



In practice

Variability debt in opportunistic reuse: A multi-project field study[☆]

Daniele Wolfart^a, Jabier Martinez^b, Wesley K.G. Assunção^{c,d,*}, Thelma E. Colanzi^e, Alexander Egyed^f

^a PPGComp, Western Paraná State University (UNIOESTE), Cascavel, Brazil

^b Tecnalia, Basque Research and Technology Alliance (BRTA), Derio, Spain

^c CSC, North Carolina State University (NCSU), Raleigh, USA

^d OPUS, Pontifical Catholic University of Rio de Janeiro (PUC-Rio), Rio de Janeiro, Brazil

^e DIN, State University of Maringá (UEM), Maringá, Brazil

^f ISSE, Johannes Kepler University Linz (JKU), Linz, Austria

ARTICLE INFO

Keywords:

Technical debt

Variability management

Clone and own

Software reuse

ABSTRACT

Technical debt is a metaphor to guide the identification, measurement, and general management of decisions that are appropriate in the short term but create obstacles in the future evolution and maintenance of systems. Variability management, which is the ability to create system variants to satisfy different business or technical needs, is a potential source of technical debt. *Variability debt*, recently characterized in a systematic literature review we conducted, is caused by suboptimal solutions in the implementation of variability management in software systems. In this work, we present a field study in which we report quantitative and qualitative analysis of *variability debt* through artifact analysis (e.g., requirements, source code, and tests) and a survey with stakeholders (e.g., analysts, developers, managers, and a user). The context is a large company with three different systems, where opportunistic reuse (a.k.a., copy-and-paste or clone-and-own reuse) of almost all project artifacts was performed to create variants for each system. We analyze the variability debt phenomenon related to opportunistic reuse, and we assess the validity of the metaphor to create awareness to stakeholders and guide technical debt management research related to variability aspects. The results of the field study show evidences of factors that complicate the evolution of the variants, such as code duplication and non-synchronized artifacts. Time pressure is identified as the main cause for not considering other options than opportunistic reuse. Technical practitioners mostly agree on the creation of usability problems and complex maintenance of multiple independent variants. However, this is not fully perceived by managerial practitioners.

1. Introduction

The *technical debt* metaphor emerged as an explanation of the consequences of technical issues to nontechnical product stakeholders (Cunningham, 1992). “*Not quite right code which we postpone making it right*” (Cunningham, 1992; Kruchten et al., 2012) is how technical debt was originally described. Since then, the technical debt concept is already well-known in the software engineering culture, and it has been discussed and investigated from many perspectives (Rios et al., 2018; Junior and Travassos, 2022) such as how to identify, measure, prioritize, transform, or monitoring it across the system life cycle (Li et al., 2013). Technical debt management is now considered a discipline by itself, responsible to handle this important aspect in systems and software engineering (Li et al., 2015).

Recently, Avgeriou et al. (2016) framed the term technical debt as “*delayed tasks and immature artifacts that constitute a ‘debt’ because*

they incur extra costs in the future in the form of increased cost of change during evolution and maintenance”. This definition of technical debt describes well the issues observed when opportunistic reuse practices (i.e., non-systematic reuse of parts of an existing product to create new ones—a.k.a., copy-and-paste or clone-and-own) are used recurrently across projects without a proper management of commonality and variability (Fischer et al., 2014; Assunção et al., 2017). This is the scenario we focus on this work. Notably, 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, software companies commonly adopt opportunistic reuse practices (Echeverría et al., 2021). This practice of copying and adapting similar products or features from existing projects lead to achieve short term benefits such as

[☆] Editor: Marcos Kalinowski.

* Corresponding author at: CSC, North Carolina State University (NCSU), Raleigh, USA.

E-mail address: wguezas@ncsu.edu (W.K.G. Assunção).

creating new product variants quickly. However, it leads to technical drawbacks in a medium- and long-term, like duplicate code and bugs, challenging the propagation of improvements and fixes across all the variants (Assunção et al., 2017). As we presented above, this problem traces back to what is known as technical debt. The complexity in the evolution of the family of software product variants as a whole is dramatically increased (Li et al., 2015). This is a context frequently observed in industry, which demonstrated the practical relevance of understanding this phenomenon (Martinez et al., 2017).

Despite the wide discussion of both technical debt (Rios et al., 2018; Junior and Travassos, 2022) and engineering of families of software products with variability (Capilla et al., 2013; van der Linden et al., 2007; Pohl et al., 2005), little is known about how they are related. To fill this gap, in a previous work we conducted an investigation of variability management from a technical debt perspective (Wolfart et al., 2021). Based on a systematic literature review, we provided an initial definition of what we referred to as “Variability Debt”. *Variability Debt* refers to technical debt caused by inadequate and/or sub-optimal solutions in the implementation of variability management in software systems (Wolfart et al., 2021). In this initial investigation, we distilled the causes, consequences, and artifacts impacted by variability debt. However, a thorough study on how variability debt manifests in practice is still missing.

The goal of this paper is to describe how variability debt incurs in a real scenario and how practitioners in this scenario perceive this phenomenon. For that, we conducted a multi-project exploratory investigation following the field study method (Stol and Fitzgerald, 2018). A *field study* focus on data collection in a natural setting with minimal intrusion of the researcher to not disturb realism. The field study reported in this paper relies on three projects in which opportunistic reuse was used to create product variants. Thus, the focus of this paper is on variability debt exclusively in the context of opportunistic reuse as a variability management approach, causing issues in evolution and maintenance. The projects are developed and maintained in a Brazilian multinational industry¹ leader in its market segment. This industry follows a business strategy of producing all goods needed in its product chain, so software variants are created to different industry branches or departments. We collected and analyzed data from artifacts of different product life-cycle phases (e.g., requirements, source code, and test/validation cases) from a total of seven system variants, from three projects. Additionally, we asked 22 stakeholders (e.g., developers, analysts, testers, managers, and a user) about their perception of technical debt in opportunistic reuse as well as about their own observations on our findings.

The results of the artifact analysis showed evidences of commonality and variability in the artifacts analyzed, and inconsistencies among product variants, challenging their future synchronization. We also observed a lack of proper test cases for new variants. The survey with different stakeholder roles showed agreements and divergences with the causes and consequences identified in our previous study (Wolfart et al., 2021). For the variability debt definition and interest of future research and development roadmap, stakeholders agreed on the relevance of looking at variability management issues from the technical debt management perspective.

The main contributions of this paper are:

- *For practitioners, awareness about the importance of Systematic Variability Management*: despite the large number of studies on variability management and engineering of system families, the use of systematic approaches (e.g., Software Product Lines (Martinez et al., 2023)) is still not widely adopted in the industry. Our findings contribute to bring the attention of practitioners to problems generated by using opportunistic and non-systematic practices for software reuse and variability management.
- *For researchers, opportunities for new studies*: the survey included questions in which participants prioritized the variability debt management activities indicating, to some extent, the aspects with more potential interest or impact.
- *For tool builders, source of information for new tooling support*: similarly to the opportunities for new studies, tooling and visualization support for variability debt management is desired, mainly because of the complexity of dealing with families of product variants.

The remainder of this paper is structured as follows: Section 2 presents background and Section 3 describes related works. Then, Section 4 presents the study design, followed by the results and the discussion in Section 5. Section 6 discusses the threats to validity. Finally, Section 7 presents the conclusions and outlines future work.

2. Background on variability debt

Variability Debt (Wolfart et al., 2021) occurs when practitioners choose to implement variability with an approach that would bring short-term benefits (e.g., using opportunistic reuse) regardless of technical and architectural drawbacks in the medium- and long-term. This leads to maintenance and evolution difficulties to manage families of systems composed of multiple individual variants (Martinez et al., 2023).

In our previous study, we characterize the variability debt from the perspective of its causes, consequences, and artifacts (Wolfart et al., 2021). Among the types of impacted artifacts, source code is the most common, followed by requirements, architecture, and models. Test cases, database schemas, and build scripts were referenced in few occasions. Variability debt is observed in different types of artifacts created along the whole software development life-cycle, and it is not exclusive to the software implementation phase. This is aligned with the current understanding of technical debt management (Li et al., 2015).

Regarding the context in which variability debt occurs, we list below the main *causes*:

- Business aspects: to decrease time to market and reduce development costs;
- Technical aspects: poor design decisions of variability implementation, lack of traceability among artifacts, and over-engineering; and,
- Operational aspects: to integrate geographically distributed teams, merge different companies, and the lack of knowledge of the system domain.

Companies are faced with several problems due to variability debt. The *consequences* observed are: complex maintenance of multiple independent variants, inability to systematically deal with customization, inability to create complex systems, poor overall internal quality, complex test cycles, creating usability problems, portfolio management problems, and repeated roles in similar projects.

We conceived a *catalog of variability debt* by crossing information on technical debt reported in the 52 case studies considered in our systematic literature review (Wolfart et al., 2021). Below, we briefly present the entries of this catalog. These entries are inspired on technical debt subtypes as defined in Li et al. (2015), but from the perspective of variability debt concerns.

- *Out-of-date, incomplete, and duplicate documentation*: The implementation of variability using opportunistic reuse usually leads to duplicate documentation. The maintenance of multiple documents becomes complex and ends up being a technical debt. The team might tend to implement the variability and not update the documentation in the variants.

¹ The name of the industry, its market segment, and the name of the subject systems are omitted to keep confidentiality.

