiliza a ferramenta RepoDriller[1], que fornece acesso ao código-fonte das versões por meio do repositório Git do XSA. A ferramenta torna possível a geração de arquivos de saída no formato CSV (*Comma-Separated Values*).

A ferramenta PMD foi incluída no projeto VBR, sendo utilizada exclusivamente para a detecção de *anti-patterns* nas versões do sistema. Já algumas métricas, como a quantidade de blocos *catch* e sinalizações, foram extraídas com a ferramenta JavaParser[2]. Essas métricas foram usadas para a análise das proporções de crescimento dos *anti-patterns* e do próprio sistema.

Os arquivos CSV, gerados durante a manipulação do código-fonte, foram posteriormente processados como auxílio de bibliotecas JavaScript (dc.js[3] e Crossfilter[4]). Elas também apoiaram a geração de gráficos exibidos em uma página HTML criada para facilitar a compreensão dos resultados.

5.2.2 Resultados

No restante do texto, o termo *violação* foi adotado para indicar a ocorrência de um *anti-pattern* do tratamento de exceção, uma vez que a ferramenta os detectou usando regras de código previamente definidas. A análise evolucionária é centrada nos quatro *anti-patterns* mais recorrentes (veja a Tabela 9).

[1] https://github.com/mauricioaniche/repodriller
[2] http://javaparser.org/
[3] https://dc-js.github.io/dc.js/
[4] https://github.com/square/crossfilter

5.2.2.1 Último Release - 2017.2

A Tabela 9 apresenta os resultados para a presença de *anti-patterns* do tratamento de exceção na última versão analisada do sistema XSA. Além dos *anti-patterns*, outras regras do tratamento disponíveis foram consideradas, sendo elas também exibidas na Tabela 9

Tabela 9 – Violações encontradas

Anti-Pattern	Violações	Tratamentos ou Sinalizações Afetadas
PMD - *Anti-Patterns*		
Catch Generic	2440	50,51%
Throws Generic	495	2,60%
Destructive Wrapping	215	4,45%
Catch and Do Nothing	115	2,38%
Throw within Finally	41	0,85%
Wrong Exception Thrown	17	0,35%
Relying on getCause()	5	0,10%
Dummy Handler	4	0,08%
Error in the Definition of Exception Class	0	0,00%
PMD - Violações de Outras Regras		
AvoidRethrowingException	43	0,89%
EmptyFinallyBlock	18	0,37%
AvoidThrowingNewInstanceOfSameException	14	0,29%
EmptyTryBlock	2	0,04%
ExceptionAsFlowControl	0	0,00%
AvoidCatchingThrowable	0	0,00%

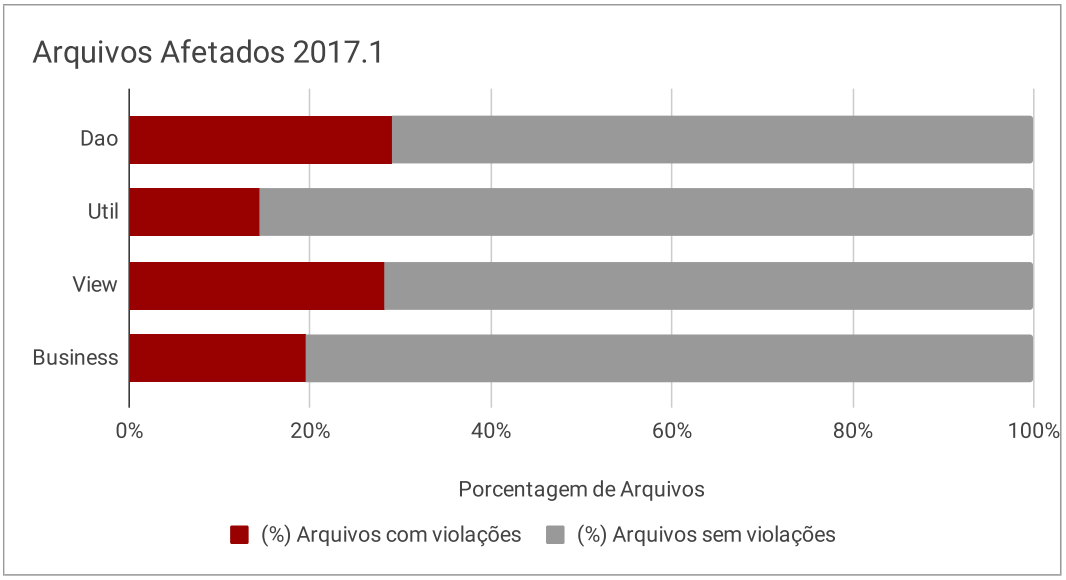
Fonte: o autor.

O *anti-pattern* relacionado à captura de exceções genéricas (ou seja, Catch Generic) é o que tem mais violações no sistema XSA. Nos resultados, encontra-se mais de 70% de violações para esse *anti-pattern*. Para essa análise, se considerou apenas exceções genéricas definidas na linguagem Java, como *NullPointerException*, *Exception* e *RuntimeException*. O *anti-pattern* Catch and Do Nothing, o quarto *anti-pattern* com o maior número de violações, é um risco de manutenção já que dificulta a depuração de código e provoca perda de informações do erro original (EBERT *et al.*, 2015). No entanto, em comparação com a regra mais violada, ela é numericamente muito menor, pois afeta apenas 2,38% do tratamento implementado no sistema XSA.

A Figura 10 mostra a distribuição dos arquivos afetados com *anti-patterns* na versão 2017. A camada de visualização e a camada de acesso a dados têm as maiores porcentagens de arquivos de código com violações, com 28,2% e 29%, respectivamente. A camada de visualização tem um papel essencial para evitar a apresentação de problemas internos do sistema

para os usuários finais. Defeitos do tratamento de exceção na camada de visualização podem afetar negativamente esse papel. Esse diagnóstico confirmou a respostas fornecidas pelos desenvolvedores no primeiro formulário aplicado (Figura 9 - Itens A7 e A8), sendo um indicativo de funcionamento inadequado do tratamento de exceção nessa camada, o que pode expor erros internos ao usuário final.

Figura 10 – Arquivos que contém *anti-patterns*



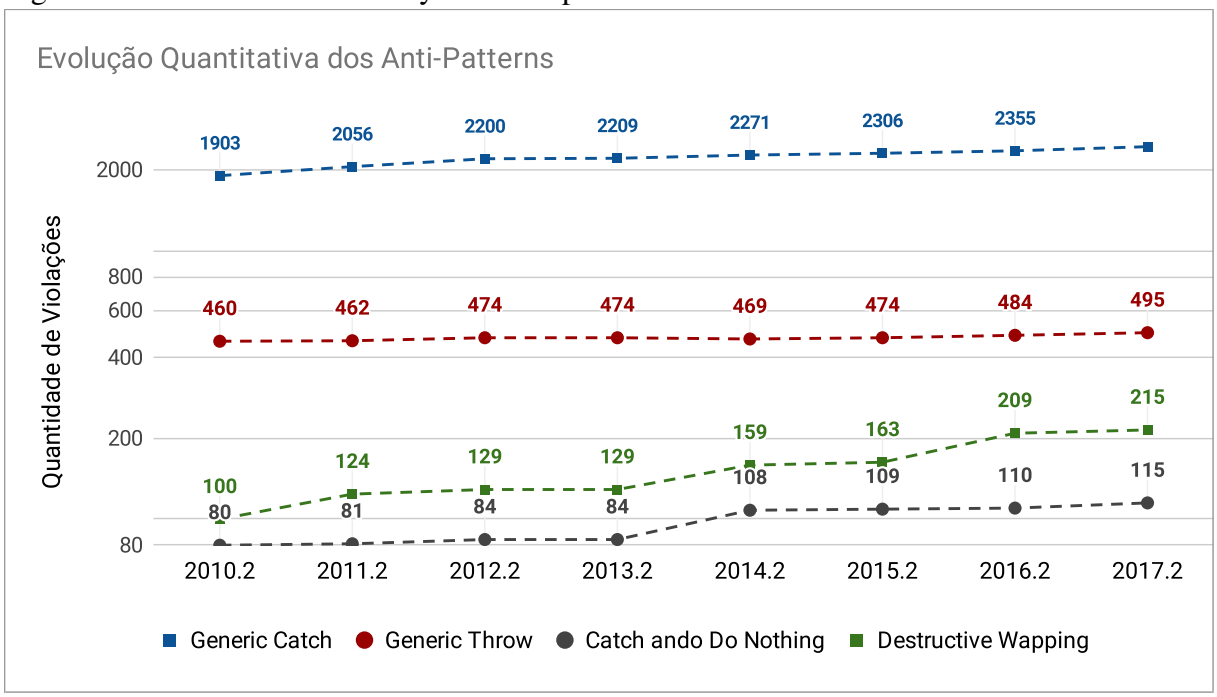
Fonte: o autor.

5.2.2.2 *Evolução dos Anti-Patterns*

Após a análise estática do código-fonte, percebeu-se um aumento do número de classes e de LOC entre 2010 e 2017. Esse resultado era esperado já que o sistema XSA está em constante manutenção e customização. Em relação ao tratamento de exceção, pode-se observar as seguintes porcentagens de crescimento: 41,91% para blocos de manuseio implementados e 38,64% para lançamento de exceção. A expansão do código-fonte também incluiu uma proliferação de *anti-patterns* (veja a Figura 11). Por exemplo, o *anti-pattern* Catch and Do Nothing tem 80 violações na versão de 2010 e 115 na versão de 2017, representando um aumento de 43,75 %. O *anti-pattern* Destructive Wrapping cresceu 115% entre os *releases* de 2010 e 2017, tendo o número de violações aumentado de 100 para 215.

Analisar apenas o número de violações pode resultar em erro de interpretação do fenômeno, uma vez que o tamanho do XSA aumentou consideravelmente. Dessa forma, algumas proporções entre o número de regras violadas e as métricas de código foram estudadas (por

Figura 11 – Evolution of XSA system anti-patterns



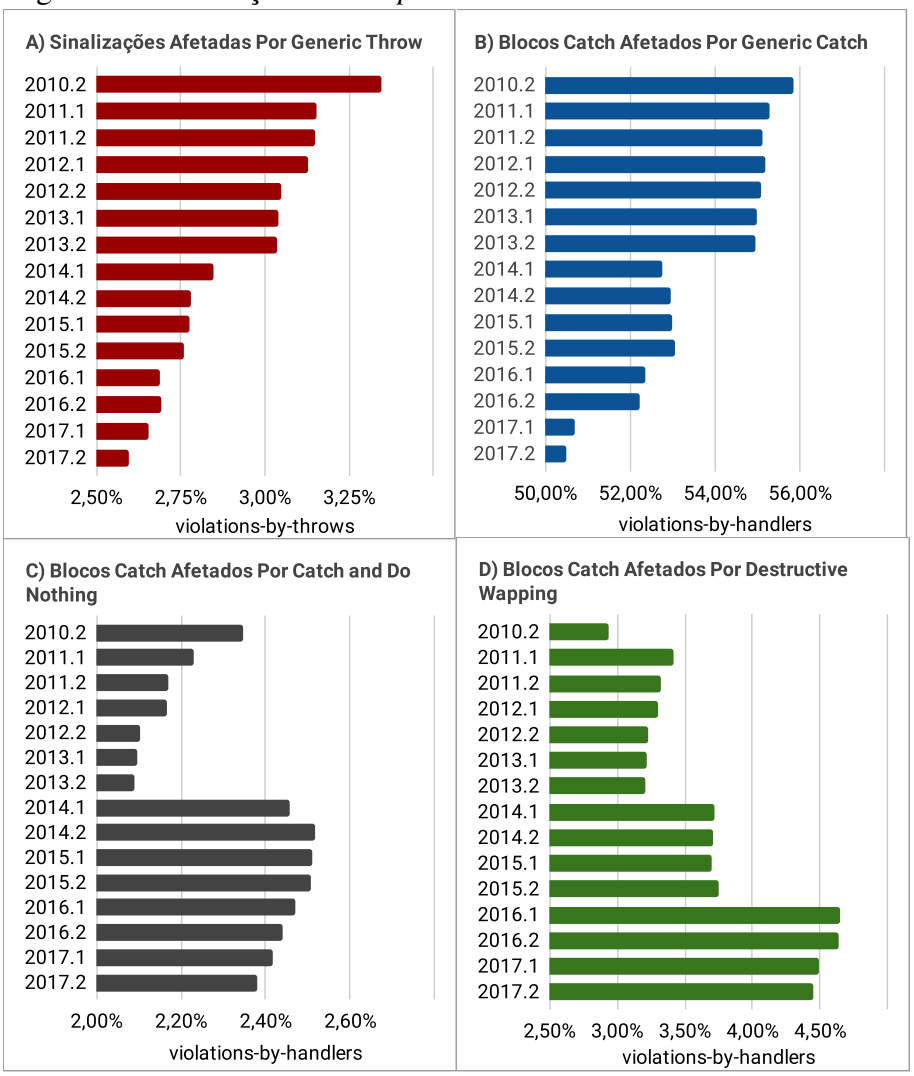
Fonte: o autor.

exemplo, número de classes, número de tratamentos). O *violations-by-handlers* é a razão entre o número de violações de um *anti-pattern* por número de tratamentos implementados no XSA. Similarmente, a razão *violations-by-throws* é calculada dividindo-se o número de violações por número de sinalizações. Finalmente, *violations-by-classes* é obtido pela proporção do número de violações pelo número total de classes do XSA.

A Figura 12 mostra a evolução dessas proporções. Observa-se um aumento das proporções referentes aos *anti-patterns* Catch e do Nothing e Destructive Wrapping, especialmente entre 2014 e 2017. O decréscimo dos *anti-patterns* mais recorrentes no sistema (Catch Generic e Throws Generic) é um fator positivo. Uma causa possível é a adoção de práticas, como a implementação de tratamentos especializados e a sinalização de exceções específicas, que contribuíram para o desenvolvimento adequado da atividade de recuperação de erros. No entanto, à medida que outros *anti-patterns* continuam a crescer na XSA (mesmo em suas proporções), não se pode confirmar que essa melhoria é devida à adoção de algumas políticas, refatoração de código ou ferramentas de inspeção.

A Figura 13 (B) exibe um aumento absoluto no número de *anti-patterns* detectados ao longo dos anos para cada camada. Essa informação revela que os desenvolvedores não refatoraram o código suficientemente, o que indica a existência de uma dívida técnica. A camada de apresentação continua sendo aquela com mais violações, seguida pelas camadas

Figura 12 – Evolução de *anti-patterns* no sistema XSA

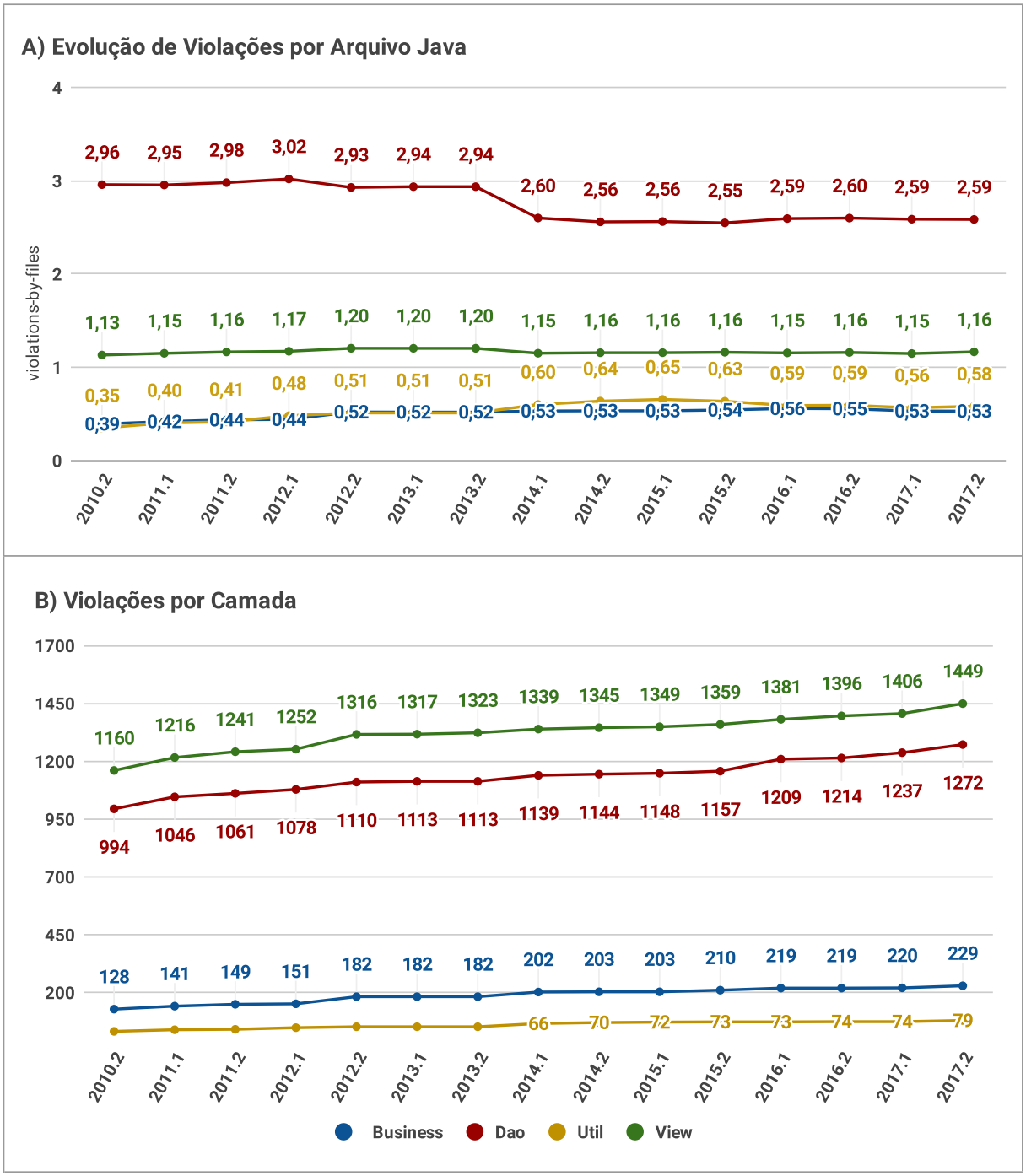


Fonte: o autor.

de acesso a dados e de negócios. Agrupando todas as violações nas quinze versões, obteve-se que o tratamento genérico de exceções e a sinalização de exceções genéricas correspondem, respectivamente, a 61.86% (12.279) e 32.42% (6.437) das violações na camada de visualização. Na camada de acesso a dados, o tratamento genérico de exceções é responsável por 93,50% (15.929) das violações.

