TECHNICAL DEBT SPECIFICATION AND CATEGORIZATION FOR SOFTWARE AS A SERVICE APPLICATIONS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF INFORMATICS
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

YASEMİN KURANEL

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
THE DEPARTMENT OF INFORMATION SYSTEMS

NOVEMBER 2022

Approval of the thesis:

TECHNICAL DEBT SPECIFICATION AND CATEGORIZATION FOR SOFTWARE AS A SERVICE APPLICATIONS

submitted by **YASEMİN KURANEL** in partial fulfillment of the requirements for the degree of **Master of Science** i**n Information Systems, Middle East Technical University** by,

Date: 28.11.2022

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name Last name : Yasemin, Kuranel

Signature :

ABSTRACT

TECHNICAL DEBT SPECIFICATION AND CATEGORIZATION FOR SOFTWARE AS A SERVICE APPLICATIONS

Kuranel, Yasemin
Master of Science, Department of Information Systems
Supervisor : Assist. Prof. Dr. Özden Özcan Top
Co-Supervisor: Prof. Dr. Altan Koçyiğit

November 2022, 70 pages

An outcome of taking poor decisions or choosing easier solutions for faster code delivery is technical debt (TD). It is important to specify technical debt in any development effort. Technical debt is also common in platform-based solutions. However, there is not much research about TD categorization for software as a service applications (SaaS). In this study, we used different categorization methods to specify the TD in organizations using SaaS applications. To understand the technical debt indicators and problems specific to such organizations, and to address TD management, we conducted two different case studies. First, we evaluated the effectiveness of existing technical debt categorization methods varying in granularity. For the second case study, we used one of the categorization methods with the highest level of detail, which takes the nature of the debt into consideration, and we performed a mapping with the TD categories and ISO/IEC 12207 software life cycle processes. We found the indicators, reasons, the problems arising due to TD, and the ways in which TD management can be performed in organizations working with SaaS applications. It was determined that TD categories and problems that exist in traditional software applications are also seen in the field of SaaS, but

there are also additions to TD problems that are specific to SaaS. The 9 different categories of TD experienced in SaaS applications and the sub-diffractions of the “SaaS Related Limitations” category are presented in the study.

Keywords: Technical Debt, Categorization, Software as a Service

ÖZ

SERVİS OLARAK YAZILIM UYGULAMALARINDA TEKNİK BORÇ BELİRTİMİ VE SINIFLANDIRMASI

Kuranel, Yasemin
Yüksek Lisans, Bilişim Sistemleri
Tez Yöneticisi: Dr. Öğrt. Üyesi. Özden Özcan Top
Ortak Tez Yöneticisi: Prof. Dr. Altan Koçyiğit

Kasım 2022, 70 sayfa

Kötü kararlar almanın veya daha hızlı kod teslimi için daha kolay çözümler seçmenin bir sonucu teknik borçtur (TD). Herhangi bir geliştirme çabasında teknik borcu belirtmek önemlidir. Teknik borç, platform tabanlı çözümlerde de yaygındır. Ancak, servis olarak yazılım uygulamaları (SaaS) için teknik borç sınıflandırması hakkında çok fazla araştırma yoktur. Bu çalışmada, servis olarak yazılım uygulamaları kullanan kuruluşlarda teknik borcu belirlemek için farklı kategorizasyon yöntemleri kullandık. Teknik borç göstergelerini ve bu tür kuruluşlara özgü sorunları anlamak ve TD yönetimini araştırabilmek için iki farklı durum çalışması yürüttük. İlk olarak, ayrıntı düzeyine göre değişen mevcut teknik borç sınıflandırma yöntemlerinin efektifliğini değerlendirdik. İkinci durum çalışması için, borcun niteliğini dikkate alan, detay seviyesi en yüksek kategorizasyon yöntemlerinden birini kullandık ve buradaki teknik borç kategorileri ve ISO/IEC 12207 yazılım yaşam döngüsü süreçleri ile bir eşleştirme gerçekleştirdik. Geleneksel yazılım uygulamalarında var olan kategorilerin servis olarak yazılım alanında da görüldüğünü ancak bu alana özgü farklılıaşmaların da olduğunu, buradaki özgün göstergeleri, sebepleri ve bunlardan kaynaklanan sorunları tespit ettik. Çalışmada servis olarak yazılım

uygulamalarında yaşanan 9 farklı teknik borç kategorisi ve “SaaS ile İlgili Sınırlamalar” kategorisinin alt kırınımları sunulmaktadır.

Anahtar Kelimeler: Teknik Borçlanma, Kategorizasyon, Servis Olarak Yazılım Uygulamaları

Dedicated to my family

ACKNOWLEDGMENTS

I would like to express my gratitude to my supervisor Assist. Prof. Dr. Özden Özcan Top. She always supported me with her ideas, vision, and encouragement. Her teachings will always stay with me, and her perseverance will always be an inspiration for me.

Secondly, I would like to thank my co-advisor Prof. Dr. Altan Koçyiğit for making this study possible. I always felt his support throughout the thesis study, helping me to stay on the right track, and introducing me new ways of thinking throughout all my studies in METU.

I would like to thank Bülent Doğan and Burak Fenercioğlu for giving me the chance to become a part of a wonderful team and encouraging me for this study. I am thankful to all my team members for supporting me and putting up with me, and for being a part of my thesis study.

I am grateful to my friend Onatkut Dağtekin for his support.

I cannot describe how thankful I am for my mother Tümay, my father Levent and my brother Yiğit. Thank you for your love and being there for me whenever I need you.