- **Code duplication:** Practitioners copy and paste code of features or entire products, leading to complex maintenance of multiple independent product variants. For example, when a bug occurs, it must be fixed in all product variants. Similarly, enhancements are difficult to synchronize among variants as it requires a change propagation effort. This is one of the most frequent type of technical debt when implementing variability with opportunistic reuse.
- **Multi-version support:** When variability is implemented in a non-systematic way, it brings short-term benefits, such as time-to-market, without requiring knowledge of variability management. However, opportunistic reuse lead to multiple product versions that can be upgraded at different times. For example, a variant A is upgraded while a variant B is not, on the other hand it is possible to have an upgrade for variant C and not for A and B. When source code and documentation must be synchronized, support becomes even more complex.
- **Architectural anti-patterns:** The implementation of variability adopting opportunistic reuse can lead to undesired anti-patterns, such as the replication of a complex integration each time a product is cloned, requiring architecture refactoring and evolution.
- **Lack of testing, expensive tests, and poor test of feature interactions:** For releasing a new product variant quickly, the testing phase might be skipped or partially conducted. With the assumption that the main product works, practitioners might decide that there is no need to test the new product variant, or the testing phase is underestimated, then the test is reduced to save time. The main consequence of this debt is the potential lack of quality in the variants. Elaborating complex and even repetitive test scenarios to cover all system variants might be a time- and resource-consuming activity, mainly when the development team have limited knowledge in variability management and testing techniques, or just have limited time to put the testing infrastructure in place. Possible feature interactions should be tested when creating new variants with different feature configurations because the interaction of features can change variants' behavior.
- **Old technology in use:** This variability debt occurs when the company has multiple variants of a legacy system, or multiple variants were created to meet different requirements with the original legacy technology and architecture. When there is a need to update the underlying technology, it becomes a difficult task.

In this present work, we relied on our catalog of variability debts to carry out the field study.

3. Related work

Li, Avgeriou, and Liang (Li et al., 2015) presented a systematic review to propose a technical debt catalog that includes 10 types of technical debt and eight activities for technical debt management. However, they do not discuss the technical debts caused by the non-systematic implementation of variability. In our previous work (Wolfart et al., 2021), we proposed a catalog of variability debts in the light of technical debt found in the literature (Li et al., 2015) and we added new debts found in our study. Our catalog, presented in Section 2, includes the technical debts that occurs when implementing variability by means of opportunistic reuse, its causes, consequence and the main artifacts involved. In this present work, we relied on our catalog of variability debts to carry out the field study.

The state of the art on technical debt has not culminated in rigorous analysis models yet. In this sense, Siebra et al. (2012) investigated decisions on technical debt and related events in an industrial large scale project with the goal of characterizing the technical debt existence as well as its parameters' evolution. The study was based on only one system (SMB - Samsung Mobile Business) and in three different scenarios: persistence layer, user interface layer, and during the inclusion of a new

programming language. However, characteristics related to the system variabilities were not investigated.

Some studies investigated strategies to manage technical debt and prioritization criteria to pay off the debts (Ribeiro et al., 2016; Arvanitou et al., 2019; Lenarduzzi et al., 2021). Ribeiro et al. (2016) presented 14 criteria to support the development team in deciding which debts should be prioritized. They also presented a list of debt types related to each criterion. In a study with 60 software engineers of 11 software companies from nine different countries, Arvanitou et al. (2019) found that stakeholders need different views of the quality dashboard on metrics related to technical debt because they usually have different interests. For instance, managers are more interested in financial concepts, whereas developers are more interested in the nature of the problems that exist in the code. This difference was also observed in our study (see Section 5). Lenarduzzi et al. (2021) performed a systematic literature review to investigate the body of knowledge on approaches proposed in both academy and industry to prioritize technical debts. The main finding is that there is no consensus on which are the most important factors and how they should be measured in the context of technical debt prioritization. Our work mainly differs from those because we investigate the technical debt related to variability, and we are not discussing when and how to pay off the technical debts in this context.

Fenske and Schulze (2015) introduced the notion code smells related to variability, proposing an initial catalog of four variability-aware code smells. Their findings point out that the proposed smells (i) have been observed in existing product lines and (ii) are considered to be problematic for common software development activities, such as program comprehension, maintenance, and evolution. Our work complements this notion because we are interested in improving the understanding of variability debt.

Digkas et al. (2019) investigated the effects of software reuse on technical debt. They carried out an empirical study on the relation between the existence of reusing code retrieved from StackOverflow and the technical debt of the target system. Their empirical results provide insights to the potential impact of small-scale code reuse on technical debt and highlight the benefits of assessing code quality before performing the reuse of pieces of code. Despite addressing software reuse and technical debt, their study did not take into account families of systems, thus, they did not study variants created using opportunistic reuse.

Verdecchia et al. (2021) focused on architectural technical debt in software-intensive systems. This is the debt incurred in the architectural level of software design (e.g., choices regarding structure, frameworks, technologies, and languages) that, while being suitable or even optimal when made, significantly hinder progress in the future. The goal of the study is understanding how software development organizations conceptualize architectural debt, and how they deal with it. Differently from their study, our work addresses systems that implement variability in a non-systematic way, observing any level of technical debt (code or architectural).

Recently, Capilla et al. (2021) carried out an exploratory study to investigate to what extent reusing components opportunistically negatively affects the quality of systems, focusing on the impact of opportunistic reuse on technical debt. They were concerned with the difficulties and impact of reusing third-party components, while we are concerned with the debt related to variability management.

Given the aforementioned context, at the time of the writing, we have not found any investigation about the relationship between implementing variability using opportunistic reuse and technical debt. Thus, the purpose of our study is to investigate this relationship and assess its importance in an industrial context.

During the review process of this manuscript, we organized a first workshop in the software and systems product line conference (SPLC) on the subject of technical debt for variability-intensive systems (TD4ViS) (Martinez et al., 2022). The discussions were fruitful, and the

subject was discussed from different perspectives.² However, because of the time limitations, we did not manage to solve the issue of characterizing the variability debt phenomenon.

4. Study design

Our work relies on a field study. As defined by [Stol and Fitzgerald \(2018\)](#), a field study has a “*natural setting that exists before the researcher enters it with a minimal intrusion of the setting so as not to disturb realism, only to facilitate data collection.*” The authors also point out that it “*facilitates study of phenomena and actors and their behavior in natural contexts.*” The exploratory nature of a field study helps “*to understand what is going on, how things work, or to generate hypotheses.*” Based on this definition of a field study, the goal of our work is:

Goal Understanding variability debt in the natural settings of a company that uses opportunistic reuse to create families of product variants.

To achieve this goal through our field study, we focus on answering two research questions regarding variability debt in opportunistic reuse:

RQ1 Which artifacts are affected by variability debt and how?

RQ2 How do practitioners from the company perceive variability debt?

Technical debt can occur at any stage or activity of a software development process ([Li et al., 2015](#)). The RQ1 of our field study serves as a confirmatory study regarding which and how artifacts from different stages of the development process are affected by opportunistic reuse. Our objective with RQ1 is to discuss the impact of opportunistic reuse from a technical debt perspective. For RQ2, we focused on studying three main aspects of the industry: (i) characterization of the practitioners; (ii) artifacts, causes, and consequences related to the technical debt because of opportunistic reuse; and (iii) definition and perspectives regarding the variability debt concept and its management.

To answer the research questions and achieve our goal, in Section 4.1 we describe in detail the industrial scenario, and in Section 4.2 we present the data collection and analysis process. Despite the advantages of a field study, it also has intrinsic threats to validity regarding generalization, which are further discussed in Section 6.

4.1. Characteristics of the industrial scenario

The field study reported in this paper was conducted in a Brazilian multinational industry leader in its market segment in Latin America. This industry produces all goods needed in its product chain, having an Information Technology (IT) department in charge of providing software solutions for all branches or departments within the company. Based on that, the team of the IT department have employed opportunistic reuse when a customized variant of a software system is needed. These practitioners have also been responsible for maintaining the variants of the systems in operation.

4.1.1. Subject systems and variants

[Table 1](#) presents a characterization of the three systems (names omitted, see previous footnote 1), in terms of their variants, the total number of features, and the number of common features among the variants.

SysX is a software system for the inventory management of finished goods, having three product variants. The Original variant was developed for stock management at headquarters. After a few years, a branch

Table 1

Subject systems and variants.

System	Variant	Year	#F	#CF
SysX	Original	2010	14	7
	Variant1	2018	10	
	Variant2	2020	11	
SysY	Original	2012	34	24
	Variant1	2016	24	
SysZ	Original	2018	26	24
	Variant1	2018	26	

Year = year in which the variant was created, #F = number of features, #CF = number of common features.

of the group, responsible for delivering and packaging, grew up substantially and had the need to control its own stock, then, Variant1 was created. Recently, the company created a branch that is a distribution center, reusing again the Original variant to create Variant2. During the opportunistic reuse, some features cease to exist in variant systems (e.g., *Check Remittance*). This happened because Variants 1 and 2 were customized for branches of the organizational group that do not carry out sales, but only store products, unlike the variant Original. This way, remittance was not needed. On the other hand, a new feature appeared in Variant 1, namely *Remove Blocked Fractional*, because, contrary to the headquarters that uses the variant Original and the branch that uses Variant2, Variant1 was deployed in a branch that deals with products that can be fractionated (e.g., delivering part of a batch to another branch of the industry).

