Daniele Wolfart

Dívida de Variabilidade: Um Estudo Multimétodo

Cascavel-PR

2022

Daniele Wolfart

Dívida de Variabilidade: Um Estudo Multimétodo

Dissertação apresentada como requisito parcial para a obtenção do grau de Mestre pelo Programa de Pós-Graduação em Ciência da Computação (PPGComp) da Universidade Estadual do Oeste do Paraná – Unioeste, campus de Cascavel.

Universidade Estadual do Oeste do Paraná – Unioeste – Cascavel

Centro de Ciências Exatas e Tecnológicas – CCET

Programa de Pós-Graduação em Ciência da Computação – PPGComp

Orientador: Wesley Klewerton Guez Assunção

Coorientador: Jabier Martinez

Cascavel-PR

2022

Daniele Wolfart
Dívida de Variabilidade: Um Estudo Multimétodo/ Daniele Wolfart. – Cascavel-PR, 2022-
94p. : il. (algumas color.) ; 30 cm.

Orientador: Wesley Klewerton Guez Assunção

Dissertação (Mestrado)– Universidade Estadual do Oeste do Paraná – Unioeste – Cascavel
Centro de Ciências Exatas e Tecnológicas – CCET
Programa de Pós-Graduação em Ciência da Computação – PPGComp, 2022.

1. Reúso oportunista. 2. Linha de Produto de Software. 2. Variante de Software. I. Wesley Klewerton Guez Assunção II. Universidade Estadual do Oeste do Paraná – Unioeste – Cascavel III. Programa de Pós-Graduação em Ciência da Computação - PPGComp. IV. Dívida de Variabilidade: Um Estudo Multimétodo

Daniele Wolfart

Dívida de Variabilidade: Um Estudo Multimétodo

Dissertação apresentada como requisito parcial para a obtenção do grau de Mestre pelo Programa de Pós-Graduação em Ciência da Computação (PPGComp) da Universidade Estadual do Oeste do Paraná – Unioeste, campus de Cascavel.

Trabalho aprovado. Cascavel-PR, 16 de Fevereiro de 2022:

Wesley Klewerton Guez Assunção
Orientador

Jabier Martinez
Co-orientador

Ivonei Freitas da Silva
Avaliador Interno

Thelma Elita Colanzi
Avaliador Externo

Cascavel-PR
2022

Este trabalho é dedicado aos meus pais, que tiveram o sonho de estudar e não puderam.

Agradecimentos

Gostaria de agradecer primeiramente a Deus pelo dom da vida e por me dar saúde, permitindo assim realizar este projeto até o fim. Também pela saúde de meus familiares, professores e colegas que me deram suporte nestes dois anos e neste período tão crítico que enfrentamos de pandemia.

Ao meu orientador Wesley Klewerton Guez Assunção e co-orientador Jabier Martinez que com muita paciência, dedicação e profissionalismo auxiliaram para a produção deste trabalho.

ÀS empresas que atuei no período do mestrado, que oportunizaram a execução do meu sonho, permitindo que eu me ausentasse do trabalho diversas vezes para estudar, além de ceder os estudos de caso para a produção deste trabalho.

Agradeço também aos meus colegas de trabalho Alexandre, Guilherme, Luciane, Jemerson, Roberta e Hermes pelo apoio e compreensão durante estes dois anos.

Aos entrevistados do estudo de caso, pela disponibilidade de tempo e interesse demonstrados. A todos que direta ou indiretamente contribuíram, muito obrigado!

Aos meus pais, que me proporcionaram uma educação sólida e por me transmitirem seus valores que moldaram meu caráter. Obrigado por sempre me incentivarem a progredir de forma honesta. Também pelo apoio e torcida o tempo todo, mesmo que às vezes de longe, e pelas palavras de estímulo.

E por fim, agradeço aos meus familiares, amigos, e namorado pela compreensão por tantos momentos de ausência e apoio em todas horas difíceis.

"O sucesso é a soma de pequenos esforços repetidos dia após dia."
(Collier, Robert)

Resumo

WOLFART, Daniele. **Dívida de Variabilidade: Um Estudo Multimétodo**. Orientador: Wesley Klewerton Guez Assunção. 2022. 94f. Dissertação (Mestrado em Ciência da Computação) – Universidade Estadual do Oeste do Paraná, Cascavel – Paraná, 2022.

Variabilidade é a capacidade de um sistema ou artefato de software ser estendido, customizado ou configurado para uso e reúso em diferentes contextos. Gerenciar variabilidade é crucial para garantir o sucesso de um projeto de software. O que ainda varia é a forma de implementar a variabilidade. Embora possa ser implementada de forma sistemática, como por exemplo utilizando Linhas de Produtos de Software (LPSs), muitas empresas optam pela forma que inicialmente exige um investimento menor e oferece um bom *time-to-market*, o reúso oportunista, como por exemplo, copiar e colar. Porém, este caminho pode aumentar a ocorrência de dívidas técnicas, característica típica de uma decisão que apresenta vantagem a curto prazo, mas gera um passivo a longo prazo, tal como dificuldade de manutenção e evolução do sistema. Dado a suas importâncias, tanto o conceito de variabilidade quanto o conceito de dívida técnica são amplamente estudados na Engenharia de Software, no entanto, ambos conceitos ainda não foram investigados juntos. Assim, este trabalho reporta os resultados de uma pesquisa para compreender como a dívida técnica é ocasionada por meio de um gerenciamento de variabilidade inadequado, incluindo a definição de um novo conceito chamado de *dívida de variabilidade*. Para isso, conduziu-se um estudo utilizando-se dois métodos de pesquisa: (i) uma revisão sistemática da literatura, e (ii) um estudo de caso multiprojetos com três sistemas reais. Os resultados apontam que as características identificadas nos estudos mapeados pela revisão sistemática da literatura: causas, artefatos e consequências de dívida de variabilidade ocorrem comumente em sistemas na prática e que os profissionais compreendem que o mau gerenciamento de variabilidade implica em diversos tipos de dívida técnica, com destaque para a causa de dívida de variabilidade de "pressão de tempo" que foi unanimidade de concordância entre os participantes da pesquisa. Entre as consequências da dívida de variabilidade identificadas na revisão sistemática e posteriormente avaliadas na pesquisa de campo com profissionais da indústria, destaca-se "problemas de usabilidade" e "dificuldade de manutenção".

Palavras-chave: Reúso oportunista. Linha de Produto de Software. Variante de Software.

Abstract

WOLFART, Daniele. **Variability Debt: A Multi-Method Study**. Orientador: Wesley Klewerton Guez Assunção. 2022. 94f. Dissertação (Mestrado em Ciência da Computação) – Universidade Estadual do Oeste do Paraná, Cascavel – Paraná, 2022.

Variability is the ability of a system or software artifact to be extended, customized or configured for use and reuse in different contexts. Managing variability is crucial to ensuring the success of a software project. What still varies is how to implement the variability. Although it can be implemented in a systematic way, such as using Software Product Lines (SPLs), many companies choose the way that initially requires a less investment and offers good time-to-market, opportunistic reuse, such as copy and paste. However, this kind of reuse can increase the occurrence of technical debts, a typical characteristic of a decision that presents a short-term advantage, but generates a long-term liability, such as difficulty in maintaining and evolving the system. Given their importance, both the concept of variability and the concept of technical debt are widely studied in Software Engineering, however, both concepts have not yet been investigated together. Thus, this work reports the results of a research to understand how technical debt is caused by inadequate variability management, including the definition of a new concept called variability debt. For this, a study was conducted using two research methods: (i) a systematic literature review, and (ii) a multi-project case study with three real systems. The results indicate that the characteristics identified in the studies mapped by the systematic review of the literature: causes, artifacts and consequences of variability debt commonly occur in practice and that professionals understand that poor variability management implies different types of technical debt, with emphasis on the debt cause of "time pressure" that was unanimous agreement among the research participants. Among the consequences of the variability debt identified in the systematic review and later evaluated in the field research with industry professionals, "usability problems" and "maintenance difficulties".

Keywords: Opportunistic Reuse. Software Product Line. Software Variant.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

ATD	Dívida Técnica Arquitetural
BD	Banco de Dados
DER	Diagrama Entidade Relacionamento
DT	Dívida Técnica
ESPLA	*Extractive Software Product Line Adoption*
LPS	Linha de Produto de Software
MER	Modelo Entidade Relacionamento
NC	Não Conformidade
OOPSLA	*Objected Oriented Programming System, Languages & Application*
RN	Regra de Negócio
SMB	*Samsung Mobile Business*
SBQS	Simpósio Brasileiro de Qualidade de Software
SPL	*software Product Line*
TDM	*Technical Debt Management*
TI	Tecnologia da Informação
URS	*User Requirement Specification*

Sumário

1 Introdução

Em todos os segmentos da sociedade, percebe-se uma grande demanda por produtos e serviços de software customizados. Pode-se associar esta necessidade a exigência de agências reguladoras[1], softwares que precisam executar em diferentes hardwares, ou ainda a necessidade de permitir a customização de um produto para satisfazer as necessidades específicas do usuário final (BERGER et al., 2020). A propriedade do software que permite lidar com sua customização, possibilitando a criação de variantes de um sistema para diferentes segmentos de mercado ou contextos de uso, é conhecida como *variabilidade*. Implementar e gerenciar variabilidade permite que o sistema seja customizável e reutilizável (CAPILLA; BOSCH; KANG, 2013).

O gerenciamento de variabilidade é a especificação das partes variáveis do software e que não são totalmente definidas durante o projeto inicial (GALSTER et al., 2014; BERGER et al., 2020). Esse gerenciamento facilita o desenvolvimento de diferentes versões de um sistema ou artefato de software. Porém, é desafiador criar sistemas ricos em variabilidade (GALSTER et al., 2014). Como a variabilidade permeia todo o ciclo de desenvolvimento de software, os engenheiros de software precisam de uma compreensão adequada, além de métodos e ferramentas apropriados. É necessário representar, gerenciar e raciocinar sobre a variabilidade desde o início do projeto. Muitas vezes, os engenheiros de software precisam prever o que as diferentes partes interessadas podem exigir de variabilidade no produto final, estarem cientes de onde a variabilidade do código será implementada, e consecutivamente testar todas as variantes possíveis (GALSTER et al., 2014).

Embora a variabilidade possa ser implementada de forma sistemática, como por exemplo, utilizando Linhas de Produtos de Software (LINDEN; SCHMID; ROMMES, 2007), para suportar combinações mais fáceis de funcionalidades, propagação de correções e melhorias personalizadas (LINDEN; SCHMID; ROMMES, 2007); existem várias barreiras para chegar a este ponto. A adoção de LPS requer um alto investimento inicial (ALI; BABAR; SCHMID, 2009; ASSUNÇÃO et al., 2017; KRUEGER; MAHMOOD; BERGER, 2020; KRUEGER, 2001). Além disso, prever de antemão a necessidade de adicionar variabilidade, por exemplo, em domínios imaturos, requer um grande esforço e por vezes é impossível (ASSUNÇÃO et al., 2017; WOLFART; ASSUNCAO; MARTINEZ, 2021). Alternativamente, mesmo muitas vezes cientes das abordagens sistemáticas, as empresas decidem adotar estratégias oportunistas de gerenciamento de variabilidade, como copiar e

[1] As agências reguladoras são órgãos governamentais que exercem o papel de fiscalização, regulamentação e controle de produtos e serviços de interesse público tais como telecomunicações, energia elétrica, serviços de planos de saúde, entre outros (IDEC, 2013).

colar (DUBINSKY et al., 2013), e adequar/adicionar as funcionalidades necessárias para a variação do novo produto (MAHDAVI-HEZAVEH; DREMANN; WILLIAMS, 2021), em detrimento de outros atributos de qualidade, como a sustentabilidade da família de produtos como um todo.

Evitar a variabilidade, com um produto incluindo todas as funcionalidades disponíveis, também não é uma solução ideal, pois pode levar a problemas de usabilidade, tais como, funcionalidades com nomes semelhantes e/ou muitas funcionalidades desnecessárias (SOUZA et al., 2019). Finalmente, independentemente do gerenciamento de variabilidade implementado, tomar decisões subótimas durante a evolução é um problema a médio e longo prazo. Por exemplo, a falta de anotação é uma barreira para o mapeamento de implementação de funcionalidades e impacta negativamente na compreensão, manutenção e consequentemente na evolução do programa (SOUZA et al., 2019).

1.1 Motivação

A metáfora da dívida técnica (DT) foi mencionada pela primeira vez por Cunningham (1992). Sua definição de "código não muito correto", continua sendo a mais citada, mas foi estendida para se referir às tarefas internas de desenvolvimento de software escolhidas para serem atrasadas, mas que correm o risco de causar problemas futuros se não forem feitas eventualmente. Assim, representa a dívida que a equipe de desenvolvimento incorre quando opta por uma abordagem fácil ou rápida para implementar ou alterar requisitos no curto prazo, mas com uma maior possibilidade de um impacto negativo a longo prazo, como dificuldade de manutenção (ALVES et al., 2016).

A implementação de variabilidade por meio de reúso oportunista é um cenário amplamente observado na indústria e sabe-se que esta prática pode aumentar a ocorrência de dívida técnica no projeto de software, como dificuldade de manutenção e evolução do sistema (BROWN et al., 2010). Por um lado, a DT é "uma coleção de construções do projeto ou implementação que são convenientes no curto prazo, mas estabelecem um contexto técnico que pode tornar as mudanças futuras mais caras ou impossíveis" (AVGERIOU et al., 2016). Por outro lado, "o gerenciamento de variabilidade engloba as atividades de representar explicitamente a variabilidade em artefatos de software ao longo do ciclo de vida, gerenciando dependências entre diferentes variabilidades e apoiando a instanciação das variabilidades" (SCHMID; JOHN, 2004). Tanto a DT quanto a gestão da variabilidade estão sendo tratadas como áreas de estudo da Engenharia de Software com suas próprias funções, metodologias e práticas (BERGER et al., 2013; KRUCHTEN; NORD; OZKAYA, 2019; LI; AVGERIOU; LIANG, 2015; BERGER et al., 2020).

1.2 Definição do Problema

Ambos conceitos de dívida técnica e gerenciamento de variabilidade ainda não foram investigados juntos, existindo uma lacuna na área de pesquisa acadêmica e na aplicabilidade no contexto industrial de desenvolvimento de software. De acordo com o conceito de DT existe um grande potencial de que aspectos de gerenciamento de variabilidade inadequado sejam fonte de dívida técnica (BROWN et al., 2010). Contudo, há uma falta de conhecimento e fonte de informação sobre as características de como um gerenciamento inadequado de variabilidade pode ocasionar ou agravar a ocorrência de dívidas técnicas. Portanto, a necessidade de compreender as características da dívida de variabilidade faz com que esse estudo seja necessário. Uma vez ciente das características da dívida de variabilidade, os profissionais podem adotar uma determinada postura frente as decisões no gerenciamento de variabilidade e assim controlar a dívida técnica em seus portfólios.

1.3 Objetivos

Esta dissertação tem como objetivo geral *"compreender como a dívida técnica é ocasionada por meio de um gerenciamento de variabilidade inadequado"*. De forma a complementar o objetivo geral proposto, os seguintes objetivos específicos foram definidos:

- Caracterizar o fenômeno de dívida técnica relacionada a gerenciamento de variabilidade;
- Reportar os principais tipos de dívida de variabilidade descritos na literatura;
- Examinar a ocorrência de dívida de variabilidade na prática;
- Avaliar a percepção de profissionais sobre dívida de variabilidade.

1.4 Organização do Trabalho

No Capítulo 2 são apresentados os conceitos fundamentais sobre a variabilidade, reúso de software e dívida técnica. O Capítulo 3 apresenta o trabalho proposto, com as questões de pesquisa, os detalhes da metodologia utilizada para revisão sistemática da literatura e para o estudo de caso, e os três objetos de estudo do estudo de caso realizado. Para finalizar, explica-se o protocolo de estudo de caso seguido. Os resultados e discussões são apresentados no Capítulo 4, descrevendo o resultado das questões de pesquisa do mapeamento sistemático, o catálogo de dívida de variabilidade, o detalhamento da coleta de dados, e as discussões sobre o resultado obtido. No Capítulo 5 serão apresentados os

trabalhos relacionados. Por fim, os conclusões do trabalho são apresentadas no Capítulo 6, juntamente com as limitações do trabalho e sugestões de trabalhos futuros.

2 Referencial Teórico

Neste capítulo, são apresentados os conceitos da Engenharia de Software referentes a ambos os tópicos de Variabilidade e Dívida Técnica. São descritos a definição, origem e aplicabilidade que permeiam cada tópico.

2.1 Variabilidade e Reúso Oportunista de Software

Variabilidade é a capacidade de um sistema de software ou artefato de software ser estendido, customizado ou configurado para uso e reúso em contextos específicos (GURP; BOSCH; SVAHNBERG, 2001). O gerenciamento de variabilidade é a especificação das partes do software que permanecem variáveis e que não são totalmente definidas durante o projeto inicial (GALSTER et al., 2014; BERGER et al., 2020). Isso facilita o desenvolvimento de diferentes versões de um sistema de software ou artefato de software (GALSTER et al., 2014). A variabilidade é uma propriedade inerente de sistemas de software. Ela representa a capacidade de criar variantes de sistema para diferentes segmentos de mercado ou contextos de uso, e até mesmo, criar variantes para diversos sistemas de um mesmo contexto de uso. Praticamente qualquer software de sucesso enfrenta eventualmente a necessidade de existir com múltiplas variantes. Permitir a variabilidade nos sistemas de software é crucial para garantir que os sistemas se adaptem com sucesso às necessidades dos clientes e para facilitar a reutilização de sistemas de software ou artefatos de software individuais (CZARNECKI, 2013; GALSTER et al., 2014).

A variabilidade introduz complexidade em todas as atividades da engenharia de software e exige métodos e ferramentas com gerenciamento de variabilidade que possam lidar com essa complexidade de maneira eficaz. A engenharia de um sistema variável equivale a projetar um conjunto de sistemas simultaneamente (CZARNECKI, 2013). Como resultado, requisitos, arquitetura, código, e os testes são inerentemente mais complexos do que na engenharia de um sistema único. Um desenvolvedor que deseja compreender uma determinada variante do sistema deve identificar as partes relevantes dos requisitos e do código e ignorar aquelas que pertencem exclusivamente a outras variantes. O responsável pela garantia da qualidade, verificando um sistema de variantes, deve garantir que todas as variantes relevantes estejam corretas. Os métodos e ferramentas que reconhecem a variabilidade potencializam as semelhanças entre as variantes do sistema, ao mesmo tempo que gerenciam as diferenças de maneira eficaz (CZARNECKI, 2013).

Implantar a variabilidade pode ser algo planejado no início do projeto de desenvolvimento ou uma mudança devido aos erros, manutenção ou novas necessidades impostas pelo(s) cliente(s) mais adiante. Esta visão da variabilidade interfere em todo o ciclo de

vida do software, desde o planejamento, desenvolvimento até a etapa de manutenção. De acordo com Svahnberg et al. (2005), uma forma para apoiar a variabilidade é atrasar as decisões de projeto concretas para o último ponto que é economicamente viável.

A variabilidade do software, além de ser modelada, fundamentada e discutida em muitas organizações, precisa ser implementada no sistema. A realização de um ponto de variação[1] pode ser alcançada por uma variedade de tecnologias e abordagens. A seleção da abordagem ideal é motivada por dois fatores. O primeiro é o nível de abstração em que o ponto de variação é explorado, desde a arquitetura até o nível de código. O segundo é o estágio do ciclo de vida em que o ponto de variação é vinculado, se a vinculação é permanente, bem como os estágios durante o qual as variantes podem ser adicionadas ao ponto de variação. A escolha do mecanismo de realização correto é de grande importância por duas razões. Uma delas é que muitas vezes é difícil alterar o mecanismo selecionado depois de escolhido. A razão para isso é que as variantes são escritas para operar com um mecanismo específico. Além disso, frequentemente, as variantes são escritas por outras unidades organizacionais ou mesmo outras organizações em conjunto, como no caso de ecossistemas de software, o que complica a mudança do mecanismo selecionado. A segunda razão é que, ao longo do tempo, muitos pontos de variação tendem a ser limitados cada vez mais tarde no ciclo de vida de desenvolvimento de software. Um mecanismo de realização rígido que complica este processo causará tensão na organização e ineficiências no desenvolvimento (CAPILLA; BOSCH; KANG, 2013).

Os mecanismos para acomodar a variabilidade incluem linhas de produtos de software, assistentes/ferramentas de configuração em software comercial, interfaces de configuração de componentes de software, composição de tempo de execução dinâmico de serviços da web e o reúso oportunista (GALSTER; AVGERIOU, 2011; KRüGER; BERGER, 2020).

A variabilidade tem sido estudada principalmente no domínio da LPS, para suportar combinações mais fáceis de características, propagação de correções e melhorias customizadas (GALSTER et al., 2014; CAPILLA; BOSCH; KANG, 2013; LINDEN; SCHMID; ROMMES, 2007). Porém, existem algumas barreiras até chegar a este ponto. Implementar LPS requer um alto investimento que nem todas empresas estão dispostas a pagar, além de ser necessário estar atrelado a um contexto com domínio maduro para mapear as funcionalidades adequadas, exigindo um grande esforço de tempo das equipes envolvidas. Por isso, muitas organizações preferem confiar em estratégias oportunistas de gerenciamento de variabilidade, como "copiar e colar" (DUBINSKY et al., 2013; BERGER et al., 2020), e adequar/adicionar as características necessárias para a variação do novo produto.

[1] Um ponto de variação identifica um ou mais locais em que a variação ocorrerá, enquanto a forma como um ponto de variação irá variar é expressa por suas variantes (TëRNAVA et al., 2022).

Segundo Krüger e Berger (2020) a reutilização de software é uma das práticas mais importantes para economizar custos de desenvolvimento e reduzir o tempo de colocação no mercado de sistemas de software. A ideia é evitar a reimplementação de funcionalidades existentes e reutilizá-las para novos sistemas, por exemplo, variantes. Assim, eles enfatizam que entre as estratégias de reutilização mais utilizadas estão "copiar e colar" e "orientação de plataforma". Copiar e colar reutiliza trechos de código para resolver problemas idênticos no mesmo ou em outro sistema. Como uma estratégia barata e prontamente disponível, as organizações normalmente começam com essa técnica, que é bem suportado pelos sistemas de controle de versão, como Git[2], e por plataformas de hospedagem de software, como GitHub[3]. No entanto, com um número crescente de variantes clonadas, a manutenção facilmente se torna um desafio, por exemplo, quando os desenvolvedores precisam propagar alterações, correções de *bugs* ou funcionalidades. Os desenvolvedores facilmente perdem a compreensão geral das variantes e são desafiados por sobrecargas de manutenção. Já a estratégia de reúso orientado a plataforma, baseia-se em um sistema inteiro ou suas partes (por exemplo, componentes, arquivos, pacotes ou módulos) para projetar uma variante nova, mas semelhante, que é customizada para cada parte interessada pelos requisitos, como hardware, recursos e ambientes de tempo de execução, ou aspectos não funcionais, como custo, desempenho, regulamentos ou consumo de energia (KRüGER; BERGER, 2020).

Seja pela necessidade de se implementar variabilidade tardiamente no ciclo de vida de desenvolvimento de software e ocasionando dificuldades técnicas e ineficiências no desenvolvimento (CAPILLA; BOSCH; KANG, 2013), ou, pela escolha de se implementar variabilidade de forma não sistematizada por meio da reutilização de software (DUBINSKY et al., 2013; BERGER et al., 2020), ambas podem acabar incorrendo em dívida técnica em seus sistemas.

2.2 Dívida técnica

A metáfora "*dívida técnica*" foi citada pela primeira vez por Ward Cunningham em 1992 no *Objected Oriented Programming System, Languages & Application* (OOPSLA). A metáfora foi descrita, em tradução livre, como (CUNNINGHAM, 1992):

> "... a primeira vez que a qualidade do código é comprometida é como se estivéssemos incorrendo em dívida. Uma pequena dívida acelera o desenvolvimento até que seja pago

[2] https://git-scm.com/
[3] https://github.com/

> através da reescrita do código. O perigo ocorre quando o dívida não é paga. Cada minuto em que o código é mantido em inconformidade, juros são acrescidos na forma de refatoração[4] ..."

Cunningham (1992) criou a metáfora da dívida técnica para descrever a dívida que a equipe de desenvolvimento assume quando escolhe um *design* ou abordagem fácil de implementar no curto prazo, mas com grande impacto negativo no longo prazo. Em poucas palavras "código não muito correto". O termo DT é usado para descrever a dívida que uma equipe de desenvolvimento incorre quando assume atalhos no processo de desenvolvimento de software, mas que pode aumentar a complexidade e manutenção a longo prazo (BROWN et al., 2010). Para Allman (2012), a dívida técnica é resultado da lacuna entre as boas práticas de desenvolvimento e fatores como a falta de tempo ou o custo de ferramentas. Em outras palavras, a dívida acontece quando membros da equipe buscam atalhos que ficam aquém das melhores práticas.

A dívida pode se tornar uma âncora que impede o sistema de evoluir de forma eficiente, ocasionando menor produtividade e maior custo de manutenção (EISENBERG, 2012). Contudo, é necessário diferenciar dívida técnica de defeitos ou falhas. Durante o período de testes, as falhas podem ser sintomas de dívida técnica, mas a maior parte dessas falhas que causam a dívida podem não ser identificadas durante esse período, deixando o software menos eficiente (CURTIS; SAPPIDI; SZYNKARSKI, 2012).

O problema da dívida técnica não é assumi-la, mas sim esquecê-la. Muitas empresas assumem a dívida para atingir um objetivo e a esquecem, deixando-a fugir de seu controle. Por esse motivo, o objetivo não deve ser eliminá-la, mas sim, gerenciá-la. Deve-se aceitar a dívida técnica como inevitável, porém um "plano para mudança" deve existir, caso algo necessite ser remediado (ALLMAN, 2012).

Em uma analogia, a dívida técnica pode ser comparada à dívida financeira. Toda vez que se incorre em uma dívida, é como se estivesse adquirindo um empréstimo financeiro. Cada minuto que essa dívida permanece no software, juros são acrescidos na forma de refatoração, da mesma forma que a cada momento em que o empréstimo não é quitado, o montante final a ser pago cresce incorporando juros em seu valor (LENARDUZZI et al., 2021). Escolher dívida como metáfora envolve um conjunto de conceitos financeiros que ajudam os executivos a pensar sobre a qualidade do software em termos de negócios. Abaixo são definidos os conceitos necessários para aplicar a metáfora da dívida técnica completa, de modo que cada fator possa ser medido e usado na análise financeira da qualidade estrutural das aplicações (CURTIS; SAPPIDI; SZYNKARSKI, 2012).

Dívida Técnica - os custos futuros atribuíveis a falhas estruturais conhecidas no código de produção que precisam ser consertadas. Um custo que inclui tanto o principal

[4] Refatoração é uma técnica utilizada na fase do desenvolvimento de software para melhorar a qualidade do sistema. É a reestruturação do código, de modo a alterar sua estrutura interna sem modificar sua estrutura externa (SHARMA, 2012).

quanto os juros. Uma falha estrutural no código de produção só é incluída nos cálculos de DT se os responsáveis pelo sistema acreditarem que é um problema obrigatório. A dívida técnica é o principal componente do custo de propriedade do software (CURTIS; SAPPIDI; SZYNKARSKI, 2012).

Principal - o custo de corrigir problemas obrigatórios no código de produção. No mínimo, o principal é calculado a partir do número de horas necessárias para corrigir problemas obrigatórios no código de produção, multiplicado pelo custo por hora totalmente onerado daqueles envolvidos no projeto, implementação e teste dessas correções (CURTIS; SAPPIDI; SZYNKARSKI, 2012; TOM; AURUM; VIDGEN, 2013).

Juros - os custos contínuos atribuíveis aos problemas obrigatórios no código de produção. Esses custos contínuos podem resultar do esforço excessivo para modificar código desnecessariamente complexo, maior uso de recursos por código ineficiente e custos semelhantes (CURTIS; SAPPIDI; SZYNKARSKI, 2012; TOM; AURUM; VIDGEN, 2013).

Risco de negócios - os custos potenciais para os negócios, se problemas devem ser corrigidos no código de produção, causando eventos operacionais prejudiciais ou outros problemas que reduzem o valor a ser derivado do aplicativo (CURTIS; SAPPIDI; SZYNKARSKI, 2012).

Problemas estruturais de qualidade dão origem a DT, que contém o principal e os juros da dívida. O custo para corrigir esses problemas estruturais constitui o principal desta dívida. Código com falhas estruturais cria ineficiências como maior esforço de manutenção ou computação excessiva, recursos cujos custos representam os juros da dívida. Além disso, os problemas estruturais subjacentes a DT também podem criar riscos de negócios. Quando esses riscos se traduzem em eventos operacionais negativos, eles criam um dano, como perdas, ou violação de segurança de informações. A remediação de dívidas técnicas requer cronograma e esforço que poderiam ter sido dedicados à criação de novas funcionalidades de negócios (CURTIS; SAPPIDI; SZYNKARSKI, 2012).

Para uma melhor compreensão da metáfora da dívida técnica alguns autores preferem categorizar as dívidas encontradas em seus estudos. É o caso de McConnell que divide dívida técnica em intencional e sem intenção. Se a dívida for classificada como intencional, o autor ainda faz outra divisão: a dívida a curto e a longo prazo (MCCONNELL, 2013).

Dívida técnica sem intenção é resultado não-estratégico de um trabalho mal planejado (MCCONNELL, 2013). A dívida sem intenção é contraída de forma involuntária. Como essa dívida não é esperada, pode-se dizer que ela causa um impacto negativo no projeto. Alguns exemplos de dívida técnica sem intenção são: “adquirimos uma nova empresa que havia acumulado dívida técnica e estamos lidando com isso”, “algumas funcionalidades não tiveram êxito na hora da apresentação para o cliente pois estas foram desenvolvidas

por um programador inexperiente" ou "havia problemas nos requisitos iniciais e por isso o design não ficou como o cliente esperava".

Já a dívida técnica intencional acontece normalmente por priorização de outras dívidas, mas também pode ser resultado de um cronograma muito rígido. A equipe pode contrair essa dívida numa atitude estratégica para otimizar o projeto. Mcconnell (2013) cita um exemplo para explicar a dívida técnica intencional: "Não temos tempo para fazer o merge dessas duas bases de dados, por isso vamos utilizar um código improvisado que as mantém sincronizadas e corrigir após o envio". Outros exemplos de dívida técnica intencional: "Hoje priorizaremos uma determinada funcionalidade pois o prazo de entrega está próximo e havíamos prometido essa parte do sistema funcionando para o cliente" ou "O código será desenvolvido sem testes e depois da entrega faremos os mesmos".

A dívida técnica a curto prazo é assumida quando não se possui uma solução mas se está próxima dela (MCCONNELL, 2013). A dívida a curto prazo é utilizada para cobrir pequenas lacunas de tempo, por exemplo, a necessidade da entrega de uma funcionalidade em um curto período. Mais exemplos desse tipo de dívida são: "Hoje não faremos a documentação completa das alterações feitas no código mas amanhã finalizaremos isso" ou "Todo custo com o café consumido esse mês será pago somente no final do mês".

Mcconnell (2013) vê a dívida técnica a longo prazo como uma decisão estratégica e proativa. Assumida com o intuito de suprir necessidades de grande porte, é uma dívida que, quando bem gerenciada, pode sobreviver anos dentro do projeto sem causar problema. Um exemplo de dívida técnica a longo prazo: "Esse algoritmo não é o melhor, mas supre as nossas necessidades pelos próximos 10 anos, então vamos continuar com ele".

Fowler (2014) define um quadrante para a classificação da dívida técnica. O autor a classifica em dívida técnica intencional e dívida técnica sem intenção, da mesma forma que Mcconnell (2013). Acrescenta-se, porém, uma nova divisão: a dívida técnica prudente e a dívida técnica imprudente, formando assim um quadrante, ilustrado na Figura 1.

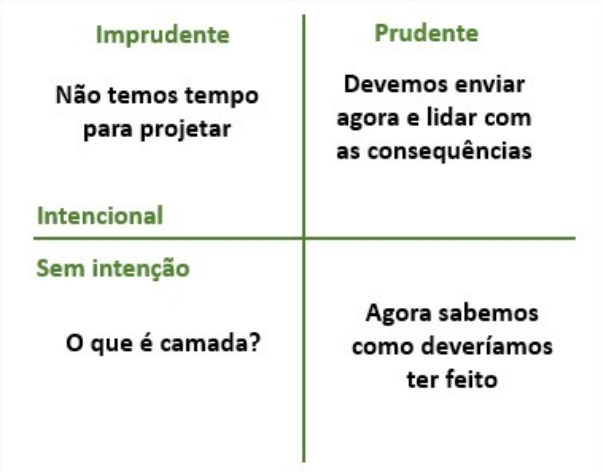

Figura 1 – Quadrante da Dívida Técnica, adaptado de (FOWLER, 2014)

Segundo Rosser e Norton (2021), a dívida imprudente geralmente surpreende as equipes de projeto e se torna um contribuinte significativo para estouros de custo e cronograma. O incorrimento imprudente em dívidas técnicas frequentemente tem suas raízes em uma necessidade urgente.A dívida técnica imprudente é uma sequência de más escolhas que podem resultar em altos juros ou em um longo período para pagamento da dívida. Uma dívida técnica imprudente, se intencional, é uma dívida assumida de forma negligente pela equipe. É perigosa pois, se for esquecida pode gerar juros impossíveis de pagar levando até mesmo à falência do projeto (FOWLER, 2014). O exemplo: "Contrataremos 5 funcionários por no mínimo 5 anos para o setor de manutenção" estaria incluído no quadrante de dívida técnica intencional-imprudente. Já o exemplo: "Como não sabemos o que é documentação, não faremos" seria classificado no quadrante de dívida técnica sem intenção-imprudente.