Finally, I would like to thank my beloved husband Şahin, for being with me always, for growing up together, loving me endlessly, and believing in me.

TABLE OF CONTENTS

ABSTRACT iv
ÖZ vi
ACKNOWLEDGMENTS ix
TABLE OF CONTENTS x
LIST OF TABLES xii
LIST OF FIGURES xiv
LIST OF ABBREVIATIONS xv
CHAPTER
1 INTRODUCTION 1
1.1 Background of the Problem 2
1.2 Motivation 3
1.3 Objectives and Scope 4
1.4 Research Strategy 4
1.5 Organization of the Thesis 6
2 BACKGROUND INFORMATION 9
2.1 Technical Debt 9
2.1.1 Level 1 Categorization – Steve McConnell 9
2.1.2 Level 2 Categorization – Martin Fowler 10
2.1.3 Level 3 Categorization – Alves et al. 11
2.2 Cloud Computing and Software as a Service - SaaS 12
2.3 Customer Relationship Management – CRM 14
2.4 Related Work on Technical Debt Analysis on Enterprise Level Solutions 16
3 CASE STUDY ON ANONYMOUS ISSUE STATEMENTS OF A CRM PLATFORM 17
3.1 Design of the Study 18
3.2 Conduct of the Study 20
3.2.1 Categorization 20
3.2.2 Validation 22
3.3 Findings of the Study 23

3.3.1 Categorization Results 23
3.3.2 Relationship Between Categorization Methods 26
3.3.3 Inferences on Level 3 Categorization – "Nature" of the TD 26
3.3.4 Evaluation of the First Case Study 27
4 CASE STUDY ON TECHNICAL DEBT AND SDLC PROCESSES WITH A SOFTWARE DEVELOPMENT TEAM 29
4.1 Design of the Study 31
4.1.1 Software Life Cycle Processes 32
4.1.2 TD Management 33
4.1.3 Coding – Categorization Practice 33
4.2 Conduct of the Study 34
4.2.1 Organizational Structure and Software Team Experience 34
4.2.2 Software Life Cycle Processes and Level 3 TD Mapping 35
4.2.3 SDLC Process Involvement 36
4.2.4 TD Awareness 38
4.2.5 Coding and Categorization 39
4.3 Findings of the Study 41
4.3.1 TD Indicators and TD Problems 41
4.3.2 TD Categories and Analysis 43
4.3.3 SaaS Related Limitations 45
5 RESULTS AND DISCUSSION 47
6 CONCLUSION 55
REFERENCES 57
APPENDIX A 61
APPENDIX B 62
APPENDIX C 65
APPENDIX D 67

LIST OF TABLES

TABLES

Table 1- The technical debt quadrant of Fowler 10
Table 2 - Technical debt types with definitions by Alves et al. 11
Table 3 - Validator Expertise in Software Development and SaaS Platform based on years of experience 20
Table 4 - Example categorization of confessions on Level 1 20
Table 5 - Example categorization of confessions on Level 2 21
Table 6 - Example categorization of confessions on Level 3 22
Table 7 - Matching Entries for each level of categorization 23
Table 8 – Number of entries in each technical debt category 24
Table 9 – Mapping of Level 1 and Level 2 categories 25
Table 10 – Total number of entries in Level 1 vs Level 2 categorization when Level 3 category is Design 25
Table 11 - Total number of entries in Level 1 vs Level 2 categorization when Level 3 category is People 26
Table 12 - ISO/IEC 12207 processes utilized in the case study 32
Table 13 - TD Management Options 33
Table 14 - TD Level 3 vs ISO/IEC 12207 Processes 35
Table 15 - SDLC processes practiced in the organization and the number of participants taking responsibility in each process 36
Table 16 – SDLC process involvement for each interviewee 37
Table 17 – Experience Level and TD Awareness for each interviewee 39
Table 18 – Coding and Categorization Process for Case Study 2 – Participant #4. 40
Table 19 - TD indicators 41
Table 20 - TD problems 42
Table 21 – Categories found by coding 43
Table 22 – Examples related to SaaS Related Limitations 45

Table 23 – TD issues experienced in SaaS applications vs TD types..... 48
Table 24 – TD awareness based on roles..... 49

LIST OF FIGURES

FIGURES

Figure 1 - Steps of Research Strategy 6
Figure 2 – Data collection process for Case Study 1 18
Figure 3 - Data collection process for Case Study 2 31
Figure 4 – Categories mentioned the most by interview participants 44
Figure 5 – Sub-categories on SaaS related limitations 50

LIST OF ABBREVIATIONS

Cons.	: Consultant
CRM	: Customer Relationship Management
Customer Sup	: Customer Support
Dev.	: Developer
ERP	: Enterprise Resource Planning
IEC	: the International Electrotechnical Commission
ISO	: the International Organization for Standardization
ISV	: Independent Software Vendor
Org	: Organization
QA	: Quality Assurance
QA Eng.	: Quality Assurance Engineer
Release Mng.	: Release Manager
RQ	: Research Question
SaaS	: Software as a Service
SDLC	: Software Development Life Cycle
TD	: Technical Debt

CHAPTER 1

INTRODUCTION

Technical debt is the outcome of taking poor decisions or choosing easier or quicker paths for code delivery, resulting in refactoring processes and more effort by the developers in long term. Given that software companies are in competition, decisions and actions taken instantly during development process affect the overall software quality such as code, user experience, architecture, and design quality and consequently the product quality. However, in software processes, especially in service or maintenance improvements that need to be delivered quickly, technical debt can become inevitable. The reasons for this debt include pressure from customers, lack of communication, tests with low code coverage, the architecture of the software, or even missing documentation. One of the evident problems of this debt result in significant decreases in product quality or agility in delivery. Therefore, it is essential to measure technical debt, in terms of delivering high quality products and services. These problems are also common in platform-based solutions, increase with platform-specific boundaries and limitations, and lead to malfunctions in both software processes and the end product delivered to the customer.