Figura 13 (A) mostra a proporção do número de violações para o número de arquivos Java em cada camada. Não há crescimento constante nessas taxas para todas as camadas. De fato, a camada de dados demonstra uma diminuição de 12,61% nessa proporção (de 2,95% para 2,58% na última versão). Observou-se um comportamento diferente para as outras camadas, já que houve um aumento nessa taxa: apresentação (3,02%), negócio (35,11%) e utilitária (67%).

Figura 13 – Evolução de *anti-patterns* por camada



Fonte: o autor.

5.3 Unidade de Análise 3: Investigação Sobre o Tratamento de Exceção

A Unidade de Análise 03 foi elaborada no decorrer do estudo de caso com a função de responder os questionamentos que surgiram com a análise dos primeiros dados (Unidades de Análise 01 e 02). Esses questionamentos eram necessários para compreender melhor o contexto em que o estudo de caso estava inserido, relacionando elementos como equipe de desenvolvimento, código e documentação. As subseções a seguir mostram como os questionamentos

abordados na unidade foram concebidos e os procedimentos necessários para respondê-los.

5.3.1 Preparação da Unidade: Entrevista Semi-Estruturada

Uma entrevista semiestruturada foi conduzida com três funcionários experientes do sistema XSA. Os entrevistados assumiram diversas funções desde a implantação do sistema na instituição, como analista de software, gerente de projetos, e lider da equipe de desenvolvimento. A entrevista ocorreu durante uma única reunião entre os entrevistados e os pesquisadores. Ela foi gravada e transcrita para posterior análise. A entrevista teve como objetivo proporcionar um melhor entendimento sobre os resultados obtidos no estudo. Assim como, identificar as causas motivadoras e o contexto que produziram esses resultados no sistema XSA. A entrevista foi dividida em 04 fases:

- Fase 01: Informações sobre a pesquisa, seus objetivos e estrutura;
- Fase 02: Apresentação e discussão dos resultados obtidos com o preenchimento do questionário *online*, o primeiro método de coleta utilizado (Unidade de Análise 01);
- Fase 03: Apresentação e discussão dos resultados da coleta de dados referentes à segunda questão de pesquisa (Unidade de Análise 02);
- Fase 04: Sugestões e avaliação do formato e conteúdo da entrevista.

A entrevista foi organizada e produzida por três pesquisadores. A análise foi efetuada segundo os seis passos propostos por Creswell (2014). A transcrição da entrevista foi estruturada tendo como base o guia de entrevista. Uma re-leitura da transcrição foi feita, ao mesmo tempo em que o texto era marcado com códigos que referenciavam o conteúdo das sentenças ou parágrafo, como tratamento de exceção, experiência do desenvolvedor, ciclo de desenvolvimento. Em seguida, os códigos criados foram agrupados em categorias, um total de 27 códigos foram classificados em 6 categorias. Essas categorias foram descritas e relacionadas com a literatura. Os resultados dessa análise são apresentadas nas próximas subseções.

5.3.1.1 Resultados

A entrevista apresentou um papel fundamental para o entendimento do contexto motivador dos resultados encontrados durante a condução deste estudo de caso. Entre eles, a falta de documentação é citada como um dos grandes problemas enfrentados pelos desenvolvedores e uma das causas de *anti-patterns* no sistema. Sem ela, não é possível ter um conhecimento sobre a arquitetura e as exceções customizadas que fazem parte do fluxo de execução, sendo

essas exceções subutilizadas. Os entrevistados acreditam que dentro da equipe não há uma visão clara sobre o uso desse tipo de exceção e nem em quais situações elas devem ser aplicadas. E concluem que essa falta de conhecimento é um indicativo de negligência com o tratamento de exceção adequado.

Não existem políticas para o tratamento de exceção na instituição. O processo de desenvolvimento dos fluxos excepcionais é visto como informal e empírico, na qual a resolução do problema se sobrepõe a maneira como ele é resolvido. Na literatura, essa atitude reativa é encontrada quando o tratamento de exceção é classificado como uma atividade de prioridade baixa (SHAH *et al.*, 2010). Outras políticas de código e processos de software já foram debatidas e adotadas, buscando uma melhoria da qualidade. Mas o tratamento de exceção nunca foi visto como um problema a ser resolvido.

Segundo os entrevistados, a falta de experiência dos desenvolvedores iniciantes impacta diretamente na qualidade do tratamento implementado. Frequentemente, os funcionários iniciantes são pessoas que acabaram de terminar a graduação e estão no primeiro emprego. Portanto, eles não tem vivência no desenvolvimento de sistemas de grande porte e com o tratamento de exceção para esses sistemas. Os entrevistados listaram alguns comportamentos adotados por esse perfil de desenvolvedores, como:

- Exibem a exceção na tela do sistema para mostrar a ocorrência de uma situação anormal;
- Acham mais importante exibir informações sobre a exceção do que buscar um tratamento adequado;
- Replicam práticas existentes no código, mesmo as que não são adequadas, como *anti-patterns* do tratamento de exceção;
- Não sabem quais ações implementar em um tratamento e optam por lançar exceções gerais;
- Dificultam o entendimento do código com tratamentos aninhados e, às vezes, sem ações específicas; e
- Geralmente, estão mais preocupados em fazer o código executar. Por isso, implementam os tratamentos obrigados pela linguagem sem pensar em ações de recuperação.

A existência de *bugs* do tratamento de exceção no sistema foi identificada logo após a apresentação do conceito. Espontaneamente, os entrevistados elencaram os seguintes *bugs/anti-patterns*: falta de documentação, propagação indevida, lançamento de exceções genéricas, dummy handler, tratamentos vazios e exceções não tratadas. No entanto, os quantitativos encontrados durante a pesquisa e apresentados durante a entrevista foram considerados elevados

por parte dos entrevistados. O esperado era a estabilização dos problemas ou um crescimento modesto. Um pensamento contrário também foi apresentado por um entrevistado, demonstrado que o valor elevado de transgressões era esperado, pois elas não foram tratadas, sendo apenas reproduzidas ao longo dos anos.

Os entrevistados conectaram a evolução dos problemas do tratamento de exceção com a replicação de erros já existentes no código original. Como a documentação formal não é completa, os desenvolvedores comumente se orientam pelo próprio código e tornam-se replicadores dos comportamentos existentes. Um outro aspecto citado pelos entrevistados é o processo que guia a importação de códigos durante a implantação de novos módulos. Ele é entendido como um possível transportador de más práticas, pois consiste na adequação de códigos vindos de um ambiente de desenvolvimento externo.

Um impacto da má implementação do tratamento de exceção, percebido pelos entrevistados, é a dificuldade de encontrar a causa do erro original. A presença do *anti-pattern* Destructive Wrapping no sistema pode ser a causa desse problema. Além disso, a complexidade para resolver os *bugs* que aparecem durante a execução do sistema aumenta, já que as informações úteis permanecem entre as diversas linhas dos arquivos de *log*. Mais esforço acaba sendo empregado na busca do erro original do que na resolução do problema causador da exceção. Essa situação é acentuada quando informações incompletas e sem utilidade são gravadas nos arquivos de *log*. Ademais, o sistema não tem um comportamento claro sobre as exceções não tratadas e os desenvolvedores não entendem a estratégia adotada por ele. Algumas vezes, as exceções acabam sendo repassadas ao usuário final em alguns momentos.

Por fim, a adoção de ferramentas que auxiliam nas atividades cotidianas dos desenvolvedores já foi abordada na instituição. Foi informado que os desenvolvedores, normalmente, não fazem objeções quanto ao uso novas ferramentas. No entanto, nenhuma análise profunda foi concluída, além de não existirem funcionários responsáveis pela melhoria e inspeção da qualidade do código. Apesar do crescimento da equipe, não há pessoas suficientes para que uma equipe de qualidade seja organizada dentro da instituição.

5.3.1.2 Ações Após a Entrevista

As informações reveladas com a entrevista expuseram situações que poderiam ser a causa dos problemas do tratamento de exceção do sistema XSA. Portanto, essas situações deveriam ser investigadas de modo a confirmar a sua relação com as resultados apresentados

na Unidade de Análise 02. Os pontos-chave a serem investigados, e possíveis causadores de *anti-patterns*, eram:

- A rotatividade da equipe de desenvolvimento;
- A inexperiência dos desenvolvedores iniciantes;
- A replicação de más práticas do sistema original; e
- A falta de documentação.

Essas questões são essenciais para responder o Questionamento 01, apresentado na Subseção 4.4.3, "Qual é a motivação para a ocorrência dos *anti-patterns*?". Elas também subsidiaram a formulação das possíveis respostas (suposições) exibidas na mesma subseção. O estudo dessas suposições é detalhado nas subseções a seguir.

5.3.2 Rotatividade da Equipe

Suposição 01: A alta rotatividade de desenvolvedores dificulta que os profissionais contratados se tornassem experientes no código e, consequentemente, no correto tratamento de exceção exigido para o sistema.

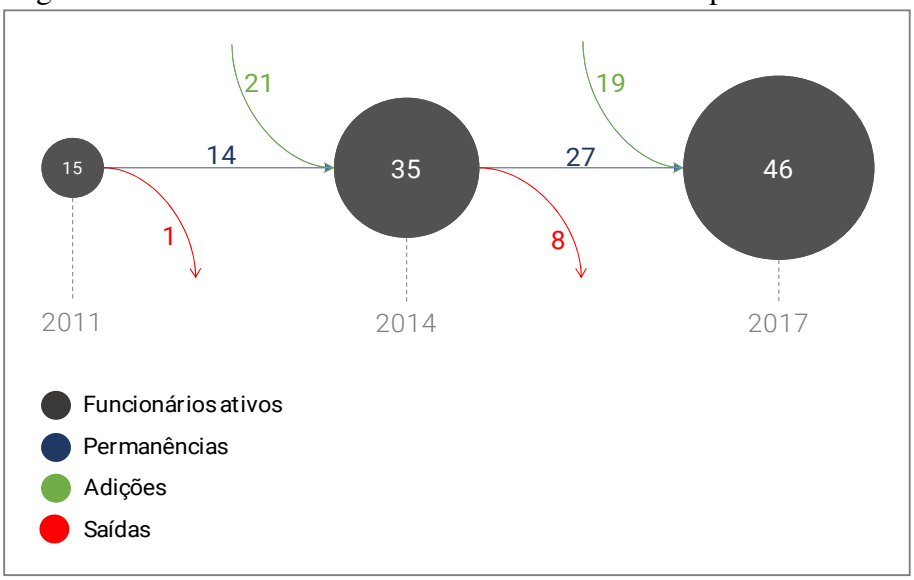
5.3.2.1 Procedimento

A suposição tem a rotatividade da equipe como parte de central da questão. Portanto, era necessário obter dados sobre a contratação e a saída de membros da equipe de desenvolvimento. O sistema que gerencia os recursos humanos da equipe, detentor dos dados sobre a rotatividade, possibilitou entender como ocorre esse fenômeno. A partir dele, foi obtida a listagem de funcionários, com a data de admissão e retirada, desde 2010.

5.3.2.2 Resultados

A Figura 14 e a Figura 15 exibem o resumo dos dados relacionados à rotatividade da equipe. Em 2011, a instituição iniciou o processo de adaptação do sistema XSA. Nessa época, a equipe tinha 15 profissionais ativos. O número de desenvolvedores cresceu 73,33% do sétimo *release* (2014.1) e 76,92% antes da versão atual de 2017. Ao longo do período analisado neste estudo de caso, a instituição contratou 43 profissionais e 12 outros mudaram de emprego. O rotatividade média por semestre foi de 14,16%, com mediana de 9,14% e desvio padrão de 16,59%.

Figura 14 – Rotatividade do time de desenvolvimento por *release*



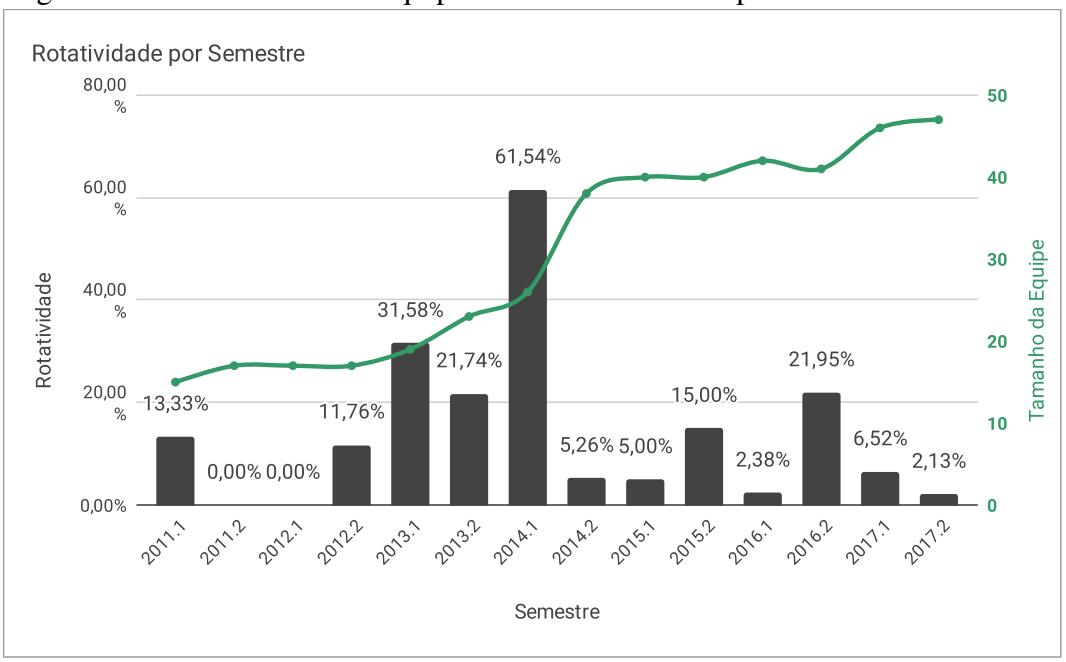
Fonte: o autor.

A instituição que mantém o XSA contrata profissionais por meio de processo de licitação pública. Esses trabalhos são atraentes para iniciantes. No entanto, eles acabam deixando a instituição por outras empresas após um curto período de tempo. Esses desenvolvedores novatos têm um nível adequado de habilidades "teóricas", o que é necessário para ter sucesso no processo de licitação pública. No entanto, eles não têm uma experiência de desenvolvimento com a linguagem Java e o próprio sistema XSA. A instituição adquiriu o sistema XSA e o desenvolveu para atender às suas necessidades específicas. No entanto, a maioria dos profissionais envolvidos na evolução da XSA foram contratados recentemente. 66,7% deles estão na instituição por um período máximo de 3 anos, que é um período após a aquisição da XSA. Portanto, o conhecimento sobre o tratamento de exceção não foi documentado adequadamente e passado para novos desenvolvedores. Segundo as opiniões da entrevista semiestruturada, esses profissionais replicam as ações já inseridas no código, mesmo aquelas que contêm *code smells*.

5.3.3 Inexperiência dos Desenvolvedores Iniciantes

Suposição 02: Desenvolvedores menos experientes são responsáveis pela inserção da maioria dos anti-patterns, pesquisas anteriores destacaram que eles não veem o tratamento de exceção como uma atividade prioritária.

Figura 15 – Rotatividade da equipe de desenvolvimento por semestre



Fonte: o autor.

5.3.3.1 Procedimento

Para analisar o impacto dos desenvolvedores iniciantes na inserção de *anti-patterns*, foram analisados os dois anos que continham o número mais substancial de profissionais recém-contratados pela instituição, em 2015 e 2017. Foram considerados dois grupos de desenvolvedores, iniciantes e experientes nessa análise. Então, foram computadas todas as mudanças incluídas no sistema e calculado o quantitativo de *anti-patterns* adicionados por cada profissional. Para isso, a aplicação anteriormente construída durante a condução da Unidade 02 foi incrementada. Considerou-se iniciantes os desenvolvedores com menos de três anos de experiência na instituição; os experientes são aqueles que têm mais de três anos de trabalho na instituição.

5.3.3.2 Resultados

Após a análise, confirmou-se que os iniciantes inseriram mais *anti-patterns* do tratamento de exceção do que os experientes. No entanto, nota-se que os recém-chegados também modificaram mais arquivos Java, o que significa que eles eram mais propensos a erros. A Figura 16 mostra uma comparação desses valores absolutos e a quantidade de violações inseridas em arquivos Java alterados em 2015 e 2017.

Em 2015, 14 pessoas alteraram os arquivos Java do sistema. O grupo de experientes

Figura 16 – Comparação entre iniciantes e experientes

Fonte: o autor.

(7 pessoas) acrescentou 20 violações. O grupo de iniciantes (7 pessoas) naquele ano implementou 46 violações. No entanto, a maioria das violações inseridas foi feita por alguns dos novatos. De fato, dois deles não adicionaram nenhuma violação. Além disso, a taxa média de violações inseridas nos arquivos Java alterados é muito próxima entre os dois grupos (box-plot da Figura 17). Quando testes de significância foram realizados, observou-se que essas diferenças não são estatisticamente significantes (Mann-Whitney $U = 22$, $n_1 = 7$, $n_2 = 7$, $p < 0.05$ two-tailed).