A dívida técnica prudente normalmente é gerenciada ao invés de excluída. É uma dívida pequena e que fica em um local pouco acessado, pois causa pouco impacto e pode ser mantida no projeto (FOWLER, 2014). O exemplo: "Amanhã faremos a documentação que não fizemos hoje" é incluído no quadrante de dívida técnica intencional-prudente, no entanto, o exemplo: "Mostramos o produto hoje para o cliente e descobrimos que acertamos na cor do *layout*" seria classificado como dívida técnica sem intenção-prudente.

De acordo com Rosser e Norton (2021) a dívida técnica aplicada com prudência, será identificada, avaliada como a melhor resposta de curto prazo, registrada e marcada para remediação antes da próxima execução de produção do sistema.

Ao analisar a implantação de variabilidade de forma não sistemática dentro do quadrante da classificação, a mesma se enquadra na categoria intencional imprudente ou prudente, dependendo de como a equipe técnica avaliou a implantação no início do projeto.

Desta forma, verifica-se que tanto o conceito de variabilidade de software quanto dívida técnica são extremamente relevantes para a área da engenharia de software, mas ainda não haviam sido investigados em conjunto, nos motivando a realizar este trabalho de pesquisa.

2.3 Considerações Finais

Este capítulo apresentou os principais conceitos envolvidos no tema do trabalho. Pode-se entender que variabilidade de software é a capacidade de um sistema ou artefato ser estendido, customizado ou configurado para uso e reúso em diferentes contextos. Além disso, podemos implementar a variabilidade de software de formas distintas, e a forma usualmente optada pelas empresas para ganhar agilidade é o reúso oportunista, conhecido por "copiar e colar".

A prática de implementar a variabilidade de uma forma não sistemática, ou seja, pelo reúso oportunista, acaba incorrendo em dívidas técnica. Este conceito foi criado em 1992 e serve para descrever a dívida que a equipe de desenvolvimento assume quando escolhe um *design* ou abordagem fácil de implementar no curto prazo, mas com grande impacto negativo no longo prazo.

O próximo capítulo apresentará um descritivo da metodologia que será utilizada para caracterizar a dívida técnica em softwares que implementam variabilidade de software por meio de práticas não sistemáticas.

3 Metodologia

Esta seção apresenta a metodologia de pesquisa utilizada para compreender como a dívida técnica é ocasionada através de um gerenciamento de variabilidade inadequado. A pesquisa baseia-se em um estudo multimétodos, focando tanto nos aspectos literários quanto práticos, sendo eles: revisão sistemática da literatura e estudo de caso multiprojetos.

3.1 Questões de Pesquisa

Para guiar os estudos e alcançar os objetivos (ver Seção 1.3) do presente estudo, formularam-se as seguintes questões, e subquestões, de pesquisa:

QP1. Quais as características de dívida técnica no gerenciamento inadequado de variabilidade? Esta questão de pesquisa tem como objetivo definir o conceito de dívida de variabilidade buscando referências na literatura de como o gerenciamento inadequado de variabilidade incorre em dívida técnica, identificando-se as causas, consequências, os artefatos impactados e os principais tipos de dívida técnica neste contexto, aqui chamados de tipos de dívida de variabilidade.

QP1.1 Quais são as causas da dívida de variabilidade? Esta QP visa explicar a causa, ou seja, motivadores ou forças, que levam as empresas a incorrer em dívidas de variabilidade.

QP1.2 Quais são os artefatos impactados pela dívida de variabilidade? Com esta QP investiga-se os diferentes tipos de artefatos impactados pela dívida de variabilidade ao longo do ciclo de vida dos produtos de software.

QP1.3 Quais são as consequências da dívida de variabilidade? Nesta QP o foco é raciocinar sobre os tipos de dívida de variabilidade e as consequências enfrentadas por empresas que acumularam dívida de variabilidade por lidar inadequadamente com o gerenciamento de variabilidade e reúso em famílias de produtos de software.

QP1.4 Quais são os principais tipos de dívida de variabilidade descritos na literatura? Essa questão de pesquisa tem como objetivo apresentar um catálogo dos principais tipos de dívida técnica que ocorrem quando há um gerenciamento de variabilidade implementado de forma inadequado.

QP2. Como a dívida de variabilidade ocorre na prática? Nesta QP o foco é compreender como a dívida de variabilidade ocorre na prática, confrontando os dados coletados na literatura, avaliando documentos e artefatos de sistemas reais e obtendo a opinião de profissionais sobre esse fenômeno.

QP2.1 Quais artefatos são afetados pela dívida de variabilidade? Esta QP tem como objetivo apresentar quais os artefatos, por exemplo, código fonte, documentação, casos de teste; são impactados pela dívida de variabilidade na prática.

QP2.2 Como os profissionais percebem a dívida de variabilidade? Essa QP tem como objetivo coletar a opinião dos profissionais quanto as características de dívida de variabilidade. Questionou-se sobre causas, artefatos, consequências e sobre a definição de dívida de variabilidade.

As questões QP1.1, QP1.2 e QP1.3 foram formuladas para atender o objetivo específico OE1 - Caracterizar o fenômeno de dívida técnica relacionada a gerenciamento de variabilidade. Enquanto que a questão QP1.4 visa atender ao objetivo específico OE2 - Reportar os principais tipos de dívida de variabilidade descritos na literatura.

A questão QP2.1 foi formulada para atender o objetivo específico OE3 - Examinar a ocorrência de dívida de variabilidade na prática. E para finalizar, a questão QP2.2 foi formulada para atender o objetivo específico OE4 - Avaliar a percepção de profissionais sobre dívida de variabilidade.

3.2 Processo de Pesquisa

Para atender os objetivos do estudo e responder as questões de pesquisa, este trabalho foi dividido e realizado em dois estudos conduzidos conforme apresenta a Figura 2. O primeiro deles realizado por meio de uma *revisão sistemática da literatura* para caracterização da dívida de variabilidade, coletando causas, consequências e artefatos envolvidos ao longo de todo este processo e para alcançar os objetivos específicos OE1 e OE2 e as questões de pesquisas QP1.1, QP1.2, QP1.3 e QP1.4. A segunda etapa realizada por meio da condução de um *estudo de caso multiprojeto* para compreender como a dívida de variabilidade ocorre na prática e qual a opinião dos profissionais envolvidos no processo sobre esse contexto. Nesta etapa, o objetivo é coletar evidências sobre as causas para a empresa ter optado por implementar reúso oportunista, se os sistemas causaram dívida técnica e, por fim, detectar os artefatos envolvidos no processo. Desta forma será possível alcançar aos objetivos específicos OE3 e OE4, respondendo as questões de pesquisas QP2.1 e QP2.2.

O resultado da primeira etapa da pesquisa foi utilizado como entrada para a segunda etapa para apoiar a formulação das questões utilizadas no questionário. Esta relação pode ser observada na Figura 2 com a fonte de informação “Características da dívida de variabilidade”. Todos os resultados encontrados no estudo da revisão sistemática foram posteriormente avaliados pelos profissionais da indústria em seu contexto.

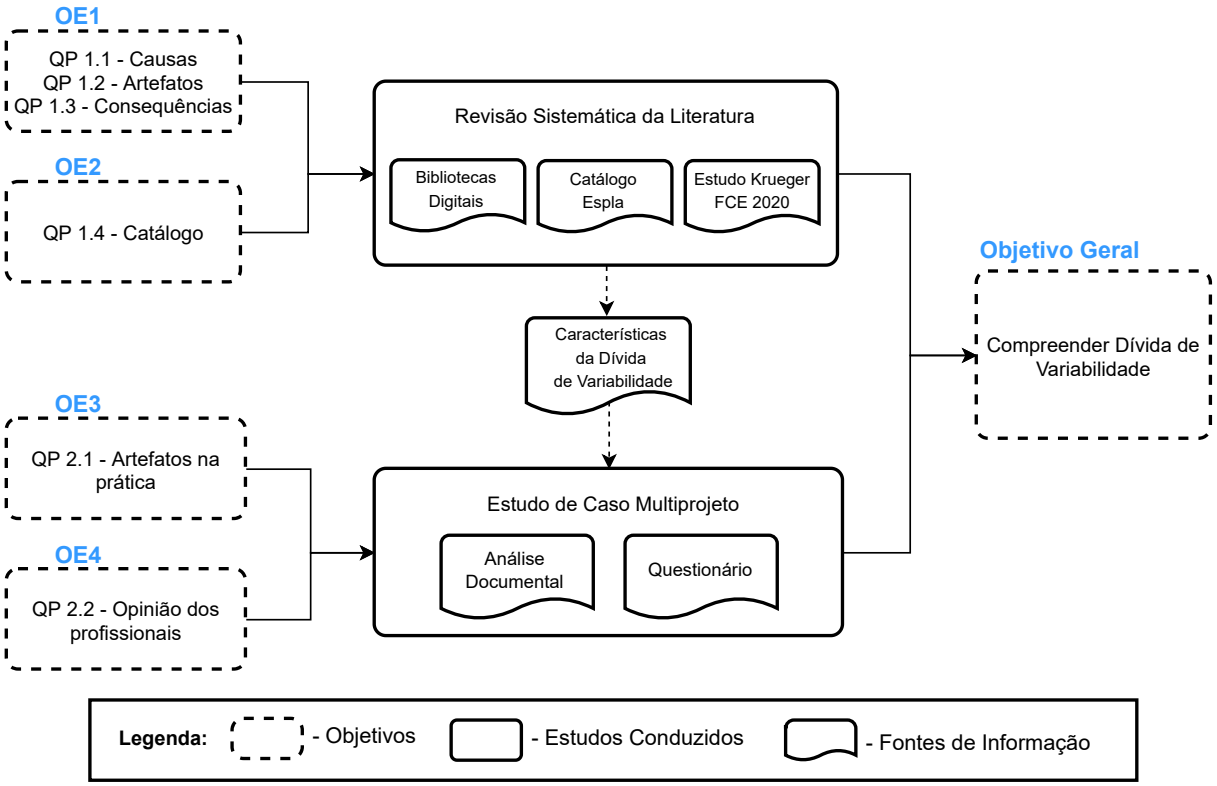

Figura 2 – Processo de Pesquisa

A estrutura de pesquisa deste trabalho está ilustrada na Figura 3. Percebe-se que a definição das questões de pesquisa foi a primeira etapa, conforme apresentado na Seção 3.1. Desta forma, o passo seguinte consistiu na definição do "tipo de pesquisa".

O tipo de pesquisa pode ser classificado como pesquisa básica ou aplicada. A pesquisa básica (também conhecida como pesquisa pura) é aplicada a um problema onde a ênfase é a compreensão do problema ao invés de fornecer uma solução para um problema, portanto, a principal contribuição é o conhecimento gerado a partir da pesquisa. Por outro lado, pesquisa aplicada é o tipo de pesquisa em que o pesquisador fornece uma solução para um problema específico, aplicando conhecimentos com o objetivo de melhorar a prática ou aplicação existente. Mas a pesquisa aplicada não pode ser realizada isoladamente. Ela conta com pesquisa básica, porque aplica o método científico do conhecimento básico em uma prática existente. Por exemplo, neste estudo investigamos qual a relação da implantação de um gerenciamento inadequado de variabilidade de software e sua implicação na dívida técnica. É provável que o resultado desta pesquisa seja o contexto de uma empresa específica, potencialmente com algumas experiências e conclusões de como ela pode ser aplicada também em outras empresas (WOHLIN; AURUM, 2014).

A próxima etapa definida para a estruturação da pesquisa deste trabalho foi referente a "lógica de pesquisa", que refere-se à direção em que a pesquisa prossegue, ou seja, se move do geral para o específico ou vice-versa. Existem duas formas comuns de raciocínio na pesquisa empírica de engenharia de software: pesquisa dedutiva versus

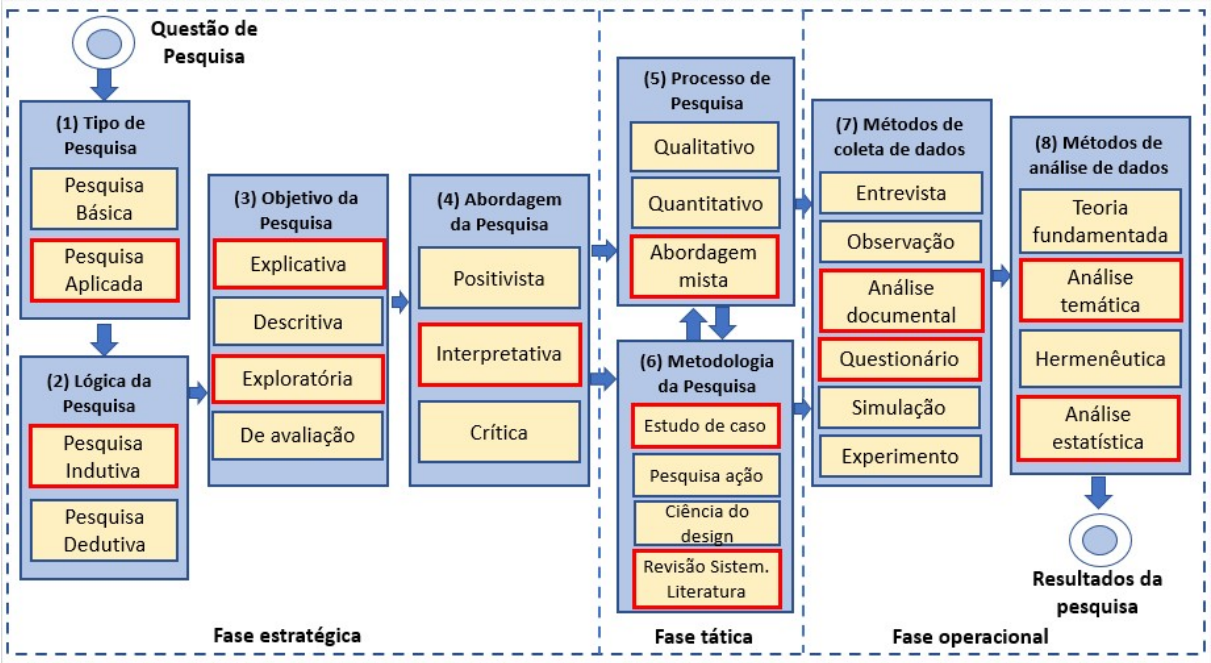

Figura 3 – Estrutura da pesquisa, adaptado de (WOHLIN et al., 2012)

pesquisa indutiva (RUNESON et al., 2012; COLLIS; HUSSEY, 2014). A pesquisa dedutiva começa com a teoria existente, estabelece hipóteses para a pesquisa e, finalmente, faz observações. Durante a análise, as observações confirmam ou rejeitam a hipótese, o que leva à confirmação ou teorias rejeitadas. A pesquisa indutiva é baseada em argumentos indutivos, ela se move do específico para o geral. O pesquisador infere conceitos teóricos e padrões de dados observados. O pesquisador segue com disciplinas específicas, detecta padrões teóricos e desenvolve algumas das teorias gerais. Neste trabalho utilizou-se uma lógica de pesquisa *indutiva* em um contexto industrial para observações dos fenômenos. Assim, o resultado também deve ser observado e interpretado a partir do contexto. Os resultados desta pesquisa podem ser aplicáveis a outros pesquisadores e profissionais que realizarão pesquisas em ambientes análogos.

Diferentes estratégias de pesquisa servem a propósitos diferentes. Um tipo de estratégia de pesquisa não atende a todos os propósitos. Podemos distinguir entre os seguintes quatro tipos gerais de fins de pesquisa, adaptados da classificação de Robson (RUNESON et al., 2012; COLLIS; HUSSEY, 2014):

- Exploratória - A pesquisa exploratória é aplicada quando não há muitas informações disponíveis na área do tópico e o pesquisador tem como objetivo reunir alguns *insights* sobre o problema. Deseja-se descobrir o que está acontecendo. O objetivo é explorar a área do problema e fornecer informações básicas que podem ser usadas para a pesquisa descritiva ou explicativa. Um estudo de caso exploratório pode gerar ideias e hipóteses para novas pesquisas.

- Descritiva - A pesquisa descritiva é, como o próprio nome sugere, aplicada para descrever um fenômeno ou características de um problema. É mais focada e vai além da pesquisa exploratória. As questões de pesquisa tendem a começar com "o quê" ou "como", uma vez que visa descrever os fenômenos.

- Explicativa - A pesquisa explicativa é aplicada ao examinar a natureza de certas relações entre os elementos de um problema. As questões de pesquisa tendem a começar com "por que", uma vez que visa explicar os fenômenos. Responder à pergunta "por que" pode envolver o propósito de pesquisa qualitativa e quantitativa, como o uso de entrevistas, pesquisas e análise de documentos. A questão crítica na pesquisa explicativa é identificar e controlar as variáveis de uma forma que as relações causais possam ser melhor explicadas.

- De avaliação - Um estudo de caso de avaliação é uma abordagem bem conhecida nas ciências sociais, pois envolve metodologias de pesquisa para julgar e melhorar o planejamento, monitoramento, eficácia e eficiência dos serviços humanos ou produtos que eles usam. Este tipo de pesquisa tenta melhorar um determinado aspecto do fenômeno estudado. A pesquisa pode usar ferramentas de pesquisa para descrever, explorar e avaliar as necessidades de diferentes grupos populacionais (por exemplo, equipes de software ou funcionários que usam um software específico), avaliando a eficácia de um determinado método, estrutura ou tecnologia de software. Para Runeson e Höst (2008) esse tipo de pesquisa é chamado de "aprimoramento".

A primeira parte desde estudo é classificado somente como pesquisa exploratória, uma vez que ela é aplicada quando não há muitas informações disponíveis na área do tópico e o pesquisador tem como objetivo reunir alguns *insights* sobre o problema. Como a primeira parte consiste de uma revisão sistemática para compreender a associação da dívida técnica a sistemas que implementam variabilidade de forma inadequada, este estudo de caso foi classificado como *exploratório*. A segunda parte da pesquisa, o estudo de caso, foi classificada como exploratória e descritiva, porque ao mesmo tempo que compreendemos como funciona a implantação de variabilidade, por meio de reúso oportunista e os impactos em dívida técnica, estamos obtendo a visão de profissionais da área técnica e explicando o que acontece na prática e como isso ocorre.

A quarta etapa, refere-se a "abordagem de pesquisa". Neste quesito Klein e Myers (1999) definem três tipos de estudos de caso, dependendo da perspectiva da pesquisa: positivista, crítico e interpretativo. Um estudo de caso positivista busca evidências para proposições formais, mede variáveis, testa hipóteses e desenha inferências de uma amostra para uma população declarada, que é próxima ao modelo de pesquisa de ciências naturais e relacionada à categoria explicativa de Robson (2002). A pesquisa crítica visa avaliar criticamente o sistema existente, com base nos pressupostos de que as variáveis sociais e

culturais impactam o sistema existente e que as interconexões não podem ser ignoradas. Na pesquisa crítica, o conhecimento é considerado subjetivo, dependendo de qual perspectiva o pesquisador assume. Esse tipo de pesquisa visa revelar contradições e conflitos dentro do sistema existente enquanto positivista e a pesquisa interpretativa visa prever ou explicar a situação atual. A pesquisa interpretativa visa compreender as atividades humanas em uma situação específica a partir da perspectiva dos participantes, portanto, enfatiza o contexto. Ela tem como objetivo compreender a estrutura mais profunda de um fenômeno dentro de situações culturais e contextuais onde o fenômeno é estudado em seu ambiente natural e da perspectiva do participante, sem incluir a compreensão prévia do pesquisador da situação. Considerando que nossa pesquisa ocorre em um contexto definido em que os resultados podem ser observados e interpretados, a *pesquisa interpretativa* é apropriada a este trabalho.

As etapas da fase estratégica da estrutura da pesquisa foram explicadas em detalhes. As fases tática e operacional descritas na Figura 3 serão explicadas ao longo da próxima seção.

3.3 Métodos de pesquisa

A seguir serão descritos detalhes dos métodos de pesquisas utilizados neste estudo: Revisão Sistemática da Literatura e Estudo de Caso. Ambos métodos fazem referência a etapa 6 da estrutura de pesquisa apresentada na Figura 3.

3.3.1 Revisão Sistemática da Literatura

A produção do conhecimento, por ser feita de forma coletiva, requer alguns cuidados por parte de quem se propõe a realizar uma pesquisa, pois quase sempre, uma nova pesquisa pretende abordar algum tópico que complemente ou que conteste aquilo que outros pesquisadores já afirmaram. Portanto, a formulação de um problema de pesquisa só se torna relevante quando o pesquisador, após uma análise crítica do estágio atual da produção científica de sua temática, consiga identificar lacunas, consensos e controvérsias sobre o tema e inserir o seu objeto de pesquisa num caminho ainda não percorrido por outros pesquisadores (BRIZOLA; FANTIN, 2016).

Neste sentido a Revisão Sistemática da Literatura é a junção de ideias de diferentes autores sobre determinado tema, conseguidas através de leituras, de pesquisas realizadas pelo pesquisador. A revisão sistemática da literatura também pode ser definida como a documentação feita pelo pesquisador sobre o trabalho, a pesquisa que está se propondo a fazer. A revisão sistemática da literatura ajuda: (i) delimitar o problema da pesquisa, (ii) auxiliar na busca de novas linhas de investigação para o problema que o pesquisador pretende investigar, (iii) evitar abordagens infrutíferas, ou seja, através da revisão

sistemática da literatura o pesquisador pode procurar caminhos nunca percorridos, (iv) identificar trabalhos já realizados, já escritos e partir para outra abordagem, e evitar que o pesquisador faça mais do mesmo, que diga o que já foi dito, tornando a sua pesquisa irrelevante (SHULL; SINGER; SJBERG, 2010).

Além disso, a revisão sistemática da literatura é de suma importância, já que é realizada para auxiliar o pesquisador a focar no seu verdadeiro objeto de pesquisa e não perder tempo com questões secundárias. Neste sentido dois aspectos da revisão da literatura devem ser levados em conta: (i) A revisão sistemática da literatura é feita para consumo próprio do pesquisador, para ajudá-lo a ter clareza sobre as principais questões teórico-metodológicas pertinentes ao tema escolhido e, (ii) feita para compor o trabalho, para ser parte integrante do trabalho, seja como um capítulo, como parte de um capítulo ou mesmo o trabalho todo (BRIZOLA; FANTIN, 2016; SHULL; SINGER; SJBERG, 2010).

Com o objetivo de caracterizar a dívida de variabilidade sob a ótica de suas causas, consequências e artefatos, realizou-se um estudo por meio da revisão sistemática de literatura em três bibliotecas digitais tradicionalmente usadas, sendo elas, Scopus[1], IEEE Xplore[2], e ACM Digital Library[3] (Acesso em 21 de abril de 2021). No entanto, apenas três publicações com quatro estudos de caso foram identificados na interseção de dívida técnica e variabilidade. Uma vez cientes da discussão sobre questões relacionadas à reutilização oportunista e reengenharia de variantes de sistema em SPLs (*Software Product Line*) para lidar com problemas de variabilidade (ASSUNÇÃO et al., 2017), decidiu-se focar adicionalmente em casos de adoção extrativa de LPS, pois considerou-se que esses são os casos mais notórios de problemas relacionados à variabilidade. Com base nisso, nossa análise conta com três fontes de informação: (i) uma busca por documento em bibliotecas digitais, (ii) uma análise sistemática dos casos disponíveis no *Catálogo de Adoção Extrativa SPL (ESPLA)* (MARTINEZ; ASSUNÇÃO; ZIADI, 2017), que é um esforço dirigido pela comunidade, e (iii) estudos de caso de uma recente revisão sistemática da literatura apresentada por *Krueger et al.* (KRüGER; BERGER, 2020) na reutilização de software orientado a clones e plataformas, que incluiu várias fontes: o próprio ESPLA, o *"SPLC hall of fame" cases* (SPLC, 2020), descrições de casos industriais de empresas que prestam serviços em torno de SPLs, uma consulta dedicada a bancos de dados científicos, outras referências baseadas em seus conhecimentos, e leituras de *snowballing*.

A Figura 4 apresenta o passo a passo da metodologia utilizada neste pesquisa para caracterização inicial de dívida de variabilidade. Mais detalhes de como a pesquisa foi realizada estão disponíveis na Seção 3 do Apêndice A deste trabalho.

[1] <https://www.scopus.com/>
[2] <https://ieeexplore.ieee.org>
[3] <https://dl.acm.org/>

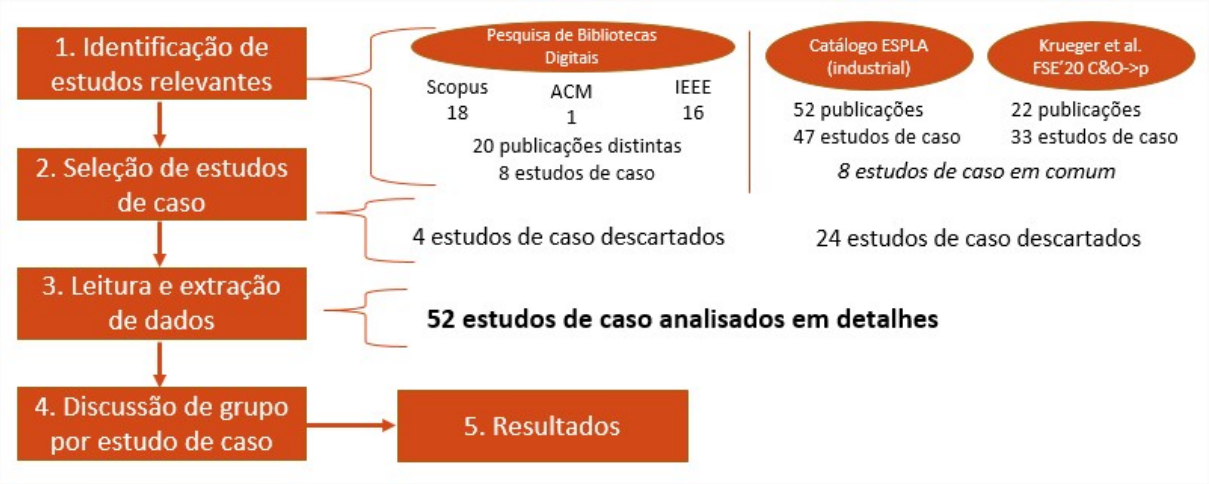

Figura 4 – Metodologia da Revisão Sistemática da Literatura

3.3.2 Estudo de Caso

O segundo método de pesquisa utilizado para alcançar os objetivos desse trabalho é o estudo de caso. O estudo de caso é uma estratégia de pesquisa geralmente usada em pesquisas que possuem caráter qualitativo, as quais procuram pesquisar e analisar dados com caráter mais profundo, descrevendo a complexidade e fornecendo maiores detalhes sobre as investigações, hábitos e tendências (LAKATOS.; MARCONI, 2006; RUNESON et al., 2012).

Neste sentido, Yin (2009) conceitua estudo de caso como uma investigação empírica que procura entender um fenômeno contemporâneo em profundidade e no seu contexto real, especialmente em situações em que o fenômeno e o seu contexto não são claramente evidentes. Os estudos de caso oferecem compreensão profunda do como e do porquê certos eventos ocorrem, e podem revelar os mecanismos de causa e efeito desse evento.

Para Runeson et al. (2012) as principais características de um estudo de caso são:

- É do tipo flexível, lidando com as características complexas e dinâmicas dos fenômenos do mundo real, como engenharia de software;
- Suas conclusões são baseadas em uma clara cadeia de evidências, sejam qualitativas ou quantitativas, coletadas de várias fontes de forma planejada e consistente; e
- Acrescenta ao conhecimento existente por ser baseado em teoria previamente estabelecida, se houver, ou por construir teoria.

O estudo de caso proposto é do tipo único e multiprojeto. Como Yin (2009) sugere estudos de caso múltiplos são aqueles conduzidos quando é possível a replicação em vários contextos/locais. Como não é o caso, este estudo é classificado como único. Além disso, ele é classificado como multiprojeto por conter mais de uma unidade de análise. Semelhante

a um estudo de caso, uma metodologia de estudo de caso multiprojeto fornece um meio de integrar métodos quantitativos e qualitativos em um único estudo de pesquisa. A abordagem de estudo de caso multiprojeto é particularmente relevante para o exame de um ambiente onde as fronteiras entre o fenômeno de interesse e o contexto não são claramente evidentes.

O estudo de caso deste trabalho foi realizado em uma indústria multinacional brasileira líder em seu segmento de mercado na América Latina. Esta indústria produz todos os bens necessário em sua cadeia de produtos, possuindo um departamento de Tecnologia da Informação (TI) responsável por fornecer soluções de software para todas as filiais ou departamentos da empresa. Com base nisso, a equipe do departamento de TI geralmente emprega a reutilização oportunista quando uma variante customizada de um sistema de software é necessária. Esses profissionais também são responsáveis por manter as variantes dos sistemas em operação, ou seja, para suas filiais ou departamentos.

3.3.2.1 Objetos de Estudo

Este estudo de caso é composto de três objetos de estudo: Sistemas[4] X, Y, e Z. Para estes três sistemas, têm-se sete variantes (X com 3 variantes, Y com 2 variantes, e Z com 2 variantes), que são as unidades de análise do estudo de caso. Estes objetos de estudo, e suas respectivas unidades de análise, serão analisados no mesmo contexto de negócio, ou seja, a mesma equipe técnica, gestores com cultura similar e que trabalham no mesmo grupo empresarial.

O Sistema X diz respeito à gestão de estoque de produtos acabados. O sistema tem por objetivo endereçar o material nos armazéns da empresa, realizar a gestão das posições, ou seja, movimentá-lo para posições livres e que caibam o material. Além disso, existem funcionalidades para separar o material de acordo com as quantidades dos pedidos de venda, realizar a devolução do material para o estoque e verificar saldo. A lista de todas funcionalidades disponíveis no sistema original e suas variantes está disponível na Tabela 1. Algumas funcionalidades deixam de existir nos sistemas variantes, por exemplo Conferir Remessa. Isso ocorre, pois o sistema foi implantado em uma filial do grupo organizacional que não pode realizar vendas, somente armazenar produtos. Já a funcionalidade Retirar Fracionado Blocado aparece somente no sistema Variante 1.

O Sistema Y tem por objetivo realizar a Gestão dos documentos e controle de impressão destes dentro da empresa. Existem funcionalidades para cadastro e aprovação de documentos, controle de impressão e reimpressão de documentos, solicitação de impressão adicional, bem como funcionalidades para conferir e armazenar as cópias impressas. A lista das funcionalidades está disponível na Tabela 2.

[4] Os nomes dos sistemas foram omitidos por motivo de confidenciabilidade.

Tabela 1 – Lista de Funcionalidades aplicáveis ao Sistema X

Funcionalidades	**Original**	**Variante 1**	**Variante 2**
Endereçar Blocado	✓	✓	
Verificar Saldo	✓	✓	✓
Movimentar Pallet	✓	✓	✓
Movimentar Começado	✓		✓
Picking Remessa	✓	✓	✓
Picking por UD	✓	✓	✓
Serviços Empilhadeira	✓	✓	✓
Fracionar Pallet	✓	✓	✓
Conferir Remessa	✓		✓
Montagem de Volume	✓		
Devolução	✓	✓	
Endereçar Pallet	✓		✓
Separar Pedidos	✓		✓
Conferir Posição	✓	✓	✓
Retirar Fracionado Blocado		✓	

O Sistema Z trata-se de um software para realizar a gestão de não conformidades, ou seja, ações ou situações que descumpriram um procedimento ou algo pre-estabelecido para toda e qualquer atividade da empresa, seja na fabricação de um produto, no desenvolvimento de software, no processo de compras, expedição, produção, etc. O sistema faz a gestão desde o lançamento da não conformidade, até a avaliação, investigação, criação do plano de ação, entre outras funcionalidades apresentadas na Tabela 3.

3.3.2.2 Protocolo do Estudo de Caso

Um protocolo de estudo de caso define os procedimentos detalhados para coleta e análise dos dados brutos, às vezes chamados de procedimentos de campo (RUNESON et al., 2012). O protocolo do estudo de caso é um documento em constante mudança. Existem vários motivos para manter uma versão atualizada de um protocolo de estudo de caso. Em primeiro lugar, um protocolo atualizado serve como um guia na condução da coleta de dados, e dessa forma evita que o pesquisador deixe de coletar os dados que foram planejados para serem coletados. Em segundo lugar, os processos de formulação do protocolo tornam a pesquisa concreta na fase de planejamento, o que pode ajudar o pesquisador a decidir quais fontes de dados usar e quais perguntas fazer. Terceiro, outros pesquisadores e pessoas interessadas no assunto podem revisar o protocolo e fornecer informações ou comentários para melhorias. Esta revisão do protocolo pode, por exemplo, diminuir o risco de perder fontes de dados relevantes, perguntas da entrevista ou funções a serem entrevistadas. Como outro exemplo, a revisão do protocolo também pode ajudar a esclarecer a relação entre as questões da pesquisa e os dados que estão sendo coletados, por exemplo, as questões da pesquisa e as questões da entrevista (RUNESON et al., 2012).