Software as a service applications are commonly used in the software development, and prefered for their benefits on low setup and infrastructure costs, and their scalability. They take the effort of managing complex software or hardware systems from software development companies. Another widely used platform by businesses is Customer Relationship Management (CRM) platforms, that provide an improvable way of managing business by increasing collaborative work, presenting easy-to-use features such as reporting tools, and therefore elevating success. These improvements -either products or services- are often delivered by software companies that are specialized in platform-based solutions. Organizations using software as a service applications and customer relationship management platforms are also experiencing technical debt throughout their development activities. The purpose of this thesis is to specify and categorize the technical debt present in organizations using software as a service applications, particularly in customer relationship management (CRM) platforms.

1.1 Background of the Problem

In the literature, there are studies which compare different technical debt identification methods such as the study by Zazworka et al. where the essential properties of TD should be captured, to approach TD with tools instead of using manual approaches (2013). The study of Ramasubbu and Kemerer focuses on TD based on interdependencies between client and vendor activities for maintenance and presenting the implications for studies for managing TD in enterprise systems (2016). Codabux et al.'s study focuses on insights from practitioners for TD identification (2017); whereas Skourletopoulos et al.'s research is on a cost estimation approach to identify the budget constraints which causes TD (2014).

There are also case studies that compare technical debt management in different companies by Zazworka et al.(2013), Iuliia (2017), Alzaghoul and Bahsoon (2013) and Klinger et al. (2011).

Kruchten et al. summarizes the evolution of technical debt and identifies this as the technical debt landscape and proposes a solution for the organization of the problem (2012). One of the studies from Alves et al. suggests an ontology of technical debt terms, to organize the different types and indicators of TD, based on its "nature" where the activity of the development process execution is taken into consideration (2014).

There are also studies which aims to detect the problems and reasons regarding TD by running surveys with software practitioners and software developers. One of these studies by Falessi and Kazman, aim to detect the "worst smells", and their frequencies, and the possible causes defined as "worst reasons", such as "lack of knowledge" for code smells type of issues in software development (2021). Another study by Ramač et al. collects information using a survey study with software practitioners, aiming to find the causes for software development teams to incur TD in their projects and the effects of TD (2022). The study identifies several TD types by occurrence as: design, test, code, architecture, and documentation debt, and provides the TD awareness and familiarity of software practitioners to the concept, and reports on the significance of other effects on the familiarity.

Although the technical debt concept is widely applicable to any software development project, there is a limited understanding of the causes and effects of it on development efforts in software as a service applications and enterprise level software. Enterprise software definition covers customizable platforms such as Enterprise Resource Planning (ERP), Human Resources Management (HRM) and Customer Relationship Management (CRM). It is crucial to investigate and specify

technical debt in these systems as the success in business highly depends on effective usage of these systems in a well-organized way.

1.2 Motivation

CRM platforms provide a streamlined way of managing businesses by improving collaborative work, presenting easy-to-use features such as reporting tools, and therefore elevating success in businesses (Salesforce, 2020). These improvements - either on products or services- are often delivered by software companies that are specialized in platform-based solutions or in-house teams who are responsible for understanding the requirements of business users and building solutions. Accordingly, the quality of such systems contributes to the overall quality of business processes. Hence, in this study, we specifically focused on technical debt in a CRM platform (i.e. Salesforce). Salesforce is a software company specialized in CRM, containing applications focused on sales, marketing, customer service, and also product/application development, providing software as a service (SaaS) and platform as a service (PaaS) (Youseff et al. 2008). Since the platform serves as a service for many organizations and application development teams, experiencing technical debt is inevitable for most of the stakeholders.

In the first case study, we've tried to understand the effect of different categorization methods on analyzing the causes of technical debt on Salesforce platform. We have used different categorization methods to understand the causes of technical debt on Salesforce platform, and we have seen that some categories – Design debt, People debt and Process debt – are standing out. However, we have concluded that our findings on this case study falls short on reflecting the debt that is specific to CRM and SaaS platforms. As SaaS applications are being used around the globe and the platform has its own development tools, and governor limitations, the technical debt management is expected to be slightly different than handling technical debt on a regular software development company. Therefore finding different categories for specifying the technical debt present in such systems become the motivation of this study, with the urge to tackle this challenge which will benefit software development teams in the industry using SaaS/PaaS. As the second case study was designed and conducted, the need to find the application specific TD issues became one of the significant objectives of the study, and findings around the methods for managing TD in organizations using SaaS was examined based on practitioner's point of view and analyzed based on their experiences in software development and their roles in the organization.

1.3 Objectives and Scope

The objective of this study is to specify the causes of technical debt in CRM platforms that provide SaaS/PaaS by deriving different technical debt categories that will reveal the debt in a more understandable way. When the causes are more apparent to software development teams, the technical debt awareness will increase. According to Suryanarayana et al. "Awareness is the first step toward managing technical debt" (2014). If the awareness is high within a project team, then managing technical debt would be easier, by identifying the causes and/or impacts of the debt.

The importance of knowing the products and services used in software development is clear, especially within the scope of CRM and SaaS applications and technical debt specification. However, difficulties in managing technical debt may arise due to difficulties in specifying this debt. For this reason, the scope of the study includes improvements for the specification of technical debt, by leveraging the categorization methods that are present today.