SysY manages documents and controls their printing in the company. The difference between the variant Original and Variant1 is that some features were removed during the reuse. The variant Original was implemented for the headquarters, which uses manufacturing orders to generate the finished product. In this variant, there are specific features that associate documents with these manufacturing orders and with the materials. Differently, Variant1 was implemented to a branch that does not have focus on product manufacturing, but that needed to control documentation, thus, some unnecessary features were removed.

SysZ manages non-conformity, that is, actions or situations that fail to comply with a pre-established procedure. The system controls the registration, evaluation, investigation, creation of the action plan related to non-conformity. The differences between the Original and Variant1 is that two features were removed (*Supplier/Manufacturer Non-Conformity Registration* and *CAC Non-Conformity Registration*) and two features were added (*Classification Officer Report* and *Investigation Officer Report*). Thus, the IT team adapted the Original to create Variant1, given that more reports were needed to submit for inspection.

4.1.2. Stakeholders and current practices

We had access to reused artifacts and stakeholders that took part during the use of opportunistic reuse for creating the variants of the subject systems. For the field study, we collected information (data collection procedure described in the next section) from several stakeholders. Most of them are from the IT department, such as managers, developers, analysts, and testers. Additionally, for the sake of representativeness, we also collected information from managers and a user, which allowed us to enrich the analysis of the causes and consequences of variability debt. As the characteristics of the stakeholders were collected through the survey, we delegate the presentation of the 22 participants to Section 5.

4.2. Data collection and analysis

We describe below the sources of information for collecting the data, analysis conducted, and tools used for our field study.

² <https://td4vis.github.io/2022/>

4.2.1. Source of information

For the data collection of our field study, we have two main sources of information: (i) *artifacts* such as source code, system requirements, feature specifications, and test/validation cases; and (ii) *survey with stakeholders* for collecting characteristics and opinions of people involved with the subject systems and their variants. For collecting the data, according to Lethbridge et al. classification (Lethbridge et al., 2005), we used a technique of the second and third level. The survey is in the second level, as we collected the answers (raw data) directly from the stakeholders using an online form, namely Google Forms.³ The data collected from artifacts is considered in the third level, since we analyzed artifacts already available.

4.2.2. Survey

We present all survey questions in [Appendix](#). It contains five groups of questions: (i) about the participant profile (e.g., experience), (ii) technical questions about the variability debt incurred because of clone-and-own practices, (iii) information about their perception of the causes, (iv) perception of the consequences, (v) questions aiming to explore the suitability of the variability debt metaphor for their case and the prioritization of variability debt management activities that they envision interesting for their scenario. The survey questions expect different types of answers, namely open text boxes, predefined alternatives, and five-point Likert scales.

Prior to answering the survey, we introduced the concepts of variability and opportunistic reuse to the participants as follows:

- **Variability:** Variability is the ability of a software system or software artifact to be extended, customized, or configured for use and reuse in specific contexts. Thus, a system variant is one of the system versions created for a specific context.
- **Opportunistic reuse:** a common practice in development companies is opportunistic (unsystematic) reuse, a.k.a., copy-and-paste or clone-and-own. In this practice, when there is a demand for a new product variant or feature, parts or complete artifacts of existing an existing product are copied/cloned and modified/adapted to satisfy the requirements.

The definition of variability debt was only presented during the survey when responding to the Definitions part of the survey.

4.2.3. Tools

For basic analysis of the source code such as files and packages structure/nomenclatures, we relied on the *Eclipse IDE*.⁴ For the comparison between variants of each system, we used the framework Bottom-Up Technologies for Reuse (*BUT4Reuse*) (Martinez et al., 2023). *BUT4Reuse*⁵ provides several tools for mining software artifact variants, allowing overlap analysis (n-way comparison of variants for commonality and variability analysis), among many other features for re-engineering variants that are out of the scope of this work. For the quantitative and qualitative analysis of the survey with stakeholders, we adopted online collaborative spreadsheets, namely *Google Sheets*.⁶

5. Results and discussions

The results and discussions are presented according to the research questions presented in Section 4.

Table 2

Available artifacts existing for each system.

Artifact	SysX			SysY		SysZ	
	Orig	Var1	Var2	Orig	Var1	Orig	Var1
Source code	✓	✓	✓	✓	✓	✓	✓
Requirements		✓	✓	✓	✓	✓	✓
Specifications	✓	✓	✓	✓	✓	✓	✓
Validation Tests	✓		✓	✓	✓	✓	

Table 3

Number of specifications for each variant.

SysX			SysY		SysZ	
Orig	Var1	Var2	Orig	Var1	Orig	Var1
14	10	11	26	17	7	7

5.1. RQ1: Artifacts affected by variability debt

In order to respond to which and how artifacts are affected by variability debt, we had to search for evidences of opportunistic reuse and analyze its consequences for evolution or maintenance. The results of artifact analysis of our field study are presented firstly for the requirements, specifications, and tests, and secondly for the source code.

5.1.1. Requirements, specifications, and tests

[Table 2](#) presents the artifacts available in each variant. For the Original variant of SysX there are no requirements. The explanation for this is that this variant was developed in 2010, when the IT team of the company was not required to create and maintain a formal document of requirements. Then, as part of the development of Variants 1 and 2, the system analysts created the requirements.

As a further refinement of the high-level requirements, a *specification* in the case of our target company consists in a detailed description of the use cases, business rules, even fine-grained indications such as the fields that the user will have in the forms of the system. These specifications are the main asset for guiding the implementation. Artifacts of type specification are present in all variants of the three systems. [Table 3](#) presents the number of specification documents for each variant. For SysX, the number of specifications is similar to the number of features, as expected. However, for SysY and SysZ, the number of specifications is lower than the number of features. In the analysis, we observed that this happens because only features that would be validated according to existing regulation had the specification created. For SysY, different variant features would be validated for regulations, leading to different number of specifications. In the case of SysZ, the same number of specifications was observed for Original and Variant1, however, the specifications are not totally similar. We closely analyzed the specifications, identifying several changes. Thus, we found evidences of the variability debt of *duplicate documentation* with modifications.

The *validation tests* are artifacts designed mainly to comply with regulation of national agencies. They are documents for performing system and acceptance test, containing checkboxes to indicate that everything is functioning as expected. These tests were performed manually by a team of around four persons, creating screenshots as evidences. In this case, two variants, in SysX and SysZ, did not have this artifact, since the branch in which these variants operate does not require following any regulation. This *lack of testing* in variants created using opportunistic reuse, even if for validation purposes, is one of the variability debt items described in the literature (see [Section 2](#)) and observed in our field study. Besides these validation tests, automatic unit tests are subject of opportunistic reuse for the creation of variants in the field study. This was confirmed in the survey results when asking about the type of artifacts that were cloned (see [Table 8](#) that will be

³ <https://www.google.com/forms/about/>

⁴ <https://www.eclipse.org/ide/>

⁵ <https://but4reuse.github.io/>

⁶ <https://www.google.com/sheets/about/>

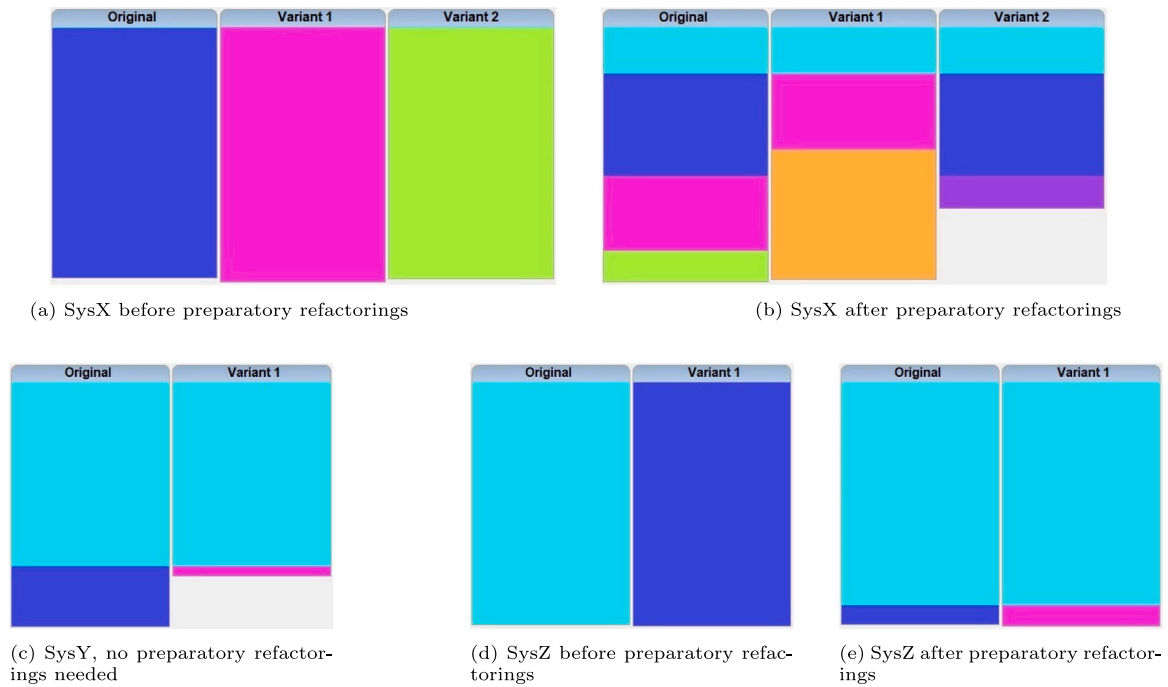


Fig. 1. Differences in the comparison of variants when comparing the source code using the “Java JDT” adapter of BUT4Reuse. All sub-figures are independent comparison analyses even if the used colors are the same. The figures “before” show the results as defined by the developers, while the “after” were fixed by the first author of the paper to recover the needed consistency for the analysis.