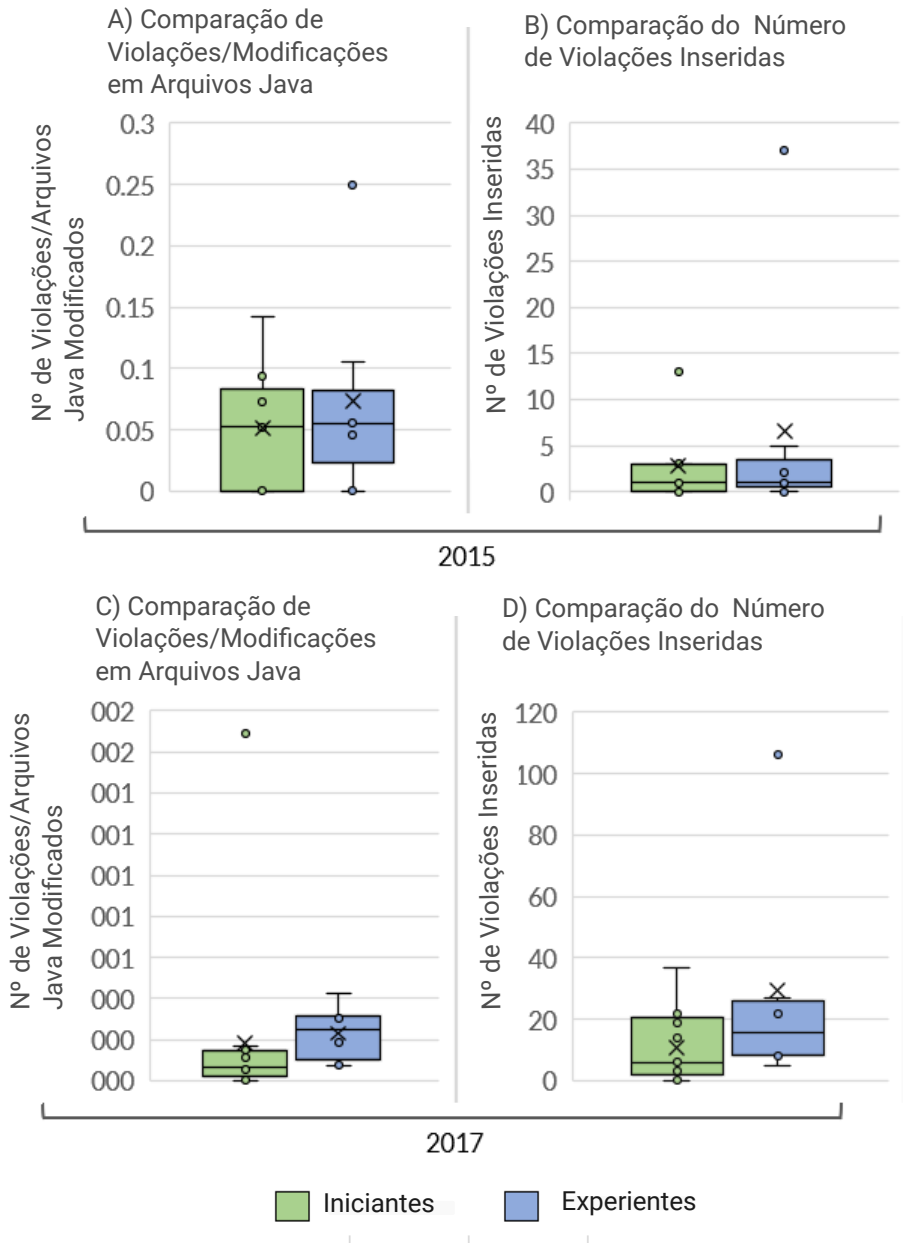
Em 2017, 20 pessoas alteraram os arquivos Java do sistema. O grupo de experientes, agora composto por 14 pessoas, acrescentou 140 violações. O grupo de novatos, composto por 6 pessoas, incluiu 177 violações. Um valor muito alto, dadas as diferenças de tamanho entre os grupos. Outro fato interessante é que os experientes removeram 56 violações e os novatos apenas 12. A taxa média de violação inserida em arquivos Java é maior no grupo iniciante (0,23) do que no grupo experiente (0,078). Os testes de significância apontaram que neste caso essas diferenças são estatisticamente significantes (Mann-Whitney $U = 12$, $n_1 = 20$, $n_2 = 6$, $p < 0.05$ two-tailed).

5.3.4 Ciência da Inserção de Anti-Patterns

Suposição 03: Os desenvolvedores não sabem que estão inserindo anti-patterns no código, pois não conhecem essas estruturas.

Para colher dados que clarificassem a veracidade dessa suposição, elaborou-se um segundo questionário. O instrumento *online* tinha como finalidade perceber se os desenvolvedores

Figura 17 – Comparação entre novatos e experientes - Inserção de violações e alteração de arquivos



Fonte: o autor.

do sistema XSA conhecem os *anti-patterns* mais recorrentes no código-fonte que implementam.

5.3.4.1 Procedimento

O instrumento foi desenvolvido seguindo algumas recomendações apresentadas por Shull *et al.* (2007). Entre elas, a elaboração do questionário mediante os seguinte passos:

- Busca na literatura;
- Construção do instrumento;

- Avaliação do instrumento;
- Documentação do instrumento.

Na busca na literatura, foi encontrado o estudo de Palomba *et al.* (2014) que serviu como base para a construção desse instrumento. Principalmente, porque a proposta de Palomba *et al.* (2014) também tem como objetivo investigar a percepção de desenvolvedores sobre a relação entre *bad smells* e o planejamento ou implementação de baixa qualidade. Em sua pesquisa, Palomba *et al.* (2014) faz uso de um questionário *online* como ferramenta de coleta de dados.

A elaboração do instrumento teve como etapa essencial a seleção de códigos que representassem a utilização dos *anti-patterns*. Os *anti-patterns* mais recorrentes no sistema XSA foram contemplados por essa busca. Além disso, também foram incluídos aqueles relacionados ao uso de exceções genéricas, que são Catch Generic, Destructive Wrapping, Throws Generic e Wrong Exception Thrown. Após a seleção de códigos, foi elaborada a primeira proposta do formulário. Questões para a identificação do perfil dos respondentes também foram inseridas, como experiência e a avaliação do próprio questionário.

Dois pesquisadores da área de Engenharia de Software auxiliaram na avaliação do instrumento. Eles responderam as questões e forneceram *feedbacks* sobre o formulário quanto à clareza das questões e ao formato do questionário. A partir das avaliações, novas versões do mesmo foram geradas. Por fim, antes do envio aos desenvolvedores, um teste piloto foi conduzido com um desenvolvedor da instituição para saber se o instrumento estava adequado ao público-alvo.

A última etapa foi a documentação do instrumento e a elaboração da mensagem introdutória que seria enviada aos desenvolvedores solicitando a participação deles no preenchimento do instrumento. A mensagem foi elaborada buscando explicitar o propósito da pesquisa e do instrumento, salientar o papel do respondente, como responder as questões, e o tempo real estimado para o preenchimento, que foi fornecido anteriormente pelos avaliadores.

Para preencher o questionário, o respondente deveria analisar o trecho de código da questão e informar se encontrou algum problema de implementação ou *design* do tratamento de exceção (Figura 18 - Parte A). Se ele encontrasse algum problema, seria encaminhado para uma página em que ele deveria descrever os problemas encontrados (Figura 18 - Parte B). Caso contrário, seria encaminhado para a próxima questão, que conteria outro código a ser avaliado. Dessa forma, seria possível identificar se os desenvolvedores conseguiam perceber os *anti-patterns* nos códigos do questionário. O instrumento é exibido integralmente no Apêndice C

Figura 18 – Exemplo de questão do Formulário 02: Aplicação do tratamento de exceção

Parte A da Questão

Em sua opinião, o seguinte trecho de código apresenta problemas na implementação do tratamento de exceção (Ex: no design, na codificação, propensão a falhas):

```
try{
    if(con != null) con.close();
}catch(SQLException sqlEx2){
    throw new DAOException(sqlEx2.getMessage());
}
```

○ Sim, apresenta
○ Não, não apresenta

Parte B da Questão

Informe os problemas do tratamento de exceção que você identificou:

Resposta

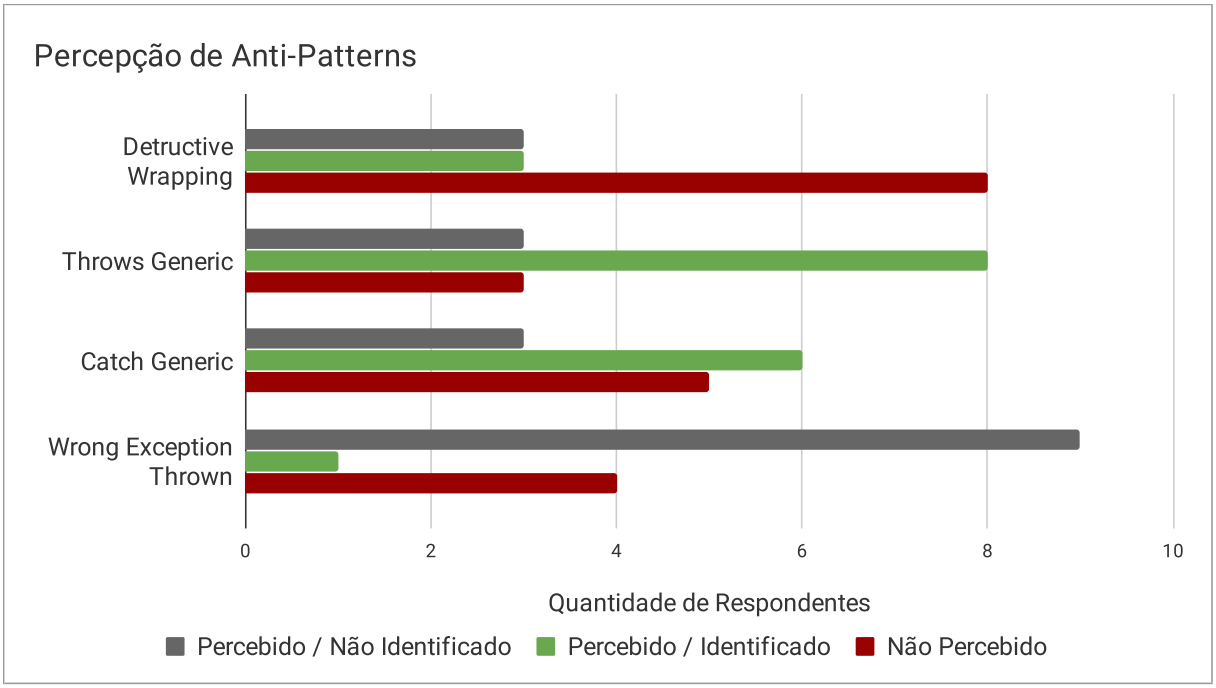
Fonte: o autor.

5.3.4.2 Resultados

O formulário *online* recebeu 14 respostas (30,43% dos profissionais envolvidos no desenvolvimento do sistema). Durante a análise, assim como no trabalho de Palomba *et al.* (2014), foram separados os respondentes que conseguiram perceber a existência de problemas do tratamento de exceção (primeira etapa da questão) e os desenvolvedores que conseguiram identificar o *anti-pattern* (segunda etapa da questão). Esses resultados são vistos na Figura 19. Os *anti-patterns* menos percebidos pelos desenvolvedores foi o Destructive Wrapping, apenas 42,86% dos respondentes. Consequentemente, apenas 21,43% conseguiram identificá-lo. Já o *anti-pattern* menos identificado foi Wrong Exception Thrown, somente 1 respondente conseguiu reconhecê-lo. Quase metade dos respondentes identificaram Catch Generic e Throws Generic, cerca de 42,86% e 57,14%, respectivamente.

Após a separação inicial, os desenvolvedores foram classificados em iniciantes (menos de três anos) e experientes (três ou mais anos) de acordo com o tempo de experiência na instituição. O objetivo dessa classificação era verificar se o tempo na instituição poderia causar

Figura 19 – Percepção de *anti-patterns* pelos desenvolvedores



Fonte: o autor.

alguma influência no reconhecimento de *anti-patterns*, visto que o código usado nas questões era muito similar ao encontrado no sistema XSA. Em média, os *anti-patterns* foram identificados em 42,86% das respostas dos veteranos e em 21,43% das respostas dos iniciantes. Essa maior identificação dos veteranos é perceptível visualmente na Figura 20.

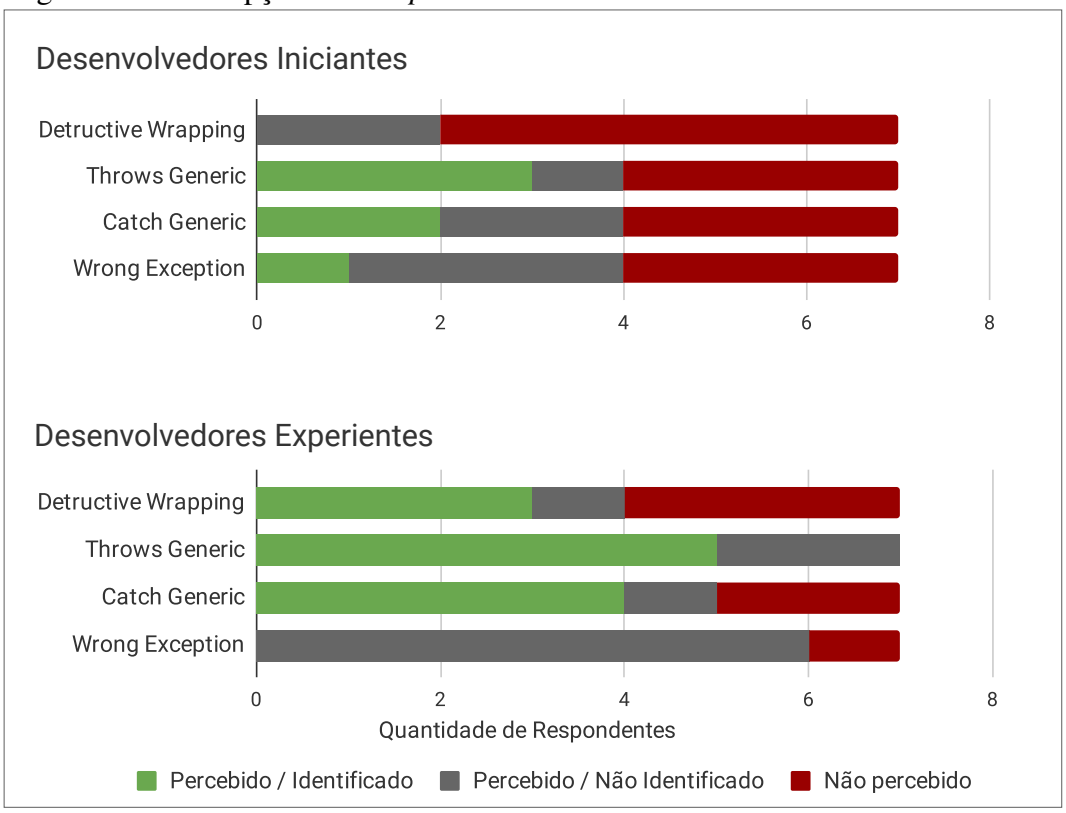
5.3.5 Falta de Documentação

Suposição 04: Os desenvolvedores não sabem como devem lidar com o tratamento de exceção ao longo das camadas do sistema porque faltam políticas que descrevam esse uso, o que propicia a inserção de anti-patterns.

5.3.5.1 Procedimento

A documentação do sistema XSA foi analisada para confirmar ou refutar as declarações dos desenvolvedores sobre a falta de políticas e guias do tratamento de exceção. Corporation (2009) define um conjunto de passos de *design* para a construção de uma estratégia para o gerenciamento de exceções. Esses passos foram usados como base para compreender a metodologia adotada para exceções no XSA. Principalmente, para averiguar se a formalização dessas decisões estava disponível aos desenvolvedores, viabilizando futuras modificações e adições.

Figura 20 – Percepção de *anti-patterns* de iniciantes e veteranos



Fonte: o autor.

5.3.5.2 Resultados

A documentação do XSA está disponível na Wiki do sistema de gerenciamento de atividades utilizado pela equipe de desenvolvimento. Ela possui duas páginas dedicadas ao tratamento de exceção. Uma página contém informações para o tratamento na camada de visão e a outra possui informações gerais, como a listagem de algumas exceções customizadas e a organização hierárquica dessas entidades. As análises foram feitas a partir das duas páginas encontradas.

A descrição dos passos de *design*, assim como a maneira como são descritos na documentação do sistema XSA, são exibidas na Tabela 10. Por meio dela, pode-se perceber a adequação da documentação somente aos Passo 4 e 7. A documentação é incompleta ou inexistente para os demais passos de *design* propostos por Corporation (2009).

Vale ressaltar ainda o fato de que a não utilização dessa documentação pelos desenvolvedores pode se dar pelas datas das suas últimas modificações. Elas remontam a maio e novembro de 2015. Consequentemente, pode haver uma não correspondência dessa documentação com o código-fonte, que segue em constante evolução.

Tabela 10 – Análise da Documentação seguindo Corporation (2009)

Passo	Descrição	Documentação XSA
1	Identificação das exceções que deverão ser tratadas na aplicação. Existem exceções para expressar uma variedade de situações, sendo importante saber quais delas precisam de tratamento.	Apresenta um conjunto de exceções que podem ser lançadas a partir dos módulos do sistema do XSA. No entanto, enfatiza apenas o tratamento de exceções dedicadas às violações negociais.
2	Situações que as exceções devem ser capturadas. No entanto, essa captura exige que alguma ação seja efetuada, o que evita a inclusão de *anti-patterns* no sistema, como Catch and Do Nothing e Dummy Handler.	Não contém informações sobre situações específicas para o tratamento das exceções próprias da linguagem e das exceções customizadas.
3	Determinar como e quais exceções serão propagadas entre as camadas que compõem o software. Portanto, definir a maneira que as exceções fluirão por toda a aplicação.	Informa que a exceção customizada DataException encapsula todas as exceções de acesso à dados. Portanto, entende-se que deve haver o remapeamento de exceções de baixo nível uma de nível mais alto, que fluirá para o restante da aplicação. Apesar disso, não foram encontrados exemplos ou instruções mais precisas de como realizar essa tarefa.
4	Definição das exceções customizadas, que são aquelas que caracterizam as situações anormais específicas do sistema desenvolvido.	A hierarquia de exceções customizadas está definida na documentação e é representada por um gráfico de classes UML. A nomenclatura das exceções definidas indicam a natureza do ocorrido. Por exemplo, exceções de segurança tem como exceção pai SegurançaException. Na hierarquia apresentada no gráfico, 7 exceções são divididas em 3 níveis.