To improve the awareness, the aim is to specify the technical debt issues specific to CRM platforms and investigate the effectiveness of different technical debt categorization methods on analyzing the causes of the debt on SaaS applications. After analyzing the possible causes of technical debt in such applications, the purpose is to find out the methods for managing technical debt in organizations that develop software in SaaS platform and make an improvement in technical debt management for such organizations.

1.4 Research Strategy

As the purpose of the study is to specify and categorize the technical debt issues related to SaaS applications and CRM platforms, we have formulated the research questions of this study by targeting these applications.

RQ1: How do different technical debt categorization or identification methods help on analyzing the causes of technical debt in a Customer Relationship Management (CRM) platform?

RQ2: What are the technical debt issues in organizations using SaaS in CRM?
RQ3: What are possible problems caused due to TD in such organizations?
RQ4: What are the methods for managing TD in organizations using SaaS?

For the research methodology, a qualitative study on Salesforce platform was performed, by using 300 issue definitions, which were collected as confessions from OrgConfessions, which provides unbiased confessions of Salesforce developers, administrators, and consultants anonymously. The study was conducted using different technical debt categorization methods, which provide increasing level of detail. The categorization helped on the identification of the root causes of technical debt on Salesforce platform and therefore aims to increase the "awareness" among software development teams in CRM platforms. The details of the study are presented in chapter 3.3 in order to answer RQ1.

After conducting the case study it was understood that the categorization methods can be formed differently for SaaS applications. Therefore we have conducted a second study with a software development company, which specializes in Salesforce application development and services for customization. Multiple interviews were conducted with the software development team members with separate roles to understand the causes of technical debt in their organization, and how they've been managing the debt. The results of the second study created the opportunity to form new technical debt categories that addresses the problems that are common in software development companies using SaaS applications. The TD indicators, problems and managerial activities are introduced in chapter 4.3 and discussed further in chapter 5, to answer the rest of the research questions. The research strategy followed in this study is provided step by step in Figure 1.

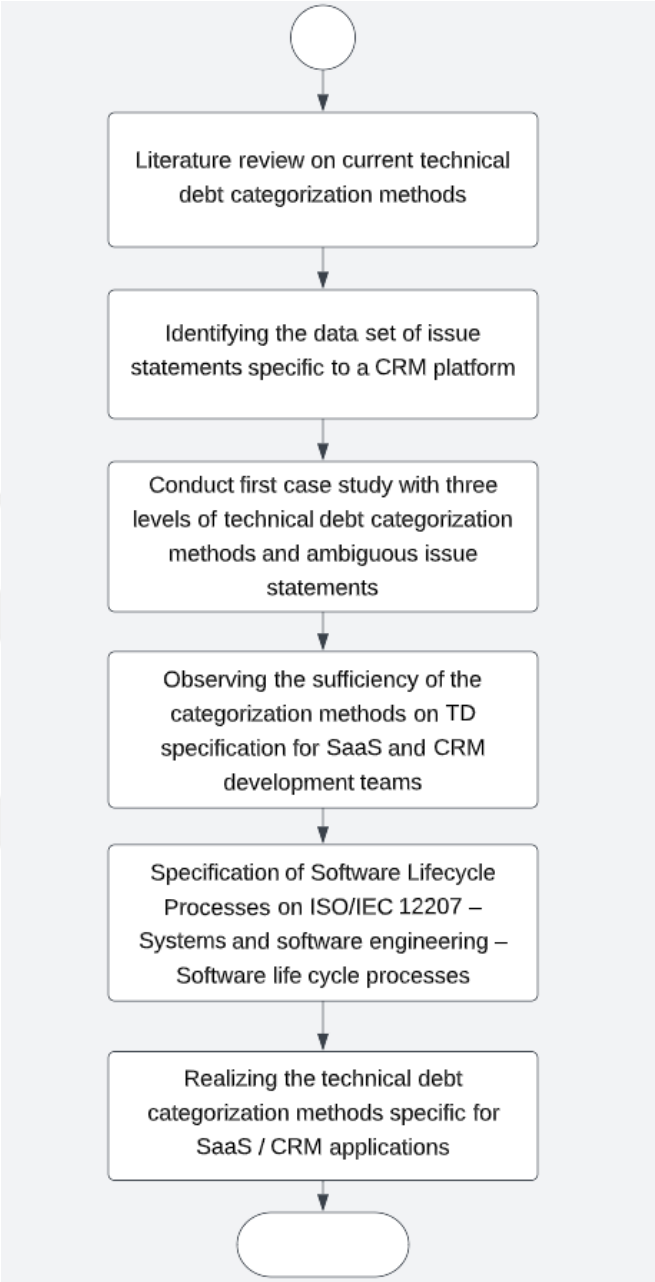

Figure 1 - Steps of Research Strategy

1.5 Organization of the Thesis

The overall study consists of six main sections. Chapter 2 is the literature review of studies on technical debt, and background information on CRM and SaaS applications. Chapter 3 is explaining the case study that was done on anonymous

issue statements of the CRM Platform. Chapter 4 presents the case study that was done with a CRM software development team using SaaS applications. Chapter 5 presents the results and discussion section. Chapter 6 concludes the thesis with findings, the significance of the study and future work.

CHAPTER 2

BACKGROUND INFORMATION

2.1 Technical Debt

The term “technical debt” was created as a metaphor by Ward Cunningham (1992), to describe all the code defects, and design and architectural mistakes made by developers, and to summarize them to “non-technical” people in software projects. Technical debt occurs when an instantaneous action adds value to the software but leads to undesirable consequences. In other words, taking shortcuts during analysis, design, implementation, testing or even documentation phases of a project might end up in more effort and time spent on the tasks to resolve a defect or to enhance the quality of the end product.