Tabela 2 – Lista de Funcionalidades aplicáveis ao Sistema Y

Funcionalidades	**Original**	**Variante 1**
Cadastro de Parâmetros Globais	✓	✓
Cadastro de Impressora	✓	✓
Cadastro de Documento exceção Material	✓	
Cadastro Tipo de Documento	✓	✓
Associar Tipo de Documento a centro	✓	✓
Cadastro de Categoria de Documento	✓	✓
Associar Categoria de Documento a centro	✓	✓
Vincular Documentos por categoria	✓	✓
Cadastro de Documento	✓	✓
Cadastro de Referências Farmacopeicas	✓	
Cadastro de Materiais Preenchimento Automático	✓	
Cadastro de Documento com Dado de entrada	✓	
Gerenciar Documentos	✓	✓
Avaliar Documentos	✓	
Consultar histórico de documentos	✓	✓
Autorizar Impressão e Reimpressão	✓	✓
Consultar Solicitações	✓	✓
Reutilizar siglas	✓	
Impressão de Documentos	✓	✓
Solicitar impressão adicional	✓	✓
Solicitar reimpressão	✓	✓
Verificar impressões de documentos de OF	✓	
Conferir impressões de documentos	✓	✓
Relatório Documentos impressos - Mensal	✓	✓
Relatório Páginas impressas - Mensal	✓	✓
Relatório Documentos impressos - Diário	✓	✓
Relatório Páginas impressas - Diário	✓	✓
Relatório Impressão por Tipo	✓	✓
Relatório Impressão: Top 5	✓	✓
Relatório Conferências de Impressão	✓	✓
Relatório Entrada de Lotes em CQ	✓	
Relatório Lista Mestre de Metodologia	✓	
Relatório Alterações Status Documentos	✓	✓
Relatório Documentos x Ref. Farmacopeicas	✓	

O protocolo de pesquisa seguido nesta pesquisa é baseado nas recomendações de Yin (2009). Mais detalhes serão apresentados na próxima seção.

3.3.2.3 Coleta de Dados

Existem muitas fontes diferentes de informações que podem ser usadas em um estudo de caso. É importante usar várias fontes de dados em um estudo de caso para limitar os efeitos de uma interpretação de uma única fonte de dados. Se a mesma conclusão pode ser obtida de várias fontes de informação, ou seja, triangulação. Essa conclusão é mais forte do que uma conclusão baseada em uma única fonte. Em um estudo de caso, também

Tabela 3 – Lista de Funcionalidades aplicáveis ao Sistema Z

Funcionalidades	**Original**	**Variante 1**
Cadastro Grupo Problema	✓	✓
Cadastro de Subgrupo Problema	✓	✓
Cadastro de Salas	✓	✓
Cadastro de Tipos de Ação	✓	✓
Administração de usuários	✓	✓
Registro de NC Interna	✓	✓
Registro de NC Fornecedor/Fabricante	✓	
Registro de NC CAC	✓	
Ações - Listagem	✓	✓
Ações - Visualizar	✓	✓
Ações - Alterar para Em Andamento	✓	✓
Ações - Gerenciar/Concluir	✓	✓
Ações - Prorrogar	✓	✓
Ações - Abortar	✓	✓
Gestão de NC - Consulta	✓	✓
Gestão de NC - Alteração de Registros	✓	✓
Gestão de NC - Classificação de Não conformidade	✓	✓
Gestão de NC - Investigação de Não conformidade	✓	✓
Gestão de NC - Gerenciamento do CAPA	✓	✓
Relatório de Nível de Criticidade por U.O	✓	✓
Relatório de Situação por U.O	✓	✓
Relatório de Grupo Problema por U.O	✓	✓
Relatório de Subgrupo Problema por Grupo e U.O	✓	✓
Relatório de Responsável pela Classificação		✓
Relatório de Responsável pela Investigação		✓
Relatório de Responsável pelo CAPA	✓	✓
Relatório de Ações por Responsável	✓	✓
Relatório do CAPA da NC	✓	✓

é importante levar em consideração pontos de vista de diferentes funções e investigar diferenças, por exemplo, entre diferentes projetos e produtos. Normalmente, as conclusões são tiradas analisando as diferenças entre as fontes de dados (RUNESON; HöST, 2008; WOHLIN et al., 2012).

De acordo com Lethbridge, Sim e Singer (2005) as técnicas de coleta de dados podem ser divididas em três níveis:

- Primeiro nível: Métodos diretos significam que o pesquisador está em contato direto com os sujeitos e coleta dados em tempo real. Este é o caso com, por exemplo, entrevistas, grupos de focais, pesquisas Delphi, e observações com "pense em voz alta".

- Segundo nível: Métodos indiretos onde o pesquisador coleta dados brutos diretamente sem realmente interagir com os sujeitos durante a coleta de dados. Essa abordagem é, por exemplo, tomada em Telemetria de Projeto de Software, onde o uso de

ferramentas de engenharia de software é monitorado automaticamente e observado por meio de gravação de vídeo.

- Terceiro nível: Análise independente de artefatos de trabalho onde já estão disponíveis e, às vezes, são usados dados compilados. Este é o caso, por exemplo, quando documentos como especificações de requisitos e relatórios de falhas de uma organização são analisados ou quando dados de bancos de dados organizacionais, como contabilidade de tempo, são analisados.

Para o estudo de caso deste projeto serão utilizadas fontes de informações de segundo e terceiro nível: *questionário* e *análise documental* (etapa 7 da Figura 3).

A análise documental é a investigação de dados históricos concretos que são arquivados por alguém ou por organizações. Pode exigir permissão para acessar os dados ou pode estar disponível publicamente. O método envolve localizar os dados, coletar sistematicamente os dados, analisar e interpretar o dados (PRODANOV; FREITAS, 2013).

Quanto a análise de dados arquivados, Wohlin et al. (2012), Runeson et al. (2012) destacam que estes se referem a, por exemplo, atas de reuniões, documentos de diferentes fases de desenvolvimento, dados de falha, organogramas, registros financeiros e outras medidas coletadas anteriormente em uma organização. Mas que estes documentos não foram originalmente desenvolvidos com o intuito de fornecer dados para a pesquisa. Obviamente, é difícil para o pesquisador avaliar a qualidade dos dados, embora algumas informações possam ser obtidas investigando o propósito da coleta de dados original e entrevistando pessoas relevantes na organização, como é a proposta deste estudo.

Neste trabalho a análise documental foi realizada utilizando os seguintes documentos: URS (*User Requirement Specification*), Especificações, Protótipos, código-fonte, MER (Modelo Entidade Relacionamento) e Casos de Teste.

Já o questionário, é uma técnica de coleta de dados que consiste em uma série ordenada de questões que devem ser respondidas por escrito pelo participante, sem que haja a presença do pesquisador (LAKATOS.; MARCONI, 2003). A linguagem do questionário deve ser simples e direta para que suas questões sejam compreendidas com facilidade pelo respondente. O questionário também deve ser objetivo, limitado em extensão e estar acompanhado de instruções que expliquem a sua natureza e a importância e a necessidade das respostas (PRODANOV; FREITAS, 2013). O pesquisador que desenvolver o questionário deve estar ciente dos tipos de perguntas que podem ser feitas e qual desses tipos trará melhores resultados. Elas podem ser abertas ou fechadas. A primeira permite ao informante responder livremente e emitir opiniões, a segunda, onde o respondente escolhe uma opção entre duas fornecidas, ou de múltipla escolha, as quais são perguntas fechadas mas que em sua resposta, apresentam mais do que apenas duas opções, abrangendo várias facetas do mesmo assunto (LAKATOS.; MARCONI, 2003).

A Tabela 4 apresenta as questões realizadas na pesquisa de questionário. O questionário utilizado no projeto foi híbrido mesclando perguntas abertas e fechadas. Para as opções de respostas foi utilizada a escala Likert[5] contemplando os extremos com "Concordo totalmente" e "Discordo totalmente", disponibilizando diferente níveis de opinião aos entrevistados. Além disso, foi adicionado uma opção de resposta "Não sei opinar", para o participante que não tenha participado daquela etapa, ou não tenha entendido o contexto da pergunta. Por fim, o questionário foi divido em cinco seções: Perfil do profissional, questões técnicas para caracterização da dívida de variabilidade, questões para compreender as causas do reuso oportunista em variabilidade de software, questões técnicas para compreensão das consequências da implementação de sistema com reuso oportunista e questões para auxiliar na definição do termo dívida de variabilidade.

O universo de possíveis profissionais a responder esse questionário, que estavam diretamente envolvidos no processo de desenvolvimento e manutenção do software, era de 28 pessoas. Uma das pessoas envolvidas é a própria desenvolvedora deste trabalho e não seria adequado responder a pesquisa pois poderia tendenciar as respostas para um pré-julgamento que a pesquisadora possui. Outros dois participantes não trabalham atualmente na empresa e não foram contactados.

Desta maneira o questionário foi enviado para 25 profissionais com envolvimento e participação no processo de desenvolvimento de software dos três sistemas objeto de estudo, ou que tenham participado do processo de evolução/manutenção do sistema. Destes, obteve-se um retorno de 22 profissionais, o que representa uma participação efetiva de 88% dos envolvidos.

3.3.2.4 Análise de Dados

A análise de dados foi conduzida de forma diferente para dados quantitativos e qualitativos. Para dados quantitativos, a análise normalmente inclui análise de estatísticas descritivas, análise de correlação, desenvolvimento de modelos preditivos e teste de hipóteses. Todas essas atividades são relevantes na pesquisa de estudo de caso. Por outro lado, o objetivo básico da análise qualitativa é obter conclusões dos dados, mantendo uma cadeia de evidências clara. A cadeia de evidências significa que um leitor deve ser capaz de seguir a derivação de resultados e conclusões a partir dos dados coletados. Portanto, apresenta-se informações suficientes de cada etapa do estudo e decisões importantes tomadas pelo pesquisador. Além disso, a análise da pesquisa qualitativa caracteriza-se por ter a análise realizada paralelamente à coleta de dados e pela necessidade de técnicas

[5] Foi desenvolvida nos Estados Unidos na década de 30, e ao contrário de uma pergunta na qual se escolhe entre o sim e o não, as questões construídas a partir da escala Likert apresentam uma afirmação autodescritiva. A escala Likert oferece como opção de resposta uma escala de pontos com descrições verbais que contemplam extremos – como "concordo totalmente" e "discordo totalmente", permitindo que seja descoberto diferentes níveis de intensidade da opinião a respeito de um mesmo assunto ou tema (Frankenthal, Rafaela, 2022).

Tabela 4 – Questões aplicadas na pesquisa

ID	Seção	Questão	Tipo	Categoria
1	1	Qual o seu nome?	Aberta	Opcional
2	1	Qual seu tempo de experiência profissional?	Fechada	Obrigatória
3	1	Indique qual o sistema você responderá as questões:	Fechada	Obrigatória
4	1	Por quanto tempo você teve envolvimento no sistema indicado anteriormente?	Fechada	Obrigatória
5	1	Qual era sua função no Projeto ou Departamento quando participou do desenvolvimento do sistema? (Caso teve mais de uma função, favor sinalizar)	Aberta	Obrigatória
6	1	Quanto tempo de experiência você possuía nesta função quando lidou com o sistema indicado anteriormente?	Fechada	Obrigatória
7	2	Para o projeto X (lê-se o nome do sistema que você teve participação), o uso da prática de copiar e colar na criação de diferentes variantes deste sistema teve um benefício inicial, mas incorreu em problemas de manutenção e evolução das variantes (sistemas) posteriormente?	Fechada	Obrigatória
8	2	Se você concordou com a pergunta acima, poderia relatar algum problema que você enfrentou?	Aberta	Opcional
9	2	Para as variantes do sistema que você teve contato, quais os artefatos que você lidou durante o processo de desenvolvimento de software e implantação do sistema? (por exemplo, requisitos, modelos de design, manuais, código-fonte, etc)	Aberta	Opcional
10	3	Para o projeto X (lê-se o nome do sistema que você teve participação) até que ponto você concorda que os seguintes fatores são motivos para utilização de uma prática não-sistemática (ex: copiar e colar) para criar diferentes variantes de software? *(Pressão de Tempo; Falta de Conhecimento; Mudanças constantes de requisitos; Restrições operacionais)*	Fechada	Obrigatória
11	3	Para os motivos que você concorda acima, você poderia fornecer mais informações citando exemplos, ou descrevendo como ocorreu este processo?	Aberta	Opcional
12	3	Existem motivos adicionais que justificaram a escolha por uma prática não sistemática na criação das variantes do projeto X?	Aberta	Opcional
13	4	Para o projeto X, até que ponto você concorda que os problemas a seguir são consequências do uso de uma abordagem não-sistemática para o gerenciamento da variabilidade da família de produtos? *(Manutenção complexa; Incapacidade de lidar sistematicamente com a customização; Incapacidade de criar sistemas complexos; Má qualidade interna do software; Ciclos complexos de testes; Problemas de usabilidade;Problemas na gestão de portfólio;Papéis repetidos em projetos similares)*	Fechada	Obrigatória
14	4	Para as consequências que você concorda acima, você poderia fornecer mais informações citando exemplos, ou descrevendo como ocorreu este impacto?	Aberta	Opcional
15	4	Você consegue indicar outras consequências no projeto X devido a implementação pela prática não-sistemática (uso de copiar-colar)?	Aberta	Opcional
16	5	Um novo termo foi criado ao longo do desenvolvimento deste trabalho: Dívida de Variabilidade. "Dívida de Variabilidade representa a Dívida técnica causada por problemas e soluções abaixo do ideal na implementação de gerenciamento de variabilidade em sistemas de software. Ou seja, as empresas podem optar por implementar a variabilidade com uma abordagem que traria benefícios de curto prazo, por exemplo, usando a reutilização oportunista, independentemente das desvantagens técnicas / arquitetônicas a médio e longo prazo. A dívida de variabilidade leva a dificuldades de manutenção e evolução para gerenciar famílias de sistemas ou sistemas altamente configuráveis."Até que ponto você concorda que a metáfora é claramente descrita com relação ao seu contexto de uso e propósito para criar consciência nas empresas?	Fechada	Obrigatória
17	5	Você tem sugestões/comentários sobre a definição do termo Dívida de Variabilidade?	Aberta	Opcional
18	5	Até que ponto você concorda que apoiar as seguintes atividades é importante? *(Identificação: Detecção de itens de dívida técnica associados à variabilidade; Medição: estimar o custo e o benefício associados a um item de dívida de variabilidade; Priorização: Identifique qual item de dívida de variabilidade deve ser tratado primeiro; Reembolso: mudanças e transformações ajudam a eliminar o efeito negativo de um item de dívida de variabilidade; Monitoramento: observe as mudanças dos custos e benefícios de itens de dívida de variabilidade não resolvidos ao longo do tempo.*	Fechada	Obrigatória

de análise sistemática. A análise deve ser realizada em paralelo com a coleta de dados, uma vez que a abordagem é flexível e novos *insights* são encontrados durante a análise.

A fim de investigar esses *insights*, novos dados devem ser frequentemente coletados e instrumentação, como questionários de entrevista, deve ser atualizada (RUNESON; HöST, 2008; RUNESON et al., 2012; WOHLIN et al., 2012).

Para finalizar a estrutura de pesquisa definindo como são conduzidas as etapas 5 e 8 (de acordo com a Figura 3) deste estudo, utilizou-se análise de dados *qualitativos* e *quantitativos*.

Os dados quantitativos analisados foram:

- Documentos duplicados;
- Pastas duplicadas entre os sistemas, e;
- Arquivos de código repetidos ou implementados diferentemente.

Para avaliar os dados qualitativos, utilizamos:

- Problemas de gestão de portfólio;
- Complexidade para evolução nos sistemas e/ou a arquitetura dos sistemas;
- Opinião quanto as características de dívida de variabilidade identificadas no processo de revisão sistemática da literatura (causas, artefatos e consequências).

3.3.2.5 Ferramentas utilizadas

Para o trabalho desenvolvido foi utilizada a IDE de desenvolvimento *Eclipse neon modeling distribution*[6]. Esta ferramenta foi utilizada para manipular o artefato de código-fonte dos objetos de estudo. Além disso, utilizamos a ferramenta But4Reuse[7] que fornece uma estrutura unificada para análise de variantes de artefatos de software, neste trabalho o código-fonte(MARTINEZ et al., 2017). But4Reuse é projetado para ser genérico e extensível. Genérico ao permitir seu uso em diferentes cenários com variantes de produto de diferentes tipos de artefatos de software (por exemplo, código-fonte em Java, C, modelos, requisitos ou arquiteturas baseadas em *plug-ins*) e extensível ao permitir adicionar diferentes técnicas concretas ou algoritmos para o atividades de adoção de SPL extrativo (ou seja, identificação de recursos, localização de recursos, restrições de recursos de mineração, extração de ativos reutilizáveis, síntese de modelos de recursos e visualizações). But4Reuse é uma ferramenta integrada ao Eclipse que fornece recursos para simplificar o processo de identificação de recursos, em particular para gerenciar a localização das variantes de software e para visualizar de diferentes perspectivas os resultados da identificação automática de recursos.

[6] Disponível para download no site da Eclipse: <http://www.eclipse.org/downloads/packages/eclipse-modeling-tools/neon2/>

[7] Disponível para download no site: <https://but4reuse.github.io/>

Após realizar a instalação da ferramenta But4Reuse, deve-se atualizar as preferências (pela ferramenta Eclipse) conforme Figura 5, que demonstra a configuração utilizada para as análises conduzidas neste trabalho:

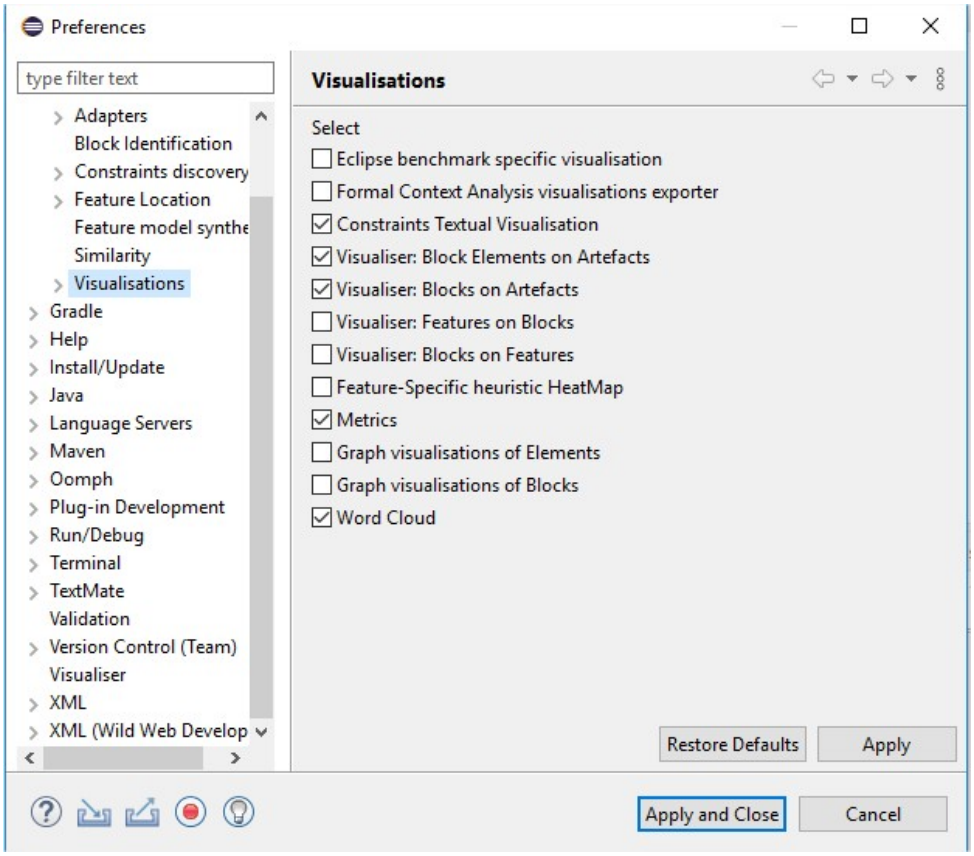

Figura 5 – Preferencias de Visualização

3.4 Considerações Finais

Este capítulo apresentou toda a metodologia utilizada neste trabalho, desde os conceitos da metologia até o descritivo do passo-a-passo de como o trabalho será executado. O trabalho foi conduzido por meio de dois métodos pesquisa: primeiramente a *Revisão Sistemática da Literatura*, para embasar a definição dos termos e caracterizar a dívida de variabilidade, identificando quais são as principais dívidas técnicas que acontecem em softwares que implementam variabilidade por meio do reúso de software, e, na sequencia o *Estudo de Caso*, para avaliar os impactos da dívida de variabilidade em cases reais e compreender como os profissionais lidam com essa situação no dia-a-dia.

Todos os resultados das etapas conduzidas seguindo a metodologia descrita aqui, são apresentadas no próximo capítulo.

4 Resultados e Discussões

Este capítulo apresenta os resultados obtidos por meio da revisão sistemática da literatura para caracterizar a dívida de variabilidade. Além disso, descreve os achados do estudo de caso multiprojetos conduzido para avaliar como o gerenciamento inadequado de variabilidade impacta a dívida técnica na prática.

4.1 Características da Dívida de Variabilidade

Os resultados da pesquisa sobre Caracterização da dívida de variabilidade foram publicados em um artigo no XX Simpósio Brasileiro de Qualidade de Software - SBQS (2021) que encontra-se na integra no Apêndice A.

Uma das contribuições deste trabalho, é a definição do conceito de "Dívida de Variabilidade", conforme apresentado abaixo como resposta a QP1. Os resultados e análises para responder as sub-questões de pesquisa são apresentados nas subseções seguintes.

> ***Resposta da QP1**: Dívida de Variabilidade é a dívida técnica causada por defeitos e soluções sub-ótimas na implementação da gestão da variabilidade em sistemas de software. Ou seja, as empresas podem optar por implementar a variabilidade com uma abordagem que traga benefícios no curto prazo, por exemplo, usando o reuso oportunista, independentemente das desvantagens técnicas/arquiteturais no médio e longo prazo. A dívida de variabilidade leva a dificuldades de manutenção e evolução para gerenciar famílias de sistemas ou sistemas altamente configuráveis.*

4.1.1 QP1.1 - Causas da Dívida de Variabilidade

A Tabela 2 do Apêndice A apresenta as forças motrizes que causam a dívida de variabilidade. Um dos principais motivos pelos quais as empresas incorrem em dívidas de variabilidade está relacionado à *pressão do tempo*. As empresas podem decidir não dedicar tempo para projetar a variabilidade adequadamente, por exemplo, usando a reutilização oportunista, para diminuir o *time-to-market* ou reduzir custos de desenvolvimento com uma visão de curto prazo.

Outra situação que leva as empresas a falhar em lidar com a gestão adequada da variabilidade, às vezes devido à pressão do tempo, diz respeito a cenários em que os *requisitos mudam constantemente*. Podem ser requisitos funcionais onde os desenvolvedores devem responder à demanda constante para alterar ou adicionar novas funcionalidades ou lidar com requisitos não-funcionais, como a implantação em um novo hardware.

A *falta de conhecimento sobre gestão da variabilidade* também é um fator que pode levar empresas a incorrer em dívidas de variabilidade. Decisões rápidas ou fracas tomadas no início do projeto para implementação da variabilidade são mencionadas em vários estudos de caso. Por exemplo, uma empresa pode se destacar no gerenciamento da variabilidade no código-fonte, mas, na maioria dos casos, as variações também estão presentes em ativos essenciais que não são de software. Por exemplo, funcionalidades opcionais podem ser relacionados a documentos de requisitos e/ou procedimentos de teste. Além da implementação da variabilidade em todos esses níveis, uma rastreabilidade global entre todos esses artefatos de funcionalidades é desejada. Quando os profissionais não conhecem nenhum mecanismo de variabilidade, percebe-se um excesso de engenharia para construir produtos por projeto, por exemplo, levando à duplicação de artefatos, integração de funcionalidades desnecessárias e indesejadas, desenvolvimento de funcionalidades raramente usadas e aumento da complexidade do código.

Diversas *restrições operacionais* podem gerar dívidas de variabilidade. Em primeiro lugar, os processos estabelecidos na empresa para lidar com similaridades e a variabilidade entre todos os clientes atuais e potenciais devem estar em vigor. A falta de práticas de análise de domínio pode impedir a possibilidade de criar uma arquitetura e infraestrutura projetada especificamente para suportar uma LPS. Em segundo lugar, configurações operacionais complexas podem comprometer uma visão global e um plano para a família de produtos, por exemplo, integração de equipes distribuídas geograficamente ou sistemas resultantes de fusão de diferentes empresas. Além disso, sistemas distribuídos nos quais apenas modificações locais são feitas podem criar problemas porque um número significativo de variantes podem surgir. As restrições operacionais também podem incluir esforços para acomodar código legado e de terceiros.

Resposta da QP1.1: *Como resultado da questão de pesquisa foram identificadas as causas da Dívida de Variabilidade comumente encontradas nos estudos de casos apresentados pela Revisão Bibliográfica da Literatura, ou seja, quais são os principais motivos que levam as empresas a incorrer em dívida técnica quando implementam sistemas com variabilidade. Entre as causas mais citadas estão pressão de tempo e as mudanças constantes de requisitos para clientes ou cenários diferentes.*

4.1.2 QP1.2 - Artefatos impactados pela Dívida de Variabilidade

Como pode-se observar na Tabela 3 do Apêndice A, a dívida de variabilidade impacta artefatos ao longo de todo o ciclo de desenvolvimento, desde os requisitos até os arquivos de configuração. Código-fonte, requisitos, arquitetura e modelos de *design* são os mais comumente afetados. Podemos observar um grande número de estudos de caso descrevendo problemas de variabilidade no *código-fonte*, uma vez que a pressão do tempo leva as empresas a se concentrarem diretamente nos artefatos de implementação.

Problemas relacionados à variabilidade de *modelos* são frequentes porque vários estudos de caso estão no domínio de sistemas embarcados e automóveis, que usam modelos como artefatos relevantes. Curiosamente, a dívida de variabilidade também afeta os *esquemas de banco de dados (BD)* e os *scripts de construção*, o que afeta indiretamente a implantação de variantes.

Resposta da QP1.2: *Vários tipos de artefatos são impactados pela variabilidade da dívida. O código-fonte é o mais comum, seguido por requisitos, arquitetura e modelos. Alguns estudos de caso também mencionaram casos de teste, esquemas de banco de dados e scripts de construção. Percebe-se que a dívida de variabilidade impacta os artefatos criados ao longo de todo o ciclo de vida do desenvolvimento de software.*

4.1.3 QP1.3 - Consequências da Dívida de Variabilidade

A Tabela 4 do Apêndice A apresenta os problemas enfrentados pelas empresas devido à dívida de variabilidade. A consequência mais comum da dívida de variabilidade é a *complexidade para manter variantes independentes* criadas com reúso oportunista. Por exemplo, um único defeito deve ser corrigido várias vezes, sendo uma tarefa sujeita a erros e demorada. Além disso, a proliferação de artefatos de software redundantes e quase semelhantes afeta diretamente a capacidade de manutenção de uma família de produtos.

Outra consequência da dívida de variabilidade é a *incapacidade de criar novos produtos customizados* como resultado de um gerenciamento de variabilidade inadequado. Por exemplo, é difícil identificar qual artefato deve ser usado em um novo produto quando existem muitos artefatos semelhantes, mas com pequenas personalizações. A terceira consequência mais comum de dívida de variabilidade refere-se à *incapacidade de criar sistemas complexos*, como por exemplo, criar um novo produto baseado em sistemas já existentes de diferentes empresas que foram fundidas. Como consequência da falta de mecanismos sistemáticos para lidar com a customização, a *qualidade interna geral dos sistemas* diminui. Por exemplo, variantes têm código morto que aumenta o tamanho e a complexidade do produto, também conhecido como software inchado. Além disso, a falta de variabilidade impacta diretamente a flexibilidade para atender a situações de mudança e operações diversificadas com o mínimo de interrupção ou atraso.

Os ciclos de teste também são afetados devido à falta de gerenciamento de variabilidade. Por exemplo, a *derivação de casos de teste para famílias de produtos é complexa* devido à variabilidade nos requisitos, com a necessidade de muitos comportamentos a serem testados. Estratégias oportunistas para lidar com a variabilidade também podem frustrar os testadores, pois defeitos repetidos com causas semelhantes permanecem presentes em diferentes variantes e comumente os ciclos de garantia de qualidade são longos.

Evitar a variabilidade com sistemas únicos com todas as funcionalidades também é uma solução alternativa problemática. Isso cria *problemas de usabilidade*. Perigosamente,

um problema de engenharia é convertido em um problema de negócios, pois afetará diretamente a satisfação do cliente. Do ponto de vista da *gestão de portfólio*, entre os problemas de falta de variabilidade podemos citar o uso de vários repositórios de software para cada variante, perdendo o controle das modificações gerais, implantação mais complexa de produtos para clientes distintos, e as variantes legadas tornam-se *outliers*.

Por fim, lidar com múltiplas variantes independentes também gera *problemas relacionados ao uso de recursos humanos*. Por exemplo, atribuições de funções repetidas: dois profissionais como arquitetos, em diferentes variantes, no entanto, desenvolvendo tarefas muito semelhantes.

Resposta da QP1.3: *Esta questão de pesquisa apresentou como resultado as principais consequências enfrentadas por empresas que acumularam dívida de variabilidade. As consequências vão desde manutenção complexa e ciclos de teste, dificuldade de customização e criação de sistemas complexos, até o gerenciamento de portfólio limitado e baixa qualidade interna ao mau uso de recursos humanos e problemas de usabilidade.*

4.1.4 QP1.4 - Catálogo de Dívidas de Variabilidade

Para responder a questão de pesquisa QP1.4 e criar o catálogo preliminar de dívidas de variabilidade, foi utilizado um estudo de mapeamento sistemático que apresenta uma lista abrangente de dívidas técnicas (LI; AVGERIOU; LIANG, 2015). A Figura 6 apresenta a estrutura lógica utilizada para criação do catálogo. O trabalho de Li, Avgeriou e Liang (2015) coletou vários tipos de dívida técnica em diferentes níveis. Esses tipos de dívida técnica podem ser classificados em 10 tipos de granularidade e cada um deles é classificado em vários subtipos com base em suas causas. Utilizou-se cinco dos 10 tipos de granularidade dívida técnica, representados na Figura 6 pela raia de cor azul (os tipos de DT selecionados estão representados pelas caixas de texto com o nome do tipo da dívida, ou demais estão representados com a caixa de texto que possui os reticências). Para seleção dos tipos de dívida de variabilidade, escolheu-se nove dos 45 subtipos de dívida técnica, representados na raia de cor verde claro (os subtipos de DT selecionados estão representados pelas caixas de texto com o nome da dívida técnica, ou demais estão representados com a caixa de texto que possui os reticências).

De acordo com Li, Avgeriou e Liang (2015), alguns atributos de qualidade (QAs) são comprometidos quando a dívida técnica existe. A maioria dos estudos argumenta que a dívida técnica afeta negativamente a manutenibilidade dos sistemas de software. Além disso, como a ISO/IEC 25010 (padrão usado para atributos de qualidade) não distingue entre facilidade de implementação de novos requisitos e correções de *bugs* como atributos de qualidade diferentes, esses autores classificaram como modificabilidade os casos em que a capacidade de implementar novos requisitos é influenciada negativamente. As dívidas técnicas relacionadas a essas categorias de QAs foram os que tiveram maior relevância

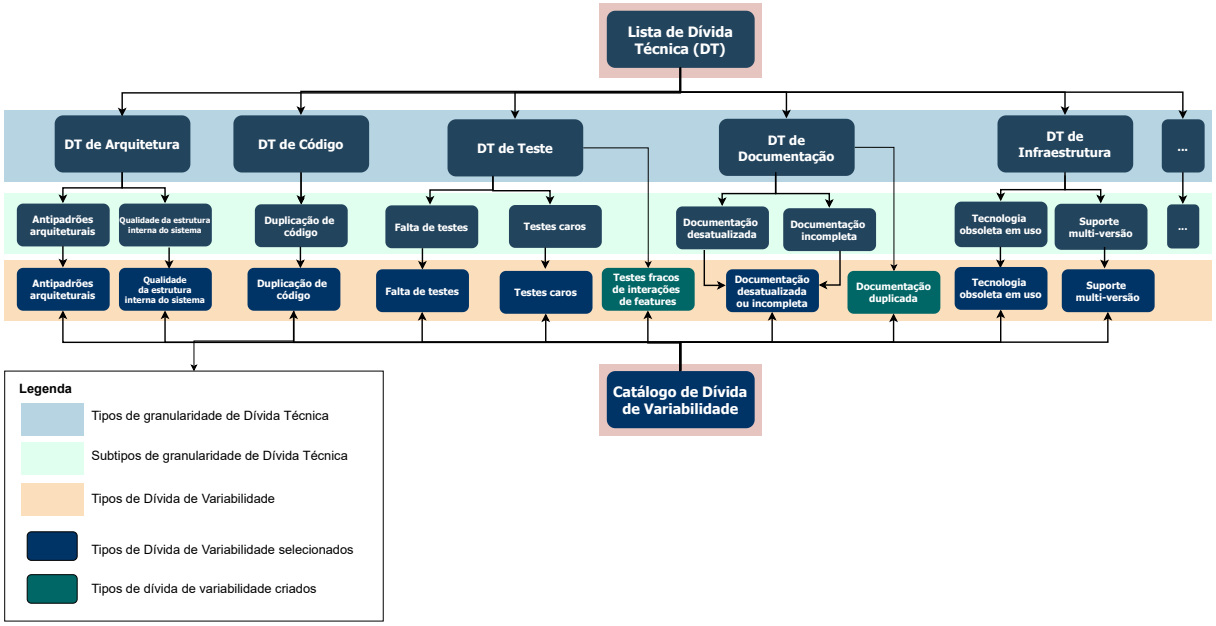

Figura 6 – Estrutura utilizada para criação do Catálogo de Dívida de Variabilidade

para classificarmos em nosso trabalho. Portanto, nossa análise considerou que a dívida de variabilidade pode estar relacionada a dívidas técnicas relacionadas aos seguintes atributos de qualidade descritas por Li, Avgeriou e Liang (2015): Capacidade de manutenção, modularidade, reutilização, analisabilidade, modificabilidade, testabilidade, adaptabilidade e substituibilidade.