(Continuação da Tabela 10)

Passo	Descrição	Documentação XSA
5	Definição das informações necessárias para tornar os dados excepcionais significativos para o público que é destinado, como usuários finais e desenvolvedores	Apenas a exceção do tipo negocial possui detalhamento quanto ao público-alvo e às informações necessárias para prover entendimento sobre a situação excepcional ocorrida. Na documentação, é informado que a exceção é destinada ao usuário final, sendo necessário que a regra negocial afetada seja explicada de forma clara.
6	Determinar mecanismos para armazenar informações dos eventos excepcionais ocorridos. Por exemplo, arquivos de texto e banco de dados.	Apesar do sistema realizar o armazenamento de eventos excepcionais em um banco de dados relacional e em arquivos, a documentação não evidencia essa informação.
7	Decidir a estratégia de notificação para que os administradores e operadores do sistema estejam cientes dos problemas ocorridos durante a execução.	Apresenta métodos que podem ser utilizados durante tratamentos para notificar a equipe de desenvolvimento por email. Assim como, exemplos para empregá-los na camada de apresentação (View).
8	Determinar como lidar com as exceções que não foram tratadas ao longo das camadas da aplicação, evitando que elas afetem os usuários finais e exponham detalhes da implementação.	O sistema apresenta estratégias para a notificação da equipe de desenvolvimento e apresentação páginas de erro quando uma exceção não é tratada e chega ao usuário final. No entanto, não foram encontradas documentações a respeito.

Fonte: o autor.

5.3.6 Replicação de Más Práticas do Sistema Original

Suposição 05: O sistema XSA, assim como outros de mesma procedência, possuem diversos anti-patterns porque eles já existiam no sistema original.

O sistema XSA foi adquirido por meio de outra instituição. Como apresentado na Unidade de Análise 02, o sistema já possuía vários *anti-patterns* originalmente. Essas más práticas podem ter sido alvo de replicações, assim como os participantes da entrevista semiestruturada

apontaram. Por isso, buscou-se investigar se ambientes originados ou customizados a partir do mesmo sistema continham problemas similares relacionados ao tratamento de exceção.

5.3.6.1 Procedimento

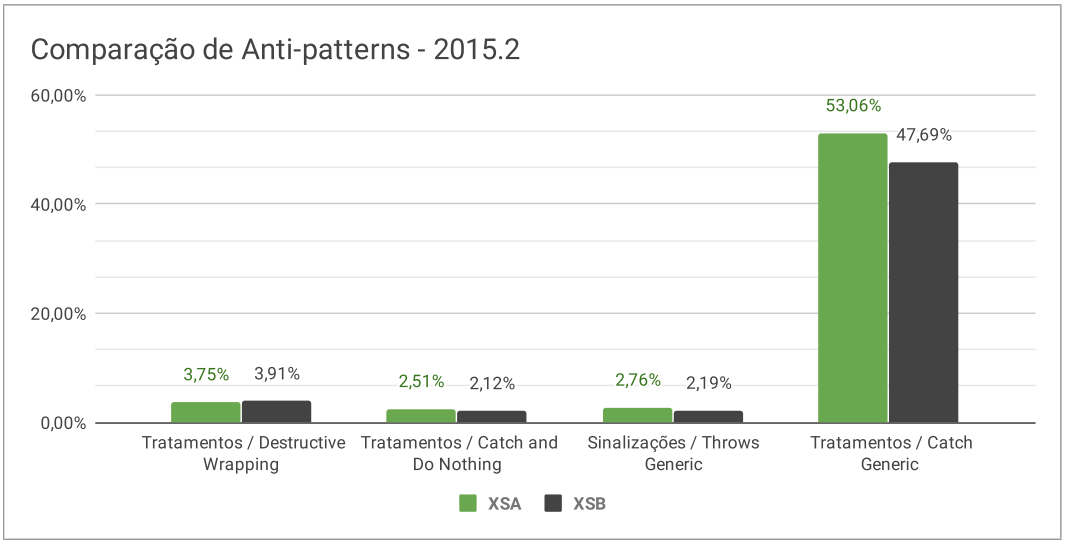
A abordagem empregada nesta análise foi a comparação do percentual de *anti-patterns* existentes no XSA com o percentual apresentado em outras versões derivadas do mesmo sistema inicial. A instituição mantedora do XSA tinha acesso ao código-fonte de um sistema semelhante, mas apenas até a versão gerada no final 2015. Por isso, a comparação ocorreu com a versão de 2015.2 do XSA. Essa comparação se deu por meio do projeto ViolationsByRepository, utilizado na Unidade de Análise 02.

5.3.6.2 Resultados

Com os dados gerados, obteve-se os resultados apresentados nas Figuras 21 e 22. O sistema utilizado na comparação foi chamado de XSB para facilitar a compreensão. Alguns percentuais foram gerados para possibilitar as comparações, que são:

- A quantidade de tratamentos pela quantidade de violações do tipo Destructive Wrapping, Catch and Do Nothing, e Catch Generic; e
- A quantidade de sinalizações pela quantidade de violações do tipo Throws Generic.

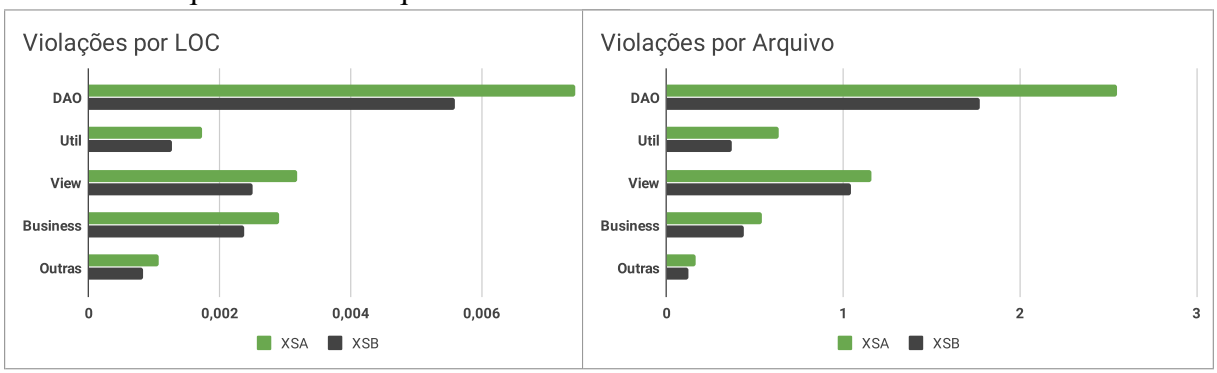
Figura 21 – Comparação entre os *anti-patterns* de sistemas semelhantes



Fonte: o autor.

Os resultados obtidos mostram que os sistemas XSA e XSB possuem percentuais próximos. Além disso, os mesmos tipos de violações são mais recorrentes nos dois sistemas.

Figura 22 – Comparação entre os *anti-patterns* de sistemas semelhantes com relação ao LOC e à quantidade de arquivos



Fonte: o autor.

Quantitativamente, o sistema XSB possui 693 violações a mais. No entanto, as divergências ocorrem porque o crescimento de código no sistema XSB é maior que o aumento de violações. Com isso, conclui-se que ao incluir novos tratamentos, os desenvolvedores responsáveis pelo XSB inserem menos violações do que a equipe responsável pelo XSA. Apensar disso, observa-se que não houve uma priorização do tratamento de exceção, visto que a quantidade de *anti-patterns* é maior que na versão original de 2010, compartilhada pelos dois sistemas.

No entanto, é necessário enfatizar que a comparação quantitativa apresentada nesta subseção não é suficiente para contrastar a manipulação do tratamento de exceção pelas equipes. Portanto, é possível que uma equipe seja mais ativa na alteração dos fluxos excepcionais. Por exemplo, modificando tratamentos nos módulos do sistema, realizando refatorações em alguns componentes, aumentando a quantidade de *anti-patterns* em outros, ou substituindo instâncias das exceções utilizadas, mesmo que isso não afete o quantitativo final observado na Figuras 21 e 22. Consequentemente, não há como perceber se os *anti-patterns* existentes na versão inicial afetaram o desenvolvimento dos sistemas ou mesmo se foram substituídos por outras instâncias dos mesmos *anti-patterns*. Uma análise histórica dos sistemas seria requisito para detectar tais práticas. Infelizmente, não foi obtido acesso ao histórico de modificações do sistema XSB. Por isso, a Suposição 05 foi verificada apenas parcialmente.

5.3.7 Exceções Que Afetam o Usuário Final

Questionamento 02: Quais são as exceções que mais afetam o usuário final do sistema alvo?

O sistema XSA contém estratégias para o armazenamento das exceções que chegam ao usuário ou são notificadas à equipe de desenvolvimento em tratamentos. Infelizmente, o

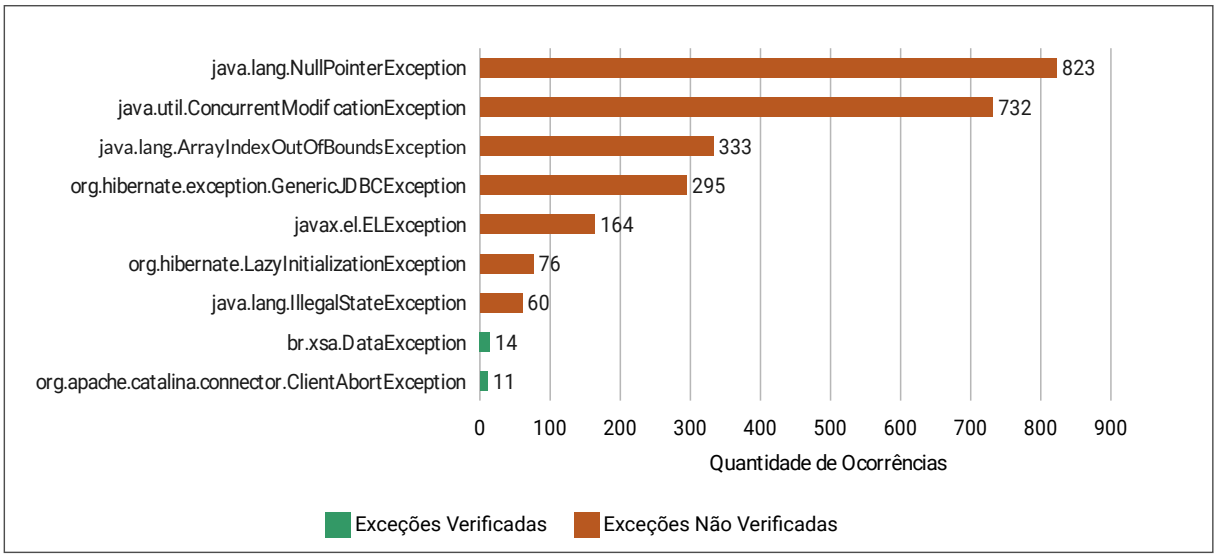
sistema não apresentava uma diferenciação entre essas duas abordagens. Por isso, foi solicitada à instituição uma pequena modificação na entidade que efetua esse armazenamento. A solicitação foi atendida, graças a isso, foi possível verificar quais são os principais problemas enfrentados pelos usuários do sistema. Assim como, perceber quais exceções são tratadas pela aplicação.

5.3.7.1 Procedimento

Relatórios foram extraídos do banco de dados em que as falhas ficam armazenadas, contemplado um período de 24 dias, entre as datas 28/09/2018 e 21/10/2018. As informações contidas nos relatórios foram separadas em exceções tratadas e não capturadas (tratadas) pela aplicação. Infelizmente, por motivações técnicas, foi possível identificar apenas exceções tratadas na camada de aplicação, pois o mapeamento das exceções tratadas em outras camadas do sistema demandaria modificações no código-fonte. No entanto, apenas 16 ocorrências não foram contempladas por esta análise. Depois foram classificadas em tipos quanto a exigência de tratamento, que são: exceções verificadas e não verificadas. Como também ordenadas pela quantidade de ocorrências.

5.3.7.2 Resultados

Figura 23 – Exceções não tratadas pela aplicação



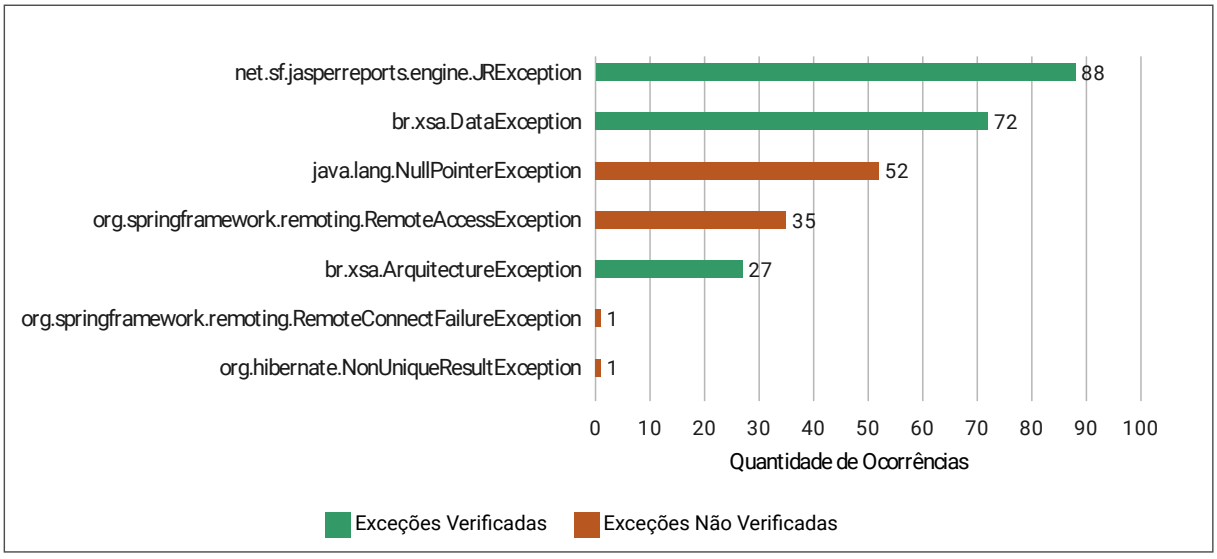
Fonte: o autor.

Durante o período observado, 2927 exceções foram registradas no sistema. 2634 não foram capturadas pela aplicação, em média, 106 por dia. As 10 exceções com as maiores

quantidade de ocorrências representam 95,22% do total registrado, 8 delas são do tipo *RuntimeException*. Em sua maioria, correspondem à exceções gerais para erros de programação relativos ao acesso indevido de objetos, a erros de consultas ao banco de dados e a problemas de conexão. A mais expressiva em quantidade é a *NullPointerException*, assim como é apresentado na Figura 23. Dentre as listadas na figura, apenas uma é customizada, *DataException*, sendo utilizada para re-mapear uma exceção do tipo *RuntimeException*, originada na camada de acesso a dados.

Apesar da quantidade de exceções não capturadas, os dados obtidos não permitem inferir a parcela do sistema XSA que origina as ocorrências. Eventos isolados, como o lançamento de uma nova funcionalidade, podem gerar diversas exceções, o que impacta diretamente na quantidade de registros de erro armazenados. As ocorrências de erro podem ainda ter seu quantitativo elevado se vários usuários requisitarem a funcionalidade defeituosa. Uma visão contextualizada do desenvolvimento do sistema possibilitaria maiores esclarecimentos. No entanto, esse tipo de intervenção e acompanhamento foge do escopo idealizado para este trabalho.

Figura 24 – Exceções tratadas pela aplicação



Fonte: o autor.

O total de 276 exceções foram tratadas pela aplicação, o que representa 9,43% das ocorrências. Dos 7 tipos de exceção encontrados, categorizados na Figura 24, 4 são *RuntimeException* relacionadas a erros de programação e falhas de comunicação com serviços remotos. As exceções verificadas equivalem a 67,75% das ocorrências. No entanto, 26,09% são fruto de re-mapeamentos de *RuntimeExceptions* que surgiram na camada de acesso a dados e não encontram tratamentos nos fluxos percorridos por elas até a camada de apresentação.

5.4 Conclusão

Neste capítulo, detalhes sobre a condução do estudo de caso foram apresentados. Para isso, listou-se procedimentos, materiais e ferramentas usados para colher e interpretar os dados necessários. Informações sobre a percepção dos desenvolvedores foram obtidas por meio de formulários online. O código-fonte do sistema XSA foi inspecionado com a finalidade de conhecer como o tratamento de exceção foi implementado, assim verificar suas deficiências e acertos. Além disso, buscou-se informações para que os dados colhidos fossem compreendidos a partir da perspectiva dos profissionais que trabalham na instituição, dos recursos utilizados diariamente por eles e do conhecimento detém sobre o tratamento de exceção implementado no sistema desenvolvido.

6 DISCUSSÃO GERAL

Neste capítulo, são discutidas as informações obtidas com a execução do estudo de caso detalhado no Capítulo 5. Por meio dessas informações, são feitas conexões com a literatura objetivando entender como está o tratamento de exceção usado no XSA em relação ao de outros sistemas anteriormente estudados. A Seção 6.1 tem foco na percepção do tratamento de exceção pelos desenvolvedores. A Seção 6.2 apresenta detalhes sobre a documentação do tratamento de exceção vista neste trabalho e em outros estudos. Os *anti-patterns*, assim como a evolução deles, são apresentados na Seção 6.2. Os efeitos da rotatividade da equipe de desenvolvimento e a inserção de *anti-patterns* são discutidos na Seção 6.4. O impacto do tratamento de exceção no usuário final é abordado na Seção 6.5. A Seção 6.6 exibe estimativas para a resolução dos problemas do tratamento de exceção encontrados no XSA. Por fim, a Seção 6.7 apresenta as conclusões deste capítulo.