Several technical debt identification methods are suggested by researchers in the literature and comparisons of these different methods are made since Cunningham’s introduction by Zazworka et al. (2013), Ramasubbu and Kemerer (2016), Codabux et al (2017), and Skourletopoulos et al.(2014), Alves et al.(2014), McConnell (2013), and Fowler (2009). Among these, we selected Steve McConnell’s, Martin Fowler’s, and Alves et al.'s approach for technical debt categorization. We ordered these three approaches as follows according to the increasing level of granularity of their descriptions and applied them to our context in this order.

2.1.1 Level 1 Categorization – Steve McConnell

The first level technical debt identification can be performed on the “intention” level as suggested by Steve McConnell (2013). Each technical debt can be categorized as “intentional” and “unintentional”. In software projects, technical debts are usually unintentional, when imperfect solutions are preferred unconsciously. We discovered that most of the architectural or structural debts fall into this category, since the consequences of the architectural/design decisions could usually be observed at later stages of a software development life cycle.

On the other hand, almost all suboptimal solutions preferred to address the needs of customers or stakeholders intentionally, lead to low product quality alongside. Such solutions may be developed to solve design or code defects which blocks the software. Consequences of these actions are mostly known by development teams at the time of the actions, and mostly marked as "to be refactored" in the following development iterations. Therefore, such cases are considered as intentional technical debt.

2.1.2 Level 2 Categorization – Martin Fowler

The second level of technical debt categorization we used is the "Technical Debt Quadrant" developed by Martin Fowler (2009). This quadrant classifies the technical debt based on the intention of the person who creates the debt but in a more detailed way than McConnell's classification. The classification quadrant shown in Table 1 includes both the classification types and the examples for each classification given by F**owler.**

Table 1- The technical debt quadrant of Fowler

	Reckless	Prudent
Deliberate	We don't have time for design	We must ship now and deal with consequences
Inadvertent	What's layering?	Now we know how we should have done it

Below, we provide the explanations of the examples given above:

1. **Reckless – Inadvertent*:*** "*What's layering?".* This example refers to the lack of knowledge on good design practices and capability of practicing them as a team of developers. This kind of technical debt is the least desirable one and usually not recognized.

2. **Reckless – Deliberate*:*** "*We don't have time for design"*. This example may refer to project planning issues and not meeting deadlines, the state of not affording the time required to come up with clean solutions. Quick solutions without proper design, causing long-term defects are considered mainly in this category.

3. **Prudent – Deliberate*:*** "*We must ship now and deal with consequences*". This example refers to meeting certain deadlines with quick and low quality solutions, but accepting the debt, where the cost of paying it is recognized.

4. **Prudent – Inadvertent:** "*Now we know how we should have done it*". This example refers to a state where the code or design had been clean but realizing that it could have been designed better to meet the requirements. The debts in this category can be seen as learning opportunities to provide higher quality on upcoming development cycles or efforts.

2.1.3 Level 3 Categorization – Alves et al.

Aside from the intention and carefulness aspects of technical debt– which were covered by McConnell's and Fowler's approaches respectively – a more comprehensive taxonomy was formed by Alves et al. (2014). They defined an ontology for the "nature" of the debt, by considering the activities of the development process where the debt occurs. They identified 13 different technical debt types which were correlated with the activity of the development process execution:

Table 2 - Technical debt types with definitions by Alves et al.

Technical Debt Types – Level 3	**Definitions**
Architecture Debt	Refers to the problems encountered in product architecture, such as, violation of modularity.
Build Debt	Refers to issues that make the build task harder, and unnecessarily time consuming.
Code Debt	Refers to the problems found in the source code that can negatively affect the legibility of the code making it more difficult to maintain.
Defect Debt	Refers to known defects, usually identified by testing activities or by the user and reported on bug tracking systems.
Design Debt	Refers to debt that can be discovered by analyzing the source code and identifying violations of the principles of good object-oriented design.

Table 2 (cont.)

Documentation Debt	Refers to the problems found in software project documentation and can be identified by looking for missing, inadequate, or incomplete documentation of any type.
Infrastructure Debt	Refers to infrastructure issues that can delay or hinder some development activities.
People Debt	Refers to people issues that can delay or hinder some development activities.
Process Debt	Refers to inefficient processes, e.g. what the process was designed to handle may be no longer appropriate.
Requirement Debt	Refers to tradeoffs made with respect to what requirements the development team needs to implement or how to implement them.
Service Debt	Refers to the inappropriate selection and substitution of web services that lead to mismatch of the service features and applications' requirements.
Test Automation Debt	Refers to the work involved in automating tests of previously developed functionality to support continuous integration and faster development cycles.
Test Debt	Refers to issues found in testing activities that can affect the quality of those activities.
Usability Debt	Refers to inappropriate usability decisions that will need to be adjusted later.
Versioning Debt	Refers to problems in source code versioning, such as unnecessary code forks.

This third level categorization creates the deepest level of understanding on identifying technical debt as the nature of the debt is considered in the categorization.

2.2 Cloud Computing and Software as a Service - SaaS

Software as a service, also known as on-demand software, is a way to deliver applications and software over the Internet, as a service. Unlike on-premise software, which usually requires installation and maintenance by users, software as a service allows users to access the required architecture and resources via the Internet. SaaS solutions are usually preferred because of their lower upfront costs when compared to traditional or on-premise software (Cusumano, 2010).