Em seguida, avaliaram-se os 52 estudos de caso selecionados em nossa revisão sistemática da literatura para compreender as situações em que a dívida de variabilidade apareceu e não foi coberta no catálogo existente de Li, Avgeriou e Liang (2015). Analisamos situações que são geradas como consequência de uma implementação de variabilidade que gerou problemas de médio e longo prazo para o sistema e, portanto, incluímos mais duas classificações: (i) Duplicação de documentação e (ii) Nenhum teste de interação de *features*. Desta forma, os tipos de dívida de variabilidade completo estão apresentados na Figura 6 na raia de cor laranja. Nosso catálogo de dívidas de variabilidade também é apresentado na Tabela 6 do Apêndice A. Este catálogo foi concebido cruzando informações sobre dívidas técnicas que ocorrem nos 52 estudos de caso considerados em nossa investigação. Para isso, analisaram-se os sistemas que implementam a variabilidade avaliando as causas, consequências e os principais artefatos impactados pela dívida técnica, gerando um conjunto de estudos de caso que passaram por esta situação. Cada dívida de variabilidade apresentada no catálogo é discutida a seguir:

- **Documentação desatualizada ou incompleta**: Ao implementar a variabilidade de forma não sistemática por meio de copiar e colar, a tendência é duplicar a documentação. Com o passar do tempo, a manutenção de múltiplos documentos

torna-se cada vez mais complexa e acaba tornando-se uma dívida. A equipe tende a implementar a variabilidade e não atualizar a documentação.

- **Duplicação de código**: É uma das dívidas técnicas mais frequentes identificadas quando a variabilidade é implementada através de reuso não sistemático. Isso porque as empresas optam, seja por falta de conhecimento ou por falta de tempo (causas), em implementar a variabilidade copiando e colando trechos de código ou sistemas inteiros. Isso leva a (consequência) dificuldade na manutenção (por exemplo: quando um *bug* é identificado, é necessário corrigi-lo em todas as variantes do sistema), repetição de regras pelo sistema, entre outros.

- **Antipadrões arquiteturais**: Implementar variabilidade de forma não sistemática através do copiar/colar, seja de trechos de código ou de todo o sistema, pode levar a sérios problemas na arquitetura dos sistemas da empresa. Por exemplo, se houver integrações complexas e cada vez que o sistema for copiado elas serem replicadas.

- **Qualidade da estrutura interna do sistema**: Ao implementar a variabilidade de forma não sistemática, os danos ao sistema podem ser inúmeros: problemas de duplicação de código, problemas de arquitetura, problemas de integração de sistema e banco de dados, além de problemas com a documentação. A qualidade tende a cair porque a família de projetos não é projetada com uma estrutura sólida e por isso é evoluído. Pelo contrário, muitas vezes é inicialmente criado como um pequeno sistema e a partir disso cria-se cópias e variações, gerando uma estrutura frágil e sem qualidade.

- **Falta de teste**: Essa dívida técnica no contexto de variabilidade está diretamente relacionada à falta de tempo e variações de cenários/variantes. Ou seja, a empresa quer lançar o mais rápido possível um novo produto com variação e falta tempo, então, uma etapa completa do teste é pulada. Ou o que também pode acontecer é que, como há certeza de que o produto principal funciona, não há necessidade de testar a variação, ou ainda, a fase de teste está subestimada podendo ser uma etapa a se eliminar para economizar tempo. Consequência disso: falta de qualidade no produto.

- **Testes caros**: Por outro lado, a ausência de testes pode ocorrer devido à falta de conhecimento em gerenciamento de variabilidade, técnicas de teste adequadas, elaboração de cenários de teste complexos e até repetitivos para incorporar todas as variantes do sistema em todos os reúsos implementados, tornando esta uma atividade que consome tempo e recursos.

- **Suporte multi-versão**: Podemos considerar esta a dívida técnica que melhor representa o significado do termo DT, pois traz benefícios de curto prazo ao usar a

variabilidade de forma não sistemática (por exemplo, *time-to-market*), não exige que a equipe conheça técnicas para implementar a variabilidade e gerenciá-la melhor, mas à medida que o copiar e colar é implementado, muitas versões do sistema são geradas e podem até ser atualizadas em momentos diferentes. A variante A pode ser atualizada, a variante B não, por outro lado é possível fazer uma atualização da variante C e não de A e B, etc. Quando o código-fonte e a documentação devem ser sincronizados, o suporte torna-se ainda mais complexo.

- **Tecnologia obsoleta em uso**: Como é possível manter múltiplas versões de um sistema com suas variantes ou múltiplos sistemas criados para atender diferentes requisitos (variações), algumas empresas mantém esses sistemas por anos com a tecnologia e arquitetura inicialmente criados, eles acabam se tornando sistemas legados importantes nas empresas e quando há a necessidade de atualização de tecnologia, torna-se uma missão difícil.

Além das dívidas no artigo de Li, Avgeriou e Liang (2015), discutidas à luz da variabilidade, incluímos novas dívidas observadas em nosso estudo:

- **Documentação duplicada**: Esta dívida de variabilidade se deve à implementação não sistemática das variantes do sistema. Em muitas ocasiões, por falta de conhecimento para implementar a variabilidade de outra forma ou pelo curto período de tempo para liberação da nova variante, as equipes acabam optando por copiar e colar o sistema (fonte mais documentação). Assim, implicam na dívida técnica de manutenção de múltiplos documentos, dificultando a manutenção futura para manter todas as informações atualizadas.

- **Teste fracos de interações de *features***: Ao criar novas variantes com configurações de *features* diferentes, a interação dessas diferentes *features* pode interferir o comportamento de uma variante. Portanto, interações de *features* devem ser testadas.

Resposta da QP1.4: *Como resultado da revisão sistemática da literatura, definiu-se o primeiro catálogo de dívidas técnicas encontradas na literatura que ocorrem quando a variabilidade é implementada de forma não sistemática. O catálogo é constituído de um total de 10 tipos de dívida de variabilidade, sendo que desse total, oito foram extraídos de um estudo similar identificado na Revisão Sistemática da Literatura, a partir do qual analisou-se os tipos de dívida técnica com características de sistemas que implementam variabilidade de software. Os outros dois tipos de dívida de variabilidade foram criados pelos autores, também a partir do mapeamento da Revisão Sistemática da Literatura, mas desta vez, avaliando os estudos de casos e os impactos que os sistemas que implementam variabilidade tiveram ao longo do tempo. Pesquisadores e profissionais podem utilizar como referência seja para compreender o que ocorrerá se implementarem variabilidade de forma não adequada, ou então, para compreenderem que os impactos são semelhantes ao de seus sistemas que já apresentam dívida de variabilidade.*

4.2 Dívida de Variabilidade na Prática

O estudo de caso multiprojetos foi realizado em duas etapas: (i) análise de artefatos e (ii) aplicação de questionário com profissionais envolvidos nos três sistemas sendo analisados.

4.2.1 Tecnologias utilizadas

Inicialmente coletou-se as informações referentes as tecnologias utilizadas nos sistemas (Original e suas variantes) com o intuito de compreender se houveram mudanças de tecnologias, linguagens, bibliotecas e/ou *frameworks* de um sistema para sua variante.

O Sistema X apresenta exatamente as mesmas configurações tanto na Variante Original quanto nas Variantes 1 e 2:

- Servidor de aplicação: Server JSE 6;
- Banco de dados: Postgre SQL 9.4;
- Endpoint através de Sockets;
- Conexão com o SAP via RFC;
- Versão Mobile: JSE 1.3;
- Interface GUI AWT;
- Autenticação de usuários com banco de dados.

O Sistema Y apresenta as seguintes configurações para a Variante Original e a Variante 1:

- Servidor de aplicação: EJB Java EE6;
- *Frameworks*: Hibernate 3, Primefaces 2.2.1; JSF 2.1;
- Conexão com o SAP via RFC;

- Banco de dados: Postgre SQL 9.1;
- Autenticação de usuários via LDAP;
- JasperReport 4.5.1.

As tecnologias utilizadas no Sistema Z, tanto para a Variante Original quanto para a Variante 1, são:

- Servidor de aplicação: EJB Java EE7;
- *Frameworks*: Hibernate 5, Spring Data JPA, JSF 2.2;
- Conexão com o SAP via RFC;
- Banco de dados: Oracle 11g;
- Autenticação de usuários via Active Directory;
- JasperReport 6.5.1.

4.2.2 QP2.1 - Análise Documental

Esta seção apresenta os resultados obtidos através das análises dos artefatos dos três objetos de estudo: Sistemas X, Y e Z. Iniciou-se a análise com a avaliação de quais eram os artefatos que cada sistema e suas variantes possuíam, obtendo o resultado encontrado na Tabela 5. Os artefatos avaliados foram: URS, também conhecida como Lista de Requisitos, o documento que expressa as necessidades do cliente e o que o sistema deve fazer; Especificação (casos de uso), o documento que descreve o passo-a-passo da interação entre o ator e o sistema, bem como os fluxos de tela e quais são as regras de negócio do sistema; Protótipos, modelos de software sem funcionalidades inteligentes que representam a versão inicial do que poderá vir a ser o sistema final; Casos de Teste, documento que apresenta um conjunto de valores de entrada, precondições de execução, resultados esperados e pós-condições de execução desenvolvidas para um determinado objetivo ou condição de teste, tais como para exercitar o caminho de um determinado programa ou verificar o atendimento de um requisito; o MER, que refere-se ao modelo utilizado durante a fase de projeto para representar os dados que serão armazenados no sistema, e; o código-fonte, que são as instruções necessárias para fazer o sistema funcionar.

Tabela 5 – Comparação de Artefatos

Sistemas	**Sistema X**			**Sistema Y**		**Sistema Z**	
Artefatos	**Orig.**	**Var. 1**	**Var. 2**	**Orig.**	**Var. 1**	**Orig.**	**Var. 1**
URS		✓	✓	✓	✓	✓	✓
Especificação	✓	✓	✓	✓	✓	✓	✓
Protótipos				✓	✓		
Casos de Teste	✓		✓	✓	✓	✓	
MER	✓			✓	✓		
Código-fonte	✓	✓	✓	✓	✓	✓	✓

Orig. = Original, Var. = Variante

Iniciamos a análise nos artefatos, exceto o código-fonte. Com base nessa análise, percebeu-se que há uma predominância em desenvolver URS e especificações para os sistemas na empresa onde o estudo de caso foi conduzido. No caso de URS, com exceção do Sistema Original X que não possuía URS, os demais tiveram seus documentos criados e adaptados para suas variantes (no caso do Sistema X, a cópia/adaptação ocorreu da Variante 1 para Variante 2). Além disso, ao analisar os documentos referente aos casos de teste, percebeu-se que estes foram replicados somente nos sistemas e/ou variantes cuja empresa/filial passa pela inspeção de uma agência reguladora, caso contrário, os mesmos não foram desenvolvidos e nem executados. Isso nos remete a evidência do tipo de dívida de variabilidade identificada na literatura de "Falta de testes", que acabam sendo negligenciados para entregar mais rápido, ou até mesmo porque a equipe subentende que o sistema original já está funcionando, então não há necessidade de testes. Em relação as especificações (artefato que apresentou uma dominância em todos os sistemas, tanto original quanto variante) realizou-se a análise da quantidade de especificações por sistema, obtendo o resultado apresentado na Tabela 6.

Tabela 6 – Comparação de Especificações

Sistemas	**Sistema X**			**Sistema Y**		**Sistema Z**	
Artefato	**Orig.**	**Var. 1**	**Var. 2**	**Orig.**	**Var. 1**	**Orig.**	**Var. 1**
Especificação	14	10	11	26	17	7	7

Orig. = Original, Var. = Variante

O Sistema X apresentou as especificações de acordo com as funcionalidades existentes. O Sistema Y apresentou documentação somente para as funcionalidades que eram relevantes para uma possível inspeção de uma agência reguladora (a equipe de TI não cria especificações para relatórios não relevantes ao processo). E no caso do Sistema Z, a quantidade de especificações é equivalente, porém elas foram copiadas de um sistema para outro e adaptadas de acordo com os tipos de formulários (cadastros) e relatórios existentes no sistema.

Ao realizar a análise dos documentos, foi possível observar cópias fiéis de documentos, alterando-se algumas regras de negócio do sistema original para sua variante, conforme mostra a Figura 7. Estas também são evidências para o tipo de dívida de variabilidade "Documentação duplicada". Nesta figura pode-se comparar uma especificação do Sistema Z, na qual há diferenças na regra de negócio RN-17, que diz respeito a página de registro de Fornecedores/Fabricantes, por exemplo nos campos de Apontamento de Auditoria Interna, Externa, Cliente e Número N.C. No entanto, as regras de negócios RN-15 e RN-16 são idênticas.

Na sequência, serão apresentados os resultados da análise do artefato código-fonte dos três sistemas.

RN-15 Campo "Data do Registro"	O sistema deve preencher o campo ***Data do Registro*** automaticamente com a data atual do dia em que o usuário está logado no sistema.
RN-16 Preenchimento do Campo "Centro"	O campo ***Centro*** deve ser uma lista de seleção única com todos os centros ativos disponíveis.
RN-17 Campos do Formulário N.C Fornecedor/Fabricante (Processo)	A página de Registro de Fornecedores/Fabricantes (Processo) deve possuir os seguintes campos: ❖ Ocorrência Detectado em **[RN-24]** ❖ Data do Registro ❖ Apontamento de Auditoria Interna ❖ Apontamento de Auditoria Externa ❖ Apontamento de Auditoria Cliente ❖ Número N.C ❖ Centro ❖ Registrado Por ❖ Detectado Por ❖ Tipo Responsável **[RN-35][RN-37]** ❖ Lote ❖ Quantidade com Ocorrência ❖ Descrição da Ocorrência ❖ Ocorrido em ❖ Lote/Produto ❖ Foi solicitado o bloqueio do Lote ME/MP

(a) Sistema Z - Variante 1

RN-15 Campo "Data do Registro"	O sistema deve preencher o campo ***Data do Registro*** automaticamente com a data atual do dia em que o usuário está logado no sistema.
RN-16 Preenchimento do Campo "Centro"	O campo ***Centro*** deve ser uma lista de seleção única com todos os centros ativos disponíveis.
RN-17 Campos do Formulário N.C Fornecedor/Fabricante (Processo)	A página de ***Registro de Fornecedores/Fabricantes (Processo)*** deve possuir os seguintes campos: ❖ Desvio Detectado em **[RN-24]** ❖ Data do Registro **[RN-15]** ❖ Apontamento de Auditoria ❖ Centro **[RN-16]** ❖ Registrado Por **[RN-14]** ❖ Detectado Por **[RN-14] [RN-32]** ❖ Origem do Desvio **[RN-35][RN-36][RN-37]** (Lote Interno ME/MP com desvio / Fornecedor) o Tipo / Código / Descrição do Material (Automático) o Lote Fornecedor do Lote com Desvio **[RN-26]** / Fornecedor do Lote (Automático) o Centro (Automático) o Fornecedor (quando a opção for fornecedor) **[RN-37]** ❖ Quantidade com Desvio ❖ Descrição do Desvio **[RN-39]** ❖ Lotes Impactacos / Relacionados **[RN-27]** ❖ Ocorrido em: PA, SA, ME, MP, Todos ou Outros o Lote / Tipo/Material / Descrição do Material / Centro / Fornecedor (Automático) ❖ Foi solicitado o bloqueio do Lote ME/MP? **[RN-45]**

(b) Sistema Z - Original

Figura 7 – Comparação de Regras de Negócios

4.2.2.1 Sistema Y

A Variante 1 do Sistema Y foi criada em um formato diferente do sistema original. A variante usou miniprojetos, dificultando a comparação na ferramenta But4Reuse. Desta forma, foi necessário, juntar os miniprojetos em um único projeto para a comparação ser possível. Como resultado da comparação obteve-se a visualização apresentada na Figura 8.

Quando analisa-se a estrutura das pasta, o *Block 0* (cor verde) representa a parte comum de pastas entre ambos sistemas, o *Block 1* (cor azul) representa a parte que existe somente no Sistema Original e o *Block 2* (cor rosa) indica a parte referente a variante. Ao desmarcar as áreas distintas das variantes é possível observar melhor a parcela de pastas comum entre os sistemas, conforme apresenta a Figura 9.

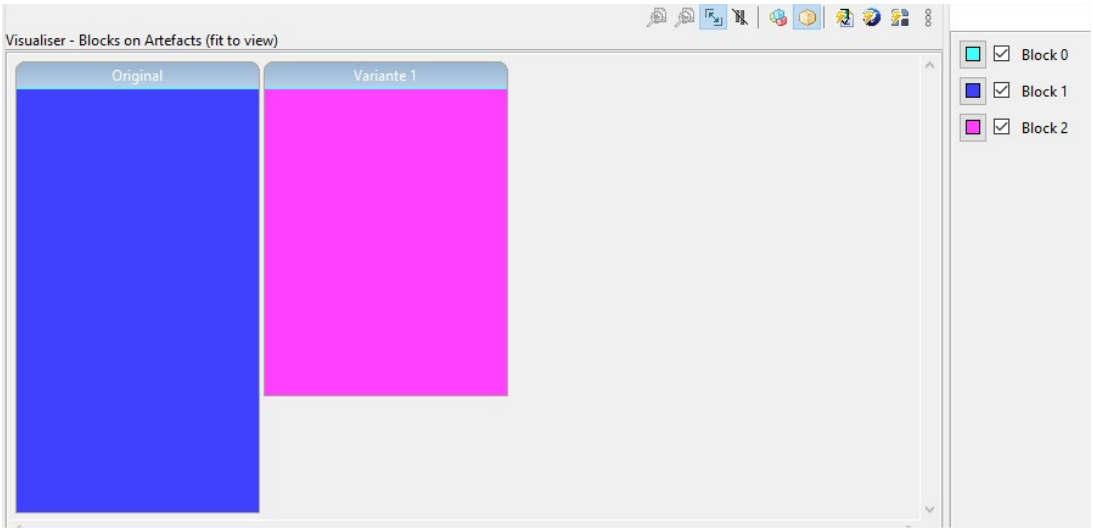

Figura 8 – Comparação na Ferramenta But4Reuse da estrutura de pastas - Sistema Y

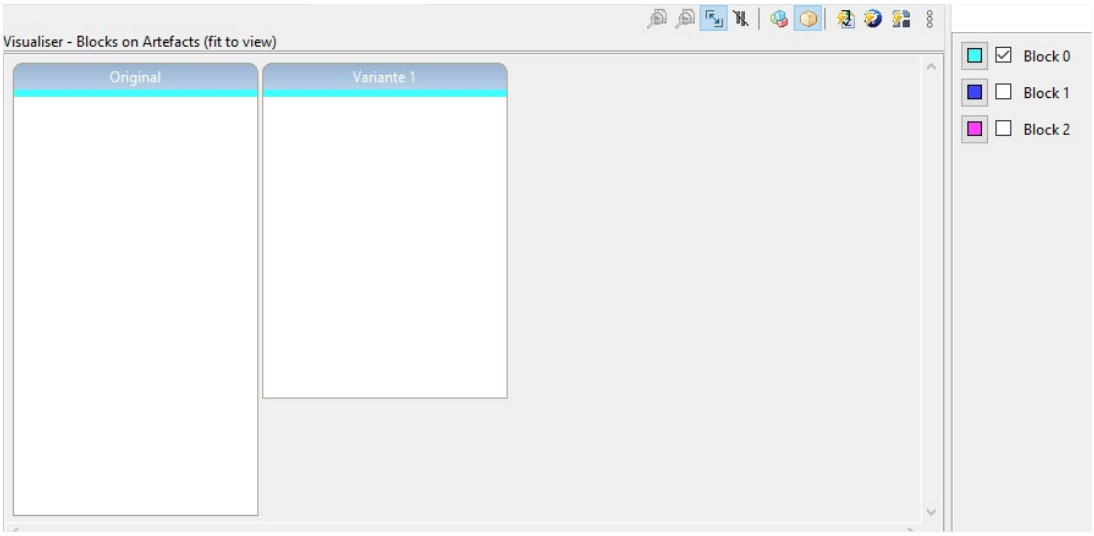

Figura 9 – Semelhanças das estruturas de pastas - Sistema Y

A Figura 10 mostra a comparação entre as variantes com o *plugin* Java JDT (*Java Development Tools*). Nesta comparação, analisa-se além da estrutura de pastas semelhantes, as demais ferramentas de desenvolvimento utilizadas, por exemplo, bibliotecas e componentes. Desta vez, pode-se observar uma área de concentração maior para as partes comum dos sistemas. Toda região verde, denominada *Block 0*, representa a parte comum entre os sistemas.

Na sequência, realizou-se uma análise das pastas comuns dos projetos Original e Variante 1 pela própria IDE de desenvolvimento. Em muitos casos, percebe-se que as

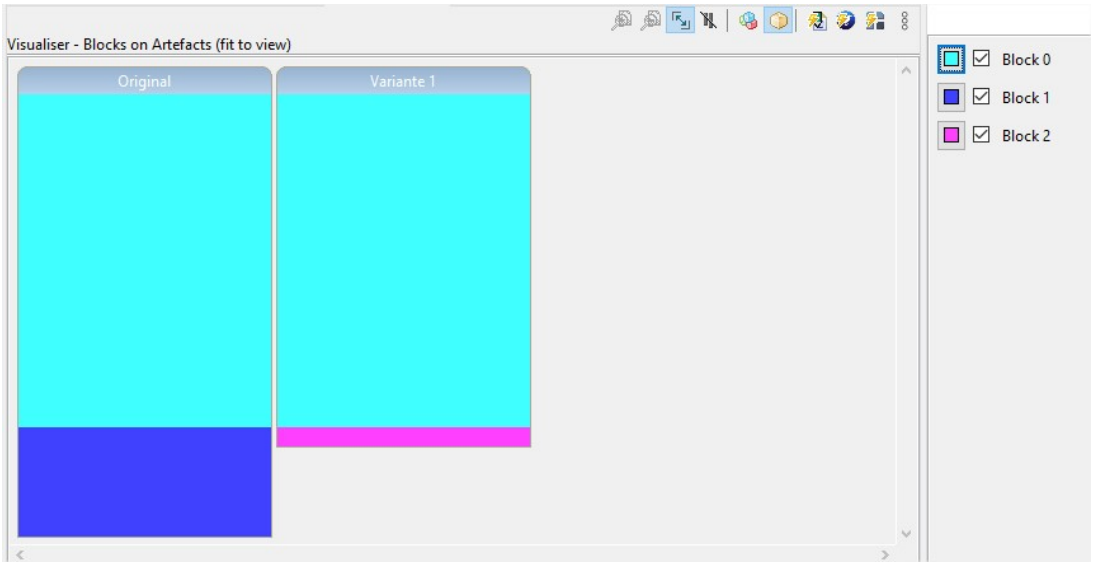

Figura 10 – Análise das ferramentas de desenvolvimento Java - Sistema Y

estruturas de pastas de um projeto para outro haviam sido alteradas adicionando um nível ou então, haviam sido renomeadas as pastas raízes. Decidiu-se então, realizar uma adequação dos nomes das pastas. Quando o conteúdo da classe era a mesma, renomeou-se as pastas para o mesmo nome. Ao aplicar novamente a ferramenta But4Reuse para a comparação de estrutura de pastas, obteve-se o resultado que é apresentado na Figura 11. Ao analisar o código Java JDT, obteve-se o resultado conforme apresenta a Figura 12.

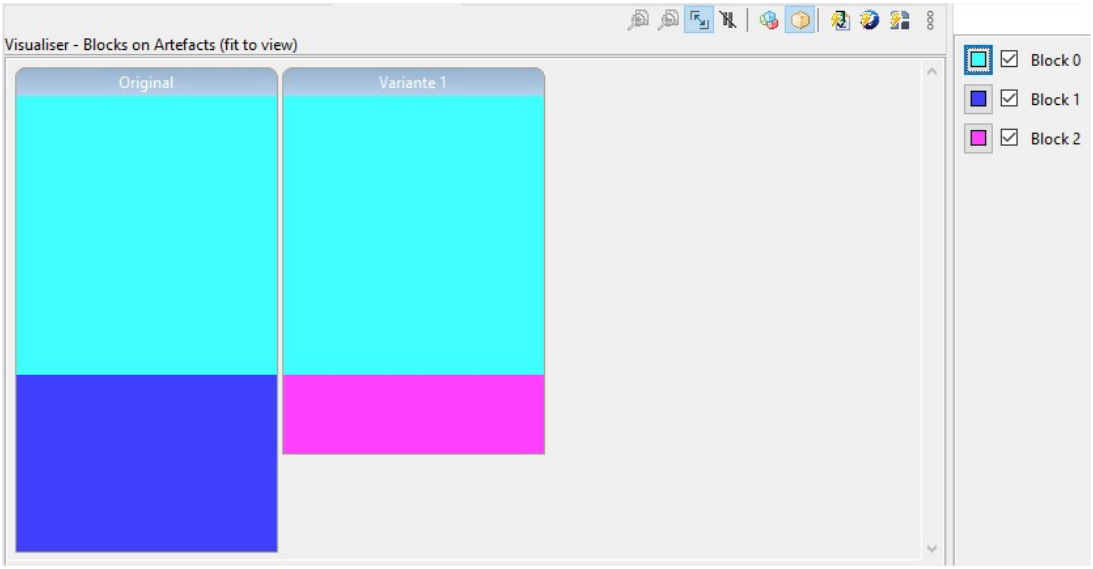

Figura 11 – Comparação da estrutura de pastas após adequação de nomes - Sistema Y

4.2.2.2 Sistema X

O Sistema X apresenta três implementações a serem comparadas: Sistema Original, Variante 1, e Variante 2. Ao executarmos a análise na ferramenta But4Reuse no quesito de estrutura de pastas obtivemos o resultado apresentado na Figura 13. O *Block 0* (cor

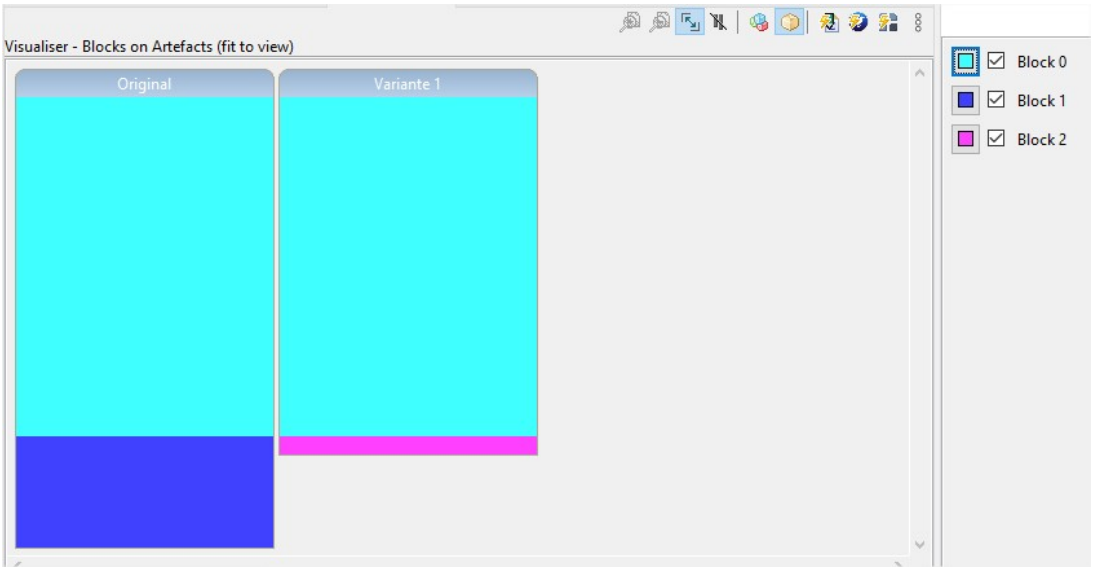

Figura 12 – Análise das ferramentas de desenvolvimento Java após adequação de nomes das pastas do projeto - Sistema Y

verde) representa a estrutura de pastas idêntica entre os três sistemas. O *Block 1* (cor azul) mostra a estrutura de pastas iguais entre o sistema Original e a Variante 1, já o *Block 2* (cor rosa) representa as semelhanças entre o sistema Original e a Variante 2. Desta forma é possível confirmar que a Variante 2 foi feito por meio de reúso oportunista do sistema Original. As partes identificadas como *Block 3* (cor verde limão) representa a parte única do sistema original (só existe neste sistema); o *Block 4* (cor laranja) representa a parte exclusiva da Variante 1 e o *Block 5* (cor roxa) representa a parte única da Variante 2.

Figura 13 – Comparação da estrutura de pastas - Sistema X

Para este caso, também realizou-se uma análise minuciosa de quais pastas possuíam conteúdos iguais e apenas foram renomeadas para indicar qual sua respectiva variante. Os nomes das pastas foram mantidos iguais ao sistema Original e executou-se uma nova análise na ferramenta But4Reuse, resultando na comparação apresentada na Figura 14.

Como resultado, é importante destacar que aumentou consideravelmente a parte semelhante entre os três sistemas e a semelhança do sistema Original com a Variante 1. Para nosso estudo, isso representa que houve mais reúso oportunista, o que aumenta a

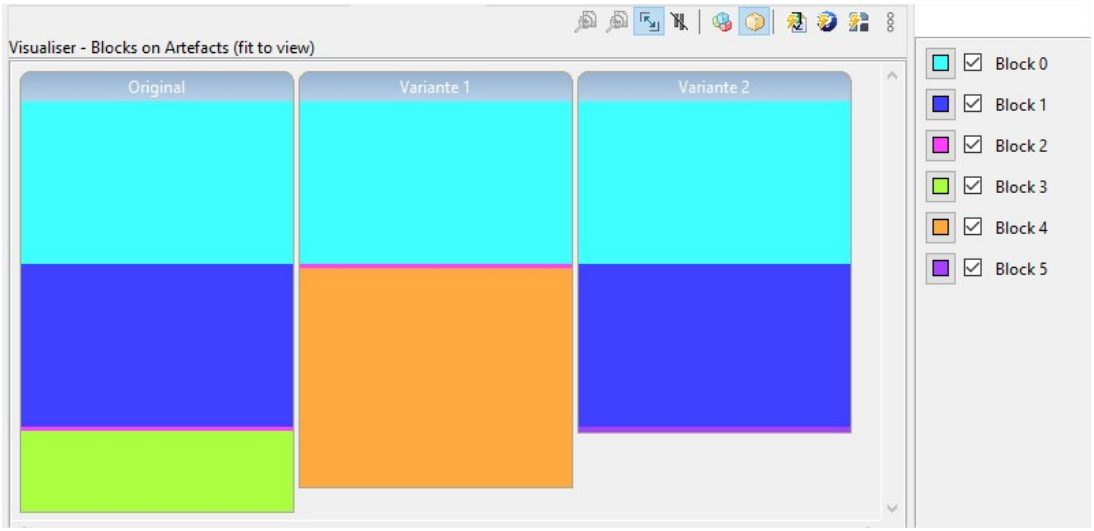

Figura 14 – Comparação da estrutura de pastas após adequação de nomes - Sistema X

incidência de partes iguais que possam gerar dívida de variabilidade. Consequentemente, isso pode gerar problemas de manutenção, retrabalho, testes repetitivos, sendo que quando ocorrer um problema, ou até mesmo uma melhoria, o desenvolvedor terá que aplicar nos três sistemas.