6.1 Percepção dos Desenvolvedores e a Importância do Tratamento de Exceção

Considerando as respostas colhidas na Unidade de Análise 01, os profissionais que cuidam do sistema XSA julgam o tratamento de exceção necessário e até concordam que as linguagens de programação obriguem a implementação do mecanismo. Esse pensamento foi expressado até mesmo por desenvolvedores iniciantes na construção de sistemas (Tabela 8). O que difere da opinião dos entrevistados pouco experientes retratados no trabalho de Shah *et al.* (2010), já que não aprovam essa determinação imposta por algumas linguagens de programação e sentem-se forçados por elas.

No entanto, foi esclarecido que o tratamento de exceção não é visto como uma questão prioritária, principalmente se comparado com outras atividades. De fato, ferramentas não são utilizadas rotineiramente para verificar possíveis violações às boas práticas dos fluxos excepcionais. Até mesmo a documentação dedicada ao tratamento de exceção não é frequentemente atualizada desde 2015.

O aprendizado sobre o tratamento de exceção também foi abordado no questionário. Cerca de 42,8% dos respondentes discorda que aprendeu a usar o tratamento de exceção em cursos ou tranamentos. O que corrobora com as respostas fornecidas em um item similar (Apêndice B - Questão 12), em que 42,9% concorda que não foi capacitado para usar o mecanismo durante o aprendizado técnico ou acadêmico. Consequentemente, a não satisfação com a forma de lidar

com o tratamento de exceção é retratada por 47,6% dos respondentes. A falta de conhecimento sobre o uso do mecanismo torna a replicação do código existente ainda mais atrativa, mesmo que ele represente uma vulnerabilidade.

A percepção dos desenvolvedores ainda revela a crença que o tratamento de exceção precisa ser melhorado, 100% dos respondentes confirmam isso. Análises realizadas posteriormente por meio da inspeção de código e de documentação reforçam esse pensamento.

6.2 Documentação do Tratamento de Exceção

Muitos *anti-patterns* foram revelados no código-fonte do sistema, mesmo nas primeiras versões analisadas. A causa de tal problema pode ser remetida à inexistência e incompletude de diretrizes que guiam os desenvolvedores durante as atividades cotidianas. O que é confirmado pelos desenvolvedores no questionário *online* (Figura 9 - Item A13), na entrevista semiestruturada (Unidade de Análise 03) e durante análise da documentação disponível para consulta dos profissionais da instituição. De maneira similar, 80% dos respondentes brasileiros da pesquisa de Ebert *et al.* (2015) afirmam a falta dessas políticas nas organizações em que trabalham. Mesmo que essas regulamentações tenham um papel importante para influenciar na escolha de um tratamento adequado em cada contexto de código trabalhado (SHAH *et al.*, 2010).

Além disso, 61% dos respondentes da pesquisa de Ebert *et al.* (2015) indicam que nenhuma ou pouca importância é dada à documentação do tratamento de exceção fase de *design*. De uma perspectiva geral, os desenvolvedores do sistema XSA informam que não são adeptos à prática de documentar os construtos do tratamento de exceção. A documentação pobre do tratamento de exceção já é um problema conhecido e impacta no entendimento dos desenvolvedores sobre as consequências de não oferecer tratamentos adequados (YUAN *et al.*, 2014) e na ciência das exceções não verificadas lançadas por um método (SENA *et al.*, 2016).

6.3 Anti-Patterns e Sua Evolução

O *anti-pattern* Generic Catch é o mais expressivo dentre os detectados no XSA. Sua quantidade significativa é uma peculiaridade não apresentada em outros trabalhos. Pádua e Shang (2017) identificam uma média de 31,9% de tratamentos afetados pelo *anti-pattern*. Para Barbosa *et al.* (2014), a quantidade de *bugs* causados pelo problema não possui um valor significativo. O mesmo também é percebido no trabalho de Ebert *et al.* (2015), cuja classificação de *bugs*

reportados atribuí um quantitativo menor do que 2% aos tratamentos genéricos.

No estudo de (PÁDUA; SHANG, 2017), o *anti-pattern* Generic Throw também não está entre os mais recorrentes. No entanto, diversas violações desse tipo foram encontradas, representando 29,81% e 14,59% para as ferramentas Parichayana e PMD, respectivamente. Além disso, contribui com a proliferação de tratamentos genéricos, tendo como consequência o aumento da presença do *anti-pattern* no código-fonte.

Durante a evolução do sistema XSA, foi observado um aumento na quantidade de *anti-patterns*. Um exemplo disso é o *anti-pattern* Catch and Do Nothing, que possui 78 violações na *release* de 2010 e 115 na *release* de 2017, o que corresponde a um aumento de 47,43%. Nogueira *et al.* (2017), no entanto, atestam uma diminuição dessas violações comparando a primeira e a última versão dos softwares analisados por eles. Mostrando a existência de uma tendência positiva de refatoração, em que ocorre a troca dos tratamentos vazios por tratamentos especializados. Porém, essa tendência ainda não é vista no sistema XSA, pois a presença do *anti-pattern* cresce desde 2014 (Figura 12 - Item C).

Mesmo assim, a quantidade de tratamentos vazios apresentados pelo XSA com relação à quantidade total de tratamentos ainda é pequena se comparada às apresentadas no estudo de Nogueira *et al.* (2017). A maior biblioteca utilizada no estudo foi aplicada em um exemplo comparativo na Figura 25. Além disso, na Figura 25 (Item A) pode-se perceber uma diminuição na quantidade de tratamentos vazios em relação ao crescimento do software, o mesmo não é identificado na Figura 12 (Item C), mesmo que o *anti-pattern* Catch and Do Nothing apareça em menor quantidade (Figura 25 - Item B).

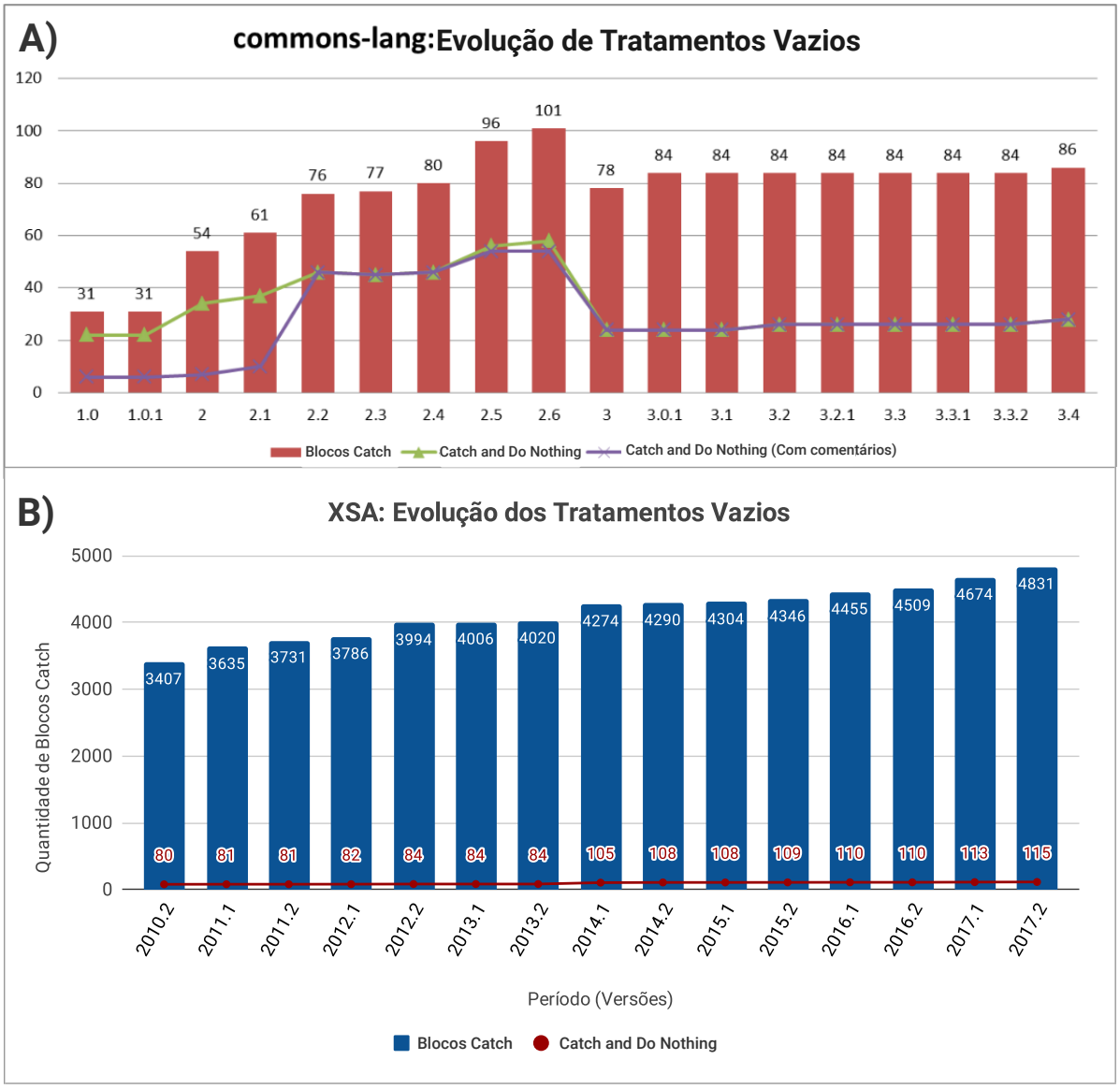
Tabela 11 – Percentuais máximos e mínimos de tratamentos ou sinalizações afetados *anti-patterns* em aplicações Java no trabalho de Pádua e Shang (2017) e no XSA

Anti-Pattern	Percentual Máximo	Percentual Mínimo	Percentual no XSA
Catch Generic	37,75%	2,26%	50,51%
Generic Throws	13,58%	0,59%	2,60%
Catch and Do Nothing	11,76%	3%	2,38%
Desctructive Wrapping	41,69%	31,62%	4,45%
Dummy Handler	10,01%	5,05%	0,08%
Throw within Finally	35,71%	3,19%	0,85%
Relying on getCause()	1,78%	0,18%	0,10%
Log and Return Null	1,7%	0,48%	6,77%
Log and Throw	0,68%	0%	4,16%

Fonte: o autor.

Por fim, a Tabela 11 foi desenvolvida objetivando a comparação dos valores cada

Figura 25 – Comparação de um dos resultados apresentados no trabalho de Nogueira *et al.* (2017) e dos dados obtidos sobre a evolução de tratamentos vazios no sistema XSA



Fonte: o autor.

anti-pattern exibido no trabalho de Pádua e Shang (2017) e os valores encontrados no sistema XSA. Os dados mostram que o sistema XSA encontra-se dentro dos limites detectados em outros sistemas. Contudo, os *anti-patterns* Catch Generic, como citado previamente, Log and Return Null e Log and Throw ainda superam esses limites. O *anti-pattern* Log and Throw dificulta a identificação da situação anormal ocorrida, já que contribui com o armazenamento desnecessário de informações nos arquivos *log*. Já o *anti-pattern* Log and Return Null mascarra a ocorrência de uma situação anormal, pois o método que detecta o erro retorna um tipo de dado geralmente usado em situações normais (*Null*).

6.4 Desenvolvedores Iniciantes e a Rotatividade da Equipe

Os desenvolvedores recém contratados são frequentes na instituição, por causa da alta taxa de rotatividade, aproximadamente 14,16% por semestre. A quantidade de desenvolvedores cresceu 31,42% entre os anos de 2014 e 2017. No ano de 2017, a diferença entre a quantidade de *anti-patterns* inseridas por desenvolvedores iniciantes é maior do que a quantidade inserida por desenvolvedores mais experientes, sendo essa diferença estatisticamente significante. A análise dos *anti-patterns* inseridos também mostraram que os desenvolvedores experientes atuam mais na remoção dessas estruturas do que os inciantes.

Esses resultados contribuem com a suspeita dos profissionais que foram entrevistados, o que indica um impacto significativo da rotatividade da equipe e a experiência dos desenvolvedores na inserção de violações. Esse comportamento de desenvolvedores novatos é semelhante ao apresentado no trabalho de Shah *et al.* (2010). Os autores identificaram que os iniciantes replicam o tratamento de exceção já estruturado no código, lançam exceções gerais e usam o tratamento de exceção sem ações de recuperação. No entanto, este trabalho de mestrado também revelou que os experientes continuam contribuindo negativamente para esse cenário.

Os experientes também apresentaram maior assertividade no reconhecimento de *anti-patterns*. A apesar disso, apenas 42,86% das repostas fornecidas pelos veteranos sobre problemas no código relativos ao tratamento de exceção identificaram os *anti-patterns* contidos nele. Portanto, não se pode afirmar que o conhecimento dessas estruturas prejudiciais é amplamente difundido nesse grupo de desenvolvedores. Os inciantes demonstram ainda um menor conhecimento do assunto, identificando os *anti-patterns* em 21,43% das respostas. Consequentemente, é provável que muitos *anti-patterns* sejam inseridos e replicados no código de maneira inconsciente.

6.5 O Usuário Final do Sistema XSA

O usuário final do sistema XSA se depara principalmente com exceções que tem origem em problemas de programação, como variáveis nulas, utilização concorrente não permitida e acesso a posições de *arrays* inexistentes. Essas exceções se apresentam na tela do sistema como uma falha genérica, exibindo alguns detalhes da implementação do sistema, como classes e pacotes.

A alta predominância de exceções do tipo *NullPointerException* no XSA é vista

também em sistemas analisados em outros trabalhos (KECHAGIA; SPINELLIS, 2014; COELHO *et al.*, 2015; KIM *et al.*, 2013). A ocorrência de diversos problemas decorrentes de defeitos de programação também é recorrente. Coelho *et al.* (2015) mostra que essa exceção está presente em 27,71% das falhas analisadas. Já no trabalho de Kim *et al.* (2013), os *stack traces* dos dois sistemas alvo apresentam prevalência de 45,36% e 38,70% desse tipo de exceção. *NullPointerException* também é a exceção mais vista nos relatórios de falha analisados por Kechagia e Spinellis (2014).

No XSA, algumas das exceções que representam erros acabaram sendo tratadas pela aplicação, pois são capturadas por tratamentos genéricos ao chegarem na camada de apresentação (Figura 24). A presença de *anti-patterns* como Catch and Do Nothing, Dummy Handler e Log and Return Null pode até mesmo impedir a reportação de alguns erros. Portanto, é provável que a quantidade de falhas ocorridas seja maior, já que algumas não são quantificadas por causa do silenciamento de exceções. Esse tipo de situação impede que o usuário seja notificado quando um comportamento inesperado ocorre, fazendo que a funcionalidade utilizada pareça simplesmente desabilitada. De forma semelhante, os desenvolvedores também não são notificados, consequentemente, ficam impedidos de solucionar o problema.

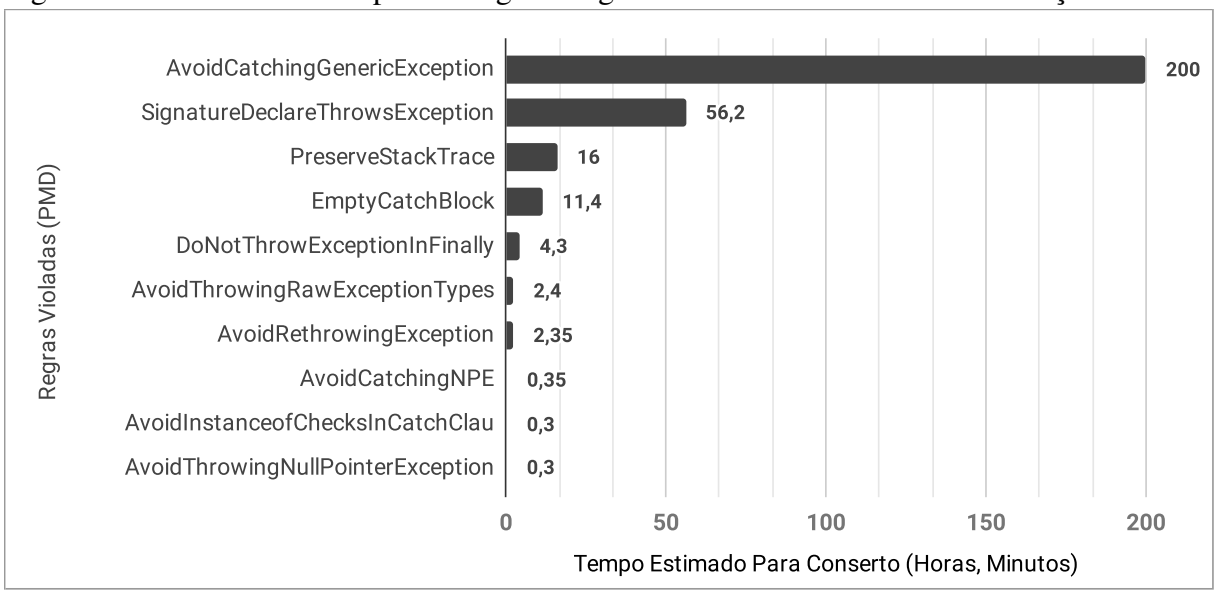
6.6 Custo Estimado Para Correção de *Anti-Patterns* do Tratamento de Exceção

Mesmo conhecendo as categorias e quantidades de *anti-patterns* encontrados no código-fonte do XSA, ainda é difícil mensurar o esforço necessário para que eles sejam corrigidos. Pensando nisso, e em fornecer um *feedback* compreensível à organização responsável pelo XSA, foi buscada uma alternativa para melhor quantificar os problemas encontrados. Baseando-se no trabalho de Digkas *et al.* (2017), foi calculado o custo estimado (em horas) para resolver os *anti-patterns* relacionados ao tratamento de exceção. Para isso, o conceito de débito técnico da ferramenta SonarQube foi empregado, que é definido como o tempo requerido para consertar todas as violações de manutenção ou *code smells*.

Foge do escopo dessa análise a comparação com outras ferramentas ou métodos que calculam débito técnico. O intuito não é validar acurácia das informações obtidas com o SonarQube, mas exibir os problemas do tratamento de exceção, apresentados ao longo deste trabalho, sob perspectiva de custos para a equipe de desenvolvimento.