SaaS applications use a multitenant architecture, where all applications and users share a single infrastructure and code base. The faster/easier innovation of SaaS applications is through multitenancy. As the software is delivered through the Internet, the ability for each user to access data is ensured. Most SaaS solutions provide no-code solutions, saving both from development time, and the cost of maintaining a code base. Allowing no-code automation and customization, it causes no harmful effects to the common infrastructure, and organizations can achieve success in their business processes.

As SaaS is usually prefered by organizations that hold back due to costs and the time spent to evaluate, analyze, install on-premise software, SaaS applications are usually compared to packaged software. Main differences of these can be defined as:

- Setup and infrastructure costs
- Scalability
- Rapidness in implementation
- Accessibility
- Update frequencies
- Maintenance costs
- Security

One of these main differences are often pointed out as ease of customization with SaaS applications, usually with no-code tools. The changes that can be made without writing code is playing a huge role in the automation and maintenance of business processes for organizations. While making the changes users use the advantage of using SaaS solutions, with the benefit of documented and ready-to-use solutions for their use cases if the service provider offers the specifications of the platform, software or service. In order to undertake all such improvements and maintenance work, organizations can employ in-house teams or administrators, as well as depending on external support, or even independent software vendors (ISV) (Mazalon, 2021).

At this point, it is also important that the developments that are made works as a complete system, within the limits of the provided service, depending on the architecture, and being compatible with the infrastructure. Also, the customizations made by external software providers, ISVs or in-house teams, should be well documented. Services and/or customizations that are not well documented, joined with less-experienced administrators or developers starts causing troubles in the SaaS applications. These problems can usually be similar to the problems experienced in on-premise services, however they can also be related to the SaaS application, the infrastructure, or may be introduced with continuous updates to the

service (Agarwal, 2011). Most SaaS applications provide solutions to such problems with bug-fixes on their release cycles, some may offer workarounds for their known issues as well. Either way, it is important for SaaS users to follow the updates from SaaS provider (Liu et al. 2011), and document their customizations and business processes. Otherwise, it is possible to say that technical debt becomes a problem in the field of SaaS applications due to such problems.

2.3 Customer Relationship Management – CRM

Being one of the pillars of Enterprise Resource Planning (ERP) systems, Customer Relationship Management (CRM) system is the complete system of services required to manage a business' interactions with its customers by organizing activities such as sales and marketing by leveraging different tools of automation and handling the life cycle of a customer with the organization wherever required. The transactions with the customers are traced with CRM systems, therefore it is beneficial to use them to maintain a long-term relationship with the customers (Tohidi and Jabbari, 2012).

The CRM as we know started in 1980s, with the name of ACT!, by Pat Sullivan and Mike Muhney (Bryant, 2018). It was founded to provide storage and organization for customer life cycle information in an effective way (Rigby and Ledingham, 2004). As CRM systems are developing in agility in the last 15 years, each day more CRM software is introduced. Some of the CRM tools are exceling at being easy-to-use, or better at managing sales teams or online transactions, some of them are preferred for being highly customizable. There are many types of CRM tools which are famous for different capabilities. Some of the most popular and highly-used CRM tools are:

- Microsoft CRM
- Salesforce CRM
- Oracle CRM
- SAP CRM (Ascendix, 2022)

As businesses has different needs and requirements, it becomes harder to choose the correct program for the organization. There are common items that are present in most CRM tools which satisfies the following needs of organizations:

- Ease of use
- Sales management
- Reporting
- Customization
- Integration
- Automation

As there are different features introduced in different CRM tools, there are different platform-specific boundaries and limitations that may lead to problems for organizations, resulting in poor business quality. When the CRM platform gets more customizable, more complex solutions to user requirements are to be produced. This usually requires a dedicated team to provide maintenance for organizations. Furthermore, lots of issues for customers are introduced as some features are constantly evolving with platform updates.

Widely used by businesses, the solutions that are created on CRM platforms are usually produced by in-house teams, or software companies that are specialized in platform-based solutions.

Being one of the world's leading cloud-based CRM platform/software, Salesforce, provides a highly customizable platform, with capabilities such as reporting and dashboard creation, automation with non-coding tools such as process builders, and flows, assisting in forecasting and additional AI solutions. These capabilities are provided with the cost of a complex environment that needs dedicated teams or admins to handle such costly customizations.

As some of these customizations can be made by in-house teams, there are also solution providers that work on products that can be easily installed using Salesforce's marketplace, AppExchange. Although Salesforce recommends using non-coding tools wherever possible, it is sometimes inevitable to write code for specific cases or even install the applications on AppExchange for requirements that will cost more to implement. All these features and alternatives makes Salesforce a complex system, which causes technical debt to be introduced, by both organization admins, and/or the solution providers of different applications.

2.4 Related Work on Technical Debt Analysis on Enterprise Level Solutions

Although the technical debt concept has been extensively studied in the literature, there are few studies in relation to analysis of technical debt for enterprise software systems, especially customer relationship management software. Klinger et al. (2011) analyzed the technical debt by conducting interviews with technical architects related to enterprise level solutions and made recommendations for organizations to manage technical debt with enterprise-level circumstances (Klinger et al, 2011). Another study in this area is on managing technical debt using an option-based approach for cloud-based solutions, where each option's ability to clear technical debt is analyzed (Alzaghoul and Bahsoon, 2013). There is also a study which focuses on the dependencies between client – vendor maintenance activities in enterprise level software systems and empirically quantifying "the negative impact of technical debt on enterprise system reliability" (Ramassubbu and Kemerer, 2016).