4.2.2.3 Sistema Z

A Figura 15 apresenta a comparação das estruturas de pastas entre o Sistema Z (Original e Variante 1). O *Block 0* (cor verde) representa a parte comum entre os sistemas, o *Block 1* (Azul) refere-se somente a parte do sistema Original e o *Block 2* (Rosa) trata-se da parte individualizada da Variante 1. Como nos demais sistemas, neste também realizou-se a análise de quais estruturas de pasta poderiam ser renomeadas, pois haviam conteúdo iguais em suas pastas. Após a adequação dos nomes das pastas, o resultado foi o que está sendo apresentado na Figura 16. De acordo com a figura, percebe-se que praticamente todo o sistema original foi mantido na Variante 1, e apenas uma pequena parte foi acrescentada.

Resposta da QP2 e QP2.1: *Com base na análise de como a dívida de variabilidade ocorre na prática, percebe-se que o código-fonte é o artefato mais copiado e colado para implantação de variabilidade na forma de reúso oportunista, gerando problemas futuros como replicação de regras de negócio, duplicidade de bugs e necessidade de correção dos mesmos, etc. Na sequência, percebe-se que os documentos de especificações (casos de uso) e URS são também replicados para as variantes de software.*

4.2.3 QP2.2 - Questionário com Profissionais

Foi realizada uma pesquisa de questionário com profissionais que tiveram contato com os três sistemas envolvidos no Estudo de Caso multiprojetos: Sistema X, Y e Z.

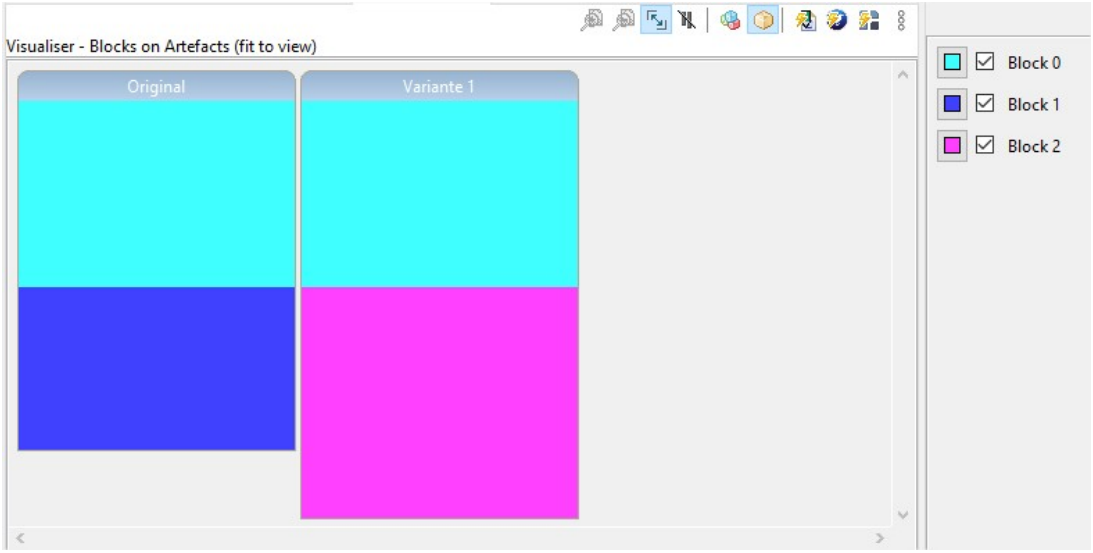

Figura 15 – Comparação da estrutura de pastas - Sistema Z

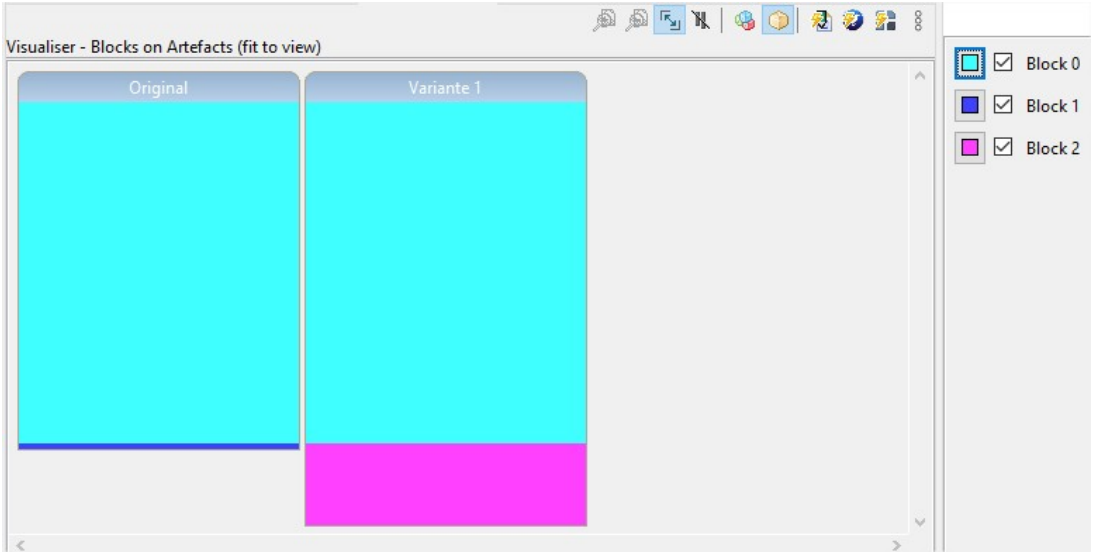

Figura 16 – Comparação da estrutura de pastas após adequação de nomes - Sistema Z

Foi elaborado um questionário com 18 perguntas (ver Seção 3.3.2.3) e enviado para 25 profissionais, obtendo um retorno de 22 respostas (88%). Cada participante foi nomeado de P1 até P22 para utilização nas citações deste trabalho.

4.2.3.1 Perfil dos Participantes

O perfil dos participantes efetivos em relação ao tempo de experiência profissional é apresentado na Figura 17. Com base nas respostas, observa-se que a maioria possui uma boa experiência profissional, sendo que 93,5% possuem mais de 5 anos de experiência. Já a Figura 18 apresenta a quantidade de participantes por sistema do estudo de caso multiprojetos.

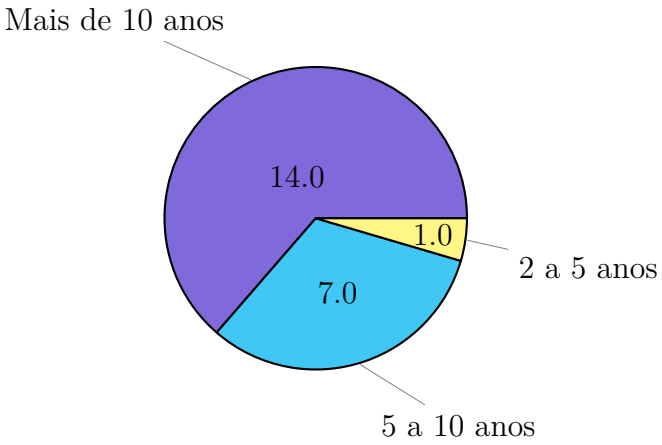

Figura 17 – Tempo de experiência profissional

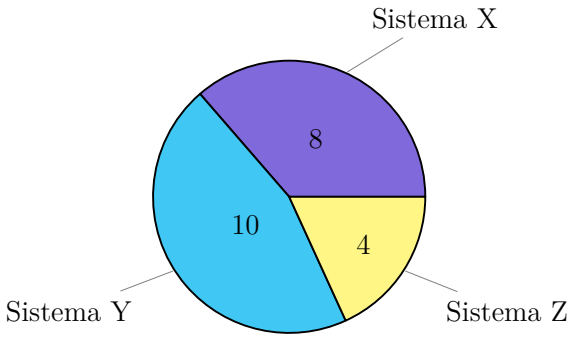

Figura 18 – Número de participantes por sistema

A pergunta de número 5 solicitava qual havia sido a função no Projeto ou Departamento quando o profissional participou do desenvolvimento do sistema. O resultado é apresentado na Tabela 7 e pode-se observar que, existe uma grande diversidade de funções, o que possibilitou uma visão mais rica e ampla sobre a dívida de variabilidade.

Tabela 7 – Função dos profissionais entrevistados

Função	**Quantidade**
Desenvolvedor	7
Analista de Sistemas	4
Analista de Testes	3
Gestor da área de negócios	3
Gestor de TI	2
Analista de Infraestrutura	1
Analista de Helpdesk	1
Usuário final	1

Após identificar em qual sistema e qual era a função do profissional quando ele lidou com o sistema indicado, questionou-se quanto tempo de experiência o participante possuía na função indicada. Com base nas respostas apresentadas na Figura 19, é possível perceber que grande parte possuía uma boa experiência na função. Mais de 70% possuíam mais de cinco anos de experiência na função. Esse resultado é muito significativo para

nossa pesquisa, pois aumenta a probabilidade dos resultados serem mais confiáveis, uma vez que tem-se um grupo com uma experiência considerável.

A tabela 8 apresenta um resumo dos dados dos participantes, a qual cargo pertencem, sua experiencia no cargo atual e em qual sistema fizeram parte para responder o questionário do estudo de caso aplicado.

Tabela 8 – Participantes da pesquisa e informações básicas

ID	**Cargo**	**Tempo de Exp.**	**Sistema**
P1	Desenvolvedor	≥ 10	Sistema Y
P2	Desenvolvedor	≥ 10	Sistema X
P3	Desenvolvedor	5 a 10	Sistema X
P4	Gestor da área de negócios	≥ 10	Sistema Y
P5	Analista de Helpdesk	5 a 10	Sistema Y
P6	Analista de Testes	≥ 10	Sistema X
P7	Analista de Sistemas	≥ 10	Sistema X
P8	Desenvolvedor	≥ 10	Sistema Z
P9	Analista de Sistemas	≥ 10	Sistema Y
P10	Usuário final	≥ 10	Sistema Y
P11	Gestor da área de negócios	≥ 10	Sistema Y
P12	Analista de Testes	5 a 10	Sistema X
P13	Gestor de TI	≥ 10	Sistema X
P14	Analista de Sistemas	5 a 10	Sistema Y
P15	Desenvolvedor	≥ 10	Sistema Y
P16	Desenvolvedor	≥ 10	Sistema X
P17	Gestor da área de negócios	5 a 10	Sistema Z
P18	Analista de Infraestrutura	≥ 10	Sistema Y
P19	Desenvolvedor	5 a 10	Sistema X
P20	Gestor de TI	≥ 10	Sistema Y
P21	Analista de Testes	5 a 10	Sistema Z
P22	Analista de Sistemas	2 a 5	Sistema Z

Tempo de Exp. = Tempo de experiência em anos, 0 a 2, 2 a 5, 5 a 10, ≥ 10

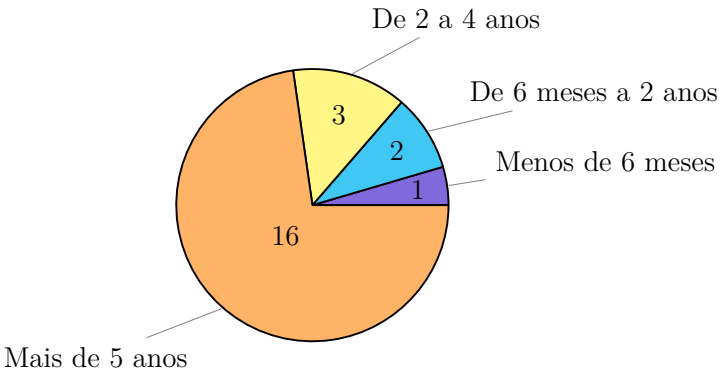

Figura 19 – Tempo de experiência na função que lidou com o sistema do estudo de caso

Para a questão de número 7, que pergunta se o participante compreende que ocorreu dívida técnica na implantação de variabilidade de software por meio do reúso oportunista, obteve-se as respostas de acordo com a Figura 20.

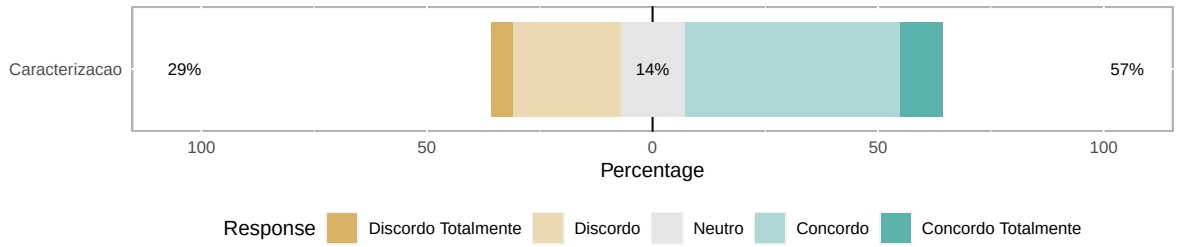

Figura 20 – Avaliação se houve dívida técnica na implantação de variabilidade de software

4.2.3.2 Percepção da Dívida de Variabilidade

Para avaliar os dados, desconsiderou-se a resposta de um participante que disse não saber opinar (o que representa 4,54% das opiniões). Ao avaliar a Figura 20, percebe-se que a maioria dos participantes concordam que ocorreu dívida técnica na implantação de variabilidade de software nos projetos analisados, pois 57% concordaram com o fato. 29% das opiniões estão concentradas para o lado da discordância e 14% são neutras sobre o assunto. Quando analisamos detalhadamente estas respostas, baseado nas funções dos participantes, podemos observar algumas curiosidades:

- 66,6% dos gestores das áreas de negócio discordaram, seja de forma parcial ou total e 33,3% concordam que existe problemas em criar sistema através de reúso oportunista.
- A opinião dos gestores de TI está dividida: 50% concordam e 50% discordam.
- Em relação aos analistas de sistema, 75% deles concordaram que há impactos de dívida técnica na variabilidade de software e 25% permaneceram neutros.
- 57% dos desenvolvedores concordam, 28% discordam e 14% permaneceram neutros.
- O analista de infraestrutura respondeu que concorda.

Quando as repostas foram analisadas separadamente por funções, percebeu-se que há uma tendência de profissionais de TI que concordam que existe dívida técnica em sistemas oriundos de variabilidade de software e profissionais da área de negócio que não sabem opinar ou não concordam, pois só enxergam o valor que existe na entrega a curto prazo. Curiosamente, há um relato de uma gestora da área de negócio (P17) afirmando que após realizar o reúso oportunista do Sistema Original para a Variante 1 e implantar na empresa que ela atuava, o sistema não atendia sua demanda: *"O sistema não refletia a nossa realidade, as nossas classificações de não conformidades não estavam de acordo"*.

Já os desenvolvedores de software incluíram relatos de partes copiadas que não são alteradas e causam impactos posteriores: P1 disse *"Alguns problemas que foram corrigidos para execução automática de jobs, realizadas no Sistema Y não foram feitas no Sistema Y [variante 1], que foi uma cópia. Também algumas regras de reimpressão de documentos*

aplicadas ao Sistema Y não foram reproduzidas no Y [Variante 1]". Já o respondente P3 afirma *"Em algumas partes da documentação de funcionalidades parecidas, alguns pequenos detalhes que não foram trocados na cópia comprometeram o resultado"*. P15 e P16 alegam que tiveram que alterar funcionalidades posteriormente pois haviam sido copiadas fielmente, mas tiveram que ser alteradas pois não se encaixaram exatamente no contexto do sistema. Apenas o desenvolvedor P2 que discordou relatou que *"As parte copiadas (estruturas de comunicação e integração com sistema legado) eram bastante estáveis e apresentavam pouca diferença estrutural"*.

A questão número 9 visa a identificação de quais foram os artefatos que fizeram parte do processo de desenvolvimento de software durante a implantação de variabilidade de software por meio de reúso oportunista. A Tabela 9 apresenta os resultados com suas respectivas quantidades de vezes citadas, em ordem decrescente.

Tabela 9 – Quantidade de Artefatos utilizados no processo de desenvolvimento

Artefato	**Quantidade Total**
Lista de requisitos	13
Especificações	8
Código-Fonte	7
Manuais	4
Protótipos de tela	4
Arquitetura	3
Diagramas UML	3
Especificações de *design*	2
Diagrama Entidade Relacionamento (DER)	2
Servidores	2
Testes unitários	1
Testes de *widgets*	1
Banco de Dados	1
Configurações	1

A diversificação de artefatos citados foi grande. Por se tratar do estudo de caso ocorrer em uma empresa regulamentada, o número de exigências para seguir o processo de desenvolvimento de software é alto. Algumas variantes do sistema foram implantadas em filiais do grupo não regulamentada, o que diminuiu o controle de alguns artefatos. Mesmo assim, foi possível perceber que a empresa dá uma importância grande ao registro dos requisitos e necessidades do cliente. O artefato "Lista de requisitos" foi o mais citado pelos participantes também foi o artefato que foi citado por praticamente todas funções entrevistadas. A única função que não indicou a Lista de requisitos como artefato utilizado no processo foi o Analista de Helpdesk (P5), mas que também não sinalizou nenhum artefato. Como era de se esperar, 71% das indicações do artefato código-fonte foram oriundas dos desenvolvedores de software. Já as especificações de software tiveram uma

divisão considerável, sendo: 50% indicados pelos desenvolvedores, 25% pelos analistas de sistemas e o restante dividido entre analista de infraestrutura e analista de testes.

A seguir, iniciou-se as questões relativas a opinião dos participantes em relação as causas da implantação de variabilidade utilizando práticas não sistemáticas e que acabam gerando dívida técnica no sistema. Foram avaliadas as causas identificadas anteriormente no estudo citado no Apêndice A. O resultado é apresentado na Figura 21.

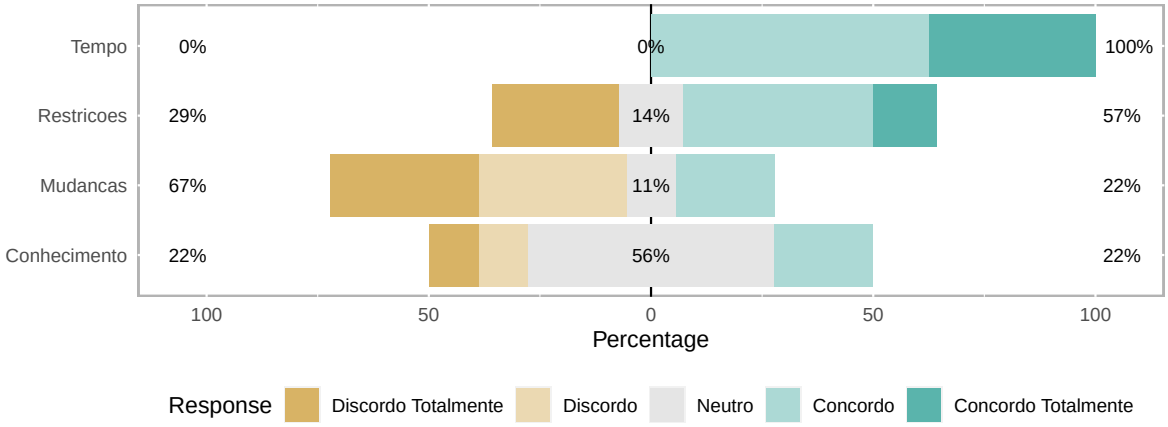

Figura 21 – Avaliação dos participantes quanto as causas de Dívida de Variabilidade

O item tempo, que refere-se a pressão de tempo, foi unanimidade dos participantes (desconsiderando o participante P3 que indicou "Não sei opinar") em concordância para uma das causas da dívida de variabilidade. Ou seja, a essência da metáfora de dívida técnica está presente neste resultado, pois ao entregar em um curto espaço de tempo temos alguns benefícios, ex: *time-to-market*, ou por questões regulatórias, ou até mesmo para satisfazer a pressão do cliente, mas acaba gerando um prejuízo a longo prazo por falta de planejamento, má qualidade, dificuldade de manutenção, etc. Isso foi constatado com as afirmações dos participantes na questão 11. P2 afirma que *"A pressão de tempo e o uso da mesma arquitetura de comunicação eram motivos para o uso da solução que já estava funcionando."*, já P4 escreve *"Quando o tempo de desenvolvimento não é o suficiente acaba-se optando por utilizar funcionalidades prontas ou já conhecidas"*. P6 relata que *"Na ocasião dos projetos de tais Sistemas, o tempo para execução do projeto era estritamente curto, bem como não era possível a implantação de uma nova ferramenta por se tratar de um processo com particularidades e com muitas mudanças devido à Empresa e processo ser extremamente dinâmico"*. P9 também faz menção a falta de tempo: *"Por falta de tempo é mais rápido sempre copiar e colar e fazer apenas a alterações no ponto necessário"*. Outros nove participantes ainda incluíram algum relato sobre sua experiência com o projeto e a pressão de tempo que sofreram para colocar o sistema no ar.

O segundo fator que teve maior aderência dos participantes com sua concordância como sendo uma causa de dívida de variabilidade foi "Restrições operacionais". O participante P1 relata *"Um agravante que cabe citar é o fato do sistema ser validado. A garantia*

da qualidade tem um calendário muito ***difícil*** *de validação de sistemas, e muitas vezes esse processo não fecha direito com o cronograma. Ainda, o sistema em questão trata de dados delicados, fazendo com que problemas ocasionem não conformidades para o setor e risco junto a inspeções da agência reguladora".* Para o participante P14 *"A falta de tempo para execução e as restrições operacionais (considerei poucos analistas para muitas demandas) motivam que a solução sistêmica seja "apenas"uma customização/revisão de um desenho já feito e implementado, que se encaixa na necessidade do cliente, por ser mais rápido e prático".* Além disso o participante P17 complementa *"O TI [Departamento de Tecnologia da Informação] que estava desenvolvendo não tinha tempo hábil e pessoas para fazer um sistema específico para cada empresa, porém havia uma pressa dos setores em controlar melhor suas não conformidades e parar de usar controles de excel".* Esta questão também teve um pequeno percentual (4,54%) de participantes que sinalizaram na resposta a opção "Não sei opinar".

O item mudanças que refere-se a "Mudanças constantes de requisitos" foi o item com maior discordância dos participantes. Não houveram muitos relatos sobre o tópico, mas alguns deles chamam a atenção: O participante P18 que concorda que mudanças constantes de requisitos é uma causa de dívida de variabilidade alega que *"Existe uma cultura da empresa de deixar para o último momento. Áreas de negócio não valorizam o levantamento de requisito e só percebem problemas após desenvolvimento. Além de as áreas de negócio não conhecerem os seus próprios processos".* O participante P11 acrescenta *"Muitas áreas participando da URS, solicitando urgência e mudanças de requisitos".* Por outro lado, o participante P21 que discorda desta causa, afirmou que: *"Mudanças constantes de requisitos: Discordo, esse pode ser um motivo para que um sistema não seja clonado".* Ou seja, na visão deste participante, na empresa existe sim muita mudança de requisitos, mas este seria um motivo para não clonar o sistema e sim, trabalhar em algo mais estruturado para melhor receber as mudanças constantes de requisitos.

E o último item foi o que teve o maior percentual de neutralidade com 56% de respostas neutras e um empate para discordância e concordância. O fato dos participantes se manterem neutros pode ser associado a questão de não desejarem medir seu próprio conhecimento, uma vez que o item refere-se sobre a "Falta de conhecimento" sobre dívida técnica e outras maneiras de implantar variabilidade de software. Ainda assim, alguns participantes que concordaram com esta causa, incluíram comentários, como o caso do participante P19 *"Pressão de tempo, pois se você já tem um código fonte que já funciona, você só agrega ele no novo código e evita retrabalho. A falta de conhecimento também entra na mesma resposta".* Além disso, o participante P22 completou: *"Uma outra empresa do mesmo grupo precisou de um sistema em comum, porém, como a área de negócio possui algumas diferenças e por falta de conhecimento, muitos requisitos foram alterados, por se tratar de uma cópia, existiu pressão para que a implantação ocorresse com agilidade".*

Na questão 12, ao serem questionados sobre motivos adicionais para a escolha de uma prática não sistemática na criação das variantes, alguns participantes detalharam causas questionadas na questão anterior. O participante P1 relatou *"Equipe reduzida, onde praticamente todo o processo é conduzido por duas pessoas, um analista e um desenvolvedor. A arquitetura que foi definida para o projeto é muito complexa, em vez de pensar em uma nova arquitetura, se optou por copiar e reproduzir os mesmos problemas."* Porém ambas situações podem se enquadrar em restrições operacionais. Além disso, o participante P11 sinalizou que o problema era tempo. E o participante P21 disse *"Na minha visão o único motivo para se clonar um sistema é a pressão ou preguiça de criar um sistema especifico. Porém não sou totalmente contrário a duplicação em alguns casos ela pode ser benéfica a empresa, cabe a avaliação da gestão sobre os pontos positivos e negativos".*

Por outro lado, pode-se observar algumas causas/fatores novos: P6 indicou que o problema eram recursos financeiros e na mesma linha o participante P12 relata *"Empresa não mensura o gasto para mudanças em sistemas devido setor de desenvolvimento ser interno".* Já os participantes P7 e 18 foram para outra linha, referente a processos e documentação, onde P7 diz que *"Falta de documentação do sistema"* e P18 afirma que *"Processo ou prática sistemática não é formalizada".* Desta forma, podemos identificar duas novas causas: Restrições financeiras e Ausência de processo de desenvolvimento de software adequado.

O estudo também verificou quais as consequências na prática quando implantado variantes de software por meio de reúso oportunista e se elas tem relação com a metáfora de dívida técnica. Questionou-se o quanto os participantes concordam com as consequências já identificadas anteriormente no mapeamento sistemático e apresentadas na Tabela 4 do Apêndice A. A Figura 22 apresenta o resultado da pesquisa. Nela observa-se que os participantes concordaram com a maioria (5/8) das consequências sugeridas como impacto ao se utilizar uma implantação não sistemática. Com destaque para problemas de usabilidade e de manutenção, enfatizando assim o conceito da dívida técnica.

Os participantes reafirmaram essas consequências nos relatos deixados na questão 14. P3 diz *"Muitos problema são identificados após a implementação por conta da falta da criação completa de uma feature".* P9 afirma *"Toda manutenção e teste no sistema é extremamente complexa. Sempre que precisamos fazer uma manutenções temos que validar praticamente todo o sistema novamente para ter a certeza de que não houve impacto em outros pontos".* P17 completa *"Qualquer mudança que era necessária, era extremamente morosa".*

Por outro lado, teve uma justificativa entre os participantes que discordou das consequências: P6 afirma *"Manutenção Complexa: Por se tratar de um sistema para cópia e cola, a manutenção torna-se mais fácil, o que justifica a implantação de um sistema cópia. Incapacidade de lidar com a customização: Por se tratar de um processo complexo, fazer*

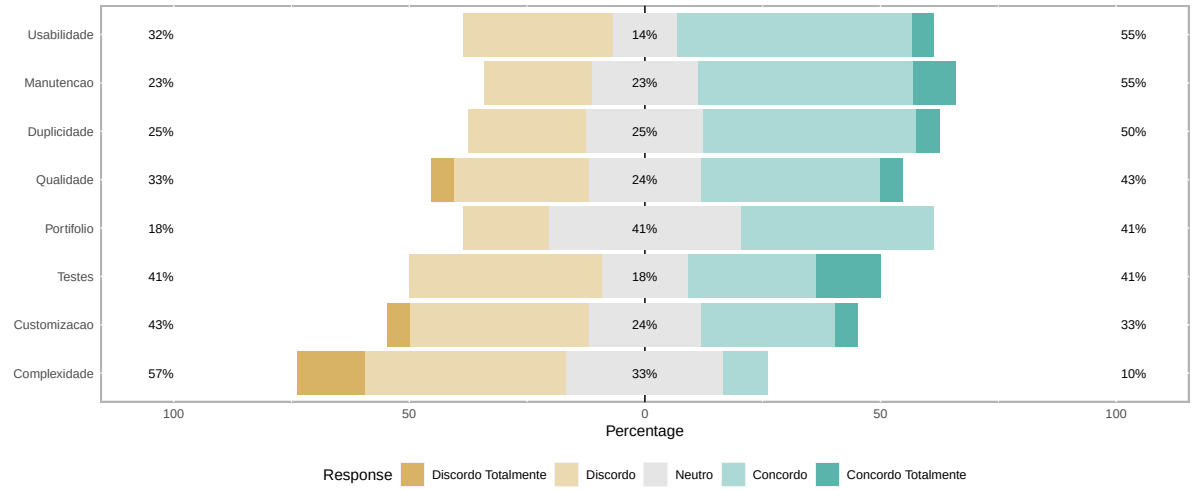

Figura 22 – Consequências da implantação variabilidade por meio de reúso oportunista

novas customizações trariam prejuízos financeiro e de tempo. Problemas na usabilidade: Devido ao curto tempo de entrega, justificou-se uma cópia do sistema para não se afetar o processo operacional.". Neste caso, o participante está dando mais ênfase ao beneficio que se ganha ao implantar variabilidade por meio do reúso oportunista e também uma visão técnica.

Apesar do item complexidade nos ciclos de testes ter ficado empatado entre os participante que não se mantiveram neutros, obteve-se comentários importantes sobre o tópico. O participante P1 afirma *"A falta de evolução tecnológica no ambiente de execução (atualização de versão de software base, servidores de aplicação, frameworks) faz com que bugs que já foram corrigidos fiquem escondidos no sistema, até alguém conseguir encontrar"*. Já o participante P3 deixa seu comentário com o seguinte registro: *"O usuário ao executar testes extensivos acaba por utilizar de recursos (cópia/cola) para facilitar/agilizar seus processos, podendo influenciar negativamente pois não necessariamente testou com atenção o que era necessário"*. E o participante P20 completa *"Problemas gerados pelo conflito de regras de negócio distintas para cada operação, causando paradas não planejadas e maior complexidade nos ciclos de testes"*.

É importante ressaltar que as consequências "Incapacidade de lidar sistemicamente com customização", "Incapacidade para criar sistemas complextos" e "Má qualidade de software" tiveram um participante em cada item, o que corresponde a 4,54% de participantes (por resposta) que indicaram a opção de resposta "Não sei opinar". Já o item "Papéis repetidos em projetos similares" teve esse número aumentado para 9,09%, o que representa dois participantes.

Quanto ao resultado da questão 15, que questionava se os participantes identificavam consequências adicionais além das citadas pelo estudo realizado anteriormente e documentado conforme Apêndice A, foram citados: O Participante P1 citou que *"Aumenta*

o risco para o Cliente"; para P6, P7 e P18 *"os erros são replicados de um sistema para outro e assim aumenta o retrabalho"*. P6 ainda destaca: *"Melhorias de processo são perdidas"*; e para finalizar o participante P21 afirma *"Além dos itens já mencionados também podemos considerar que as melhorias similares que muitas vezes acontecem em ambos os sistemas podem se tornar mais morosas devido a haver mais pessoas envolvidas e muitas vezes empresas diferentes, deixando o processo inicial mais demorado até o fechamento de escopo e testes"*.

4.2.3.3 Clareza do Termo e Gerenciamento da Dívida de Variabilidade

A última parte do questionário estava atrelado a clareza que os participantes compreenderam o termo dívida de variabilidade e suas perspectivas em relação as atividades envolvidas na gestão de dívida de variabilidade.

A questão 16, foi referente a clareza do termo e perguntava aos participantes até que ponto ele concorda que a definição da dívida de variabilidade está claramente descrita em relação ao seu contexto de uso e propósito para criar consciência na empresa. A Figura 23 apresenta o resultado da pesquisa e mostra que o termo está claro e bem definido, com exceção de um participante que se manteve neutro.

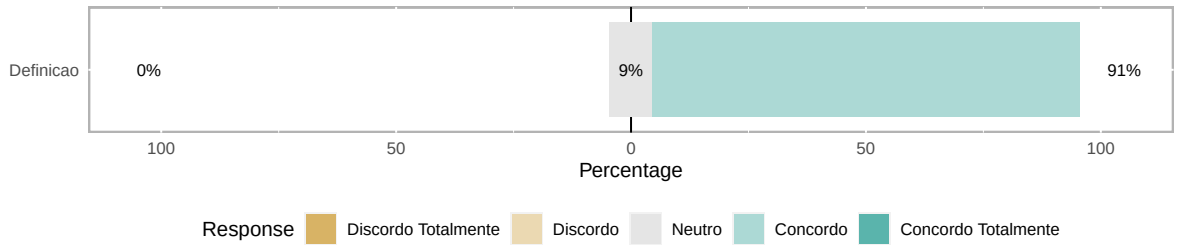

Figura 23 – Clareza quanto a Definição do termo Dívida de Variabilidade

Na questão 17, que complementava a questão anterior, poucos participantes incluíram sugestões. O Participante P15 disse *"Concordo que inicialmente ela se demonstra mais simples no início, mas com o passar do tempo, manter essas arquiteturas se torna muito mais complexo. Entendo também que manter softwares altamente configuráveis é extremamente complexo e que se for mal aplicado, pode causar manutenção ainda maior ou prejudicar a performance do sistema, principalmente devido a grande ocorrência de mudanças de requisitos durante seu ciclo de vida. Acredito que ter uma aplicação dívida em pequenas partes e funcionando como um todo, de forma distribuída ou não, facilite a adição de novas funcionalidade customizadas bem como a desativação de funcionalidades não mais necessárias"*. P20 afirma *"O maior desafio é quantificar esta dívida para o negócio antes do desenvolvimento das soluções, pois a necessidade de curto prazo acaba levando ao caminho mais fácil"*. E por fim o participante P21 acrescenta *"Acredito que deve ser levado em consideração as diferenças entre os processos que são realizados nas empresas que usam o mesmo sistema (Clonado), muitas vezes o processo definido para a empresa*

que inicialmente o software foi desenvolvido não é o melhor processo a ser adotado pela empresa que vai implementar o software clonado, tendo que fazer adaptações em seu processo ou customizações dentro do sistema por este motivo, muitas vezes se houver um levantamento de processo ou criar um escopo com as premissas para a implantação do sistema poderiam ser detectados vários motivos para inviabilizar a clonagem de sistemas".