Primeiramente, realizou-se o mapeamento das regras implementadas pelo PMD e pelo SonarQube, como é exemplificado na Tabela 12. O mapeamento foi desenvolvido para

Figura 26 – Custo estimado para corrigir as regras violadas do tratamento de exceção



Fonte: o autor.

obter o custo de reparo quando uma é regra violada, valor previamente definido no SonarQube, e calculá-lo para todo o sistema XSA. De acordo com a documentação da ferramenta SonarQube, diferentes funções podem ser utilizadas para calcular o tempo de reparo de uma violação. No entanto, as regras utilizadas neste trabalho possuem custo constante. O tempo necessário para resolver todas as violações de uma regra de custo constante é único por arquivo. Um *script* foi desenvolvido a partir dessas informações para calcular o débito técnico do tratatamento de exceção.

Os seguintes resultados foram encontrados:

- O custo estimado para consertar todas as violações do último *release* é de 295 horas e 20 minutos;
- A regra com o maior débito é AvoidCatchingGenericException, referente ao *anti-pattern* Catch Generic, pois tem custo estimado em 200 horas, 67,75% de todo débito técnico calculado para a versão de 2017. O custo das demais regras é apresentado na Figura 26.

O débito técnico para o primeiro *release* (2010) também foi calculado a fim de estimar o seu crescimento até 2017. Foi obtido custo 229 horas e 20 minutos, o que resulta em um crescimento de 66 horas em 7 anos (28,80%). Esse crescimento é menor se comparado a outras métricas de evolução do sistema, como 41,91% para tratamentos implementados e 42,65% para linhas de código.

Tabela 12 – Mapeamento das regras do PMD e SonarQube

Regra PMD	Regra SonarQube	*Anti-Pattern*
Do Not Throw Exception In Finally	Exceptions should not be thrown in finally blocks	Throw withing Finally
Avoid Instance of Checks In Catch Clause	Exception types should not be tested using "instanceof"in catch blocks	Relying on getCause()
Avoid Throwing Raw Exception Types	Generic exceptions should never be thrown	Wrong exception thrown
Preserve StackTrace	Exception handlers should preserve the original exceptions	Destructive Wrapping
Avoid Catching Generic Exception	"Exception"should not be caught when not required by called methods	Catch Generic
Empty Catch Block	Exception handlers should preserve the original exceptions	Catch and Do Nothing
Signature Declare Throws Exception	Generic exceptions should never be thrown	Throws Generic
Avoid Re throwing Exception	"catch"clauses should do more than re-throw	-
Avoid Catching NPE	"NullPointerException"should not be caught	Catch Generic
Avoid Throwing "NullPointerException"	"NullPointerException"should not be explicitly thrown	Wrong exception thrown

Fonte: o autor.

6.7 Conclusão

Este capítulo discutiu os achados desta pesquisa de mestrado. Primeiramente, a percepção dos desenvolvedores foi abordada, envolvendo a importância dada ao tratamento de exceção na instituição de trabalho e ao longo da vida acadêmica dos profissionais. A documentação e os *anti-patterns* abordados neste trabalho foram relacionados com exemplos encontrados na literatura. Além disso, discutiu-se a contribuição da rotatividade da equipe com na inserção de *anti-patterns*. Por fim, as exceções que mais impactam os usuários e o custo para corrigir os *anti-patterns* do tratamento de exceção foram destacados.

7 CONCLUSÃO

Este trabalho apresentou a análise do tratamento de exceção em um contexto real de desenvolvimento. Na Seção 7.1, este capítulo resume os resultados alcançados ao longo da pesquisa por meio da resposta de cada questão de pesquisa. A Seção 7.2 lista as ameaças à validade deste estudo. As limitações são apresentadas na Seção 7.3. As produções bibliográficas produzidas ao longo deste trabalho são destacadas na Seção 7.4. A apresentação dos trabalhos futuros na Seção 7.5 conclui este capítulo.

7.1 Resultados Alcançados

O tratamento de exceção é tipicamente associado à capacidade de software se recuperar de situações anormais. No entanto, apesar de sua importância, ele é comumente negligenciado pelos desenvolvedores, o que resulta em vários problemas. Estudos anteriores mostram que esses problemas vêm de aspectos humanos ou técnicos. Considerando esses dois aspectos, este trabalho teve como objetivo compreender como os desenvolvedores de software de uma instituição percebem o tratamento de exceção e quais as principais situações técnicas enfrentadas naquele ambiente real que podem afetar a qualidade do tratamento de exceção em um sistema particular de grande porte.

7.1.1 QP1: Quais percepções o time de desenvolvimento de um sistema de larga escala tem sobre o tratamento de exceção?

Um estudo de caso foi conduzido com um sistema da Web de larga escala. O primeiro questionário *online* aplicado mostrou que os desenvolvedores estão cientes da importância do tratamento de exceção, mesmo sem uma política explícita para regular o mecanismo na instituição. No entanto, como a análise de código revelou, *anti-patterns* foram encontrados, e eles não são uma novidade no sistema. Pode-se concluir que o reconhecimento da importância do tratamento de exceção não foi suficiente para evitar a inserção e a proliferação de *anti-patterns* no código do sistema.

Os profissionais da instituição concordam que o tratamento de exceção precisa ser aprimorado no sistema mantido. Além disso, a maioria deles acredita que deve melhorar o tratamento de exceção diariamente. Ademais, também informaram que não usam ferramentas que suportam o desenvolvimento de fluxos excepcionais e que não há uma documentação

unificada que possa guiá-los durante o desenvolvimento do sistema.

7.1.2 QP2: Quais anti-patterns do tratamento de exceção podem ser encontrados em um sistema de larga escala e como eles evoluem ao longo do tempo?

Na análise de código XSA, os mais frequentes *anti-patterns* encontrados foram o Catch Generic e o Generic Throw. Eles denotam falta de compreensão sobre quais exceções devem ser tratadas e quais devem ser lançadas nas camadas superiores do sistema. Várias anomalias de código já estavam no sistema desde suas versões iniciais e foram replicadas ao longo dos anos, crescendo junto com o sistema. No entanto, essa tendência crescente parece estar diminuindo, especialmente ao comparar a evolução do sistema com os dois *anti-patterns* mencionados anteriormente. Foi confirmado pela comparação da dívida técnica estimada para o primeiro e o último lançamento, que está crescendo menos que o próprio sistema. Infelizmente, essa tendência não se aplica a todas as anomalias encontradas no código, uma vez que algumas delas, como Catch e Do Nothing e Wrapping Destructive, continuam aumentando.

7.1.3 QP3: Quais fatores motivaram a inserção de anti-patterns no sistema?

Os estudos conduzidos na Unidade de Análise 03 possibilitaram a identificação de possíveis causas da inserção de *anti-patterns* e a investigação delas. Revelou-se que o sistema realmente apresenta uma documentação básica do tratamento de exceção, o que impede o entendimento de como o mecanismo está implementado. Os desenvolvedores, sem a apoio documental, aprendem a utilizar o mecanismo por meio do código-fonte, que já continha vários *anti-patterns* desde o início da sua customização. Essa situação induz a replicação das más práticas vistas no código. Principalmente, porque mais da metade dos desenvolvedores que participaram do estudo não reconhecem os *anti-patterns* como estruturas prejudiciais. A rotatividade de profissionais, frequente na instituição, contribui para que poucos consigam dominar o ferramental utilizado. As análises mostram que os desenvolvedores recém chegados contribuem mais com a proliferação dos *anti-patterns*, e menos com a remoção deles.

7.2 Ameaças à Validade

7.2.1 Validade Interna

A Unidade de Análise 01 considera apenas as informações fornecidas no questionário *online*. Esse método de coleta pode ter viés dependendo do interesse dos respondentes. No entanto, alguns dados foram validados a partir dos resultados da segunda e terceira unidades de análise. Por exemplo, a ocorrência de *anti-patterns* na camada de apresentação, a incompletude de políticas, e o conhecimento dos desenvolvedores sobre as más práticas do tratamento de exceção.

Outra ameaça à validade diz respeito à porcentagem de participantes das pesquisas *online*, cerca de 53% e 30,43% do público alvo. No entanto, vale salientar que nem todos os profissionais da equipe de desenvolvimento de sistemas trabalham diretamente com o planejamento e implementação de fluxos excepcionais. Além disso, esse percentual de respostas é bastante similar ao obtido em estudos correlatos da literatura (EBERT *et al.*, 2015; COELHO *et al.*, 2017; QUEIROZ; COELHO, 2016).

Este trabalho não analisa todos os aspectos relevantes para o tratamento de exceção (por exemplo, a situação em que os desenvolvedores adicionam *anti-patterns* ao código, os testes usados no sistema XSA e o treinamento recebido pelos desenvolvedores). Também não inspecionamos todos os *anti-patterns* existentes. O estudo de caso é limitado pelas ferramentas de análise adotadas no estudo. No entanto, essas limitações não impediram o alcance dos objetivos. Foi possível entender o estado de implementação do tratamento de exceção em um sistema real, apresentando suas vulnerabilidades e suas possíveis causas a partir das percepções dos profissionais envolvidos no desenvolvimento dele.

7.2.2 Validade Externa

Como estudo de caso, esta pesquisa se encaixa em um contexto particular. A seleção dos participantes do estudo não foi aleatória. Eles trabalham no mesmo ambiente de desenvolvimento e compartilham experiências (treinamento, documentação, código e dificuldades de trabalho). Esse contexto dificulta a generalização dos resultados obtidos por outras instituições públicas e empresas que desenvolvem software. Por exemplo, organizações com mais regulamentação e inspeção de código, ou aquelas que usam outras linguagens de programação para desenvolvimento de aplicativos da Web, podem produzir resultados distintos.

7.2.3 Validade de Construção

Os formulários *online*, respondidos sem o auxílio dos condutores da pesquisa, podem não ser completamente entendidos pelos respondentes. Para evitar essa ameaça foram feitos testes preliminares com pesquisadores e profissionais da área de desenvolvimento e análise de sistemas, que ajudaram a testar os instrumentos quanto a organização e as questões.

O projeto Java usado durante as análises evolutivas e de rotatividade foi desenvolvido para suprir às necessidades deste estudo de caso. Portanto, foi desenvolvido e testado pelos condutores da pesquisa. No entanto, o projeto usa uma ferramenta externa para detectar os *anti-patterns*, não sendo responsável pelas violações detectadas por ela. A ferramenta em questão, PMD, é produzida e testada por vários desenvolvedores.

A inspeção de documentação, abordada na Subseção 4.4.3, é um processo manual, complexo e suscetível a erros. Uma inspeção mal planejada e controlada pode levar a conclusões equivocadas. Assim, como um objetivo de combater essa ameaça, segui-se as recomendações descritas em Corporation (2009). A viabilidade do material adotado se confirma devido à concordância das recomendações propostas nele com as já adotadas por desenvolvedores ou elaborados pela literatura, como os *guidelines* e definições de política do tratamento de exceção apresentados no Capítulo 2 deste documento. Dentre essas práticas, estão recomendações sobre as exceções que devem ser tratadas em um sistema, a definição dos contextos de tratamento, o fluxo de exceções entre níveis de abstração, as informações fundamentais para que o eventos excepcionais sejam compreendidos, a notificação de eventos para que a equipe de desenvolvimento tenha ciência das falhas ocorridas, e a modelagem de exceções específicas do sistema desenvolvido.

7.3 Limitações

As seguintes limitações foram percebidas durante a elaboração e execução deste trabalho:

- Os *anti-patterns* buscados neste trabalho são limitados pela quantidade contemplada pela ferramenta adotada, assim como pela implementação das regras de detecção, que podem ser abrangentes ou limitadoras, como é exemplificado no Apêndice A;
- Este estudo também é limitado ao contexto do caso. Por isso, os dados coletados são pertencentes à mesma fonte, especificamente, à instituição mantedora do sistema XSA. No entanto, tal limitação já é esperada em estudos de caso; e

- Outras abordagens poderiam ter sido agregadas a este trabalho para prover um melhor entendimento de como o tratamento de exceção é usado dentro do sistema, mapeando os contextos de uso para revelar vulnerabilidades ou políticas que são usadas praticamente, mas não estão formalmente descritas. Um exemplo, seria o uso de ferramentas que verificam a conformidade arquitetural do tratamento de exceção. No entanto, o tempo também foi um fator limitante, pois restringe a quantidade de métodos usados.

7.4 Produção Bibliográfica

O processo de análise e os resultados alcançados nesta pesquisa foram submetidos para duas conferências internacionais e duas nacionais. Proporcionando duas publicações:

- Avaliando o Tratamento de Exceção em um Sistema Web Corporativo: Um Estudo de Caso. Publicado no *Workshop* de Teses e Dissertações do XXIII Simpósio Brasileiro de Sistemas Multimídia e Web (WebMedia '2017). O artigo apresentava a problemática abordada neste trabalho, os trabalhos relacionados, a proposta da pesquisa, as possíveis contribuições, e os resultados da Unidade de Análise 01. Autores: Dêmora Bruna Cunha de Sousa, Windson Viana de Carvalho e Lincoln Souza Rocha;
- *Analysing the Evolution of Exception Handling Anti-Patterns in Large-Scale Projects: A Case Study*. Publicado no XII Simpósio Brasileiro de Componentes de Software, Arquiteturas e Reutilização (SBCARS '18). O artigo apresentava os resultados encontrados a partir da execução das Unidades de Análise 01 e 02. Além da entrevista semiestruturada e a estimação do tempo necessário para o conserto dos *anti-patterns*. O trabalho ganhou a premiação "*Best Paper Awards*". Autores: Dêmora Bruna Cunha de Sousa, Windson Viana de Carvalho Lincoln Souza Rocha, e Paulo Henrique Maia; e
- A premiação do evento SBCARS '18 resultou em um convite para publicar a versão de detalhada do artigo no *Journal of the Brazilian Computer Society* (JBCS) no início de 2019.

7.5 Trabalhos Futuros

Este estudo buscou esclarecer as dificuldades inerentes ao desenvolvimento do tratamento de exceção em ambientes reais de desenvolvimento por meio de um estudo de caso. Ao longo do estudo, foram percebidos obstáculos que prejudicam o desenvolvimento

de qualidade do mecanismo. Esses obstáculos serviram de inspiração para a proposição dos seguintes trabalhos futuros:

- Contextualizar os valores quantitativos obtidos neste trabalho sobre os *anti-patterns* e as exceções não capturadas com as funcionalidades do sistema. Essa nova perspectiva permitiria identificar as regiões mais afetadas por problemas do tratamento de exceção, e possibilitaria a priorização de funcionalidades mais críticas em futuras refatorações;
- Buscar técnicas para o desenvolvimento de políticas do tratamento de exceção que previnam a ocorrência dos problemas encontrados neste trabalho, de forma que possam ser implantadas em sistemas já em produção, sem prejuízo à evolução deles e à equipe de desenvolvimento;
- Desenvolver ferramentas e guias para a medição da qualidade do tratamento de exceção em ambientes reais de desenvolvimento. Essas tecnologias seriam usadas para prover um *feedback* rápido e significativo, visto que análises manuais e o uso de diversas ferramentas consomem tempo e mão de obra qualificada;
- Criar materiais de qualificação para o treinamento de desenvolvedores recém contratados ou iniciantes no desenvolvimento de software. O material seria voltado ao ensino prático da implementação do tratamento de exceção em sistemas de grande porte e ao conhecimento das más práticas comprovadamente utilizadas; e
- Medir o impacto da adoção de políticas e do treinamento de desenvolvedores na diminuição da presença de *anti-patterns*, assim como no aparacimento de *bugs* relacionados ao tratamento de exceção;
- Analisar se o tamanho e a natureza do sistema (sistemas legados ou desenvolvidos pela instituição mantedora desde o início) impactam no desenvolvimento de qualidade do tratamento de exceção.

REFERÊNCIAS

AVIZIENIS, A.; LAPRIE, J.-C.; RANDELL, B.; LANDWEHR, C. Basic concepts and taxonomy of dependable and secure computing. **IEEE transactions on dependable and secure computing**, IEEE, v. 1, n. 1, p. 11–33, 2004.

BARBOSA, E. A.; GARCIA, A. Analyzing exceptional interfaces on evolving frameworks. In: IEEE. **Fifth Latin-American Symposium on Dependable Computing Workshops**. [S.l.], 2011. p. 17–20.

BARBOSA, E. A.; GARCIA, A. Global-aware recommendations for repairing violations in exception handling. **IEEE Transactions on Software Engineering**, PP, n. 99, p. 1–1, 2017. ISSN 0098-5589.

BARBOSA, E. A.; GARCIA, A.; BARBOSA, S. D. J. Categorizing faults in exception handling: A study of open source projects. In: IEEE. **Software Engineering (SBES), 2014 Brazilian Symposium on**. [S.l.], 2014. p. 11–20.

BARBOSA, E. A.; GARCIA, A.; ROBILLARD, M. P.; JAKOBUS, B. Enforcing exception handling policies with a domain-specific language. **IEEE Transactions on Software Engineering**, IEEE Computer Society, Los Alamitos, CA, USA, v. 42, n. 6, p. 559–584, 2016. ISSN 0098-5589.

BONIFÁCIO, R.; CARVALHO, F.; RAMOS, G. N.; KULESZA, U.; COELHO, R. The use of c++ exception handling constructs: A comprehensive study. In: IEEE. **Source Code Analysis and Manipulation (SCAM), 2015 IEEE 15th International Working Conference on**. [S.l.], 2015. p. 21–30.

BUHR, P. A.; MOK, W. Y. R. Advanced exception handling mechanisms. **IEEE Transactions on Software Engineering**, IEEE Press, Piscataway, NJ, USA, v. 26, p. 820–836, September 2000. ISSN 0098-5589.

CACHO, N.; BARBOSA, E. A.; ARAUJO, J.; PRANTO, F.; GARCIA, A.; CESAR, T.; SOARES, E.; CASSIO, A.; FILIPE, T.; GARCIA, I. How does exception handling behavior evolve? an exploratory study in java and c# applications. In: IEEE. **Software Maintenance and Evolution (ICSME), 2014 IEEE International Conference on**. [S.l.], 2014. p. 31–40.