Table 4
Technology stack.

Technology	SysX	SysY	SysZ
Platform	Mobile	Web	Web
Application server	Server JSE 1.6	EJB Java EE6	EJB Java EE7
Database Management Systems	PostgreSQL 9.4	PostgreSQL 9.1	Oracle 11 g
Frameworks	JSE 1.3	Hibernate 3 and Primefaces 2.2.1	Hibernate 5 and Spring Data JPA
Graphical User Interface	AWT	JSF 2.1	JSF 2.2
Reporting Engine	–	JasperReport 4.5.1	JasperReport 6.5.1
Connection with the ERP	RFC	RFC	RFC
Authentication	User Database	LDAP	Active Directory

introduced later, where we can see the mention to unit test cases). In the case of RQ1, we did not distinguish between source code and unit tests (which are also source code) when performing the analysis. Therefore, as we will see in the next subsection, unit tests also have the *code duplication* and synchronization issues for *multi-version support*.

5.1.2. Technology stack and source code

We analyzed the technology stack used in the implementation of the systems, presented in Table 4. During our analysis, we observed that the variants of the same system use the same technology stack. This helps in the maintenance and evolution of the system variants as a whole, so apparently no issues were identified regarding this.

The next analysis relied on the source code of the variants. The three systems have Java-based source code. We used BUT4Reuse that allows the analysis of source code through reverse engineering techniques, including the comparison of several variants (n-way comparison). Table 5 shows some metrics obtained with the tool to provide an overview of the size of the systems. SysY is the largest one, but both SysX and SysZ are also complex systems of relatively large size.

To provide evidences and measures of commonality and variability within these large systems, we used the variants’ comparison techniques of BUT4Reuse. Fig. 1 shows the bars and stripes visualization (Martinez et al., 2014) with the results of the comparisons between variants for the three systems. For this analysis, we used the Java JDT (Eclipse Java development tools) source code adapter of BUT4Reuse. Thus, each bar is a variant, and the stripes (blocks) with different

Table 5

Source code size of the variants (packages = Java Packages, .java = Java files, methods = Java methods).













Sys.	Var.	#packages	#.java	#methods
SysX	Orig	4	91	1497
	Var1	4	90	1517
	Var2	5	91	1495
SysY	Orig	82	605	6270
	Var1	63	475	4971
SysZ	Orig	23	128	2846
	Var1	23	132	2838

colors are the results of the static commonality and variability analysis. A shared block color among variants means that these elements are shared. The size of the blocks correspond to the number of Java elements (e.g., compilation units, types, methods, and fields). Each of the five subfigures in Fig. 1 are independent comparison analyses. That means that the colors of the blocks do not mean that they have any relation with the other subfigures. The colors and numbers of the block are determined according to the identified intersections of the variants during the comparison of the elements. Table 6 will present later information about the identified blocks for the most interesting figures, which are Figs. 1(b), 1(c), and 1(e).

For SysX and SysZ in Fig. 1, we show two cases, namely “before preparatory refactorings” and “after preparatory refactorings”. When

Table 6

Source code size of the blocks (pack. = Java Packages, .java = Java files, me. = unique implementations of Java methods). The rows correspond to Figs. 1(b), 1(c), and 1(e).

Sys.	Blocks	Var. presence	#pack.	#.java	#me.
SysX	 Common	Orig,Var1,Var2	3	23	208
	 Block 1	Orig,Var2	0	26	599
	 Block 2	Orig,Var1	1	40	520
	 Block 3	Orig	0	1	41
	 Block 4	Var1	0	25	596
	 Block 5	Var2	0	1	52
SysY	 Common	Orig,Var1	61	466	4622
	 Block 1	Orig	21	139	1648
	 Block 2	Var1	0	2	272
SysZ	 Common	Orig, ar1	23	126	2431
	 Block 1	Orig	0	2	415
	 Block 2	Var1	0	6	407

we analyzed the source code of the systems without any modification, we found a lack of common source code between variants that were supposed to be created through clone-and-own. As we can observe in Figs. 1(a) and 1(d), the results incorrectly suggested that the variants were unique or almost unique (e.g., just one shared Java file in SysX variants). However, by the specifications and requirements, we knew that these variants have most of their implementation similar to each other. Then, we realized that after cloning the variant Original, many packages and Java types were renamed, making inconsistent the file structure between the variants.

These non-propagated changes might already challenge the future evolution of the system, as it complicates the manual propagation of bug fixes and the future consolidation of variants (Fenske et al., 2017; Benmerzoug et al., 2020). A characteristic found in the three systems is to rename packages based on the industry branch that was targeting the variant. Other changes seemed more with the intention to improve the file structure quality. SysY, where the source code was consistent between the two variants, shows that it is possible to create a variant removing ten features (see Table 1) while keeping the source code structure consistent. There will be technical debt for propagating changes, but at least the propagation will require less effort as the developers will not need to reason in the refactorings.

To continue with our variability analysis, we needed to perform what has been defined as *preparatory refactorings* for analyzing cloned variants (Fenske et al., 2017). This practice has been already reported analyzing a real set of variants called Apo-Games (Fenske et al., 2017; Lima et al., 2019). After a manual analysis of the source code, the first author of this paper refactored the variants through several packages renaming to make them consistent. Finally, after rerunning BUT4Reuse to analyze the commonality and variability, we obtained the result presented in Figs. 1(b) and 1(e). We can clearly see how similar are the variants “after” the preparatory refactorings in comparison to the variants “before” the renaming. The preparatory refactoring was needed for the automated static commonality and variability analysis technique to identify the resulting blocks. As mentioned before, all the subfigures in Fig. 1 represent independent analysis. This applies also for the “before” and “after” analysis. For example, it must not be considered that the block with green color in Fig. 1(a) (the block present only in Variant 2) has relation with the same block color of Fig. 1(b).

Table 6 provides details on the distinguishable blocks shown in Figs. 1(b), 1(c), and 1(e). As discussed above, modifications in the source code of the blocks corresponding to more than one variant (i.e., Common, Block 1 and 2 for SysX, and Common for SysY and SysZ) will suggest the need for changes propagation. We can observe that these are the larger blocks, except for SysX Block 4 where Variant1 has significant source code specific for this variant, namely 25 Java

files and 596 unique implementations of methods in these Java files or in other Java files from the Common or Block 2. Notice that certain numbers between Tables 5 and 6 does not necessarily need to match. For example, in SysX Variant2 we have five packages while the total number of the SysX blocks is four. This is because of the preparatory refactorings, as Table 5 represented the version of the developers and one package was renamed. Also, the number of methods in Table 6 does not represent the number of method declarations, but the number of unique implementations of a method. That means that, for example, the same method declaration can have one implementation in one block and another one in another (i.e., at least one source code statement inside the method were different between variants).

Based on our catalog of variability debt items and the commonality and variability evidences in the source code, we can observe that *code duplication* is a prevalent issue. Moreover, the fact that refactorings were performed in the variants challenges the synchronization of the variants for *multi-version support*.

Answering RQ1: Based on the analysis of how variability debt happens in practice, we observe that source code is the artifact more cloned for implementing variability in an opportunistic reuse way. This created issues like the replication of business rules, duplicate bugs and the need to correct them. The complexity in maintenance is aggravated because of non-synchronized refactorings. Also, specification documents are also replicated for software variants.

5.2. RQ2: Survey with stakeholders to analyze their perception of technical debt related to opportunistic reuse

This section describes the results and analysis of the survey with stakeholders to investigate the phenomenon of variability debt from the point of view of practitioners.

To answer RQ2, based on the survey in Appendix, we ask stakeholders about the context, causes, and consequences, they faced about opportunistic reuse without mentioning the variability debt concept. Then, after these questions parts were answered, we introduced to them our definition of variability debt as a way to name the phenomenon. This last part aims to understand if the practitioners agree with this category of technical debt, independently whether they considered that they had it in their systems or not. In the following analysis, we present the results without making the distinction between variability debt and the issues derived from opportunistic reuse regarding evolution and maintenance of the family as a whole.

During the study design, we identified 25 stakeholders that could be part of the survey. We invited all of them, and the survey was responded by 22 practitioners (88% of participation rate). The following subsections present the survey results of our field study.

5.2.1. Characterization

Table 7 presents the survey participants. For the analysis, we identified the participants of the survey from P1 to P22. From the data in the table, we did different analysis. The pie-charts presented in Figs. 2, 3, and 4, show that most of the participants had more than 10 years of experience, the three systems are well represented with more participants in SysY and SysX, and Developer is the most represented role, but a diversity of roles is observed.