E a última questão (18) teve como objetivo coletar a opinião dos participantes em relação as atividades na gestão da dívida técnica. Avaliou-se a perspectiva em relação ao apoio ou não dos participantes para cinco atividades-chave na gestão de DT, conforme relatado em Li, Liang e Avgeriou (2013). Desta forma as definições das atividades adaptadas à dívida de variabilidade são: i) Identificação: Detecção de itens de dívida técnica associados à variabilidade; ii) Medição: Estimar o custo e benefício associado a um item de dívida de variabilidade; iii) Priorização: Identifique qual item de dívida de variabilidade deve ser tratado primeiro; iv) Reembolso: Mudanças e ajudantes de transformação para eliminar o efeito negativo de um item de dívida de variabilidade e, v) Monitoramento: Observe as mudanças nos custos e benefícios de itens de dívida de variabilidade não resolvidos ao longo do tempo. Apesar de sua importância, não foi incluso, prevenção técnica de dívidas, representação/documentação e comunicação, que foram posteriormente acrescentadas em outro trabalho de Li, Avgeriou e Liang (2015). Estes itens foram excluídos para reduzir a pesquisa aos principais, conforme destacado por Li, Liang e Avgeriou (2013) e reduzir o esforço dos entrevistados.

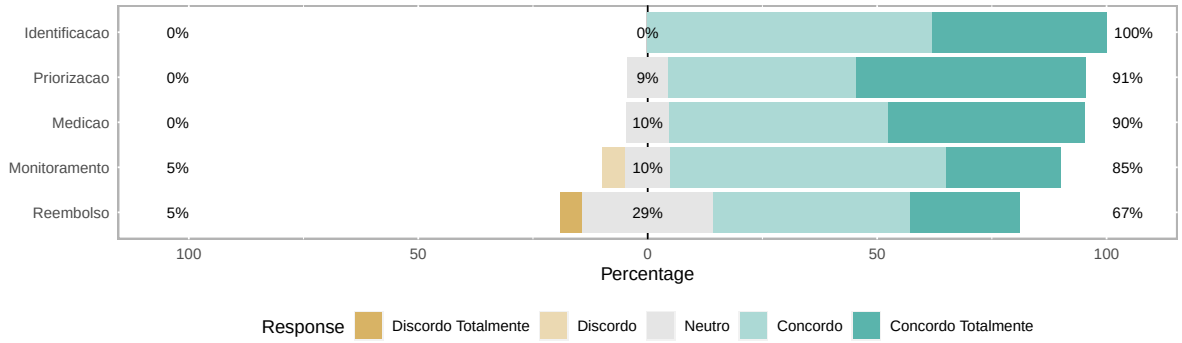

Figura 24 – Perspectivas em relação as atividades de gestão de dívida de variabilidade

Como pode-se perceber na Figura 24, a maioria dos participantes compreende que todas atividades são extremamente importantes. A atividade de identificação da dívida técnica teve 100% de concordância, seguida da priorização com 91%, item este que teve uma grande parcela de concordância total. O item que gerou uma parcela de neutralidade (29%) foi o re-embolso.

Resposta da QP2.2: *Com base nas características coletadas na literatura sobre dívida de variabilidade (definição, causas, artefatos e consequências) diferentes tipos de profissionais foram questionados sobre sua opinião a respeito dessas características. Em geral, a maioria está de acordo com o que foi encontrado na literatura e acrescentaram as causas de "restrições financeiras" e "ausência de processo de desenvolvimento de software adequado". Além de acrescentarem as consequências de "aumenta o risco de negócio pro cliente", "aumento de retrabalho" e "melhorias de processo são perdidas".*

4.3 Ameaças a validade

Validade interna: Quanto ao uso inadequado da ferramenta de análise de artefatos: Para minimizar a impacto a validade do estudo, primeiramente realizou-se a leitura do material fornecido na *wiki* da ferramenta, além de um treinamento da ferramenta But4Reuse com um dos seus criadores, assim adquirindo conhecimento suficiente para iniciar o manuseio da ferramenta. Outra ameaça refere-se a subjetividade das respostas para perguntas fechadas cuja resposta possuía uma régua de valores (escala Likert). Para minimizar este fato, foram incluídas questões abertas para que os profissionais pudessem reportar suas opiniões e relatar exemplos enfatizando assim suas respostas. Além disso, coletou-se respostas de diferentes visões (tipos de profissionais).

E por fim, a ameaça quanto aos participantes não compreenderem ou distorcerem as questões do estudo. Para minimizar este fato, adotou-se duas ações. A primeira delas, enviando o questionário para funções diversificadas e assim coletar opiniões de diferentes perspectivas. A segunda, foi a elaboração de questões abertas e fechadas, para que nas questões abertas os participantes tivessem a oportunidade de expor melhor sua opinião.

Validade externa: O estudo de caso (análise documental e questionário) foi realizado apenas em uma única empresa. Para minimizar esta ameaça, coletou-se evidências e aplicou-se o questionário para profissionais envolvidos em três objetos de estudo diferentes, ocorridos inclusive em períodos distintos, assim foi possível ampliar a visão e a opinião dos profissionais.

4.4 Considerações Finais

Avaliando o conceito de dívida técnica e o que acontece na prática quando implementado sistemas de variabilidade, fica evidente que a dívida existe, pois entrega-se o produto mais rapidamente, porém a longo prazo existe incorrência de dívidas como dificuldade de manutenção, duplicidade de código, documentação, falta de documentação atualizada, falta de testes, etc. Desta forma, caracterizamos esse fato de dívida de variabilidade e por meio da revisão sistemática da literatura e dos diversos casos de implementação de variabilidade por meio de reúso oportunista identificados nesses estudos, elencamos quais

são as dívidas técnicas comumente incorridas nestes cenários, criando assim um catálogo, chamado de catálogo de dívida de variabilidade.

Na sequência, realizou-se um estudo de caso em uma empresa para avaliar se o que foi encontrado na literatura é o que está ocorrendo na prática em indústrias e coletar a percepção dos profissionais quanto a implantação de sistemas de variabilidade por meio de práticas não sistemáticas e o impacto disso nas organizações.

Os resultados mostraram que o que ocorre na literatura também é o que vem acontecendo na prática e que dívidas técnicas de variabilidade acontecem efetivamente. Obteve-se também diversos relatos de profissionais evidenciando o quanto essa prática é prejudicial, por exemplo, para manutenção e usabilidade do sistema.

Desta forma, após a realização da revisão sistemática da literatura e da execução do estudo de caso multiprojetos, percebe-se que em casos que a implantação da variabilidade de software acontece de forma não sistemática, ou seja, através do reúso oportunista, a incorrência de dívida técnica é imediata.

O próximo capítulo apresenta trabalhos com assuntos semelhantes a este, mas que apresentam diversas limitações, tornando o presente estudo inédito.

5 Trabalhos relacionados

A pesquisa de Li, Avgeriou e Liang (2015) teve como objetivo coletar estudos sobre dívida técnica e gestão de dívida técnica, e fazer uma classificação e análise temática sobre esses estudos, para obter uma compreensão abrangente sobre o conceito de divida técnica e uma visão geral sobre o estado atual da pesquisa em gestão de dívida técnica. O estudo consistiu de um mapeamento sistemático com 94 publicações selecionadas entre 1992 e 2013, na qual a dívida técnica foi classificada em 10 tipos e foram identificadas 8 atividades distintas para gestão de dívida técnica. Como resultado os autores perceberam que o termo "dívida" tem sido usado de maneiras diferentes por pessoas diferentes, o que leva a interpretações variadas. Eles também concluíram que existe uma necessidade de mais estudos empíricos com evidências de alta qualidade sobre todo o processo de gerenciamento de dívida técnica - TDM (*Technical Debt Management*) e sobre a aplicação de abordagens específicas de TDM em ambientes industriais, Nesse contexto, nosso estudo se diferencia por realizar a caracterização de DT e sua relação com o conceito de variabilidade implementado de forma inadequada. Nosso estudo prático, foi avaliando este cenário e não todo o TDM. O catálogo apresentado por Li, Avgeriou e Liang (2015) aborda os tipos de dívida técnica de modo geral. O nosso estudo utilizou este catálogo como base, porém apresenta as dívidas técnicas na ótica de gestão de variabilidade, mapeando quais são as dívidas causadas quando se implementa variabilidade por meio do reúso oportunista.

Para Siebra et al. (2012), o processo de desenvolvimento de grandes sistemas de software certamente requer vários tipos de decisões de projeto, que afetam o futuro das atividades de desenvolvimento. O uso da metáfora da dívida técnica é adequada para apoiar este processo de decisão. Na verdade, a DT fornece duas habilidades: de monitorar a evolução do efeito das decisões e representa aspectos de desenvolvimento de baixo nível em uma linguagem mais abrangente e direta para as partes interessadas. Assim, a dívida técnica pode ser vista como, ou tem potencial para ser, uma poderosa ferramenta para atividades de gestão e comunicação. Infelizmente, o estado da arte em dívida técnica ainda não culminou em modelos de análise rigorosos, principalmente para projetos de grande escala, deixando uma lacuna em teorias formais sobre isso. Considerando esse contexto, o grupo de pesquisa de Siebra et al. (2012) realizou um estudo para analisar um projeto industrial de grande escala na perspectiva de suas decisões e eventos relacionados, com objetivo de caracterizar a existência de dívida técnica e mostrar a evolução de seus parâmetros. Trata-se de uma investigação exploratória para identificar fontes de DT e compreender sua evolução. O objeto de estudo foi o sistema SMB (*Samsung Mobile Business*). Como este aplicativo era constantemente modificado, o processo de manutenção e evolução apresentou várias dificuldades. Os autores argumentaram que a teoria de dívida

técnica poderia ter sido usada para atenuar tais dificuldades. Esta foi a principal motivação da Samsung em investir neste estudo exploratório, para que seus resultados pudessem ser utilizados em projetos futuros. O estudo foi baseado em um único sistema mas sobre três cenários distintos: Camada de persistência, camada de interface e na etapa de inclusão de uma nova linguagem. Porém, o estudo não foi realizado com o aspecto de avaliar a relação das características de variabilidade deste sistema: se existiam, como estavam relacionadas a DT e se interferiam na DT como proposto neste trabalho.

Um objetivo importante da gestão da dívida técnica é avaliar o momento apropriado para pagar um item de DT e aplicar efetivamente os critérios de tomada de decisão para equilibrar os benefícios de curto prazo com os custos de longo prazo. No entanto, embora existam diversos estudos que propõem estratégias para a gestão da dívida técnica, os critérios de decisão muitas vezes são discutidos em segundo plano e, às vezes, nem sequer são mencionados. Assim, Ribeiro et al. (2016) realizaram uma pesquisa para identificar, por meio de um estudo sistemático de mapeamento da literatura, os critérios de tomada de decisão que têm sido propostos para subsidiar a gestão da DT. Como resultado foram obtidos 14 critérios de tomada de decisão que podem ser usados pelas equipes de desenvolvimento para priorizar o pagamento de itens de DT e uma lista de tipos de dívida relacionada aos critérios Ribeiro et al. (2016). Em um estudo similar, Arvanitou et al. (2019) desenvolveu uma ferramenta de monitoramento da qualidade por meio de um painel com várias visões, no qual são apresentadas as métricas relativas ao fenômeno de interesse da DT. O objetivo foi criar indicadores para as partes interessadas (como por exemplo gerentes, desenvolvedores e arquitetos) para medição, priorização e reembolso da DT. A pesquisa envolveu 60 engenheiros de software que trabalham para 11 empresas de desenvolvimento de software localizadas em 9 países, para entender suas necessidades de TDM. Os resultados do estudo sugerem que diferentes partes interessadas precisam de uma visão diferente do painel de qualidade, mas também alguns pontos em comum podem ser identificados. Por exemplo, por um lado, os gerentes estão mais interessados em conceitos financeiros, enquanto por outro lado, os desenvolvedores estão mais interessados na natureza dos problemas que existem no código.

Ainda seguindo a linha de pesquisa de priorização de dívida técnica, recentemente Lenarduzzi et al. (2021) realizaram uma revisão sistemática da literatura de 557 artigos originais publicados até 2020 com o objetivo de investigar o corpo de conhecimento existente em engenharia de software para entender quais abordagens de priorização de dívida técnica foram propostas na pesquisa e na indústria. Para eles, as empresas de software precisam gerenciar e refatorar questões de dívida técnica. Portanto, é necessário entender se e quando a refatoração da dívida técnica deve ser priorizada em relação ao desenvolvimento de recursos ou correção de *bugs*. Após a pesquisa, (LENARDUZZI et al., 2021) concluíram que não há consenso sobre quais são os fatores importantes e como medi-los.

Os últimos 3 trabalhos relacionados apresentados possuem uma característica em comum: discutir a priorização do pagamento da dívida técnica. Neste contexto, eles diferem-se do presente estudo, pois apresentamos uma caracterização sobre a dívida técnica no contexto de variabilidade e não estamos discutindo a forma e o momento mais adequado de realizar o pagamento da DT nesta situação.

Fenske e Schulze (2015) discutem que a variabilidade aumenta a complexidade do código-fonte e, portanto, impede a compreensão e manutenção. Além disso, a variabilidade impede o uso de algumas técnicas, como a análise do código-fonte, porque as abordagens atuais não tratam da variabilidade. Com base nessas limitações, os autores revisitaram os *code smells* à luz da variabilidade. Para esse fim, eles introduzem a noção de variabilidade nos *code smells* existentes, resultando em um catálogo inicial de *code smells* que reconhecem a variabilidade. Eles propuseram quatro *code smells* que levam em conta a variabilidade. Esses *code smells* sensíveis à variabilidade foram avaliados em uma pesquisa com 15 pesquisadores que são especialistas em linhas de produtos de software. Os resultados revelaram que a maioria dos praticantes observaram os *code smells* em sistemas do mundo real e reconheceram que tais *code smells* podem atrapalhar a manutenção e a evolução. Nosso estudo complementa este estudo relacionado, à medida que exploramos a literatura existente a fim de fornecer mais caracterizações de dívida de variabilidade.

Digkas et al. (2019) realizaram uma investigação sobre o efeito da reutilização de software na dívida técnica. No entanto, seu foco era reutilizar partes de código copiados e colados do site StackOverflow. StackOverflow é o maior site de perguntas e respostas (ou seja, aproximadamente 15 bilhões de perguntas respondidas) e altos níveis de lealdade dos usuários, que sugerem que mais de 60% deles visitam o site pelo menos uma vez por dia. Hoje em dia, os engenheiros de software são membros de várias comunidades de compartilhamento de conhecimento, que na forma de perguntas e respostas (Q&A) fornecem soluções prontas para uso para vários problemas de desenvolvimento e facilitam a troca de ideias entre os profissionais. Este artigo, realizou um estudo empírico para comparar a densidade da Dívida Técnica dos fragmentos de código StackOverflow com a dos projetos nos quais esses fragmentos foram reutilizados. Os resultados fornecem *insights* sobre o impacto potencial da reutilização de código em pequena escala na dívida técnica e destacam os benefícios de avaliar a qualidade do código antes de realizar o reúso oportunista (copiar/colar). Além disso, apesar de lidar com reúso e dívida técnica, os autores deste artigo não consideram famílias de sistemas, ou seja, não geram variantes a partir de reúso oportunista.

O estudo realizado por Verdecchia et al. (2021), atua no conceito de Dívida Técnica Arquitetural (ATD). Esta é a dívida técnica incorrida no nível arquitetural de *design* de software, ou seja, nas decisões relacionadas à escolha da estrutura (por exemplo, camadas, decomposição em subsistemas, interfaces), as opções de tecnologias (por exemplo,

frameworks, pacotes, bibliotecas, abordagem de implantação), ou mesmo linguagens, processo de desenvolvimento e plataforma. À medida que os sistemas de software crescem em tamanho e sua vida útil se estende por muitos anos, muitas dessas opções de *design* originais tornam-se restrições e limitam a evolução futura ou até a evitam. Para evoluir o sistema, os desenvolvedores devem encontrar soluções alternativas e, muitas vezes, soluções complicadas, que apresentam problemas de qualidade e atrasos. Este estudo, teve como objetivo compreender como as empresas de desenvolvimento de software conceituam a dívida arquitetural e como eles lidam com ela em sistemas intensivos de software. Diferentemente desta pesquisa que focou em sistemas intensivos de software e na dívida em nível arquitetural, nosso estudo abordou sistemas que implementam a variabilidade de forma não sistemática e observou qualquer tipo de dívida técnica, independentemente do seu nível (código ou arquitetura).

Recentemente, Capilla et al. (2021) investigou a tendência de reúso oportunista que afeta as taxas de dívida técnica em vários projetos de código aberto. Além disso, analisou a taxa de incidência de dívida técnica antes e depois de reutilizar componentes de terceiros, e quais são as implicações para a arquitetura de software. Mais especificamente, os autores usaram duas ferramentas, SonarQube e Arcan, para analisar o código e as taxas de dívida da arquitetura em três aplicações diferentes investigando os efeitos da reutilização de funcionalidade em repositórios de código aberto. Diferentemente do nosso objetivo de investigar dívidas na gestão da variabilidade, eles se preocuparam com as dificuldades e impactos no reaproveitamento de bibliotecas de terceiros.

Desta forma, percebe-se que nossa pesquisa é inédita, nenhum outro trabalho já realizado e publicado na literatura investiga a relação de implementar variabilidade de forma inadequada e seus impactos com a dívida técnica.

5.1 Considerações finais

Dívida técnica refere-se a decisões de projeto ou implementação para alcançar resultados de curto prazo, mas levam a inconvenientes técnicos a médio e longo prazo. O gerenciamento de variabilidade consiste em lidar explicitamente com as características variantes de um sistema de software ao longo de seu ciclo de vida. Tanto a dívida técnica quanto o gerenciamento da variabilidade estão sendo estabelecidas como disciplinas de engenharia com seus próprios papéis, metodologias e práticas. Nas empresas, são um indicador de maiores níveis de maturidade no desenvolvimento de sistemas intensivos em software. No entanto, percebe-se pelos trabalhos relacionados que, essas duas disciplinas ainda não foram devidamente investigadas em conjunto. Assim, o presente trabalho, investiga a interseção entre dívida técnica e gestão da variabilidade para caracterizar como

as empresas incorreram em dívida técnica relacionada aos aspectos da variabilidade e suas consequências.

6 Conclusão

A variabilidade de software refere-se a capacidade de um produto ser estendido, customizado ou configurado para uso ou reúso em contexto específicos. Existem várias maneiras de se implementar variabilidade, dentre elas incluem linhas de produtos de software, interfaces de configuração de componentes de software, e o método que em geral as empresas acabam optando, o reúso oportunista (utilizando por exemplo o copiar e colar). Já a dívida técnica é uma metáfora criada em 1992 para descrever a dívida que uma organização de software incorre quando escolhe uma abordagem de projeto ou construção que é conveniente no curto prazo, mas que aumenta a complexidade e os problemas de manutenção do software a longo prazo.

Ambos os conceitos de variabilidade e dívida técnica são extremamente relevantes na área de Engenharia de Software. Contudo, ainda não haviam sido investigados em conjunto para compreender sua relação. Diante disso, esse trabalho foi desenvolvido para compreender como a dívida técnica é ocasionada por meio de um gerenciamento de variabilidade inadequado, investigando esse fenômeno em estudos de casos existentes na literatura por meio de uma revisão sistemática da literatura e compreendendo como acontece na prática e qual a opinião deste assunto dos profissionais envolvidos por meio de um estudo de caso multiprojetos.

Após o desenvolvimento deste trabalho, conclui-se que implementar variabilidade de forma inadequada impacta em diferentes tipos de dívidas técnicas no ecossistema da empresa. Para facilitar a compreensão e fortalecer a relação que existe entre ambos conceitos o termo *dívida de variabilidade* foi criado para indicar dívida técnica relacionada a variabilidade. Além disso, pode-se confirmar que as características descritas referentes aos termos variabilidade e dívida técnica na literatura é o que de fato acontece na prática nas empresas, e também o que é compreendido dentro das empresas pela maioria dos profissionais. Este trabalho também apresenta um catálogo de tipos de dívida de variabilidade com o objetivo de divulgar o conceito de dívida de variabilidade e apresentar os principais tipos de dívida de variabilidade já identificados na literatura e confirmados pelos profissionais que trabalham com isso no dia-a-dia. Além disso, o catálogo tem como objetivo divulgar conhecimentos e despertar o interesse de investigadores da área.

6.1 Contribuições

A partir do exposto, pode-se resumir a contribuição do presente trabalho em três tipos:

- Contribuição Teórica: Com o desenvolvimento deste trabalho espera-se auxiliar pesquisadores e profissionais das industrias a encontrem conteúdo referente ao estudo envolvendo dois conceitos importantes da engenharia de software e a sua relação: Variabilidade e Dívida Técnica. Além disso, este trabalho foi desenvolvido utilizando dois métodos de pesquisa: Revisão Sistemática da Literatura e Estudo de Caso. Desta forma, os leitores podem verificar como aplicar esses métodos em suas pesquisas com o passo-a-passo para execução;

- Contribuição Empírica: Foi possível relatar os dados observados em três objetos de estudo práticos, coletando evidências e reportando os resultados observados para evidenciar a pesquisa realizada inicialmente;

- Contribuição Prática: os profissionais de outras empresas poderão consultar nosso catálogo para entender o que ocorrerá caso optarem por implementar variabilidade de forma inadequada.

6.2 Limitações e trabalhos futuros

Uma das limitações apresentadas por este trabalho é que o estudo de caso foi realizado apenas em um ambiente onde o Departamento de Tecnologia de Informação faz parte de uma corporação maior. Ou seja, esse estudo não corresponde ao contexto de uma fábrica de software onde o resultado da quantidade de variabilidade e dívida técnica poderia ser diferente. Desta forma, existe uma oportunidade de trabalhos futuros para avaliar a dívida de variabilidade com estudos de casos em outros contextos.

Além disso, o catálogo proposto ainda se limita a ser uma forma de consulta inicial aos profissionais que o desejam utilizar. O catálogo não apresenta uma forma de priorizar as dívidas técnicas e nem de como corrigi-las. Como trabalho futuro pretende-se adicionar ao catálogo uma forma de solução para cada tipo de dívida técnica e assim ajudar ainda mais os pesquisadores e profissionais que tiverem acesso a este catálogo.

Por fim, este trabalho se limitou a caracterizar o novo conceito de dívida de variabilidade, permanecendo somente na fase de identificação. Para abordar o gerenciamento de dívida completamente, ainda existem as etapas de: mensuração, priorização, pagamento ou reembolso, e monitoramento. Assim, pretendemos expandir nossa pesquisa para compreender o ciclo total de gerenciamento de dívida de variabilidade. Um trabalho futuro pode ter como foco a identificação e medição de dívida técnica para tópicos de variabilidade com abordagens concretas, por exemplo, diretrizes e ferramentas para apoiar a criação de casos de negócios para raciocinar sobre a variabilidade da dívida. Além disso, pesquisas futuras incluem a validação e gerenciamento de tipos de dívidas de variabilidade com novos estudos de caso industriais.

Referências

ALI, M. S.; BABAR, M. A.; SCHMID, K. A comparative survey of economic models for software product lines. In: *35th Euromicro Conf. on Software Engineering and Advanced Applications*. [S.l.: s.n.], 2009. p. 275–278. Citado na página 14.

ALLMAN, E. Managing technical debt. *Commun. ACM*, Association for Computing Machinery, New York, NY, USA, v. 55, n. 5, p. 50–55, may 2012. ISSN 0001-0782. Disponível em: <https://doi.org/10.1145/2160718.2160733>. Citado na página 21.

ALVES, N. S. et al. Identification and management of technical debt: A systematic mapping study. *Information and Software Technology*, v. 70, p. 100 – 121, 2016. ISSN 0950-5849. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0950584915001743>. Citado na página 15.

ARVANITOU, E.-M. et al. Monitoring technical debt in an industrial setting. In: *Proceedings of the Evaluation and Assessment on Software Engineering*. New York, NY, USA: Association for Computing Machinery, 2019. (EASE '19), p. 123–132. ISBN 9781450371452. Disponível em: <https://doi.org/10.1145/3319008.3319019>. Citado na página 72.

ASSUNÇÃO, W. K. G. et al. Reengineering legacy applications into software product lines: a systematic mapping. *Empirical Software Engineering*, v. 22, n. 6, p. 2972–3016, 2017. Citado 2 vezes nas páginas 14 e 32.

AVGERIOU, P. et al. Managing technical debt in software engineering (dagstuhl seminar 16162). *Dagstuhl Reports*, v. 6, n. 4, p. 110–138, 2016. Citado na página 15.

BERGER, T. et al. A survey of variability modeling in industrial practice. In: *7th Int. Workshop on Variability Modelling of Software-intensive Systems*. [S.l.]: ACM, 2013. p. 7:1–7:8. Citado na página 15.

BERGER, T. et al. The state of adoption and the challenges of systematic variability management in industry. *Empirical Software Engineering*, v. 25, n. 3, p. 1755–1797, 2020. Citado 5 vezes nas páginas 14, 15, 18, 19 e 20.

BRIZOLA, J.; FANTIN, N. Revisão da literatura e revisão sistemática da literatura. *Revista de Educação do Vale do Arinos-RELVA*, v. 3, n. 2, 2016. Citado 2 vezes nas páginas 31 e 32.

BROWN, N. et al. Managing technical debt in software-reliant systems. In: *Proceedings of the FSE/SDP Workshop on the Future of Software Engineering Research, FoSER 2010*. [S.l.: s.n.], 2010. (Proceedings of the FSE/SDP Workshop on the Future of Software Engineering Research, FoSER 2010), p. 47–51. ISBN 9781450304276. FSE/SDP Workshop on the Future of Software Engineering Research, FoSER 2010 ; Conference date: 07-11-2010 Through 11-11-2010. Citado 3 vezes nas páginas 15, 16 e 21.

CAPILLA, R.; BOSCH, J.; KANG, K. *Systems and Software Variability Management: Concepts, Tools and Experiences*. [S.l.]: Springer, 2013. ISBN 9783642365843. Citado 3 vezes nas páginas 14, 19 e 20.

CAPILLA, R. et al. Impact of opportunistic reuse practices to technical debt. In: *2021 IEEE/ACM International Conference on Technical Debt (TechDebt)*. [S.l.: s.n.], 2021. p. 16–25. Citado na página 74.

COLLIS, J.; HUSSEY, R. *Business research: A practical guide for undergraduate and postgraduate students*. [S.l.: s.n.], 2014. ISBN 978-0-230-30183-2. Citado na página 29.

CUNNINGHAM, W. The wycash portfolio management system. *SIGPLAN OOPS Mess.*, Association for Computing Machinery, New York, NY, USA, v. 4, n. 2, p. 29–30, dez. 1992. ISSN 1055-6400. Disponível em: <https://doi.org/10.1145/157710.157715>. Citado 3 vezes nas páginas 15, 20 e 21.

CURTIS, B.; SAPPIDI, J.; SZYNKARSKI, A. Estimating the size, cost, and types of technical debt. In: . [S.l.: s.n.], 2012. p. 49–53. ISBN 978-1-4673-1748-1. Citado 2 vezes nas páginas 21 e 22.

CZARNECKI, K. Variability in software: State of the art and future directions. In: CORTELLESSA, V.; VARRÓ, D. (Ed.). *Fundamental Approaches to Software Engineering*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013. p. 1–5. ISBN 978-3-642-37057-1. Citado na página 18.

DIGKAS, G. et al. Reusing code from stackoverflow: The effect on technical debt. In: *2019 45th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. [S.l.: s.n.], 2019. p. 87–91. Citado na página 73.

DUBINSKY, Y. et al. An exploratory study of cloning in industrial software product lines. In: *17th European Conference on Software Maintenance and Reengineering*. [S.l.]: IEEE Computer Society, 2013. p. 25–34. Citado 3 vezes nas páginas 15, 19 e 20.

EISENBERG, R. J. A threshold based approach to technical debt. *SIGSOFT Softw. Eng. Notes*, Association for Computing Machinery, New York, NY, USA, v. 37, n. 2, p. 1–6, apr 2012. ISSN 0163-5948. Disponível em: <https://doi.org/10.1145/2108144.2108151>. Citado na página 21.

FENSKE, W.; SCHULZE, S. Code smells revisited: A variability perspective. In: *Proceedings of the Ninth International Workshop on Variability Modelling of Software-Intensive Systems*. New York, NY, USA: Association for Computing Machinery, 2015. (VaMoS '15), p. 3–10. ISBN 9781450332736. Citado na página 73.

FOWLER, M. *Technical Debt Quadrant*. 2014. Disponível em: <https://martinfowler.com/bliki/TechnicalDebtQuadrant.html>. Citado 3 vezes nas páginas 9, 23 e 24.

Frankenthal, Rafaela. *Entenda a escala Likert e saiba como aplicá-la em sua pesquisa*. 2022. Disponível em: <https://mindminers.com/blog/entenda-o-que-e-escala-likert/>. Citado na página 39.

GALSTER, M.; AVGERIOU, P. Handling variability in software architecture: Problems and implications. In: *2011 Ninth Working IEEE/IFIP Conference on Software Architecture*. [S.l.: s.n.], 2011. p. 171–180. Citado na página 19.

GALSTER, M. et al. Variability in software systems—a systematic literature review. *IEEE Transactions on Software Engineering*, v. 40, n. 3, p. 282–306, 2014. Citado 3 vezes nas páginas 14, 18 e 19.

GURP, J. van; BOSCH, J.; SVAHNBERG, M. On the notion of variability in software product lines. In: *Proceedings Working IEEE/IFIP Conference on Software Architecture*. [S.l.: s.n.], 2001. p. 45–54. Citado na página 18.

IDEC. *Conheça o papel das agências reguladoras*. 2013. Disponível em: <https://idec.org.br/consultas/dicas-e-direitos/conheca-o-papel-das-agencias-reguladoras>. Citado na página 14.

KLEIN, H. K.; MYERS, M. D. A set of principles for conducting and evaluating interpretive field studies in information systems. *MIS Q.*, Society for Information Management and The Management Information Systems Research Center, USA, v. 23, n. 1, p. 67–93, mar. 1999. ISSN 0276-7783. Disponível em: <https://doi.org/10.2307/249410>. Citado na página 30.

KRUCHTEN, P.; NORD, R.; OZKAYA, I. *Managing Technical Debt: Reducing Friction in Software Development*. 1st. ed. [S.l.]: Addison-Wesley Professional, 2019. ISBN 013564593X. Citado na página 15.

KRUEGER, C. W. Easing the transition to software mass customization. In: *Software Product-Family Engineering, 4th Int. Workshop, PFE 2001, Bilbao, Spain, October 3-5, 2001, Revised Papers*. [S.l.]: Springer, 2001. (LNCS, v. 2290), p. 282–293. Citado na página 14.

KRUEGER, J.; MAHMOOD, W.; BERGER, T. Promote-pl: A round-trip engineering process model for adopting and evolving product lines. In: *Proceedings of the 24th ACM Conference on Systems and Software Product Line: Volume A - Volume A*. New York, NY, USA: Association for Computing Machinery, 2020. (SPLC '20). ISBN 9781450375696. Citado na página 14.

KRüGER, J.; BERGER, T. An empirical analysis of the costs of clone- and platform-oriented software reuse. In: . New York, NY, USA: Association for Computing Machinery, 2020. (ESEC/FSE 2020), p. 432–444. ISBN 9781450370431. Disponível em: <https://doi.org/10.1145/3368089.3409684>. Citado 3 vezes nas páginas 19, 20 e 32.

LAKATOS., E. M.; MARCONI, M. de A. *Fundamentos da Metodologia científica*. São Paulo: Atlas, 2003. ISBN 8522433976. Citado na página 38.

LAKATOS., E. M.; MARCONI, M. de A. *Metodologia científica*. São Paulo: Atlas, 2006. Citado na página 33.

LENARDUZZI, V. et al. A systematic literature review on technical debt prioritization: Strategies, processes, factors, and tools. *Journal of Systems and Software*, v. 171, p. 110827, 2021. ISSN 0164-1212. Disponível em: <https://www.sciencedirect.com/science/article/pii/S016412122030220X>. Citado 2 vezes nas páginas 21 e 72.

LETHBRIDGE, T.; SIM, S.; SINGER, J. Studying software engineers: Data collection techniques for software field studies. *Empirical Software Engineering*, v. 10, p. 311–341, 07 2005. Citado na página 37.

LI, Z.; AVGERIOU, P.; LIANG, P. A systematic mapping study on technical debt and its management. *J. Syst. and Software*, v. 101, p. 193–220, 2015. Citado 6 vezes nas páginas 15, 46, 47, 49, 68 e 71.