CAMPIONE, M.; WALRATH, K.; HUML, A. **The Java tutorial: a short course on the basics**. [S.l.]: Addison-Wesley Professional, 2001. v. 1.

CASSEE, N.; PINTO, G.; CASTOR, F.; SEREBRENIK, A. How swift developers handle errors. In: **Mining Software Repositories (MSR), 2018 15th International Conference on**. [S.l.: s.n.], 2018.

CHANG, B.-M.; CHOI, K. A review on exception analysis. **Inf. Softw. Technol.**, Butterworth-Heinemann, Newton, MA, USA, v. 77, n. C, p. 1–16, set. 2016. ISSN 0950-5849.

CHEN, C.-T.; CHENG, Y. C.; HSIEH, C.-Y.; WU, I.-L. Exception handling refactorings: Directed by goals and driven by bug fixing. **Journal of Systems and Software**, Elsevier, v. 82, n. 2, p. 333–345, 2009.

COELHO, R.; ALMEIDA, L.; GOUSIOS, G.; DEURSEN, A. van. Unveiling exception handling bug hazards in android based on github and google code issues. In: IEEE. **Mining Software Repositories (MSR), 2015 IEEE/ACM 12th Working Conference on**. [S.l.], 2015. p. 134–145.

COELHO, R.; ALMEIDA, L.; GOUSIOS, G.; DEURSEN, A. V.; TREUDE, C. Exception handling bug hazards in android. **Empirical Software Engineering**, Springer, v. 22, n. 3, p. 1264–1304, 2017.

CORPORATION, M. **Microsoft® Application Architecture Guide**. [S.l.]: Microsoft Press, 2009.

CORREA, A. L.; WERNER, C. M.; ZAVERUCHA, G. Object oriented design expertise reuse: An approach based on heuristics, design patterns and anti-patterns. In: SPRINGER. **International Conference on Software Reuse**. [S.l.], 2000. p. 336–352.

CRESWELL, J. W. **Research design: Qualitative, quantitative, and mixed methods approaches**. [S.l.]: Sage publications, 2014. 245–250 p.

DIGKAS, G.; LUNGU, M.; CHATZIGEORGIOU, A.; AVGERIOU, P. The evolution of technical debt in the apache ecosystem. In: SPRINGER. **European Conference on Software Architecture**. [S.l.], 2017. p. 51–66.

DIKERT, K.; PAASIVAARA, M.; LASSENIUS, C. Challenges and success factors for large-scale agile transformations: A systematic literature review. **Journal of Systems and Software**, Elsevier, v. 119, p. 87–108, 2016.

DONY, C. Exception handling and object-oriented programming: towards a synthesis. **ACM Sigplan Notices**, ACM, v. 25, n. 10, p. 322–330, 1990.

DOSHI, G. **Best Practices for Exception Handling**. 2003. <http://www.onjava.com/pub/a/onjava/2003/11/19/exceptions.html>. Accessed: 2017-08-23.

EBERT, F.; CASTOR, F. A study on developers' perceptions about exception handling bugs. In: IEEE. **Software Maintenance (ICSM), 2013 29th IEEE International Conference on**. [S.l.], 2013. p. 448–451.

EBERT, F.; CASTOR, F.; SEREBRENIK, A. An exploratory study on exception handling bugs in java programs. **Journal of Systems and Software**, Elsevier, v. 106, p. 82–101, 2015.

FILHO, J. L. M.; ROCHA, L.; ANDRADE, R.; BRITTO, R. Preventing erosion in exception handling design using static-architecture conformance checking. In: ____. **Software Architecture: 11th European Conference, ECSA 2017, Canterbury, UK, September 11-15, 2017, Proceedings**. Cham: Springer International Publishing, 2017. p. 67–83. ISBN 978-3-319-65831-5.

GARCIA, A. F.; RUBIRA, C. M.; ROMANOVSKY, A.; XU, J. A comparative study of exception handling mechanisms for building dependable object-oriented software. **Journal of systems and software**, Elsevier, v. 59, n. 2, p. 197–222, 2001.

GURGEL, A.; MACIA, I.; GARCIA, A.; STAA, A. von; MEZINI, M.; EICHBERG, M.; MITSCHKE, R. Blending and reusing rules for architectural degradation prevention. In: ACM. **Proceedings of the 13th international conference on Modularity**. [S.l.], 2014. p. 61–72.

GURP, J. V.; BOSCH, J. Design erosion: problems and causes. **Journal of systems and software**, Elsevier, v. 61, n. 2, p. 105–119, 2002.

HUANG, J.; ZHU, W.; BASTANI, F.; YEN, I.-L.; FU, J. Automated exception handling in service composition using holistic planning. In: IEEE. **Computational Science and Engineering (CSE), 2012 IEEE 15th International Conference on**. [S.l.], 2012. p. 251–258.

JOSHUA, B. **Effective Java**. [S.l.]: Addison Wesley, 2012. 241–258 p.

KECHAGIA, M.; FRAGKOULIS, M.; LOURIDAS, P.; SPINELLIS, D. The exception handling riddle: An empirical study on the android api. **Journal of Systems and Software**, Elsevier, v. 142, p. 248–270, 2018.

KECHAGIA, M.; SPINELLIS, D. Undocumented and unchecked: Exceptions that spell trouble. In: **Proceedings of the 11th Working Conference on Mining Software Repositories**. New York, NY, USA: ACM, 2014. (MSR 2014), p. 312–315. ISBN 978-1-4503-2863-0.

KIM, S.; ZIMMERMANN, T.; PREMRAJ, R.; BETTENBURG, N.; SHIVAJI, S. Predicting method crashes with bytecode operations. In: ACM. **Proceedings of the 6th India software engineering conference**. [S.l.], 2013. p. 3–12.

LETHBRIDGE, T. C.; SIM, S. E.; SINGER, J. Studying software engineers: Data collection techniques for software field studies. **Empirical software engineering**, Springer, v. 10, n. 3, p. 311–341, 2005.

MALAYERI, D.; ALDRICH, J. Practical exception specifications. In: **Advanced Topics in Exception Handling Techniques**. [S.l.]: Springer, 2006. p. 200–220.

MARINESCU, C. Are the classes that use exceptions defect prone? In: ACM. **Proceedings of the 12th International Workshop on Principles of Software Evolution and the 7th annual ERCIM Workshop on Software Evolution**. [S.l.], 2011. p. 56–60.

MARINESCU, C. Should we beware the exceptions? an empirical study on the eclipse project. In: IEEE. **Symbolic and Numeric Algorithms for Scientific Computing (SYNASC), 2013 15th International Symposium on**. [S.l.], 2013. p. 250–257.

NOGUEIRA, A. F.; RIBEIRO, J. C.; ZENHA-RELA, M. A. Trends on empty exception handlers for java open source libraries. In: IEEE. **Software Analysis, Evolution and Reengineering (SANER), 2017 IEEE 24th International Conference on**. [S.l.], 2017. p. 412–416.

OLIVEIRA, J.; BORGES, D.; SILVA, T.; CACHO, N.; CASTOR, F. Do android developers neglect error handling? a maintenance-centric study on the relationship between android abstractions and uncaught exceptions. **Journal of Systems and Software**, v. 136, n. Supplement C, p. 1 – 18, 2018. ISSN 0164-1212.

OLIVEIRA, J.; BORGES, D.; SILVA, T.; CACHO, N.; CASTOR, F. Do android developers neglect error handling? a maintenance-centric study on the relationship between android abstractions and uncaught exceptions. **Journal of Systems and Software**, Elsevier, v. 136, p. 1–18, 2018.

OLIVEIRA, J.; CACHO, N.; BORGES, D.; SILVA, T.; CASTOR, F. An exploratory study of exception handling behavior in evolving android and java applications. In: ACM. **Proceedings of the 30th Brazilian Symposium on Software Engineering**. [S.l.], 2016. p. 23–32.

OSMAN, H.; CHIŞ, A.; CORRODI, C.; GHAFARI, M.; NIERSTRASZ, O. Exception evolution in long-lived java systems. In: IEEE PRESS. **Proceedings of the 14th International Conference on Mining Software Repositories**. [S.l.], 2017. p. 302–311.

OSMAN, H.; CHIŞ, A.; SCHAERER, J.; GHAFARI, M.; NIERSTRASZ, O. On the evolution of exception usage in java projects. In: IEEE. **Software Analysis, Evolution and Reengineering (SANER), 2017 IEEE 24th International Conference on**. [S.l.], 2017. p. 422–426.

PÁDUA, G. B. de; SHANG, W. Studying the prevalence of exception handling anti-patterns. In: IEEE PRESS. **Proceedings of the 25th International Conference on Program Comprehension**. [S.l.], 2017. p. 328–331.

PALOMBA, F.; BAVOTA, G.; PENTA, M. D.; OLIVETO, R.; LUCIA, A. D. Do they really smell bad? a study on developers' perception of bad code smells. In: IEEE. **Software maintenance and evolution (ICSME), 2014 IEEE international conference on**. [S.l.], 2014. p. 101–110.

PAWLAK, R.; MONPERRUS, M.; PETITPREZ, N.; NOGUERA, C.; SEINTURIER, L. Spoon: A library for implementing analyses and transformations of java source code. **Software: Practice and Experience**, Wiley-Blackwell, v. 46, p. 1155–1179, 2015. Disponível em: <https://hal.archives-ouvertes.fr/hal-01078532/document>.

PERRY, D. E.; WOLF, A. L. Foundations for the study of software architecture. **ACM SIGSOFT Software Engineering Notes**, ACM, v. 17, n. 4, p. 40–52, 1992.

QUEIROZ, F. D.; COELHO, R. Characterizing the exception handling code of android apps. In: IEEE. **Software Components, Architectures and Reuse (SBCARS), 2016 X Brazilian Symposium on**. [S.l.], 2016. p. 131–140.

RUNESON, P.; HOST, M.; RAINER, A.; REGNELL, B. **Case study research in software engineering: Guidelines and examples**. [S.l.]: John Wiley & Sons, 2012.

SAWADPONG, P.; ALLEN, E. B. Software defect prediction using exception handling call graphs: A case study. In: IEEE. **High Assurance Systems Engineering (HASE), 2016 IEEE 17th International Symposium on**. [S.l.], 2016. p. 55–62.

SAWADPONG, P.; ALLEN, E. B.; WILLIAMS, B. J. Exception handling defects: An empirical study. In: IEEE. **High-Assurance Systems Engineering (HASE), 2012 IEEE 14th International Symposium on**. [S.l.], 2012. p. 90–97.

SENA, D.; COELHO, R.; KULESZA, U.; BONIFÁCIO, R. Understanding the exception handling strategies of java libraries: An empirical study. In: IEEE. **Mining Software Repositories (MSR), 2016 IEEE/ACM 13th Working Conference on**. [S.l.], 2016. p. 212–222.

SHAH, H.; GORG, C.; HARROLD, M. J. Understanding exception handling: Viewpoints of novices and experts. **IEEE Transactions on Software Engineering**, IEEE, v. 36, n. 2, p. 150–161, 2010.

SHAHROKNI, A.; FELDT, R. A systematic review of software robustness. **Information and Software Technology**, Butterworth-Heinemann, Newton, MA, USA, v. 55, n. 1, p. 1–17, jan. 2013. ISSN 0950-5849.

SHULL, F.; SINGER, J.; SJØBERG, D. I. **Guide to advanced empirical software engineering**. [S.l.]: Springer, 2007.

SIEDERSLEBEN, J. Errors and exceptions–rights and obligations. In: **Advanced Topics in Exception Handling Techniques**. [S.l.]: Springer, 2006. p. 275–287.

SILVA, L. D.; BALASUBRAMANIAM, D. Controlling software architecture erosion: A survey. **Journal of Systems and Software**, Elsevier, v. 85, n. 1, p. 132–151, 2012.

SINHA, S.; ORSO, A.; HARROLD, M. J. Automated support for development, maintenance, and testing in the presence of implicit control flow. In: IEEE COMPUTER SOCIETY. **Proceedings of the 26th International Conference on Software Engineering**. [S.l.], 2004. p. 336–345.

TERRA, R.; VALENTE, M. T. A dependency constraint language to manage object-oriented software architectures. **Software: Practice and Experience**, Wiley Online Library, v. 39, n. 12, p. 1073–1094, 2009.

WIRFS-BROCK, R. J. Toward exception-handling best practices and patterns. **IEEE Software**, v. 23, n. 5, p. 11–13, Sept 2006. ISSN 0740-7459.

WOHLIN, C. Guidelines for snowballing in systematic literature studies and a replication in software engineering. In: ACM. **Proceedings of the 18th international conference on evaluation and assessment in software engineering**. [S.l.], 2014. p. 38.

WOHLIN, C.; RUNESON, P.; HÖST, M.; OHLSSON, M. C.; REGNELL, B.; WESSLÉN, A. **Experimentation in software engineering**. [S.l.]: Springer Science & Business Media, 2012. 55–72 p.

YUAN, D.; LUO, Y.; ZHUANG, X.; RODRIGUES, G. R.; ZHAO, X.; ZHANG, Y.; JAIN, P.; STUMM, M. Simple testing can prevent most critical failures: An analysis of production failures in distributed data-intensive systems. In: **OSDI**. [S.l.: s.n.], 2014. p. 249–265.

APÊNDICE A – ANÁLISE INICIAL DA VERSÃO 2017.2 DO SISTEMA XSA

Dentre as ferramentas selecionadas para a inspeção de código no sitema XSA, 3 são abrangentes, já que não detectam apenas problemas do tratamento de exceção: PMD, Checkstyle e FindBugs. Por isso, foram configuradas para considerar apenas as regras do tratamento de exceção durante as suas execuções ou tiveram seus resultados filtrados. A ferramenta Parichayana, desenvolvida apenas esse fim, não necessitou de configurações adicionais.

As ferramentas tiveram sua execução iniciada por linha de comando, gerando um arquivo no formato XML (*eXtensible Markup Language*) como saída, com exceção da ferramenta Parichayana, que gera apenas um documento de texto como resultado. Já que os documentos obtidos não eram padronizados, uma análise individual dos resultados foi feita.

Tabela 13 – Violações encontradas nas ferramentas Checkstyle, FindBugs e Parichayana

Anti-Pattern	Violações
Checkstyle	
Catch Generic	2425
Catch and Do Nothing	86
Throws Generic	7
Nested Try	2
FindBugs	
Catch Generic	580
Catch and Do Nothing	37
Parichayana	
Catch Generic	1034
Throws Generic	721
Log and Return Null	327
Log and Throw	201
Destructive Wrapping	103
Catch and Ignore	20
Catch NullPointerException	11
Relying on getCause()	4
Throw NullPointerException	3

Fonte: o autor.

Os resultados obtidos com a execução de todas as ferramentas deixaram evidente a presença de *anti-patterns* no sistema XSA (Tabela 9 e Tabela 13). As ferramentas foram executadas sem que o funcionamento de suas regras fosse modificado. É importante frisar que essas ferramentas possuem implementações distintas para cada regra, mesmo que elas tenham finalidades semelhantes. Portanto, alguns dos *anti-patterns* possuem resultados quantitativos diferentes em cada ferramenta. Para esclarecer de forma ilustrativa a diferença entre as regras, verificou-se divergências nas implementações de duas regras com finalidades semelhantes nas

ferramentas. O resultado dessa verificação é exibido na Tabela 14. Algumas ferramentas, como PMD e Checkstyle, permitem a customização das regras por meio dos arquivos de configuração, efetuando passagem de parâmetros. Como as ferramentas apresentadas possuem o código-fonte disponível em repositórios públicos, também é possível customizar as regras modificando o projeto de cada uma.

Tabela 14 – Comparação de regras similares implementadas nas ferramentas

	PMD	Checkstyle	Parichayana	FindBugs
Catch Generic				
Regra	Avoid Catching Generic Exception	IllegalCatch	CTGE - Catching Generic Exception	Exception is caught when Exception is not thrown
Exceções genéricas consideradas	NullPointerException, RuntimeException, Exception	Throwable,Error, RuntimeException, Exception	Exception, Throwable	Exception
Catch and Do Nothing				
Regra	EmptyCatchBlock	EmptyCatchBlock	RNHR - returns null instead of handling or re-throwing	Method might ignore exception
Divergências	Blocos que possuem apenas comentários são considerados vazios	Blocos que possuem apenas comentários não são considerados vazios	Considera somente blocos que retornam null (Catch and Ignore)	Apenas tratamentos que ignoram Exception são considerados

Fonte: o autor.

APÊNDICE B – QUESTIONÁRIO 01: PERCEPÇÕES SOBRE O TRATAMENTO DE EXCEÇÃO

O primeiro questionário *online* aplicado no estudo de caso. Foi elaborado para obter a percepção dos desenvolvedores e analistas sobre o tratamento de exceção.

Percepções Sobre o Tratamento de Exceção

Olá!

Esse formulário é um instrumento de coleta de dados para uma pesquisa de mestrado. Um dos objetivos da pesquisa é entender como desenvolvedores/analistas de uma instituição pública lidam com o tratamento de exceção diariamente. Portanto, os questionamentos a seguir devem ser respondidos considerando o contexto do seu trabalho atual (instituição, atividades desenvolvidas, etc). Também pretendemos identificar em nossa instituição problemas, desafios e soluções possíveis para melhorar a qualidade do código dos softwares Web por nós desenvolvidos e/ou mantidos, principalmente, com relação ao tratamento de situações excepcionais.

É importante salientar que as informações coletadas serão utilizadas somente para fins acadêmicos. Além disso, asseguramos a confidencialidade dos dados, não sendo possível a identificação de indivíduos por meio das respostas fornecidas.

Agradecemos a sua contribuição.

*Obrigatório

1. **Há quanto tempo você trabalha com o desenvolvimento de software Web?** *
   Marcar apenas uma oval.
   - ◯ Menos de 1 ano
   - ◯ Entre 1 e 3 anos
   - ◯ Entre 4 e 6 anos
   - ◯ Entre 7 e 9 anos
   - ◯ Mais de 9 anos

2. **Há quanto tempo você está empregado na instituição em que trabalha?** *
   Marcar apenas uma oval.
   - ◯ Menos de 1 ano
   - ◯ Entre 1 e 3 anos
   - ◯ Entre 4 e 6 anos
   - ◯ Entre de 7 e 9 anos
   - ◯ Mais de 9 anos

Responda os itens abaixo indicando o grau de concordância com as assertivas

3. **Sou um desenvolvedor/analista experiente considerando o nível de conhecimento que tenho sobre os sistemas Web que desenvolvo.** *

Marcar apenas uma oval.