As part of the survey, we asked the participants if the use of opportunistic reuse in the system they know led to initial benefit to the business, but incurred in maintenance and evolution problems afterward. Fig. 5 presents the percentage of answers according to the Likert scale. This figure does not consider the answer of P10, an end user that selected “not answer”. Most of the participants (12 = 57%) agreed or strongly agreed that there were problems related to the maintenance and evolution in the systems they worked on, which were created with opportunistic reuse. Five participants disagree and only one strongly disagree (summing 29%) of the existence of such problems

Table 7
Survey participants and basic information.

ID	Role	Prof. Exp.	System
P1	Developer	≥10	SysY
P2	Developer	≥10	SysX
P3	Developer	5 to 10	SysX
P4	Business manager	≥10	SysY
P5	Help-desk analyst	5 to 10	SysY
P6	Test analyst	≥10	SysX
P7	System analyst	≥10	SysX
P8	Developer	≥10	SysZ
P9	System analyst	≥10	SysY
P10	End user	≥10	SysY
P11	Business manager	≥10	SysY
P12	Test analyst	5 to 10	SysX
P13	IT manager	≥10	SysX
P14	System analyst	5 to 10	SysY
P15	Developer	≥10	SysY
P16	Developer	≥10	SysX
P17	Business manager	5 to 10	SysZ
P18	Infrastructure analyst	≥10	SysY
P19	Developer	5 to 10	SysX
P20	IT manager	≥10	SysY
P21	Test analyst	5 to 10	SysZ
P22	System analyst	2 to 5	SysZ

Prof. Exp. = professional experience in years, options: 0 to 2, 2 to 5, 5 to 10, ≥10.

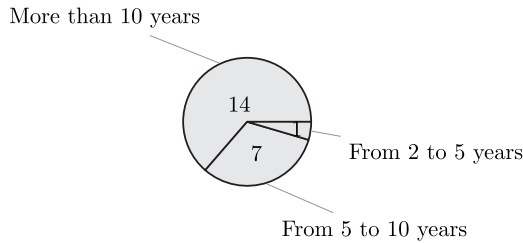


Fig. 2. Years of professional experience.

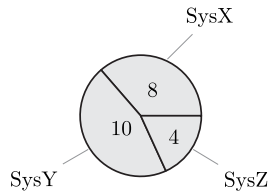


Fig. 3. Number of participants per system.

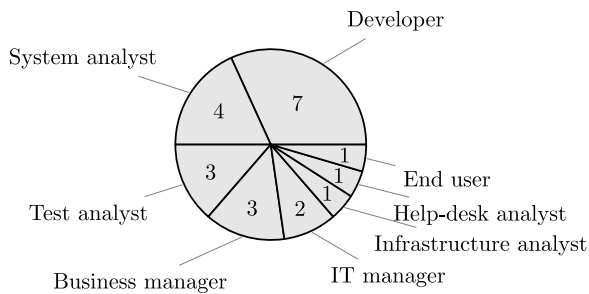


Fig. 4. Participant's roles.

in the system. Finally, three participants do not agree or disagree (neutral = 14%).

For a deeper understanding of the practitioners' opinions, we analyzed the answers based on the role of the participants. We found

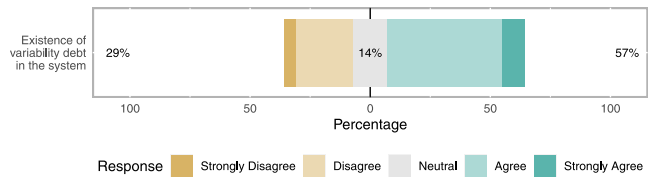


Fig. 5. Opinion of practitioners regarding the existence of variability debt in the systems/variants they interact with.

that: (i) 66.6% of the business managers does not agree that the systems created with opportunistic reuse had maintenance and evolution problems, whereas IT managers have split opinions; (ii) among system and infrastructure analysts, 80% (strongly) agree that the systems have technical debt related to variability; (iii) the majority of developers (57%) also (strongly) agree with the existence of maintenance and evolution problems in the systems they worked on, whereas 28% disagree and 14% responded as neutral.

Based on the results, we can see a tendency that practitioners in managerial roles not perceive variability debt. On the other hand, practitioners in technical roles, mostly agreed that the opportunistic reuse practices led the systems to incur in technical debt related to variability. However, we can observe some exceptions. For example, a business manager (P17) commented about the Variant1 of SysZ:

'The system [Variant1] did not reflect our reality, the non-conformity classifications were not in accordance to our needs.'

Interestingly, this is related to the external quality of the systems, which is not directly related to technical debt. However, the quality of the variant created with opportunistic reuse was not satisfactory, and the manager could see issues. Variability debt related to *system-level structure quality issues, lack of testing, or multi-version support* might be the source of problems for this external quality of the variant. This finding is supported by the answers from developers. P1 reported:

'Some bugs for automatic job execution were fixed on the main system [SysY, Original]. However, they were not fixed in the other system [SysY, Variant1], which was a copy. Also some improvements in the document reprint rules applied during the evolution of the main system [SysY, Original] were not reproduced in the other system Y [SysY, Variant1].'

Similarly, P15 (for SysY) and P16 (for SysX) also reported that they had to change features of variants created with opportunistic reuse because they had been copied the whole original variants during the reuse. However, changed were needed in Variant1 because they did not fully fit the context of the system (variant Original). This statement relates to *multi-version support*. Additionally, P3 provided further comments about documentation:

'In some parts of the documentation of similar features [of SysX], some small details that were not changed in the copy [Variant1] compromised the result.'

This is the occurrence of the variability debt of *duplicate documentation*. The developer P2 described a different situation:

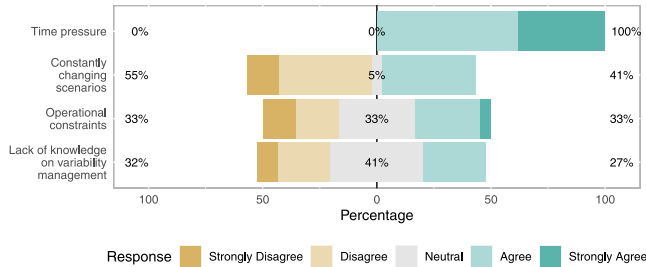
'the copied parts [from SysX Original] (communication structures and integration with legacy system) were quite stable and had little structural difference.'

This was the justification of the developer for answering that opportunistic reuse in SysX did not lead to technical debt. But as we can see, this developer considered a very specific point, perhaps not taking part in issues raised by the other developers.

Table 8

Artifacts mentioned in the survey.

Artifact	# mentions
Requirements	13
Specifications	8
Source code	7
Technical Manuals	4
Screen prototypes	4
Documented architecture	3
UML diagrams	3
Entity-relationship diagram	2
Unit test cases	1
Widget test cases	1
DB schemas	1

**Fig. 6.** Evaluation of the participants about variability debt causes.

Answering RQ2 - Characterization: Based on the survey responses, we can observe that managerial practitioners usually do not perceive variability debt, where technical practitioners clearly agree with the variability debt existence. Also, in the projects analyzed, we could see instances of variability debt described in the literature, namely system-level structure quality issues, lack of testing, multi-version support, and duplicate documentation.

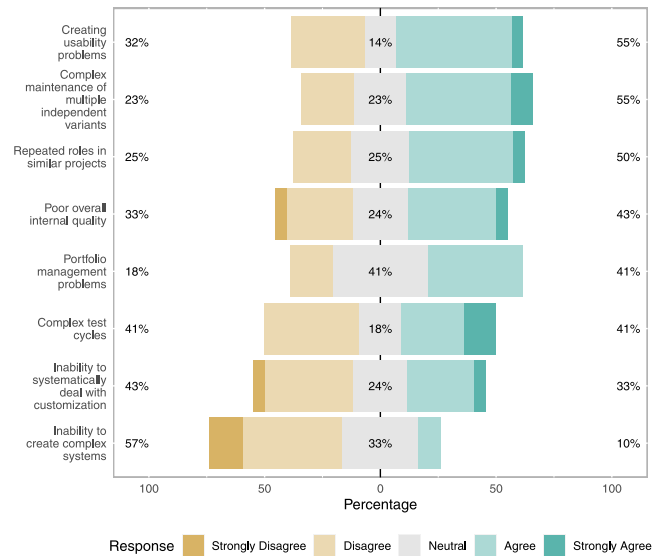
5.2.2. Artifacts, causes, and consequences

To investigate how variability debt incurs in artifacts, we inquired which types of artifacts were part of the opportunistic reuse when creating the variants. Table 8 presents the list of artifacts and the number of mentions from different participants, sort by the latter.

We observe a large diversity of mentioned artifacts. As the field study took place in an industry that must follow regulations, several artifacts of the software development process are mandatory. However, some variants (e.g., SysX - Variant1 and SysZ - Variant1) were deployed in unregulated group branches, which reduced the concerns of reusing some artifacts. We can see that the company pays important attention to requirements that describes customer needs, as requirements is the artifact type mentioned the most. Requirement was mentioned by all practitioners from all roles. As expected, 71% of the mentions to source code artifact came from software developers. The artifact specifications had a considerable division, being 50% indicated by developers, 25% by system analysts and the rest by the infrastructure analyst and the test analyst. Some artifacts are only mentioned once or few times. However, identifying them through the survey was valuable to understand the diversity of artifacts.

Considering that most of the practitioners pointed the existence of maintenance and evolution problems because of opportunistic reuse, we asked about the causes that lead the practitioners to incur in such a type of debt. The basis for the predefined options of answer were the findings of our previous study (Wolfart et al., 2021) (see Section 2). Using the Likert scale, the participant could answer about each cause.