LI, Z.; LIANG, P.; AVGERIOU, P. Architectural debt management in value-oriented architecting. In: *Economics-Driven Software Architecture*. [S.l.]: Morgan Kaufmann / Academic Press / Elsevier, 2013. p. 183–204. Citado na página 68.

LINDEN, F. van der; SCHMID, K.; ROMMES, E. *Software product lines in action - the best industrial practice in product line engineering*. [S.l.]: Springer, 2007. Citado 2 vezes nas páginas 14 e 19.

MAHDAVI-HEZAVEH, R.; DREMANN, J.; WILLIAMS, L. Software development with feature toggles: practices used by practitioners. *Empirical Software Engineering*, v. 26, n. 1, 2021. Citado na página 15.

MARTINEZ, J.; ASSUNÇÃO, W. K. G.; ZIADI, T. ESPLA: A catalog of extractive SPL adoption case studies. In: *21st Int. Systems and Software Product Line Conference*. [S.l.]: ACM, 2017. p. 38–41. Citado na página 32.

MARTINEZ, J. et al. Bottom-up technologies for reuse: Automated extractive adoption of software product lines. In: *Proceedings of the 39th International Conference on Software Engineering Companion*. IEEE Press, 2017. (ICSE-C '17), p. 67–70. ISBN 9781538615898. Disponível em: <https://doi.org/10.1109/ICSE-C.2017.15>. Citado na página 41.

MCCONNELL, S. Managing technical debt. *Construx*, p. 1–14, 01 2013. Citado 2 vezes nas páginas 22 e 23.

PRODANOV, C. C.; FREITAS, E. C. d. *Metodologia do trabalho científico: métodos e técnicas da pesquisa e do trabalho acadêmico*. Novo Hamburgo: Feevale, 2013. ISBN 9788577171583. Citado na página 38.

RIBEIRO, L. et al. Decision criteria for the payment of technical debt in software projects: A systematic mapping study. In: . [S.l.: s.n.], 2016. p. 572–579. Citado na página 72.

ROBSON, C. *Real world research : a resource for social scientists and practitioner-researchers*. 2st. ed. [S.l.]: Blackwell Publishers, 2002. Citado na página 30.

ROSSER, L. A.; NORTON, J. H. A systems perspective on technical debt. In: *2021 IEEE Aerospace Conference (50100)*. [S.l.: s.n.], 2021. p. 1–10. Citado na página 24.

RUNESON, P. et al. *Case Study Research in Software Engineering: Guidelines and Examples*. 1st. ed. [S.l.]: Wiley Publishing, 2012. ISBN 1118104358. Citado 5 vezes nas páginas 29, 33, 35, 38 e 41.

RUNESON, P.; HöST, M. Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering*, v. 14, 2008. Citado 3 vezes nas páginas 30, 37 e 41.

SCHMID, K.; JOHN, I. A customizable approach to full lifecycle variability management. *Science of Computer Programming*, Elsevier, v. 53, n. 3, p. 259–284, 2004. Citado na página 15.

SHARMA, T. Quantifying quality of software design to measure the impact of refactoring. In: *2012 IEEE 36th Annual Computer Software and Applications Conference Workshops*. [S.l.: s.n.], 2012. p. 266–271. Citado na página 21.

SHULL, F.; SINGER, J.; SJBERG, D. I. K. *Guide to Advanced Empirical Software Engineering*. 1st. ed. [S.l.]: Springer Publishing Company, Incorporated, 2010. ISBN 1849967121. Citado na página 32.

SIEBRA, C. A. et al. Managing technical debt in practice: An industrial report. In: *Proceedings of the 2012 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*. [S.l.: s.n.], 2012. p. 247–250. Citado na página 71.

SOUZA, I. S. et al. Investigating variability-aware smells in spls: An exploratory study. In: *33rd Brazilian Symposium on Software Engineering*. [S.l.]: ACM, 2019. p. 367–376. Citado na página 15.

SPLC. *Product Line Hall of Fame*. 2020. Disponível em: <https://splc.net/fame.html>. Citado na página 32.

SVAHNBERG et al. A taxonomy of variability realization techniques: Research articles. John Wiley & Sons, Inc., USA, v. 35, n. 8, p. 705–754, jul. 2005. ISSN 0038-0644. Citado na página 19.

TOM, E.; AURUM, A.; VIDGEN, R. An exploration of technical debt. *Journal of Systems and Software*, v. 86, n. 6, p. 1498–1516, 2013. ISSN 0164-1212. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0164121213000022>. Citado na página 22.

TëRNAVA, X. et al. Identification and visualization of variability implementations in object-oriented variability-rich systems: a symmetry-based approach. *Automated Software Engineering*, v. 29, 05 2022. Citado na página 19.

VERDECCHIA, R. et al. Building and evaluating a theory of architectural technical debt in software-intensive systems. *Journal of Systems and Software*, v. 176, p. 110925, 2021. ISSN 0164-1212. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0164121221000224>. Citado na página 73.

WOHLIN, C.; AURUM, A. Towards a decision-making structure for selecting a research design in empirical software engineering. *Empirical Software Engineering*, v. 20, 05 2014. Citado na página 28.

WOHLIN, C. et al. *Experimentation in Software Engineering*. [S.l.]: Springer Publishing Company, Incorporated, 2012. ISBN 3642290434. Citado 5 vezes nas páginas 9, 29, 37, 38 e 41.

WOLFART, D.; ASSUNCAO, W. K. G.; MARTINEZ, J. Variability debt: Characterization, causes and consequences. In: *XX Brazilian Symposium on Software Quality*. New York, NY, USA: Association for Computing Machinery, 2021. (SBQS '21). ISBN 9781450395533. Disponível em: <https://doi.org/10.1145/3493244.3493250>. Citado na página 14.

YIN, R. K. *Case Study Research: Design and Methods*. 4st. ed. [S.l.]: SAGE Publications, 2009. Citado 2 vezes nas páginas 33 e 36.

Apêndices

APÊNDICE A – Artigo publicado na XX SBQS - Simpósio Brasileiro de Qualidade de Software

Variability Debt: Characterization, Causes and Consequences

Daniele Wolfart
PPGComp, Western Paraná State University (UNIOESTE)
Cascavel, Brazil
danielewolfart@gmail.com

Wesley K. G. Assunção
Pontifical Catholic University of Rio de Janeiro (PUC-Rio)
Rio de Janeiro, Brazil
wesleyklewerton@gmail.com

Jabier Martinez
Tecnalia, Basque Research and Technology Alliance (BRTA)
Derio, Spain
jabier.martinez@tecnalia.com

ABSTRACT

Variability is an inherent property of software systems to create families of products dealing with needs of different customers and environments. However, some practices to manage variability may incur technical debt. For example, the use of opportunistic reuse strategies, e.g., clone-and-own, harms maintenance and evolution activities; or deciding to abandon variability management and deriving a single product with all the features might threaten system usability. These examples are common problems found in practice but, to the best of or knowledge, not properly investigated from the perspective of technical debt. To expand the knowledge on the research and practice of technical debt in the perspective of variability management, we report results of this phenomenon, which we defined as *variability debt*. Our work is based on 52 industrial case studies that report problems observed in the use of opportunistic reuse. The results show that variability debt is caused by business, operational and technical aspects; leads to complex maintenance, creates difficulties to customize and create new products, misuse of human resources, usability problems; and impacts artifacts along the whole life-cycle. Although some of these issues are investigated in the field of *systematic variability management*, e.g., software product lines, our contribution is to present them from a *technical debt* perspective to enrich and create synergies between the two fields. As additional contribution, we present a catalog of variability debts in the light of technical debts found in the literature.

CCS CONCEPTS

• **Software and its engineering → Software product lines**; **Maintaining software**; • **General and reference → Surveys and overviews**.

KEYWORDS

Technical Debt, Variability management, Software Product Lines, Variability Debt

SBQS '21, November 8–11, 2021, Virtual Event, Brazil

ACM ISBN 978-1-4503-9553-3/21/11…$15.00
https://doi.org/10.1145/3493244.3493250

ACM Reference Format:
Daniele Wolfart, Wesley K. G. Assunção, and Jabier Martinez. 2021. Variability Debt: Characterization, Causes and Consequences. In *XX Brazilian Symposium on Software Quality (SBQS '21), November 8–11, 2021, Virtual Event, Brazil.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3493244.3493250

1 INTRODUCTION

Variability is a paramount software property to allow the creation of variants. Variability is used for tailoring products for different markets according to diverse regulations, software that needs to work for diverse target hardware, or just the need to allow end-user customization to satisfy their specific needs [7]. Engineering families of software products with variability is a costly and challenging task [9]. While variability can be implemented in a systematic fashion, e.g., Software Product Lines (SPLs) [32], to support easier combinations of features, propagation of fixes, and customized improvements [32]; there are several barriers to reach this point. For instance, the extractive adoption of SPLs from existing variants requires a high up-front investment [2, 3, 19, 21]. Also, predicting beforehand the need for adding variability, e.g., immature domains, requires extensive effort, and sometimes is impossible. Alternatively, even being aware of the systematic approaches, companies rather rely on opportunistic strategies for variability management, such as clone-and-own [12] and feature toggles [24], in detriment of other quality attributes such as the maintainability of the product family as a whole. Yet, avoiding variability, with a one-size-fits-all product, is not an ideal solution either, since it might lead to usability problems. Finally, independently of the variability management implemented, taking sub-optimal decisions during evolution is also a problem in the mid- and long-term [30].

To further investigate the situations presented above, in this work we present results of the investigation of variability management from a technical debt perspective. Technical debt refers to design or implementation decisions to achieve short term results, but lead to technical drawbacks in a mid- and long-term [4]. Variability management consists of explicitly dealing with variant characteristics of a software system along its life-cycle [29]. Both technical debt [4, 16, 18, 22] and variability management [5, 6, 7] are being set up as engineering disciplines with their own roles and methodologies and practices. In companies, they are an indicator of higher levels of maturity in the development of software-intensive systems [9, 35]. However, to the best of or knowledge, these both disciplines were not properly investigated together yet. Thus, in this paper, we investigate the intersection of technical debt and variability management to characterize how companies incurred

technical debt related to variability aspects and its consequences. We use the term *variability debt* to indicate technical debt related to variability.

For our analysis, we revisit 52 industrial cases of dedicated literature to provide a summary from a technical debt perspective. The results revealed that variability debt is caused by organizational, operational, and technical aspects. Along the development life-cycle, several artifacts are impacted by variability debt, and the debt leads to complex maintenance, difficulties to customize and create new products, misuse of human resources, and usability problems. As a contribution of this study, we aim to bring the technical debt community to variability issues and vice-versa. Notably, the technical debt research roadmap [17] includes the identification of the most common causes of technical debt, so our contribution is aligned through a first characterization of variability debt. Additionally, we contribute to the research and practice with a catalog of variability debts in the light of technical debts found in the literature added of new debts found in our study. From this catalog, one can find the technical debt that occurs when variability is implemented in a non-systematic way, its causes, consequence and the main artifacts involved.

This paper is structured as follows: Section 2 introduces the definition of variability debt. Section 3 details the methodology and design of the analysis. Then, Section 4 presents our findings and Section 5 discusses them. Section 6 presents the related work and describes differences to our study. Finally, Section 8 concludes and outlines future work.

2 VARIABILITY DEBT

On the one hand, technical debt is "*a collection of design or implementation constructs that are expedient in the short term, but set up a technical context that can make future changes more costly or impossible. Technical debt presents an actual or contingent liability whose impact is limited to internal system qualities, primarily maintainability and evolvability*" [4]. On the other hand, "*Variability management encompasses the activities of explicitly representing variability in software artifacts throughout the life-cycle, managing dependencies among different variabilities, and supporting the instantiation of the variabilities.*" [29]. Based on these definitions of technical debt and variability management, we introduce a new category of technical debt, as follows:

Variability Debt. *Technical debt caused by defects and sub-optimal solutions in the implementation of variability management in software systems. That is, companies may choose to implement variability with an approach that would bring short-term benefits, e.g., using opportunistic reuse, regardless of technical/architectural drawbacks in the medium and long term. Variability debt leads to maintenance and evolution difficulties to manage families of systems or highly-configurable systems.*

3 STUDY DESIGN

Our study has as goal the characterization of variability debt from the perspective of its causes, consequences, and artifacts. Based on that, we pose the following research questions (RQs):

RQ1: What has led the company to incur variability debt? This RQ aims to explain the cause, i.e., drivers or forces, that lead companies to incur variability debt.

RQ2: Which are the artifacts impacted by variability debt? With this RQ we want to investigate the different types of artifacts impacted by variability debt along the product life-cycle.

RQ3: What are the consequences of variability debt? In this RQ the focus is to reason about the types of variability debt and the consequences faced by companies that accumulated variability debt for dealing with families of software products.

For a first characterization of variability debt, we performed a focused search in three traditionally used digital libraries, namely Scopus[1], IEEE Xplore[2], and ACM Digital Library[3] (Accessed: 21 April 2021); to find pieces of work to answer our RQs. However, only 3 publications with 4 case studies were identified in the intersection of technical debt and variability. Since we are aware of the discussion about issues related to opportunistic reuse and re-engineering of system variants into SPLs to deal with variability problems [3], we decided to additionally focus on extractive SPL adoption cases, as we consider those are the most notorious cases of variability-related problems. Based on that, our analysis relies on three sources of information: (i) a search for document in digital libraries, (ii) a systematic analysis of the cases available in the *Extractive SPL Adoption catalog (ESPLA)* [25], which is a community-driven effort, and (iii) case studies from a recent systematic literature review presented by *Krueger et al.* [20] on clone- and platform-oriented software reuse, which included several sources: the ESPLA itself, the "SPLC hall of fame" cases [31], industrial case descriptions from companies providing services around SPLs, a dedicated query to scientific databases, other references based on their knowledge, and snowballing readings.

Figure 1 presents the methodology we use in this work. In *Step 1*, we identified the relevant studies in the three sources of information. Firstly, using the string *("variability" OR "product line" OR "configurable system") AND ("techdebt" OR "technical debt")* in the title, abstract, and author keywords; we searched into Scopus, resulting in 18 retrieved documents; IEEE Xplore, retrieving 1 document; and ACM Digital Library, with 6 documents found. Among the 25 retrieved results, due to repetitions, 20 publications remained. Then, we applied a inclusion criterion for selecting papers that clearly mention reuse, variability, and SPLs together with technical debt. From ESPLA, we filtered only the industrial cases, and From the list in Krueger et al. [20], we focused on the ones where the strategy was from Clone & Own to Platform (i.e., C&O->P in their notation).

We identified a total of 72 unique case studies, with some case studies in several papers and eight cases in both sources. *Step 1* was performed by the second and third authors who are experts on the subject. Then, all decisions and actions were taken by the three authors, after discussions and reaching a consensus. In *Step 2* we performed a screening of the case studies where we discarded 17 publications found in digital libraries, and 24 one found in ESPLA and Krueger et al. [20]. These publications were discarded because

[1]https://www.scopus.com/
[2]https://ieeexplore.ieee.org
[3]https://dl.acm.org/

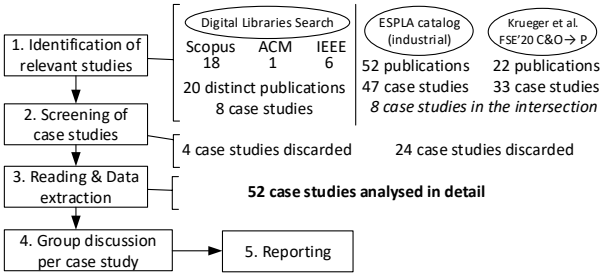

Figure 1: Methodology

(i) the content of the paper was not related to our topics of interest (industrial case studies, technical debt or variability debt) or (ii) they only described a tool to identify features or help migrating to SPLs but did not present the motivation that started the migration process. The complete reading and data extraction (*Step 3*) were performed in 52 case studies (4 found in digital libraries and 48 found in ESPLA + Krueger et al. [20]) from 38 primary sources. The references of all these primary sources are available in Appendix A. As this work was an exploratory study, the process used for Steps 3 and 4 was to read each paper carefully, noting every detail related to the RQs we wanted to answer and defining coding schemes. The information were collected exclusively from the paper, without using other artifacts of the case study. Then, we held weekly meetings to present what was found and discuss possible disagreements and integrate the data. Basically, we followed the steps of thematic analysis to collect and analyze the data obtained in the paper. In *Step 4*, we discussed each case study among us to agree on the conclusions from the technical debt perspective. Based on the readings, we conducted sessions assigning and harmonizing the names of the identified categories for the responses to the different RQs. Finally, in *Step 5*, we conducted the reporting that resulted in this paper.

Table 1 presents the 52 case studies considered in our study. The first column (#) of the table is used as an identifier in the next sections and the second column presents the name of the case study. In the third column (Ref.) we can observe that some studies were found in different papers. Also, in the last column (Source), we can see that the case studies came from different sources, namely ESPLA, Krueger et al. [20] (FSE'20), and digital libraries (DL).

Table 1: Case studies obtained from the primary sources

#	Case Study Name	Ref.	Source
1	Alcatel-Lucent IXM-PF	[S8, S32, S40]	ESPLA+FSE'20
2	BMD .NET-based Business Software	[S6]	ESPLA
3	Body Comfort System	[S36]	ESPLA
4	BSH induction hob families	[S1, S12]	ESPLA
5	Danfoss Drives frequency converters	[S11, S19, S20]	ESPLA+FSE'20
6	Defense domain systems	[S9]	ESPLA
7	Aerospace	[S9]	ESPLA
8	Electric motor controllers	[S9, S34]	ESPLA
9	DOPLER tool	[S14]	ESPLA
10	Fujitsu Kyushu Network TechNologies	[S31]	ESPLA+FSE'20
11	Global trading and settlement system	[S10]	ESPLA+FSE'20
12	Deutsche Bank	[S10]	ESPLA+FSE'20
13	HomeAway	[S25]	ESPLA
14	IEC 61499 Industrial Automation System	[S6]	ESPLA
15	Image Memory Handler (IMH)	[S24]	ESPLA
16	Industrial Automation System	[S4]	ESPLA
17	Industrial Pascal Modules	[S33]	ESPLA
18	KePlast platform	[S28]	ESPLA
19	Pharmaceutical wholesaler logistics	[S21, S22]	ESPLA
20	Power control and protection system	[S4]	ESPLA
21	Siemens VAI MSS tool	[S6, S32]	ESPLA+FSE'20
22	Siemens VAI Steel Plant Automation	[S6]	ESPLA
23	Traffic management systems	[S29]	ESPLA
24	Wingsoft Financial Management System	[S38, S39]	ESPLA
25	LGIS Elevator Control Software	[S26]	FSE'20
26	Business Unit	[S8]	FSE'20
27	Dialect Solutions	[S35]	FSE'20
28	Engenio	[S2, S16]	FSE'20
29	Common Avionics Architecture System	[S5]	FSE'20
30	ABB	[S32]	FSE'20
31	Boeing Company	[S32]	FSE'20
32	CelsiusTech Systems AB	[S32]	FSE'20
33	Cummins Inc.	[S32]	FSE'20
34	Hewlett-Packard	[S32]	FSE'20
35	LG Industrial Systems Co., Ltd.	[S32]	FSE'20
36	MARKET MAKER Software AG	[S32]	FSE'20
37	Philips Electronics & Medical Systems	[S32]	FSE'20
38	Robert Bosch GmbH	[S32]	FSE'20
39	Salion Inc.	[S32]	FSE'20
40	Testo AG	[S23, S32]	FSE'20
41	The National Reconnaissance Office	[S32]	FSE'20
42	The Naval Undersea Warfare Center AG	[S32]	FSE'20
43	Overwatch Textron Systems	[S17]	FSE'20
44	ORisk Consulting	[S18]	FSE'20
45	FISCAN	[S27]	FSE'20
46	Live Training Transformation (LT2)	[S7]	FSE'20
47	AEGIS Weapon System	[S13]	FSE'20
48	TIZEN	[S15]	FSE'20
49	Study in Start-Ups	[S30]	DL
50	Study in Scale-Ups	[S30]	DL
51	EA management system	[S37]	DL
52	Bosch Diesel Gasoline Systems	[S3]	DL

4 RESULTS

This section presents the results obtained from the analysis of the case studies to answer the posed research questions.

4.1 RQ1 - Context in which variability debt is incurred

Table 2 presents the drivers that cause debt of variability management. One of the main reasons in which companies incur variability debt is related to time pressure. Companies can decide not to pay attention on how to engineer variability properly, e.g., using opportunistic reuse, to decrease time to market [S3, S9, S10, S16, S18, S20, S23, S25, S32, S40] or reduce development costs with a short-term vision [S2, S5, S10, S11, S15, S16, S18, S20, S24, S25, S26, S29, S30, S31, S32, S35, S37].

Another situation that leads companies to fail in dealing with proper variability management, sometimes due to time pressure, concerns scenarios in which requirements constantly change. It can be related functional requirements where developers must respond to constant demand for changing or adding new functionalities [S2, S5, S6, S17, S20, S21, S32, S34, S38, S39, S40] or dealing with non-functional requirements, such as the deployment in new hardware and devices [S10, S11, S23, S32].

The lack of knowledge on variability management is also a factor that can lead to incur variability debt. Initial quick or poor design decisions in implementing variability are mentioned in several case studies [S14, S17, S23, S26, S32, S33, S34, S38, S39]. For instance, a company can excel in managing variability in source code but, in most of the cases, variations are also present in non-software

Table 2: Drivers that led to variability debt

Category	References
Time pressure	#1 [S40]; #5 [S11, S20]; #8 [S9]; #10 [S31]; #11 #12 [S10]; #13 [S25]; #15 [S24]; #23 [S29]; #25 [S26]; #27 [S35]; #28 [S2, S16]; #29 [S5]; #30 #31 #32 #33 #34 #35 #36 #37 #38 #39 [S32]; #40 [S23, S32]; #41 #42 [S32]; #44 [S18]; #47 [S13]; #48 [S15]; #49, #50 [S30]; #51 [S37]; #52 [S3]
Constantly changing scenarios	#1 [S40]; #3 [S6]; #5 [S11, S20]; #11 #12 [S10]; #16 [S4]; #19 [S21]; #22 [S6]; #24 [S38, S39]; #28 [S2]; #29 [S5]; #30 #31 #32 #33 #34 #35 #36 #37 #38 #39 [S32]; #40 [S23, S32]; #41 #42 [S32]; #43 [S17]; #47 [S13]; #49, #50 [S30]
Lack of knowledge on variability management	#1 [S8]; #8 [S34]; #9 [S14]; #17 [S33]; #24 [S38, S39] #25 [S26]; #30 #31 #32 #33 #34 #35 #36 #37 #38 #39 [S32]; #40 [S23, S32]; #41 #42 [S32]; #43 [S17];
Operational constraints	#11 #12 [S10]; #13 [S25]; #28 [S2]; #43 [S17]; #47 [S13]; #52 [S3];

core assets. For example, optional features can be related to requirement documents and/or test procedures. Apart from the variability implementation at all those levels, a global traceability among all these feature artifacts is desired [S17]. When practitioners do not know any variability mechanism, we see over-engineering to build products on a per-project basis, leading to artifact duplication, integration of unnecessary and undesired features [S37], development of features rarely used, increase of code complexity, and disappearance of links between implementation artifacts to business values [S8].

Diverse operational constraints can lead to incur variability debt. First, established processes in the company to handle the commonality and variability across all current customers and potential customers should be in place [S17]. The lack of domain analysis practices might hinder the possibility to create an architecture and infrastructure specifically designed to support an SPL. Second, complex operational settings can jeopardize a global vision and plan for the product family, e.g., integrating geographically distributed teams [S2, S13], or merging systems of different companies [S25]. For instance, distributed systems in which only local modifications are made can create problems because a significant number of variants can emerge [S10]. Operational constraints might also include efforts to accommodate legacy and third-party code [S3].

> ***Answering RQ1:*** *the drivers that cause companies to incur variability debt are related to: (i) business aspects, e.g., to decrease time to market and reduce development costs; (ii)technical aspects, e.g., poor design decisions of variability implementation, lack of traceability among artifacts, and over-engineering; (iii) operational aspects, e.g., to integrate geographically distributed teams, merge different companies, and the lack of knowledge of the system domain.*

Table 3: Artifact types where variability debt was identified

Category	References
Requirements	#6 #7 #8 [S9]; #10 [S31]; #12 [S10]; #13 [S25]; #16 [S4]; #17 [S33]; #18 [S28]; #23 [S29]; #25 [S26]; #26 [S8]; #28 [S16]; #29 [S5]; #30 #31 #32 #33 #34 #35 #36 #37 #38 #39 #40 #41 #42 [S32]; #43 [S17]; #45 [S27]; #47 [S13]
Architecture	#5 [S20]; #10 [S31]; #12 [S10]; #13 [S25]; #15 [S24]; #16 [S4]; #25 [S26]; #26 [S8]; #28 [S16]; #29 [S5]; #30 #31 #32 #33 #34 #35 #36 #37 #38 #39 #40 #41 #42 [S32]; #43 [S17]; #45 [S27]; #47 [S13]; #51 [S37]; #52 [S3]
Models	#2 [S6]; #3 [S36]; #4 [S1, S12]; #6 #7 #8 [S9]; #9 [S14]; #10 [S31]; #13 [S25]; #17 [S33]; #19 [S21]; #30 #31 #32 #33 #34 #35 #36 #37 #38 #39 #40 #41 #42 [S32]; #43 [S17]
Source code	#1 [S40]; #2 [S6]; #3 [S36]; #4 [S1]; #5 [S11, S20]; #6 #7 [S9]; #8 [S9, S34]; #9 [S14]; #10 [S31]; #12 [S10]; #13 [S25]; #15 [S24]; #16 [S4]; #17 [S33]; #18 [S28]; #19 [S22]; #24 [S38]; #25 [S26]; #26 [S8]; #27 [S35]; #28 [S2, S16]; #29 [S5]; #30 #31 #32 #33 #34 #35 #36 #37 #38 #39 [S32]; #40 [S23, S32]; #41 #42 [S32]; #43 [S17]; #44 [S18]; #45 [S27]; #46 [S7]; #47 [S13]; #48 [S15]; #49 #50 [S30]
Test cases	#6 #7 #8 [S9]; #25 [S26]; #30 #31 #32 #33 #34 #35 #36 #37 #38 #39 #40 #41 #42 [S32];
DB schemas	#24 [S38]
Build scripts	#24 [S38]

4.2 RQ2 - Artifacts impacted by variability debt

As we can observe in Table 3, variability debt impacts artifacts along the whole development life-cycle, from requirements to configuration files. Source code, requirements, architecture, and design models are the most commonly impacted. We expected a great number of case studies describing variability problems in source code, since the time pressure leads to companies to focus directly on the implementation artifacts. Problems related to variability of models are frequent because several case studies are in the domain of embedded systems and automobile, which use models as relevant artifacts. Interestingly, variability debt also affects database (DB) schemes and build scripts, which indirectly affects the deployment of variants.

> ***Answering RQ2:*** *several types of artifacts are impacted by variability debt. Source code is the most common, followed by requirements, architecture, and models. Some case studies also mentioned test cases, DB schemas, and build scripts. We can see that variability debt transverses the artifacts created along the software development life-cycle.*

4.3 RQ3 - Types and consequences of variabilty debt

Table 4 presents the problems faced by companies due to variability debt. The most common consequence of variability debt is the complexity to maintain independent variants created with ad-hoc

Table 4: Types and consequences of variability debt

Category	References
Complex maintenance of multiple independent variants	#3 [S36]; #4 [S1, S12]; #10 [S31]; #12 [S10]; #13 [S25]; #15 [S24]; #16 [S4]; #17 [S33]; #18 [S28]; #19 [S21]; #24 [S38]; #25 [S26]; #26 [S8]; #29 [S5]; #30 #31 #32 #33 #34 #35 #36 #37 #38 #39 [S32]; #40 [S23, S32]; #41 #42 [S32]; #45 [S27]; #47 [S13]; #49 #50 [S30]; #51 [S37]
Inability to systematically deal with customization	#1 [S40]; #3 [S36]; #4 [S1, S12]; #12 [S10]; #13 [S25]; #19 [S21]; #23 [S29]; #24 [S38]; #27 [S35]; #28 [S2, S16]; #30 #31 #32 #33 #34 #35 #36 #37 #38 #39 #40 #41 #42 [S32]; #43 [S17]; #47 [S13]; #48 [S15]; #49 #50 [S30]
Inability to create complex systems	#1 [S40]; #5 [S11, S20]; #10 [S31]; #13 [S25]; #15 [S24]; #30 #31 #32 #33 #34 #35 #36 #37 #38 #39 #40 #41 #42 [S32]; #43 [S17]; #52 [S3]
Poor overall internal quality	#1 [S40]; #3 [S36]; #13 [S25]; #15 [S24]; #28 [S16]; #30 #31 #32 #33 #34 #35 #36 #37 #38 #39 #40 #41 #42 [S32]; #44 [S18]; #52 [S3]
Complex test cycles	#13 [S25]; #30 #31 #32 #33 #34 #35 #36 #37 #38 #39 #40 #41 #42 [S32]; #45 [S27]
Creating usability problems	#2 [S6]; #17 [S33]; #44 [S18]
Portfolio management problems	#13 [S25]; #16 [S4]; #17 [S33]
Repeated roles in similar projects	#45 [S27]

reuse [S33]. For instance, a single bug must be fixed multiple times being an error-prone and time-consuming task [S36]. Furthermore, the proliferation of redundant and almost-alike software artifacts directly affects the maintainability of a family of products [S33].

Another type of variability debt is the inability to create new customized products as results of inadequate variability management. For example, it is hard to identify which artifact should be used in a new product when there are many similar artifacts, but with small customizations [S38]. The third most common type of variability debt refers to the inability to create complex systems, as for example, creating a new product based on already existing systems from different companies that were merged [S25]. As a consequence of the lack of systematic mechanisms to deal with customization, the overall internal quality of the systems decreases. For example, variants have dead code that increases product size and complexity, a.k.a. software bloat [S25]. Also, the lack of variability directly impacts flexibility to meet changing situations and diversified operations with minimum disruption or delay [S24].

Test cycles are also impacted due to the lack of variability management. For example, the derivation of test cases for product families is complex due to variability in the requirements, with the need of many behaviors to be tested [S32]. Ad-hoc strategies for dealing with variability can also frustrate testers, as repeated bugs with similar causes remain present in different variants [S27] and commonly the cycles for quality assurance are long [S25].

Avoiding variability with single systems with all features is also a problematic workaround. This creates usability problems [S6, S18, S33]. Dangerously, an engineering problem is converted to a business problem, since it will directly impact customer satisfaction. From the perspective of portfolio management, among the problems of the lack of variability we can mention the use of multiple software repositories for each variant, losing track of overall modifications [S4], more complex deployment of products for distinct customers [S25], and legacy variants become outliers [S33]. Finally, dealing with multiple independent variants also generates problems related to the use of human resources. For example, repeated role assignments, e.g., two practitioners as architects, in different variants, however, developing very similar tasks [S27].

> ***Answering RQ3:*** *the consequences caused by variability debt span from complex maintenance and test cycles, difficult to customize and create complex systems, limited portfolio management, and poor internal quality to misuse of human resources and usability problems.*

5 DISCUSSION

The overall challenge of variability is to use all the potential of technology platforms, while still offering tailored products [S8]. Many companies, such as Alcatel [S40], Fujitsu [S31], Siemens [S6], ABB [S32], Boeing [S32], and HP [S32], have observed this challenge and faced problems related to variability debt. However, the debt makes the cost of product development and maintenance grow too fast, or it just becomes infeasible to derive new products.

We have shown that variability debt is easily incurred in industrial practice, creating an obstacle for companies current or future competitiveness. However, how the variability debt is incurred is not always the same. The well-known technical debt quadrant [15] can help us to discuss these aspects. Table 5 introduces the "*variability debt quadrant*". Similar to technical debt, variability debt can be incurred intentionally or unintentionally, including the subdivision of reckless and prudent debt. Reckless and inadvertent debt, certainly the most undesired case, is incurred when companies do not have basic awareness on systematic reuse, knowledge about advanced variability implementation techniques, or no control on how the variants evolve. The state of adoption of SPL concepts in industry has been identified [7], the alignment between research

Table 5: Variability debt quadrant (adapted from [15])

	Reckless	Prudent
Deliberate	"We do not have time for systematic reuse and core assets' development, let's just copy-paste"	"We must ship now and deal with the consequences"
Inadvertent	"What is a Software Product Line and which are its state-of-the-art techniques?"	"Now we know how we should have done it"

and industry in the SPL domain has been analysed [27], the status and challenges in SPL teaching in educational centres and for companies has been identified [1], and public repositories of teaching material are available covering a wide range of SPL topics and techniques [13]. Despite this, simple solutions for variability management, which are easier to adopt, are used instead of more sophisticated ones creating deliberate and reckless debt. Unfortunately, there are no technical debt identification methods that help to identify the different variability debt types that we presented in this work. This is a research opportunity.

According to our observations in the analyzed case studies, deliberate and prudent debt is quite common. Before the adoption of more systematic reuse, companies usually know that their variability management is problematic and would soon become cost-prohibitive with the growth of the family of variants. Methods for the assessment and measurement of variability debt are required, perhaps taking inspiration from existing SPL adoption assessment approaches such as qualitative ones [28] and cost models [2].