There is also a study made by Kumar et al. (2019), for identifying and estimating the technical debt for service composition in SaaS cloud, which aims to propose a way to help the decision making process for managing TD based on estimates for future debt. As in this study, Kumar puts emphasis on the concepts of "good debt" and "bad debt" in terms of classification of TD in the context of SaaS applications (Kumar, 2021).

CHAPTER 3

CASE STUDY ON ANONYMOUS ISSUE STATEMENTS OF A CRM PLATFORM[1]

The first case study was conducted on anonymous issue statements about Salesforce confessed by practitioners on OrgConfessions, and the aim was to use three levels of categorization methods, to understand the intentions, indicators, and the causes of technical debt on CRM platforms. Figure 2 presents the flow of the first case study based on the design and conduct processes and the analysis with respect to research questions.

As it can be understood from the three main sections in the figure, the case study consists of three main stages: Design, Conduct and Analysis. In the study, which is carried out by a certified platform developer, TD categorization methods in the literature were examined and 3 categories at different levels of granularity were determined as the initial step of the design stage. Afterwards, research was conducted on the issues in SaaS/CRM platforms, and the anonymous issue set was found which was defined as "confessions" from platform users having different roles in the business. A sample set which is 50% of the data that was available at the time of the study was selected. The design stage was concluded with the determination of the experts that will contribute to the study for reviewing the categorizations for the issues. The experts were selected in terms of experience of the platform and experience in software development areas.

In the conduct stage, the sample set of issues were categorized based on the three different technical debt categorization methods that were determined. When the categorization was completed, the experts categorized a smaller set of issues, and their categorization was used for validating the study using majority voting method. The details of the validation can be found in section 3.2.2.

[1] The following section of the thesis was published as conference proceeding on SERP'20 – The 18th Int'l Conference on Software Engineering Research and Practice – American Council on Science and Education. (Doğancı, Özcan-Top & Koçyiğit, 2021)

In analysis stage, the similarities between the two levels of TD categorization were stated, and the most frequent categories of all three levels were found for the SaaS/CRM platform. The effectiveness of the categorization methods in identifying the causes of TD has been understood in the analysis stage as an answer to the first research question of the thesis.

Details of these stages are presented in the following sections of the study.

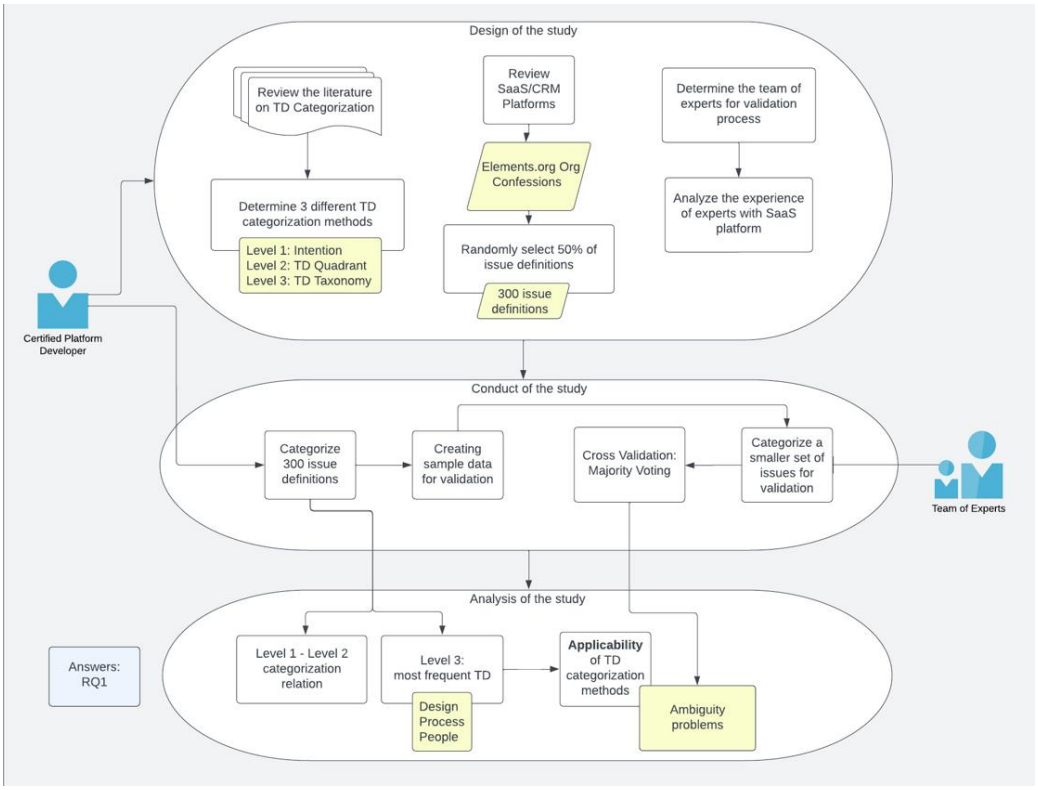

Figure 2 – Data collection process for Case Study 1

3.1 Design of the Study

The study is based on issue statements published as anonymous confessions by an Independent Software Vendor (Confessions.org). These issue statements are written by developers, administrators, consultants, and users working on the Salesforce platform. As there are different stakeholders in any Salesforce organization, the confessions of these stakeholders, their daily routines, the mistakes they've made over the course of their projects or customizations, the problems that they experience are also varying.