Fig. 6 presents a summary of the answers. Time pressure is convincingly pointed as one of the main causes for variability debt, in which all participants agree (only P3 chose “not answer”). This highlights the phenomenon of technical debt, because when delivering in a short

**Fig. 7.** Consequences of variability debt.

period of time we have some benefits, reduced time-to-market, or for regulatory reasons, satisfy customer pressure; but ends up generating a mid- long-term loss due to lack of planning, poor quality, and maintenance difficulties. Some comments from the stakeholders about time pressure are presented in what follows:

‘P2: Time pressure and the use of the same communication architecture were reasons for using the solution that was already working.’

‘P4: When the development time is not enough, you end up choosing to use ready-made or already known functionalities.’

‘P6: The time to execute the project was strictly short, and it was not possible to implement a new tool because it is a process with particularities and with many changes. The company and the process is extremely dynamic.’

‘P9: Due to lack of time, it is always faster to copy and paste, and only make changes where necessary.’

Other nine participants also reported their experience with the projects and the time pressure they had faced to build and deploy the variants.

For the other causes, namely Constantly changing scenarios, Operational constraints, and Lack of knowledge on variability management, we can observe, generally, a disagreement or neutral position.

The survey results of the consequences are presented in Fig. 7. The consequence list were also obtained from our previous study (Wolfart et al., 2021). We can observe that creating usability problems and complex maintenance of multiple independent variants are the ones with more agreement. This complex maintenance is comprehensible, as observed in our results of the artifacts’ analysis.

Examples of comments about usability problems are:

‘P12: Users do not understand the usability of the system and tend to be resistant to changing the process to use the system without customization.’

‘P21: There can be usability problems when the same user executes processes within two cloned systems, which can be different on each system.’

And examples of participants discussing the complex maintenance and inability to systematically deal with customization.

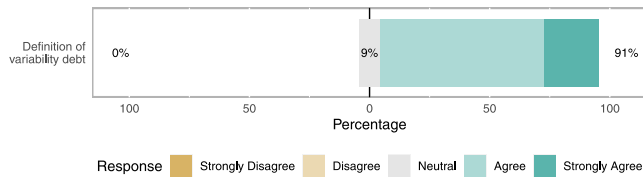


Fig. 8. Clarity of the variability debt definition and concept.

'P17: Any change that was needed was extremely time consuming.'

'P20: Problems generated by the conflict of different business rules for each operation, were causing unplanned downtime and greater complexity in test cycles.'

'P21: Similar improvements might be necessary in cloned systems, but this can be difficult due to the specific customizations of each project.'

Answering RQ2 - Artifacts, Causes, and Consequences: Based on the characteristics collected in the literature on variability debt (definition, causes, artifacts, and consequences) practitioners with different roles were asked about their opinion regarding these characteristics. In general, most agree with what was found in the literature and added the causes of "financial constraints" and "lack of adequate software development process". Additionally, they mentioned the consequences of "increased business risk for the client", "increased rework" and "process improvements are lost".

5.2.3. Definition and perspectives

In the last part of the survey, we presented our definition of variability debt and asked the participants about the clarity of such definition. Concretely, to which extent do they agree that the variability debt definition is clearly described in relation to its usage context and purpose for creating awareness in the company. The response is shown in Fig. 8 evidencing an agreement.

One participant made an interesting observation:

'P1: I think that the term technical debt is enough. Technical debt occurs for several reasons, and variability is one dimension among them.'

We agree with P1. As already mentioned, the research line of variability debt is to investigate the role of variability as source of technical debt. In this work we focus on opportunistic reuse as source of technical debt, but technical debt can also be incurred in a software product line, as also pointed out by another participant:

'P15: I agree that initially clone-and-own proves to be simpler, but as time goes by, maintaining these architectures becomes much more complex. I also understand that maintaining highly configurable software is extremely complex and that if it is poorly applied, it can cause even greater maintenance or impair system performance, mainly due to the large occurrence of requirements changes during its life cycle.'

Then we asked participants about perspectives with regard to variability debt management. Notably, to understand to which extent do they agree that supporting certain activities is important. We ask for the five key activities in technical debt management as reported in Li et al. (Li et al., 2013, 2015). Following Li et al. definitions of the activities, adapted to variability debt, they are: (i) *Identification*: Detecting technical debt items associated with variability, (ii) *Measurement*: Estimating the cost and benefit associated with a variability debt item, (iii) *Prioritization*: Identify which variability debt item should be addressed first, (iv) *Repayment*: Changes and transformation helps to eliminate the negative effect of a variability debt item, and (v) *Monitoring*: Check the changes of the costs and benefits

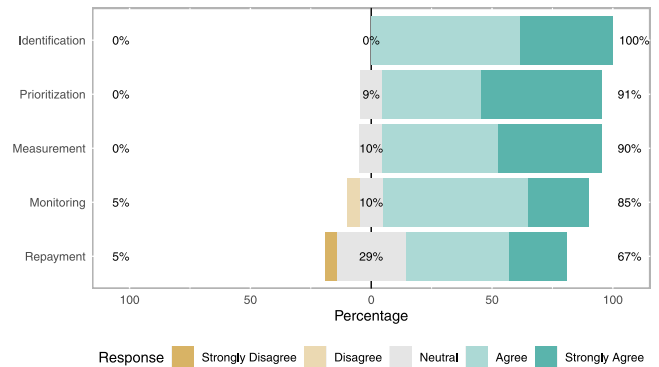


Fig. 9. Perspectives for the activities of variability debt management.

of unresolved variability debt items over time. Despite their importance, we did not include, technical debt prevention, representation/documentation, and communication which were later added in another work of Li et al. (2015). We excluded them to shorten the survey to the key ones as included in Li et al. (2013) and reduce the participant fatigue.

The result is presented in Fig. 9, showing a general agreement of the importance of all activities. Full agreement corresponds to the Identification activity, followed by Prioritization and Measurement.

One participant emphasized the quantitative measurement of the debt as a relevant input for the business. This will also include its identification.

'P20: The biggest challenge is to quantify this Debt for the business before developing the solutions, as the short-term need ends up taking the "easy" path.'

Answering RQ2 - Definition and Perspectives: Variability debt is perceived as a valuable metaphor to create awareness within the company about the concerns of opportunistic reuse in the creation of variants. Methods and functionalities will be desired for variability debt management activities.

6. Threats to validity

As mentioned by Stol and Fitzgerald (2018), known problems of field studies are the lack of statistical generalizability, no control over events, and the low precision of measurement. Every company will have their own scenario and characteristics, but the insights can be generalized to companies with similar settings. Therefore, applying this study to a company and three projects might not be enough to generalize the answers of our RQs. However, considering that we are dealing with an industrial field study, we are focusing on the scope of this company and its settings.

For the preparatory refactorings needed for the source code analysis, the systems are complex and of significant size, thus, it is possible that this effort was not complete (e.g., we did not investigate renaming at method level). However, we consider that for the purpose of showing evidence and size of commonality and variability, the refactorings identified and reverted for the variants' comparison purpose were already valid. For an actual re-engineering of the variants to a software product line, this effort will require much more fine-grained analysis, but this was out of the scope of the study.

There are also threats to the validity regarding the survey. The total number of participants can be considered a small number compared with other more general surveys in software engineering research. Being a field study, as mentioned, we tried to cover most of the involved stakeholders, and we consider that we succeed in covering the diversity of involved roles, namely, developers, system analysts, test analysts, business managers, IT managers, infrastructure analyst, help-desk analyst, and even one end user. In fact, the majority of participants (except for the end-user) represent stakeholders with knowledge on the internal and the development of the targeted systems, so for a field study, our sample of participants can be considered as representative. However, there are other known intrinsic threats to validity regarding surveys, as for example, user fatigue while responding, or the interpretation of Likert scales. We mitigated this issue by limiting the amount of questions to an acceptable number such that it did not take longer than half-an-hour, and by interacting among the authors to define clear questions.

7. Conclusions and future work

The focus of the present work is to analyze and understand variability debt in a real scenario of a company that uses opportunistic reuse. To do so, we performed a field study in a Brazilian multinational industry leader in its market segment in Latin America. Our field study relied on three subject systems and their seven variants. Our study also included a survey with 22 stakeholders of the three systems to observe their perception of variability debt.

We can synthesize our findings as follows: (i) the most cloned artifact for implementing variability in an opportunistic reuse way is the source code, which creates future issues like the replication of business rules, duplicate bugs, and the need to correct them independently; (ii) different from the managerial practitioners, technical practitioners clearly perceived variability debt; (iii) we could see instances of variability debt described in the literature, namely system-level structure quality issues, lack of testing, multi-version support, and duplicate documentation; (iv) the participants of the survey confirmed the causes and consequences of variability debt of our previous study (Wolfart et al., 2021) and added few ones; and (v) the variability debt metaphor was considered useful to highlight the concerns of opportunistic reuse in the creation of variants.

Our findings contribute to bringing the attention of practitioners to problems that can be generated by using opportunistic reuse practices for software reuse and variability management. Furthermore, the results of our field study generate opportunities for new studies and tooling for variability debt management in the opportunistic reuse scenario including more advanced visualization support for variants evolution (Medeiros et al., 2023) that could improve, for example, the automation and quality of the analyses described in Section 5.1.