- Concordo totalmente
- Concordo parcialmente
- Indiferente
- Discordo parcialmente
- Discordo totalmente

4. **Ainda não domino completamente a linguagem de programação (backend ou frontend) e o ferramental que utilizo no desenvolvimento dos sistemas Web que participo.** *

Marcar apenas uma oval.

- Concordo totalmente
- Concordo parcialmente
- Indiferente
- Discordo parcialmente
- Discordo totalmente

5. **Considero o tratamento de exceção crucial no desenvolvimento de um sistema Web** *

Marcar apenas uma oval.

- Concordo totalmente
- Concordo parcialmente
- Indiferente
- Discordo parcialmente
- Discordo totalmente

6. **Sistemas Web não dependem do tratamento de exceção para que sejam desenvolvidos e utilizados adequadamente.** *

Marcar apenas uma oval.

- Concordo totalmente
- Concordo parcialmente
- Indiferente
- Discordo parcialmente
- Discordo totalmente

7. **Sempre me preocupo com a implementação do tratamento de exceção (e.g., refletir sobre as exceções e as medidas que devem ser realizadas quando elas ocorrem) durante as atividades de desenvolvimento Web que executo.** *

Marcar apenas uma oval.

- ◯ Concordo totalmente
- ◯ Concordo parcialmente
- ◯ Indiferente
- ◯ Discordo parcialmente
- ◯ Discordo totalmente

8. **A equipe de desenvolvimento que trabalho raramente se preocupa com a implementação do tratamento de exceção durante as atividades de desenvolvimento Web.** *

Marcar apenas uma oval.

- ◯ Concordo totalmente
- ◯ Concordo parcialmente
- ◯ Indiferente
- ◯ Discordo parcialmente
- ◯ Discordo totalmente

9. **A obrigação imposta por algumas linguagens de programação de backend para que o tratamento de exceções seja implementado corrobora com a minha visão de uma boa linguagem de programação.** *

Marcar apenas uma oval.

- ◯ Concordo totalmente
- ◯ Concordo parcialmente
- ◯ Indiferente
- ◯ Discordo parcialmente
- ◯ Discordo totalmente

10. **Uma boa linguagem de programação Web usada no backend não deveria exigir que o tratamento de exceções seja sempre implementado** *

Marcar apenas uma oval.

- ◯ Concordo totalmente
- ◯ Concordo parcialmente
- ◯ Indiferente
- ◯ Discordo parcialmente
- ◯ Discordo totalmente

11. **Aprendi em cursos ou treinamentos como utilizar o tratamento de exceção adequadamente.** *
Marcar apenas uma oval.

- Concordo totalmente
- Concordo parcialmente
- Indiferente
- Discordo parcialmente
- Discordo totalmente

12. **Durante minha vida acadêmica ou aprendizado técnico, não fui capacitado para lidar com o tratamento de exceção.** *
Marcar apenas uma oval.

- Concordo totalmente
- Concordo parcialmente
- Indiferente
- Discordo parcialmente
- Discordo totalmente

13. **O desenvolvimento de fluxos de exceção exigem maior atenção pois são mais complexos.** *
Marcar apenas uma oval.

- Concordo totalmente
- Concordo parcialmente
- Indiferente
- Discordo parcialmente
- Discordo totalmente

14. **Pensar no comportamento excepcional de uma funcionalidade não exige tanto esforço quanto o empregado na elaboração de fluxos normais do sistema.** *
Marcar apenas uma oval.

- Concordo totalmente
- Concordo parcialmente
- Indiferente
- Discordo parcialmente
- Discordo Totalmente

15. **Estou satisfeito com a maneira que lido com o tratamento de exceção durante a execução de tarefas diárias em minha equipe de desenvolvimento.** *
Marcar apenas uma oval.

- Concordo totalmente
- Concordo parcialmente
- Indiferente
- Discordo parcialmente
- Discordo totalmente

16. **Percebo que a forma que implemento/projeto o tratamento de exceção ainda não é adequada.** *

Marcar apenas uma oval.

- ◯ Concordo totalmente
- ◯ Concordo parcialmente
- ◯ Indiferente
- ◯ Discordo parcialmente
- ◯ Discordo totalmente

17. **A instituição em que trabalho possui especificações, políticas documentadas ou padrões relacionados à implementação do tratamento de exceção.** *

Marcar apenas uma oval.

- ◯ Concordo totalmente
- ◯ Concordo parcialmente
- ◯ Indiferente
- ◯ Discordo parcialmente
- ◯ Discordo totalmente

18. **Nenhum tipo de documentação ou padrão de programação é exigido sobre o tratamento de exceção pela instituição em que trabalho.** *

Marcar apenas uma oval.

- ◯ Concordo totalmente
- ◯ Concordo parcialmente
- ◯ Indiferente
- ◯ Discordo parcialmente
- ◯ Discordo totalmente

19. **Geralmente, não documento elementos de código destinados ao tratamento de exceção.** *

Marcar apenas uma oval.

- ◯ Concordo totalmente
- ◯ Concordo parcialmente
- ◯ Indiferente
- ◯ Discordo parcialmente
- ◯ Discordo totalmente

20. **Tenho a documentação do tratamento de exceção como parte das minhas obrigações como desenvolvedor.** *

Marcar apenas uma oval.

- Concordo totalmente
- Concordo parcialmente
- Indiferente
- Discordo parcialmente
- Discordo totalmente

21. **Erros ocasionados por problemas do tratamento de exceção já foram reportados à equipe de desenvolvimento.** *

Marcar apenas uma oval.

- Concordo totalmente
- Concordo parcialmente
- Indiferente
- Discordo parcialmente
- Discordo totalmente

22. **São raros os erros do tratamento de exceção nos sistemas que desenvolvo.** *

Marcar apenas uma oval.

- Concordo totalmente
- Concordo parcialmente
- Indiferente
- Discordo parcialmente
- Discordo totalmente

23. **Acredito que usuários finais dos sistemas Web que desenvolvo são afetados por erros do tratamento de exceção.** *

Marcar apenas uma oval.

- Concordo totalmente
- Concordo parcialmente
- Indiferente
- Discordo parcialmente
- Discordo totalmente

24. **Utilizo rotineiramente ferramentas/técnicas para detectar bad smells do tratamento de exceção.** *

Marcar apenas uma oval.

- Concordo totalmente
- Concordo parcialmente
- Indiferente
- Discordo parcialmente
- Discordo totalmente

25. **O tratamento de exceção implementado nos sistemas em que trabalho é satisfatório e contribui para atender aos requisitos funcionais e não-funcionais.** *

Marcar apenas uma oval.

- Concordo totalmente
- Concordo parcialmente
- Indiferente
- Discordo parcialmente
- Discordo totalmente

26. **Acredito que o tratamento de exceção dos sistemas em que trabalho precisa ser melhorado.** *

Marcar apenas uma oval.

- Concordo totalmente
- Concordo parcialmente
- Indiferente
- Discordo parcialmente
- Discordo totalmente

27. **O estabelecimento/melhoria de normas institucionais do tratamento de exceção resultaria em benefícios para os desenvolvedores e usuários finais.** *

Marcar apenas uma oval.

- Concordo totalmente
- Concordo parcialmente
- Indiferente
- Discordo parcialmente
- Discordo totalmente

28. **Definir padrões institucionais de como deve ser empregado o tratamento de exceção em um sistema não será garantia da melhoria da qualidade** *
Marcar apenas uma oval.

- ◯ Concordo totalmente
- ◯ Concordo parcialmente
- ◯ Indiferente
- ◯ Discordo parcialmente
- ◯ Discordo totalmente

Sobre os seus conhecimentos do tratamento de exceção, responda:

29. **Marque a alternativa que melhor define o tratamento de exceção:** *
Marcar apenas uma oval.

- ◯ É um evento que modela situações do fluxo normal de execução de um sistema.
- ◯ O tratamento de exceção provê meios para estruturar as atividades de tolerância a faltas através da recuperação de erros.
- ◯ Mecanismo que possibilita a recuperação de erros na linguagem Java através da definição de tratamentos para exceções padrões da linguagem.
- ◯ É um sistema de controle de bugs.

30. **Marque a opção correta sobre o tratamento de exceção na linguagem Java:** *
Marcar apenas uma oval.

- ◯ Cada bloco try pode ser seguido por zero ou mais blocos catch.
- ◯ O bloco finally é executado apenas quando uma exceção é capturada.
- ◯ Como exceções funcionam de maneira hierárquica, exceções mais específicas englobam as mais gerais.
- ◯ Exceções são definidas através da especialização da classe Throws.

De acordo com a sua opinião, responda:

31. **Existem cenários/situações em que você evita ou ignora o uso do tratamento de exceção? Quais?**

_____

_____

_____

_____

_____

32. **Qual a sua opinião sobre o tratamento de exceção?**

33. **Como você avalia esse questionário com relação ao tempo e ao conteúdo apresentados?** *
Marcar apenas uma oval.

	1	2	3	4	5	
Totalmente Insatisfatório	◯	◯	◯	◯	◯	Totalmente Satisfatório

APÊNDICE C – QUESTIONÁRIO 02: APLICAÇÃO DO TRATAMENTO DE EXCEÇÃO

Segundo questionário aplicado. Tem como propósito identificar quais estruturas prejudiciais do tratamento de exceção eram conhecidas e passíveis de reconhecimento pelos desenvolvedores da instituição alvo do estudo de caso.

Aplicação do Tratamento de Exceção

#Objetivo

O propósito deste questionário é perceber quais estruturas prejudiciais do tratamento de exceção (que apresentam maior suscetibilidade à defeitos) são conhecidas e passíveis de identificação por profissionais do desenvolvimento de software.

Devido ao objetivo deste instrumento, alguns códigos contém estruturas de implementação não recomendadas na literatura ou no mercado de desenvolvimento. Não se preocupe se não puder identificá-las, é mais importante para nós saber quais estruturas não estão corretas na SUA OPINIÃO.

#Estrutura

O questionário contém duas etapas. A primeira etapa possui 5 códigos para avaliação. A segunda contém 3 itens de múltipla escolha sobre perfil profissional e 1 campo para sugestões (opcional).

Estimamos um tempo de **10 minutos** para o preenchimento.

#Confidencialidade

As informações serão enviadas sem a necessidade de identificação. Além disso, garantimos a confidencialidade e o uso dos dados apenas para fins acadêmicos.

*Obrigatório

(Código 1)

1. **Em sua opinião, o seguinte trecho de código apresenta problemas na implementação do tratamento de exceção (Ex: no design, na codificação, propensão a falhas):** *

```
try{
    if(con != null) con.close();
}catch(SQLException sqlEx2){
    throw new DAOException(sqlEx2.getMessage());
}
```

Marcar apenas uma oval.

- ◯ Sim, apresenta
- ◯ Não, não apresenta *Ir para a pergunta 3.*

(Código 1)

2. **Informe os problemas do tratamento de exceção que você identificou:** *

```
try{
    if(con != null) con.close();
}catch(SQLException sqlEx2){
    throw new DAOException(sqlEx2.getMessage());
}
```

(Código 2)

3. **Em sua opinião, o seguinte trecho de código apresenta problemas na implementação do tratamento de exceção (Ex: no design, na codificação, propensão a falhas):** *

```
public Object nomeMetodo(HttpServletRequest req) throws Exception {

        GenericDAO dao = getGenericDAO(req);
        Object obj = getCommandClass().newInstance();
        int id = RequestUtils.getIntParameter(req, "id");
```

Marcar apenas uma oval.

- ◯ Sim, apresenta
- ◯ Não, não apresenta *Ir para a pergunta 5.*

Avaliação do Tratamento de Exceção - (Código 2)

4. **Informe os problemas do tratamento de exceção que você identificou:** *

```
public Object nomeMetodo(HttpServletRequest req) throws Exception {

        GenericDAO dao = getGenericDAO(req);
        Object obj = getCommandClass().newInstance();
        int id = RequestUtils.getIntParameter(req, "id");
```

(Código 3)

5. **Em sua opinião, o seguinte trecho de código apresenta problemas na implementação do tratamento de exceção (Ex: no design, na codificação, propensão a falhas):** *

```
try {
    executeWithoutClosingSession(mov);
} catch (NegocioException e) {
    addMensagemErro(e.getMessage());
    tratarErroNegocio(e);
}
```

Marcar apenas uma oval.

- ◯ Sim, apresenta
- ◯ Não, não apresenta *Ir para a pergunta 7.*

(Código 3)

6. **Informe os problemas do tratamento de exceção que você identificou:** *

```
try {
    executeWithoutClosingSession(mov);
} catch (NegocioException e) {
    addMensagemErro(e.getMessage());
    tratarErroNegocio(e);
}
```

(Código 4)

7. **Em sua opinião, o seguinte trecho de código apresenta problemas na implementação do tratamento de exceção (Ex: no design, na codificação, propensão a falhas):** *

```
try {
    cal = CalendarioAcademicoHelper.getCalendario(getUsuarioLogado());
} catch (Exception e) {
    e.printStackTrace();
    tratamentoErroPadrao(e);
}
```

Marcar apenas uma oval.

- ◯ Sim, apresenta
- ◯ Não, não apresenta *Ir para a pergunta 9.*

(Código 4)

8. **Informe os problemas do tratamento de exceção que você identificou:** *

```
try {
    cal = CalendarioAcademicoHelper.getCalendario(getUsuarioLogado());
} catch (Exception e) {
    e.printStackTrace();
    tratamentoErroPadrao(e);
}
```

(Código 5)

9. **Em sua opinião, o seguinte trecho de código apresenta problemas na implementação do tratamento de exceção (Ex: no design, na codificação, propensão a falhas):** *

```
try {
    inicializar();
} catch (DAOException e) {
    e.printStackTrace();
    throw new RuntimeException(e);
}
```

Marcar apenas uma oval.

- ◯ Sim, apresenta
- ◯ Não, não apresenta *Ir para a pergunta 11.*

(Código 5)

10. **Informe os problemas do tratamento de exceção que você identificou:** *

```
try {
    inicializar();
} catch (DAOException e) {
    e.printStackTrace();
    throw new RuntimeException(e);
}
```

Sobre você e o questionário

11. **Há quanto tempo você está empregado na instituição em que trabalha?** *
Marcar apenas uma oval.

- ◯ Menos de 1 ano
- ◯ 1 ano
- ◯ 2 anos
- ◯ 3 anos
- ◯ 4 anos
- ◯ 5 anos
- ◯ 6 anos
- ◯ 7 anos
- ◯ 8 anos
- ◯ 9 anos ou mais

12. **Qual o seu nível de formação acadêmica na área de TI?**
Marcar apenas uma oval.

- ◯ Técnico
- ◯ Graduando
- ◯ Graduado
- ◯ Mestrando
- ◯ Mestre
- ◯ Doutorando
- ◯ Doutor
- ◯ Outro: _____

13. **Como você avalia esse questionário com relação ao tempo e ao conteúdo apresentados?** *
Marcar apenas uma oval.

	1	2	3	4	5	
Totalmente Insatisfatório	◯	◯	◯	◯	◯	Totalmente Satisfatório

14. **Críticas ou sugestões:**

_____

_____

_____

_____

_____

APÊNDICE D – EXEMPLOS DE *ANTI-PATTERNS* EM CÓDIGOS JAVA

Código-fonte 1 – **Unhandled Exceptions:** Exceções não são tratadas ao longo da cadeia de chamada de métodos. Se a exceção chegar ao ponto de entrada do programa, ela causará o fim da execução dele.

```
1 public static void main(String[] args) {
2     try {
3         this.salario = calcularSalario();
4     } catch (CalculoSalarioException e) {
5         throw new RuntimeCalculoSalarioException(e);
6     }
7 }
```

Código-fonte 2 – **Catch Generic:** O tratamento captura muitas exceções de baixo nível por meio da definição de uma exceção de alto nível.

```
1 try {
2     salario = calcularSalario();
3 } catch (Exception e) {
4     notificarDesenvolvedores(e);
5 }
```

Código-fonte 3 – **Destructive Wrapping:** Uma exceção é propagada como se fosse outra, sem a passagem dos devidos parâmetros e causa a perda de informação referente à exceção original.

```
1 try {
2     salario = calcularSalario();
3 } catch (CalculoSalarioException e) {
4     throw new MyException("Erro durante calculo do salario"
          );
5 }
```

Código-fonte 4 – **Catch and Do Nothing:** Apesar da exceção ser capturada, nada é feito para tratá-la.

```
1 try {
2     salario = calcularSalario();
3 } catch (CalculoSalarioException e) {
4
5 }
```

Código-fonte 5 – **Dummy Handler:** O tratamento não possui ações para recuperação de erros.

```
1 try {
2     salario = calcularSalario();
3 } catch (CalculoSalarioException e) {
4     e.printStackTrace();
5 }
```

Código-fonte 6 – **Ignoring Interrupted Exception:** Uma exceção do tipo *InterruptedException* é ignorada.

```
1 try {
2     Thread.sleep(15000);
3 } catch (InterruptedException ex) {
4 }
```

Código-fonte 7 – **Throw Within Finally:** Se o código presente no bloco *finally* lançar uma exceção, as exceções lançadas anteriormente serão perdidas.

```
1 try {
2     salario = calcularSalario();
3 } catch (CalculoSalarioException e) {
4     notificarDesenvolvedores();
5     throw new NegocioException(e);
6 } finally{
7     esteMetodoLancaUmaExcecao();
8 }
```

Código-fonte 8 – **Wrong Exception Thrown:** Lançamento de uma exceção genérica.

```
1 try {
2     salario = calcularSalario();
3 } catch (CalculoSalarioException e) {
4     throw new RuntimeException(e);
5 }
```

Código-fonte 9 – **Throws Generic:** Propagação de uma exceção genérica.

```
1 public void metodo1() throws Exception {
```