In general, technical debt management activities, such as identification, measurement, prioritization, prevention, monitoring, repayment, representation/documentation and communication [22]; should be revisited to handle the specific features, causes, types, and consequences of variability debt.

5.1 Catalog of Variability Debt

In order to propose a preliminary catalog of variability debts, we used a systematic mapping study that presents a comprehensive list of technical debts [22]. More specifically, the work of Zengyang et al. [22] collected numerous types and instances of technical debt at different levels reported in their systematic mapping. These types of technical debt can be classified into 10 coarse-grained types, and each of them is further classified into several sub-types based on their causes. We use 5 out of 10 for the variability debt types, and 9 out of the 45 sub-types.

According to Zengyang et al. [22], some quality attributes (QAs) are compromised when technical debt is incurred. Most studies argue that technical debt negatively affects the maintainability of software systems. Furthermore, as ISO / IEC 25010 (standard used for Quality attributes) does not distinguish between ease of implementation of new requirements and bug fixes as different Quality attributes, these authors classified as modifiability the cases in which the ability to implement new requirements is negatively influenced. The technical debts that were related to these categories of Quality attributes were the ones that had the greatest relevance for us to classify in our work. Therefore, our analysis considered that variability debt can be related to technical debts related to the following ones described by Zengyang et al. [22]: Maintainability, Modularity, Reuse, Analyzeability, Modibility, Testability, Adaptability, and Replaceability.

Next, we evaluated the 52 case studies selected for our work to understand situations where variability debt appeared and were not covered in the existing catalog of Zengyang et al. [22]. We analyze situations that are generated as a consequence of an implementation of variability that generated mid- and long-term problems for the system, and thus include two more classifications: (i) Documentation duplication and (ii) No feature interactions tests. Our catalog of variability debts is presented in Table 6. This catalog was conceived by crossing information on technical debts that occur in the 52 case studies considered in our investigation. For that, we analyzed the systems that implement variability evaluating causes, consequences and the main artifacts impacted by the technical debt, generating a set of case studies that passed by this situation. Each variability debt presented in the catalog is discussed next.

- **Out-of-date or incomplete documentation**: By implementing variability non-systematically through copy and paste, the tendency is to duplicate documentation. As time goes by, the maintenance of multiple documents becomes more and more complex and ends up being a debt. The team tends to just implement the variability and not update the documentation.
- **Code duplication**: This is one of the most frequent technical debts identified when variability is implemented through non-systematic reuse. This is because companies choose either for lack of knowledge or for lack of time (causes) to implement the variability by copying and pasting code snippets or entire systems, thus generating (consequence) difficulty in maintenance (for example: when a bug occurs, it is necessary to correct in all variants of the system), repetition of rules by the system, among others.
- **Architectural anti-patterns**: Implementing variability in a non-systematic way through copy-paste, whether of code snippets or the entire system, can lead to serious problems in the architecture of the company's systems. Ex: If there are complex integrations and each time it is copied.
- **System-level structure quality issues**: When implementing variability in a non-systematic way, the damages to the system can be numerous: problems with code duplication, architectural problems, problems with system and database integration, in addition to problems with documentation. The quality tends to fall because it is not designed with a solid structure and so it is evolved, on the contrary, it is often initially created as a small system and from this it creates copies and variations, generating a fragile structure without quality.
- **Lack of testing**: The technical debt of lack of testing when it comes to variability is closely related to lack of time and variations of scenarios/variants. That is, the company wants to release as soon as possible a new product with a variation and lacks of time, so a complete test step is skipped. Or what can also happen is that, as there is certainty that the main product works, there is no need to test the variation, or it is underestimated and this may be a step to eliminate to save time (eliminating the test). The consequence of this: lack of quality in the product.
- **Expensive tests**: On the other hand, the absence of tests can occur by the company due to lack of knowledge in variability management, adequate testing techniques, elaborating complex and even repetitive test scenarios to incorporate all system variants in all implemented reuses, making this a time- and resource-consuming activity.
- **Multi-version support**: We can consider this the technical debt that best represents the meaning of this term, as it brings

Table 6: Catalog of Variability Debts

Name	Cause/Driver	Consequence	Artifacts	References
Out-of-date or incomplete documentation(*)	Time pressure, Lack of knowledge on variability management	Complex maintenance of multiple independent variants, Repeated roles in similar projects	Requirements, Models, DB schema	[S5, S10, S13, S25, S26, S31, S32, S33, S38, S39]
Architectural anti-patterns(*)	Time pressure, Lack of knowledge on variability management, Operational constraints	Complex maintenance of multiple independent variants, Inability to systematically deal with customization, Inability to create complex systems	Architecture, Models, DB schemas, Build scripts	[S2, S3, S5, S10, S11, S13, S16, S17, S19, S20, S24, S25, S26, S31, S32, S33, S37, S38, S39]
System-level structure quality issues(*)	Time pressure, Lack of knowledge on variability management, Operational constraints	Complex maintenance of multiple independent variants, Inability to create complex systems, Portfolio management problems	Architecture, Models, Source code, DB schemas, Build scripts	[S2, S3, S5, S10, S13, S16, S17, S24, S25, S26, S30, S31, S32, S33, S37, S38, S39]
Code Duplication(*)	Time pressure, Lack of knowledge on variability management	Complex maintenance of multiple independent variants, Repeated roles in similar projects	Source code	[S5, S9, S10, S13, S24, S25, S26, S30, S31, S32, S33, S38, S39];
Lack of tests(*)	Time pressure, Constantly changing scenarios	Poor overall internal quality	Requirements, Test cases	[S2, S16, S25, S32]
Expensive tests(*)	Lack of knowledge on variability management, Operational constraints	Complex test cycles	Requirements, Test cases	[S2, S16, S25, S32]
Multi-version support(*)	Time pressure, Constantly changing scenarios, Lack of knowledge on variability management	Complex maintenance of multiple independent variants, Portfolio management problems, Complex test cycles	Source code	[S4, S5, S10, S13, S21, S22, S24, S25, S26, S30, S31, S32, S36, S38, S39]
Old technology in use(*)	Time pressure, Operational constraints	Inability to create complex systems, Portfolio management problems	Architecture, Models, Source code	[S3, S11, S17, S19, S20, S24, S25, S31, S32, S40]
Duplicate documentation	Time pressure	Complex maintenance of multiple independent variants, Repeated roles in similar projects	Requirements, Models, DB schemas	[S5, S9, S10, S13, S25, S26, S31, S32]
Poor test of feature interactions	Time pressure, Lack of knowledge on variability management, Operational constraints	Complex maintenance of multiple independent variants, Inability to create complex systems, Poor overall internal quality	Architecture, Models, Source code, Test cases	[S2, S3, S5, S10, S13, S15, S16, S17, S18, S25, S26, S30, S31, S32, S33, S37, S38, S39, S40]

(*) Variability debts related to technical debts described in Zengyang et al. [22]

short-term benefits in using variability in a non-systematic way (e.g., time-to-market), it does not require the team to know techniques to implement variability and better manage it, but as copy-paste is implemented, many system versions are generated that can even be updated at different times. Variant A can be upgraded, variant B cannot, on the other hand it is possible to have an upgrade from variant C and not from A and B, etc. When source code and documentation must be synchronized, support becomes even more complex.

- **Old technology in use**: As it is possible to maintain multiple versions of a system with their variants or multiple systems created to meet different requirements (variations), some companies take these systems for years with the technology and architecture initially created, they end up becoming important legacy systems in companies and when there is a need to update technology, it becomes a difficult mission.

In addition to the debts in the paper of Zengyang et al. [22], discussed in light of the variability, we include new debts observed in our study:

- **Duplicate documentation**: This debt of variability is due to the non-systematic implementation of system variants. On many occasions, due to lack of knowledge to implement the variability in another way or the short period of time to release the new variant, teams end up choosing to copy and paste the system (source plus documentation). Thus, they imply the technical debt of keeping multiple documents, making future maintenance difficult to keep all information up to date;
- **Poor test of feature interactions**: When creating new variants with different feature configurations, the interaction of these different joint features can infer a variant's behavior. Therefore, possible interactions should be tested.

This catalog is presented with two main objectives: To disseminate the concept and map types of variability debt already identified in the literature, to disseminate knowledge and arouse the interest of researchers in the area. How can we help to solve these problems? Is it possible to avoid or mitigate them, identify them earlier, measure them? In addition, our goal is to help professionals who need to implement software variability and have not yet decided how to do it, whether systematically or not. The catalog can support them in the problems/consequences they will experiment when choosing to implement a non-systematic reuse. For example, imagine that a company already has a sales system but needs to open a branch in another country with new legislation and some other sales rules/-formats. The company's board establishes a short time for the IT Department to build and deploy the system, and the first idea of the team might be to implement variability through non-systematic reuse (copying the current sales system and making the necessary adjustments). Through our catalog, they can see the impact that they might have if they encounter some items. However, our catalog is still limited to being a form of consultation for foundation, as a future work we intend to add it to a form of solution.

6 RELATED WORK

Despite the broad discussion around both technical debt and variability individually, there are only few pieces of work that consider them together. Yet, some studies only superficially mention technical debt and/or variability, without an in-depth investigation.

Wille et al. [33] present an industrial case study in the automotive domain for mining the variability of technical architectures. These authors mentioned potential technical debt when there are varying parts of architectures in an arbitrary number of variants in parallel. This technical debt can hamper identifying reuse potential, and the management of this debt can help to make well-founded maintenance decisions. However, these authors not explore technical debt deeply. Yli-Huumo et al. [34] investigated technical debt in a Finnish company with product lines. Similarly, Ludwig et al. [23] performed a study on technical debt in a company dealing with the US Air Force satellite communication SPL. However, in both studies, the insights on technical debt were exclusively on a per-product basis. They did not report any cause of debt related to the variability management in the target company. In this work, we bring the discussion to create awareness on variability debt so that future studies could explicitly ask for this aspect when investigating technical debt in companies dealing with SPLs.

Another case study was conducted by Njima and Demeyer [26]. They focused on start-ups to investigate if adopting a systematic reuse with SPLs helps to avoid and manage technical debt and to reduce the time-to-market. Differently from our study, in their case study the authors were focused on the perception towards SPLE practice than in reported variability management implementations. Our work is complementary to this one of Njima and Demeyer.

Digkas et al. [11] performed an investigation about the effect of software reuse on technical debt. However, their focus was on reusing code chunks copy-pasted from the StackOverflow website. Despite dealing with reuse and technical debt, these authors do not consider families of systems, i.e., generating variants based on opportunistic reuse. Also, in the mapping study of Li et al. [22], existing studies on technical debt were mapped to ISO/IEC 25010 quality attributes. These included some attributes relevant for variability such as modularity, reusability, adaptability, modifiability, or replaceability. However, the authors did not focus completely on variability management. Our study brings a new perspective in comparison to those studies, since we focus on variability debt.

In the work of Boss et al. [8], the starting point is an already developed SPL at Bosch Group, in which the goal was to reduce internal technical debt to improve the quality of the architecture. For this, the authors proposed an approach for architectural checking based on metrics. Differently from our goal, Boss et al. focused exclusively on a case study, without providing broad insights or discussion related to causes and consequences of variability debt. The authors also highlight that dealing with technical debt to a large-scale product line is challenging.

Variability-aware smells [30] is a research direction that, even not explicitly mentioned, can have a relation with variability debt. These smells are relevant when the SPL is already in place and there are defects on SPL assets, e.g., feature model and variability-related annotations in the source code. Despite the similarity, we report other variability debt categories apart from sub-optimal variability-related decisions in the implementation.

Capilla et al. [10] investigated opportunistic reuse trend affects the technical debt ratios in several open-source projects and analyzed such TD ratios before and after reusing third-party components, and what are the implications for the software architecture. More specifically, the authors used two tools, namely SonarQube and Arcan, to analyze the code and architectural debt ratios in three different applications to investigate the effects of reusing functionality in open source repositories. Differently from our goal of investigating debts in variability management, they were concerned about difficulties and impacts when reusing third-party libraries.

Fenske and Schulze [14] discusses that variability increases the complexity of the source code and therefore impedes understanding and maintenance. Also, variability hinders the use of some techniques, such as source code analysis, because current approaches do not address variability. Based on these limitations, the authors revisited code smells in the light of variability. To this end, they introduce the notion of variability into existing code smells, resulting in an initial catalog of variability-aware code smells. They proposed four code smells that take variability into account. These variability-aware code smells were evaluated in a survey with 15 researchers that are experts on software product lines. The results revealed that most of the practitioners observed the smells in real-world systems and acknowledged that such smells could hinder maintenance and evolution. Our study complements this related study, as we explore the existing literature in order to provide further characterizations of variability debt.

Technical debt is an economic metaphor that can have synergies with SPL economic models. The economic aspects of systematic management of variability have been analyzed and modelled with diverse approaches [2]. However, a decay of the interest on the topic has been observed [27]. The metric Relative Cost of Writing for Reuse (RCWR) [2], which is used in various cost models to represent the ratio of effort to develop reusable software compared to the cost of developing it to use it just once, is related to the debt principal.

The advances of the technical debt community on quantitative measurements can make it worth revisiting those models from a variability debt perspective, but this is out of the scope of this paper.

7 THREATS TO VALIDITY

In this section, we present the threats to the validity of our study, and how we mitigated/reduced them.

Internal Validity: The main threat to internal validity concerns the sources to identify the case studies. For the query in the digital libraries, we used core terms in variability management and technical debt research. Then, the inclusion of the ESPLA and Krueger et al. [20] was an attempt to guarantee certain completeness in the search for case studies.

External Validity: Despite our efforts in including a diversity of sources, the findings of this preliminary literature review might not generalize as the results might not be applicable for different scenarios other than the ones found in the primary sources. We also consider that validation in industrial scenarios and with the participation of professionals might be necessary.

8 CONCLUSIONS

A systematic review of the literature on variability-related problems was carried out to help characterize the technical output related to issues in managing the variability of a family of systems. Although these issues have been recognized for many years by companies trying to deal with the systematic management of variability, that is, SPLs, this is the first time that the issue is analyzed from the perspective of the technical debt domain. We feel that this initial discussion of variability debt can help cross-fertilize the fields of variability management and technical debt. The term variability debt was created and a catalog with debts of variability, their causes, consequences and impacts is presented to help researchers and professionals in this matter. Future research includes bringing advances in technical debt identification and measurement to variability topics with concrete approaches, for example, guidelines and tools to support the creation of business cases to reason about variability debt. Also, future research includes the validation and management of variability debt types with new industrial case studies.

ACKNOWLEDGMENTS

This research was funded by Carlos Chagas Filho Foundation for Supporting Research in the State of Rio de Janeiro (FAPERJ), PDR-10 program, grant no. 202073/2020.

REFERENCES

[1] Mathieu Acher, Roberto E. Lopez-Herrejon, and Rick Rabiser. 2017. Teaching Software Product Lines: A Snapshot of Current Practices and Challenges. *ACM Transactions on Computing Education* 18, 1, Article 2 (Oct. 2017), 31 pages.

[2] Muhammad Sarmad Ali, Muhammad Ali Babar, and Klaus Schmid. 2009. A Comparative Survey of Economic Models for Software Product Lines. In *35th Euromicro Conf. on Software Engineering and Advanced Applications*. 275–278.

[3] Wesley K. G. Assunção, Roberto E. Lopez-Herrejon, Lukas Linsbauer, Silvia R. Vergilio, and Alexander Egyed. 2017. Reengineering legacy applications into software product lines: a systematic mapping. *Empirical Software Engineering* 22, 6 (2017), 2972–3016.

[4] Paris Avgeriou, Philippe Kruchten, Ipek Ozkaya, and Carolyn B. Seaman. 2016. Managing Technical Debt in Software Engineering (Dagstuhl Seminar 16162). *Dagstuhl Reports* 6, 4 (2016), 110–138.

[5] M. Ali Babar, L. Chen, and F. Shull. 2010. Managing Variability in Software Product Lines. *IEEE Software* 27 (2010), 89–91.

[6] Thorsten Berger, Ralf Rublack, Divya Nair, Joanne M. Atlee, Martin Becker, Krzysztof Czarnecki, and Andrzej Wasowski. 2013. A survey of variability modeling in industrial practice. In *7th Int. Workshop on Variability Modelling of Software-intensive Systems*. ACM, 7:1–7:8.

[7] Thorsten Berger, Jan-Philipp Steghöfer, Tewfik Ziadi, Jacques Robin, and Jabier Martinez. 2020. The state of adoption and the challenges of systematic variability management in industry. *Empirical Software Engineering* 25, 3 (2020), 1755–1797.

[8] Birgit Boss, Christian Tischer, Sreejith Krishnan, Arun Nutakki, and Vinod Gopinath. 2016. Setting up Architectural SW Health Builds in a New Product Line Generation. In *Proccedings of the 10th European Conference on Software Architecture Workshops* (Copenhagen, Denmark) *(ECSAW '16)*. Association for Computing Machinery, New York, NY, USA, Article 16, 7 pages.

[9] R. Capilla, J. Bosch, and K.C. Kang. 2013. *Systems and Software Variability Management: Concepts, Tools and Experiences*. Springer.

[10] Rafael Capilla, Tommi Mikkonen, Carlos Carrillo, Francesca Arcelli Fontana, Ilaria Pigazzini, and Valentina Lenarduzzi. 2021. Impact of Opportunistic Reuse Practices to Technical Debt. In *2021 IEEE/ACM International Conference on Technical Debt (TechDebt)*. 16–25.

[11] Georgios Digkas, Nikolaos Nikolaidis, Apostolos Ampatzoglou, and Alexander Chatzigeorgiou. 2019. Reusing Code from StackOverflow: The Effect on Technical Debt. In *2019 45th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. 87–91. https://doi.org/10.1109/SEAA.2019.00022

[12] Yael Dubinsky, Julia Rubin, Thorsten Berger, Slawomir Duszynski, Martin Becker, and Krzysztof Czarnecki. 2013. An Exploratory Study of Cloning in Industrial Software Product Lines. In *17th European Conference on Software Maintenance and Reengineering*. IEEE Computer Society, 25–34.

[13] FAMILIAR-project. 2021. Repository of teaching material for product lines and variability. http://teaching.variability.io/

[14] Wolfram Fenske and Sandro Schulze. 2015. Code Smells Revisited: A Variability Perspective. In *Proceedings of the Ninth International Workshop on Variability Modelling of Software-Intensive Systems* (Hildesheim, Germany) *(VaMoS '15)*. Association for Computing Machinery, New York, NY, USA, 3–10.

[15] Martin Fowler. 2014. Technical Debt Quadrant. https://martinfowler.com/bliki/TechnicalDebtQuadrant.html

[16] Yuepu Guo and Carolyn B. Seaman. 2011. A portfolio approach to technical debt management. In *2nd Workshop on Managing Technical Debt*. ACM, 31–34.

[17] Clemente Izurieta, Ipek Ozkaya, Carolyn B. Seaman, and Will Snipes. 2017. Technical Debt: A Research Roadmap Report on the Eighth Workshop on Managing Technical Debt. *ACM SIGSOFT Software Engineering Notes* 42, 1 (2017), 28–31.

[18] Philippe Kruchten, Robert Nord, and Ipek Ozkaya. 2019. *Managing Technical Debt: Reducing Friction in Software Development* (1st ed.). Addison-Wesley Professional.

[19] Charles W. Krueger. 2001. Easing the Transition to Software Mass Customization. In *Software Product-Family Engineering, 4th Int. Workshop, PFE 2001, Bilbao, Spain, October 3-5, 2001, Revised Papers (LNCS)*, Vol. 2290. Springer, 282–293.

[20] Jacob Krüger and Thorsten Berger. 2020. An Empirical Analysis of the Costs of Clone- and Platform-Oriented Software Reuse. In *28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (Virtual Event, USA) *(ESEC/FSE 2020)*. ACM, 432–444.

[21] Jacob Krüger, Wardah Mahmood, and Thorsten Berger. 2020. Promote-Pl: A Round-Trip Engineering Process Model for Adopting and Evolving Product Lines. In *Proceedings of the 24th ACM Conference on Systems and Software Product Line: Volume A - Volume A* (Montreal, Quebec, Canada) *(SPLC '20)*. Association for Computing Machinery, New York, NY, USA, Article 2, 12 pages.

[22] Zengyang Li, Paris Avgeriou, and Peng Liang. 2015. A systematic mapping study on technical debt and its management. *J. Syst. and Software* 101 (2015), 193–220.

[23] Jeremy Ludwig, Devin Cline, and Aaron Novstrup. 2020. A Case Study Using CBR-Insight to Visualize Source Code Quality. In *2020 IEEE Aerospace Conference*. 1–12. https://doi.org/10.1109/AERO47225.2020.9172526

[24] Rezvan Mahdavi-Hezaveh, Jacob Dremann, and Laurie Williams. 2021. Software development with feature toggles: practices used by practitioners. *Empirical Software Engineering* 26, 1 (2021).

[25] Jabier Martinez, Wesley K. G. Assunção, and Tewfik Ziadi. 2017. ESPLA: A Catalog of Extractive SPL Adoption Case Studies. In *21st Int. Systems and Software Product Line Conference*. ACM, 38–41.

[26] Mercy Njima and Serge Demeyer. 2019. Value-Based Technical Debt Management: An Exploratory Case Study in Start-Ups and Scale-Ups. In *Proceedings of the 2nd ACM SIGSOFT International Workshop on Software-Intensive Business: Start-Ups, Platforms, and Ecosystems* (Tallinn, Estonia) *(IWSiB 2019)*. Association for Computing Machinery, New York, NY, USA, 54–59.

[27] Rick Rabiser, Klaus Schmid, Martin Becker, Goetz Botterweck, Matthias Galster, Iris Groher, and Danny Weyns. 2019. Industrial and academic software product line research at SPLC: perceptions of the community. In *23rd Int. Systems and Software Product Line Conference*. ACM, 27:1–27:6.

[28] Luisa Rincón, Raúl Mazo, and Camille Salinesi. 2018. APPLIES: A framework for evaluAting organization's motivation and preparation for adopting product lines. In *12th Int. Conference on Research Challenges in Information Science*. 1–12.

[29] Klaus Schmid and Isabel John. 2004. A customizable approach to full lifecycle variability management. *Science of Computer Programming* 53, 3 (2004), 259–284.

[30] Iuri Santos Souza, Ivan do Carmo Machado, Carolyn Seaman, Gecynalda Soares da Silva Gomes, Christina Chavez, Eduardo Santana de Almeida, and Paulo César Masiero. 2019. Investigating Variability-aware Smells in SPLs: An Exploratory Study. In *33rd Brazilian Symposium on Software Engineering*. ACM, 367–376.
[31] SPLC. 2020. Product Line Hall of Fame. https://splc.net/fame.html
[32] Frank van der Linden, Klaus Schmid, and Eelco Rommes. 2007. *Software product lines in action - the best industrial practice in product line engineering*. Springer.
[33] David Wille, Kenny Wehling, Christoph Seidl, Martin Pluchator, and Ina Schaefer. 2017. Variability Mining of Technical Architectures *(SPLC '17)*. ACM, New York, NY, USA, 39–48.
[34] Jesse Yli-Huumo, Andrey Maglyas, and Kari Smolander. 2014. The Sources and Approaches to Management of Technical Debt: A Case Study of Two Product Lines in a Middle-Size Finnish Software Company. In *Product-Focused Software Process Improvement*. Springer International Publishing, Cham, 93–107.
[35] Jesse Yli-Huumo, Andrey Maglyas, and Kari Smolander. 2016. How do software development teams manage technical debt? - An empirical study. *J. Syst. and Software* 120 (2016), 195–218.

A PRIMARY SOURCES

[S1] Manuel Ballarin, Raúl Lapeña, and Carlos Cetina. 2016. Leveraging Feature Location to Extract the Clone-and-Own Relationships of a Family of Software Products. In *ICSR 2016*. Springer, 215–230.
[S2] BigLever. 2005. *BigLever Software Case Study: Engenio*. Technical Report 2005-06-14-1. BigLever Software, Inc.
[S3] Birgit Boss, Christian Tischer, Sreejith Krishnan, Arun Nutakki, and Vinod Gopinath. 2016. Setting up Architectural SW Health Builds in a New Product Line Generation. In *Proccedings of the 10th European Conference on Software Architecture Workshops* (Copenhagen, Denmark) *(ECSAW '16)*. Association for Computing Machinery, New York, NY, USA, Article 16, 7 pages.
[S4] Hongyu Pei Breivold, Stig Larsson, and Rikard Land. 2008. Migrating Industrial Systems towards Software Product Lines: Experiences and Observations through Case Studies. In *34th Euromicro Conference Software Engineering and Advanced Applications*. IEEE.
[S5] Paul Clements and John Bergey. 2005. *The U.S. Army's Common Avionics Architecture System (CAAS) Product Line: A Case Study*. Technical Report CMU/SEI-2005-TR-019. Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA. http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=7707
[S6] Deepak Dhungana, Paul Grünbacher, and Rick Rabiser. 2010. The DOPLER meta-tool for decision-oriented variability modeling: a multiple case study. *Automated Software Engineering* 18, 1 (nov 2010), 77–114.
[S7] Michael Dillon, Jorge Rivera, and Rowland Darbin. 2014. A Methodical Approach to Product Line Adoption. In *18th Int. Software Product Line Conference* (Florence, Italy) *(SPLC '14)*. ACM, New York, NY, USA, 340–349.
[S8] Christof Ebert and Michel Smouts. 2003. Tricks and traps of initiating a product line concept in existing products. In *25th Int. Conference on Software Engineering, 2003. Proceedings*. IEEE.
[S9] Magnus Eriksson, Henrik Morast, Jürgen Börstler, and Kjell Borg. 2005. The PLUSS toolkit?. In *20th Int. Conference on Automated software engineering*. ACM.
[S10] D. Faust and Chris Verhoef. 2003. Software product line migration and deployment. *Software: Practice and Experience* 33, 10 (2003), 933–955.
[S11] Thomas Fogdal, Helene Scherrebeck, Juha Kuusela, Martin Becker, and Bo Zhang. 2016. Ten Years of Product Line Engineering at Danfoss: Lessons Learned and Way Ahead *(SPLC '16)*. ACM, New York, NY, USA, 252–261.
[S12] Jaime Font, Lorena Arcega, Øystein Haugen, and Carlos Cetina. 2016. Feature Location in Model-Based Software Product Lines Through a Genetic Algorithm. In *ICSR 2016*. Springer, 39–54.
[S13] Susan P. Gregg, Rick Scharadin, Eric LeGore, and Paul Clements. 2014. Lessons from AEGIS: Organizational and Governance Aspects of a Major Product Line in a Multi-Program Environment *(SPLC '14)*. ACM, New York, NY, USA, 264–273.
[S14] Paul Grünbacher, Rick Rabiser, Deepak Dhungana, and Martin Lehofer. 2009. Model-Based Customization and Deployment of Eclipse-Based Tools: Industrial Experiences. In *IEEE/ACM Int. Conf. on Automated Software Engineering*. IEEE.
[S15] MyungJoo Ham and Geunsik Lim. 2019. Making Configurable and Unified Platform, Ready for Broader Future Devices. In *2019 IEEE/ACM 41st Int. Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. 141–150.
[S16] William A. Hetrick, Charles W. Krueger, and Joseph G. Moore. 2006. Incremental return on incremental investment. In *Companion, 21st ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications*. ACM.
[S17] Paul Jensen. 2007. Experiences with Product Line Development of Multi-Discipline Analysis Software at Overwatch Textron Systems. In *11th Int. Software Product Line Conference (SPLC 2007)*. IEEE.
[S18] Paul Jensen. 2008. Experiences with Software Product Line Development. *CrossTalk: The Journal of Defense Software Engineering* 22, 1 (2008).
[S19] Hans Peter Jepsen, Jan Gaardsted Dall, and Danilo Beuche. 2007. Minimally Invasive Migration to Software Product Lines. In *11th Int. Software Product Line Conference (SPLC 2007)*. IEEE.
[S20] Hans Peter Jepsen, Jan Gaardsted Dall, and Danilo Beuche. 2007. Minimally Invasive Migration to Software Product Lines. In *11th Int. Software Product Line Conf.* IEEE, 203–211.
[S21] András Kicsi, László Vidács, Árpád Beszédes, Ferenc Kocsis, and Istvan Kovacs. 2017. Information retrieval based feature analysis for product line adoption in 4GL systems. In *2017 17th Int. Conference on Computational Science and Its Applications (ICCSA)*. IEEE.
[S22] András Kicsi, László Vidács, Viktor Csuvik, Ferenc Horváth, Árpád Beszédes, and Ferenc Kocsis. 2018. Supporting Product Line Adoption by Combining Syntactic and Textual Feature Extraction. In *New Opportunities for Software Reuse*. Springer, 148–163.
[S23] Ronny Kolb, Isabel John, Jens Knodel, Dirk Muthig, Uwe Haury, and Gerald Meier. [n.d.]. Experiences with Product Line Development of Embedded Systems at Testo AG. In *10th Int. Software Product Line Conference (SPLC'06)*. IEEE.
[S24] Ronny Kolb, Dirk Muthig, Thomas Patzke, and Kazuyuki Yamauchi. 2006. Refactoring a legacy component for reuse in a software product line: a case study. *J. Software Maintenance and Evolution: Research and Practice* 18, 2 (2006), 109–132.
[S25] Charles W. Krueger, Dale Churchett, and Ross Buhrdorf. 2008. HomeAway's Transition to Software Product Line Practice: Engineering and Business Results in 60 Days. In *2008 12th Int. Software Product Line Conference*. 297–306.
[S26] Kwanwoo Lee, Kyo C. Kang, Eunman Koh, Wonsuk Chae, Bokyoung Kim, and Byoung Wook Choi. 2000. Domain-Oriented Engineering of Elevator Control Software. In *Software Product Lines*. Springer US, 3–22.
[S27] Dong Li and Carl K. Chang. 2009. Initiating and Institutionalizing Software Product Line Engineering: From Bottom-Up Approach to Top-Down Practice. In *33rd Annual IEEE Int. Computer Software and Applications Conference*. IEEE, 53–60.
[S28] Lukas Linsbauer, Roberto Erick Lopez-Herrejon, and Alexander Egyed. 2014. Feature Model Synthesis with Genetic Programming. In *Search-Based Software Engineering*. Springer, 153–167.
[S29] Nan Niu, Juha Savolainen, Zhendong Niu, Mingzhou Jin, and Jing-Ru C. Cheng. 2014. A Systems Approach to Product Line Requirements Reuse. *IEEE Systems Journal* 8, 3 (sep 2014), 827–836.
[S30] Mercy Njima and Serge Demeyer. 2019. Value-Based Technical Debt Management: An Exploratory Case Study in Start-Ups and Scale-Ups. In *Proceedings of the 2nd ACM SIGSOFT International Workshop on Software-Intensive Business: Start-Ups, Platforms, and Ecosystems* (Tallinn, Estonia) *(IWSiB 2019)*. Association for Computing Machinery, New York, NY, USA, 54–59.
[S31] Jun Otsuka, Kouichi Kawarabata, Takashi Iwasaki, Makoto Uchiba, Tsuneo Nakanishi, and Kenji Hisazumi. 2011. Small inexpensive core asset construction for large gainful product line development. In *15th Int. Conference on Software Product Line*. ACM.
[S32] Klaus Pohl, Günter Böckle, and Frank J van Der Linden. 2005. *Software product line engineering: foundations, principles and techniques*. Springer.
[S33] Kamil Rosiak, Oliver Urbaniak, Alexander Schlie, Christoph Seidl, and Ina Schaefer. 2019. Analyzing variability in 25 years of industrial legacy software. In *23rd Int. Systems and Software Product Line Conference*. ACM.
[S34] Julia Rubin, Krzysztof Czarnecki, and Marsha Chechik. 2015. Cloned product variants: from ad-hoc to managed software product lines. *Int. Journal on Software Tools for Technology Transfer* 17, 5 (aug 2015), 627–646.
[S35] Mark Staples and Derrick Hill. 2004. Experiences Adopting Software Product Line Development without a Product Line Architecture. In *11th Asia-Pacific Software Engineering Conference*. IEEE.
[S36] David Wille, Sandro Schulze, and Ina Schaefer. 2016. Variability mining of state charts. In *Proceedings of the 7th Int. Workshop on Feature-Oriented Software Development*. ACM.
[S37] David Wille, Kenny Wehling, Christoph Seidl, Martin Pluchator, and Ina Schaefer. 2017. Variability Mining of Technical Architectures *(SPLC '17)*. ACM, New York, NY, USA, 39–48.
[S38] Pengfei Ye, Xin Peng, Yinxing Xue, and Stan Jarzabek. 2009. A Case Study of Variation Mechanism in an Industrial Product Line. In *11th International Conference on Software Reuse, ICSR*, Vol. 5791. 126–136.
[S39] Xue Yinxing. 2012. *Reengineering legacy software products into software product line*. Ph.D. Dissertation. https://scholarbank.nus.edu.sg/handle/10635/36403
[S40] G. Zhang, L. Shen, X. Peng, Z. Xing, and W. Zhao. 2011. Incremental and iterative reengineering towards Software Product Line: An industrial case study. In *27th IEEE Int. Conference on Software Maintenance (ICSM)*. IEEE, 418–427.