For further work, we consider that more work and community consensus is needed on the understanding of the variability debt phenomenon. Notably, as confirmed in the TD4ViS workshop mentioned in Section 3, we desire to have a better understanding in other scenarios not related to opportunistic reuse. For instance, limitations in the design of a software product line architecture. We can consider that defects in (or the absence of) software product line engineering processes can create maintenance or evolution issues of the product family as a whole. This way, variability debt can be investigated in the mentioned case of opportunistic reuse, but also, for instance in the case of implementing variability management with feature toggles, creating variants with component-based software engineering, feature-based SPL with an annotative approach, feature selection for

components composition, feature-based manual creation of the variants (feature selection is available, but there is no derivation mechanism and the variant need to be composed almost in a manual way). Also, we consider that field studies, similar to the one present in this work, are a good direction to achieve this goal. This way, research works can investigate the subject from a bottom-up perspective (as the presented field study), while others, more oriented to a theoretical and conceptual framework can, in parallel, investigate it in a top-down approach.

Finally, besides the conceptual characterization of variability debt, concrete tools for variability debt management are to be designed, implemented, and evaluated. This way, identification, measurement, prioritization, repayment, monitoring, prevention, representation/documentation, and communication of variability debt can be possible. Some already existing tools can be integrated in the conceptual framework of variability debt management, however, we consider that new ones are to be developed leveraging the conceptual framework and the advances in the general technical debt management field.

CRedit authorship contribution statement

Daniele Wolfart: Conceptualization, Methodology, Validation, Investigation, Data curation, Writing – original draft, Writing – review & editing, Visualization. **Jabier Martinez:** Conceptualization, Methodology, Validation, Investigation, Data curation, Writing – original draft, Writing – review & editing, Visualization. **Wesley K.G. Assunção:** Conceptualization, Methodology, Validation, Investigation, Data curation, Writing – original draft, Writing – review & editing, Visualization. **Thelma E. Colanzi:** Writing – original draft, Writing – review & editing, Visualization. **Alexander Egyed:** Conceptualization, Methodology, Validation.

Declaration of competing interest

The authors declare that they have no financial or non-financial conflict of interest that are directly or indirectly related to this work submitted for publication.

Data availability

The authors do not have permission to share data.

Acknowledgments

The research reported in this paper has been funded by CNPq Grant No. 428994/2018-0; FAPERJ PDR-10 program under Grant No. 202073/2020; BMK, BMDW, and the State of Upper Austria in the frame of SCCH, part of the COMET Programme managed by FFG; the Secure and Correct System Lab funded by the State of Upper Austria; and the Austrian Science Fund (FWF), grand no. P31989.

Appendix. Questionnaire

Table A.9 presents the complete information about the questionnaire used for the survey (see Section 5.2).

Table A.9

Question of the survey with the stakeholders.

Group	ID	Question	Mand.	Type
1 Characterization	1	What is your name?	No	Open
	2	How long is your professional experience? <i>Options (in years): <2, 2-5, 5-10, >10</i>	Yes	Alt
	3	Indicate which system you will answer the questions. <i>Options: SysX, SysY, SysZ</i>	Yes	Alt
	<i>All the questions below are related to the system indicated above</i>			
	4	How long have you been involved with the system? <i>Options (in months): <6, 6-12, 12-24, >24</i>	Yes	Alt
	5	What was your role in the system or department when you participated in the development of the system? (If you had more than one function, please indicate)	Yes	Open
2 Context	6	How much experience did you have in this role when dealing with the system? <i>Options (in months): <6, 6-24, 24-48, >48</i>	Yes	Alt
	7	Did the use of copy and paste practice in creating different variants of this system have an initial benefit, but did it incur maintenance and evolution problems of the variants (systems) later?	Yes	Likert
	8	If you agreed with the above question, could you report any issues you faced?	No	Open
	9	For the system variants that you came across, what artifacts did you deal with during the software development and system deployment process? (e.g. requirements, design templates, manuals, source code, etc.)	No	Open
3 Causes	10	To what extent do you agree that the following factors are reasons for using an unsystematic practice (e.g., copying and pasting) to create different software variants? - Time Pressure - Lack of knowledge - Constant changes in requirements - Operational constraints	Yes Yes Yes Yes	Likert Likert Likert Likert
	11	For the reasons you agree with above, could you provide more information by citing examples, or describing how this process took place?	No	Open
	12	Are there additional reasons that justified the choice of a non-systematic practice in creating the variants?	No	Open
	13	Are the following problems consequences of using a non-systematic approach to managing product family variability? - Complex maintenance - Inability to systematically deal with customization - Inability to create complex systems - Poor internal software quality - Complex test cycles - Usability issues - Portfolio management issues - Repeated roles in similar projects	Yes Yes Yes Yes Yes Yes Yes Yes	Likert Likert Likert Likert Likert Likert Likert Likert
4 Consequences	14	For the consequences you agree with above, could you provide more information by citing examples, or describing how this impact occurred?	No	Open
	15	Can you indicate other consequences in the system due to the use of non-systematic practice (copy-paste use)	No	Open
	16	A new term was created during the development of this work. “ <i>Variability Debt</i> represents Technical Debt caused by suboptimal problems and solutions in implementing variability management in software systems. That is, companies may choose to implement variability with an approach that would bring short-term benefits, for example, using opportunistic reuse, regardless of medium to long-term technical/architectural disadvantages. The debt of variability leads to maintenance and evolution difficulties to manage families of systems or highly configurable systems.” - To what extent do you agree that the metaphor is clearly described with respect to its context of use and purpose for creating awareness in companies?;	Yes	Likert
5 Definitions	17	Do you have any suggestions/comments on the definition of the term Variability Debt?	No	Open
	18	To what extent do you agree that supporting the following activities is important? - Identification: Detection of technical debt items associated with variability - Measurement: estimating the cost and benefit associated with a variability debt item - Prioritization: Identify which variability debt item should be addressed first - Repayment: Changes and transformations help eliminate the negative effect of a debt item of variability - Monitoring: Watch for changes in the costs and benefits of unresolved variability debt items over time	Yes Yes Yes Yes Yes	Likert Likert Likert Likert Likert

Mand.: Mandatory = Yes/No. Type: Open = open-ended question, Alt = alternative selection, Likert = five points Likert scale.

References

- Arvanitou, E.-M., Ampatzoglou, A., Bibi, S., Chatzigeorgiou, A., Stamelos, I., 2019. Monitoring technical debt in an industrial setting. In: *Proceedings of the Evaluation and Assessment on Software Engineering. EASE '19*, Association for Computing Machinery, New York, NY, USA, pp. 123–132. <http://dx.doi.org/10.1145/3319008.3319019>.
- Assunção, W.K., Lopez-Herrejon, R.E., Linsbauer, L., Vergilio, S.R., Egyed, A., 2017. Reengineering legacy applications into software product lines: a systematic mapping. *Empir. Softw. Eng.* 22 (6), 2972–3016. <http://dx.doi.org/10.1007/s10664-017-9499-z>.
- Avgeriou, P., Kruchten, P., Ozkaya, I., Seaman, C., 2016. Managing technical debt in software engineering (Dagstuhl Seminar 16162). In: Avgeriou, P., Kruchten, P., Ozkaya, I., Seaman, C. (Eds.), *Dagstuhl Rep.* 6 (4), 110–138. <http://dx.doi.org/10.4230/DagRep.6.4.110>.
- Benmerzoug, A., Yessad, L., Ziadi, T., 2020. Analyzing the impact of refactoring variants on feature location. In: *19th International Conference on Software and Systems Reuse*.
- Capilla, R., Bosch, J., Kang, K., 2013. *Systems and Software Variability Management: Concepts, Tools and Experiences*. Springer, <http://dx.doi.org/10.1007/978-3-642-36583-6>.
- Capilla, R., Mikkonen, T., Carrillo, C., Fontana, F.A., Pigazzini, I., Lenarduzzi, V., 2021. Impact of opportunistic reuse practices to technical debt. In: *2021 IEEE/ACM International Conference on Technical Debt*.
- Cunningham, W., 1992. The WyCash portfolio management system. *ACM SIGPLAN OOPS Messenger* 4 (2), 29–30. <http://dx.doi.org/10.1145/157709.157715>.
- Digkas, G., Nikolaidis, N., Ampatzoglou, A., Chatzigeorgiou, A., 2019. Reusing code from StackOverflow: The effect on technical debt. In: *2019 45th Euromicro Conference on Software Engineering and Advanced Applications. SEAA*, pp. 87–91. <http://dx.doi.org/10.1109/SEAA.2019.00022>.
- Echeverría, J., Pérez, F., Panach, J.L., Cetina, C., 2021. An empirical study of performance using Clone & Own and Software Product Lines in an industrial context. *Inf. Softw. Technol.* 130, 106444. <http://dx.doi.org/10.1016/j.infsof.2020.106444>.
- Fenske, W., Meinicke, J., Schulze, S., Schulze, S., Saake, G., 2017. Variant-preserving refactorings for migrating cloned products to a product line. In: *IEEE 24th International Conference on Software Analysis, Evolution and Reengineering. SANER*, IEEE Computer Society, pp. 316–326. <http://dx.doi.org/10.1109/SANER.2017.7884632>.
- Fenske, W., Schulze, S., 2015. Code smells revisited: A variability perspective. In: *Proceedings of the Ninth International Workshop on Variability Modelling of Software-Intensive Systems. VaMoS '15*, Association for Computing Machinery, New York, NY, USA, pp. 3–10. <http://dx.doi.org/10.1145/2701319.2701321>.
- Fischer, S., Linsbauer, L., Lopez-Herrejon, R.E., Egyed, A., 2014. Enhancing clone-and-own with systematic reuse for developing software variants. In: *IEEE International Conference on Software Maintenance and Evolution*, pp. 391–400. <http://dx.doi.org/10.1109/ICSME.2014.61>.
- Junior, H.J., Travassos, G.H., 2022. Consolidating a common perspective on *Technical Debt* and its Management through a Tertiary Study. *Inf. Softw. Technol.* 149, 106964. <http://dx.doi.org/10.1016/j.infsof.2022.106964>.
- Kruchten, P., Nord, R.L., Ozkaya, I., 2012. Technical debt: From metaphor to theory and practice. *IEEE Softw.* 29 (6), 18–21. <http://dx.doi.org/10.1109/MS.2012.167>.
- Lenarduzzi, V., Besker, T., Taibi, D., Martini, A., Arcelli Fontana, F., 2021. A systematic literature review on Technical Debt prioritization: Strategies, processes, factors, and tools. *J. Syst. Softw.* 171, 110827. <http://dx.doi.org/10.1016/j.jss.2020.110827>.
- Lethbridge, T., Sim, S., Singer, J., 2005. Studying software engineers: Data collection techniques for software field studies. *Empir. Softw. Eng.* 10, 311–341. <http://dx.doi.org/10.1007/s10664-005-1290-x>.
- Li, Z., Avgeriou, P., Liang, P., 2015. A systematic mapping study on technical debt and its management. *J. Syst. Softw.* 101, 193–220. <http://dx.doi.org/10.1016/j.jss.2014.12.027>.
- Li, Z., Liang, P., Avgeriou, P., 2013. Architectural debt management in value-oriented architecting. In: *Economics-Driven Software Architecture*. Morgan Kaufmann, Academic Press, Elsevier, pp. 183–204. <http://dx.doi.org/10.1016/b978-0-12-410464-8.00009-x>.
- Lima, C., Assunção, W.K.G., Martinez, J., Mendonça, W., do Carmo Machado, I., von Flach, G. Chavez, C., 2019. Product line architecture recovery with outlier filtering in software families: the Apo-Games case study. *J. Braz. Comput. Soc.* 25 (1), 7:1–7:17. <http://dx.doi.org/10.1186/s13173-019-0088-4>.
- Martinez, J., Assunção, W.K.G., Wolfart, D., Schmid, K., Ampatzoglou, A., 2022. TD4ViS 2022: 1st international workshop on technical debt for variability-intensive systems. In: *26th ACM International Systems and Software Product Line Conference*.
- Martinez, J., Assunção, W.K.G., Ziadi, T., 2017. ESPLA: A catalog of extractive SPL adoption case studies. In: *21st International Systems and Software Product Line Conference. SPLC*, ACM, pp. 38–41. <http://dx.doi.org/10.1145/3109729.3109748>.
- Martinez, J., Ziadi, T., Bissandé, T.F., Klein, J., Traon, Y.L., 2023. Bottom-up technologies for reuse: A framework to support extractive software product line adoption activities. In: Lopez-Herrejon, R.E., Martinez, J., Assunção, W.K.G., Ziadi, T., Acher, M., Vergilio, S.R. (Eds.), *Handbook of Re-Engineering Software Intensive Systems Into Software Product Lines*. Springer International Publishing, pp. 355–377. http://dx.doi.org/10.1007/978-3-031-11686-5_14.
- Martinez, J., Ziadi, T., Klein, J., Traon, Y.L., 2014. Identifying and visualising commonality and variability in model variants. In: *10th European Conference on Modelling Foundations and Applications. ECMFA@STAF*, In: *Lecture Notes in Computer Science*, vol. 8569, Springer, pp. 117–131. http://dx.doi.org/10.1007/978-3-319-09195-2_8.
- Medeiros, R., Martinez, J., Díaz, O., Falleri, J., 2023. Visualizations for the evolution of Variant-Rich Systems: A systematic mapping study. *Inf. Softw. Technol.* 154, 107084. <http://dx.doi.org/10.1016/j.infsof.2022.107084>.
- Pohl, K., Böckle, G., van der Linden, F., 2005. *Software Product Line Engineering - Foundations, Principles, and Techniques*. Springer, <http://dx.doi.org/10.1007/3-540-28901-1>.
- Ribeiro, L., Farias, M., Mendonça, M., Spínola, R., 2016. Decision criteria for the payment of technical debt in software projects: A systematic mapping study. In: *Proceedings of the 18th International Conference on Enterprise Information Systems, Vol. 1. ICEIS*, pp. 572–579. <http://dx.doi.org/10.5220/0005914605720579>.
- Rios, N., de Mendonça Neto, M.G., Spínola, R.O., 2018. A tertiary study on technical debt: Types, management strategies, research trends, and base information for practitioners. *Inf. Softw. Technol.* 102, 117–145. <http://dx.doi.org/10.1016/j.infsof.2018.05.010>.
- Siebra, C.A., Tonin, G.S., da Silva, F.Q.B., Oliveira, R.G., Antonio, L.C., Miranda, R.C.G., Santos, A.L.M., 2012. Managing technical debt in practice: An industrial report. In: *Proceedings of the 2012 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, pp. 247–250. <http://dx.doi.org/10.1145/2372251.2372297>.
- Stol, K.-J., Fitzgerald, B., 2018. The ABC of software engineering research. *ACM Trans. Softw. Eng. Methodol.* 27 (3), <http://dx.doi.org/10.1145/3241743>.
- van der Linden, F., Schmid, K., Rommes, E., 2007. *Software Product Lines in Action - the Best Industrial Practice in Product Line Engineering*. Springer, <http://dx.doi.org/10.1007/978-3-540-71437-8>.
- Verdecchia, R., Kruchten, P., Lago, P., Malavolta, I., 2021. Building and evaluating a theory of architectural technical debt in software-intensive systems. *J. Syst. Softw.* 176, 110925. <http://dx.doi.org/10.1016/j.jss.2021.110925>.
- Wolfart, D., Assunção, W.K.G., Martinez, J., 2021. Variability debt: Characterization, causes and consequences. In: *SBQS 2021: 20th Brazilian Software Quality Symposium*, pp. 1–10. <http://dx.doi.org/10.1145/3493244.3493250>.



Daniele Wolfart is a faculty member at Biopark Education, Brazil. She obtained her M.Sc. in Computer Science from Western Paraná State University (Unioeste) also in Brazil. Daniele has several years of industrial experience as a software engineer and systems analyst. She co-organized the International Workshop on Technical Debt for Variability-Intensive Systems, as part of the 26th ACM International Systems and Software Product Line Conference (SPLC 2022).



Jabier Martinez is an Applied Research Engineer of the industry and mobility unit of Tecnalia. He currently works on AI and Industry 4.0 challenges. His background is on providing methods and tools for systems modeling and variability management. After several years of industrial experience, he received his Ph.D. from the Luxembourg and Sorbonne Universities with an awarded thesis about product line adoption and analysis. With an active participation in the software and systems product line community, he co-organized the first Variability Debt workshop and the Reverse Variability Engineering series of workshops. He is one of the co-editors of the Springer Handbook on Re-Engineering Software Intensive Systems into Software Product Lines.



Wesley K.G. Assunção is an Associate Professor with the Department of Computer Science at North Carolina State University. Wesley was a University Assistant in the Institute of Software Systems Engineering (ISSE) at Johannes Kepler University Linz (JKU), Austria (2021–2023); a Postdoctoral Researcher at Pontifical Catholic University of Rio de Janeiro (PUC-Rio), Brazil (2019–2023); and an Associate Professor at Federal University of Technology - Paraná, Brazil (2013 to 2020). He obtained his M.Sc. and Ph.D. in Computer Science from Federal University of Paraná (UFPR) also in Brazil. Wesley has been serving as reviewers for many conferences and journal, and as organizer of conference, symposiums, workshops, competitions, and meetings. Further information: <https://wesleyklewerton.github.io/>.



Thelma E. Colanzi is an Associate Professor at State University of Maringá (UEM), Brazil. Thelma was also a Postdoctoral Researcher at Pontifical Catholic University of Rio de Janeiro (PUC-Rio), Brazil (2019–2021). She received the DS degree in 2014 from Federal University of Paraná (UFPR) and a master's degree in Computer Science and Computational Math from University of São Paulo (ICMC/USP), both universities in Brazil. Her main area of interest is Software Engineering and her current research project focuses on combining search-based software engineering techniques with software architecture, and system modernization with microservices. Thelma has regularly published scientific papers in premier conferences and journals. She has served as an associate editor of JSERD and also as a reviewer to premier journals, such as JSS, TSE and IST. She has participated in conference organizations and in numerous program committees, which are relevant to her research area.



Alexander Egyed is Professor for Software-Intensive Systems at the Johannes Kepler University, Austria. He received his Doctorate from the University of Southern California, USA, and worked in industry for many years. He is most recognized for his work on software and systems design — particularly on collaborative engineering, software modeling, variability, consistency, traceability, and testing.