As of writing this thesis, there are currently more than 1200 entries published. During the design and conduct of the study, there were 700 entries published. The entries are not always written in-detail and the confessors are usually using an informal voice. Moreover, they used the domain knowledge to express the issues succinctly. Hence, many of the confessions are ambiguous and cannot be categorized without knowing the context and the nature of the pertinent implementation. Therefore, the decision was made for categorizing almost half of the entries published for the design of the study. The total number of entries analyzed in this study is covering approximately the 25% of the current total.

Technical debt categorization was carried out by a certified Salesforce platform developer, having more than four years of experience in product development and consultancy in the platform. The confessions were gathered randomly from the OrgConfessions site, and the identification numbers were also collected.

The categorization process mainly included analysis of the issue definitions on OrgConfessions, and determination of relevant technical debt categories based on three categorization approaches for each entry. The technical debt categorizations were described in the Technical Debt chapter of the thesis, giving background information about the three different levels of categorization. The issue definitions that were gathered were categorized in three iterations for the three different categorization methods. The first iteration was made with the first level of categorization which was covering the intention aspect, the second iteration was for the level 2 categorization which was the carefulness, and the third level of categorization was on the nature of the debt, which was the final iteration for categorization process. The details of each iteration and example categorizations are provided in the following chapter of the study.

The team of experts that would take part in the validation process of the study were also determined in the design of the study, based on their experience with software development and their familiarity with the SaaS application. The team of experts consist of developers that work in a Salesforce consultancy and product development team. Four of these people work as software developers, whereas one of them works as a tester/quality assurance specialist. These experts have varying experiences in both software development and the Salesforce platform. Hence, to minimize the effect of validator experience on analysis, we employed cross validation. To this end, the experts with less than 1 year of experience on the Salesforce platform but having nearly 10 years of experience in software development; and those having around 2 years of experience both in Salesforce and software development evaluated the same set of confessions. The level of experience is given in years and the experiences on

the platform and in software development for each validator is overlapping, meaning that a validator's experience in the software development was developing during working with SaaS platform.

Table 3 - Validator Expertise in Software Development and SaaS Platform based on years of experience

Validator Role	Experience in Software Development (in years)	Experience in SaaS Platform (in years)
QA Engineer	3	2.5
Technical Lead	11	Less than a year
Software Developer	1	1
Software Developer	2	2
Software Developer	15	12
SaaS Consultant	10	15

3.2 Conduct of the Study

In this section examples for three different levels of categorization, and the details of the validation process will be presented.

3.2.1 Categorization

At the first level of detail, the entries are evaluated from the **intention** aspect to see whether the developers foresaw the consequences when they made an inappropriate decision (intentionally) or not (unintentionally). Example confessions for these categories are as follows:

Table 4 - Example categorization of confessions on Level 1

Confession #	**Confession Content**	**TD Category on Level 1**
#231	People refuse to do changes anywhere but in production, since there "is no real test data	Intentional

Table 4 (cont.)

#129	We installed a managed package that added a currency field onto every standard AND custom object.	Unintentional

For the second level of categorization, the intentions of the decision maker who eventually caused the debt was the criteria on evaluating the entries. These intentions are found by using the "Technical Debt Quadrant" introduced by Fowler. Each category in the quadrant can be identified as *Reckless* (R), *Prudent (P)*, *Inadvertent (I)*, and *Deliberate (D)*. The union of these categories constitute the technical debt quadrant. The example entries falling in each quadrant section are as follows:

Table 5 - Example categorization of confessions on Level 2

Confession #	**Confession Content**	**TD Category on Level 2**
#201	Users doing a Trailhead course and started installing apps in our prod org vs their own playground.	Reckless-Inadvertent
#198	35 users logging in with shared username/password that had System Admin access.	Reckless-Deliberate
#394	Validation rule to stop one spammer (hardcoded email address) from creating email-to-case.	Prudent-Deliberate
#307	Invited to new Chatter group called "ISV-ABC." We all ignored it because it looked like a test. ABC actually stands for a territory: AMER – BUILD – CENTRAL.	Prudent-Inadvertent

The categorization at the third level was performed by considering the software development process during which the debt injected. This evaluation is based on Alves et al.'s (2014) 13 different categories. A sample set of confessions for these categories are as follows:

Table 6 - Example categorization of confessions on Level 3

Confession #	**Confession Content**	**TD Category on Level 2**
#74	40 lookup fields on the Product object.	Design
#84	Request for a multi-select picklist with 98 values. When advised this was not the best practice and to rethink the need, they came back with a request for a picklist with 78 values.	People
#164	Sandbox not updated 10 years.	Process
#147	Multiple fields with same label.	Documentation
#39	Hard coded user names in Apex classes.	Code

3.2.2 Validation

The example confessions can be seen in Appendix A and the evaluation form attached is Appendix B. Forty-five entries were chosen randomly from the set of categorized entries (confessions). This refers to validation of the 15% of the categorizations given by the researcher. As the confessions are vaguely stated, the experts' categorizations were not always consistent. For this reason, we employed the majority voting method in deciding on the correct technical debt categories for each categorization level. The categories found by the researcher were compared with the ones found by validators and in the case of a tie, the researcher's categorization was assumed to be valid.

In the validation phase of this study, the focus was also on understanding whether different roles or experiences in software development or the Salesforce platform influenced the categorization process. Table 7 shows that most of the Salesforce experts were in consensus in categorization of the confessions (91% at Level 1, 77% at Level 2 and 75% at Level 3). However, due to ambiguities in the issue statements, and the existence of non-mutually exclusive categories (e.g. Design and Architecture) in technical debt categorization at Level 3, we concluded that the categorization levels can be revised to consider the ambiguous or unknown sourced debt issues.

Table 7 - Matching Entries for each level of categorization

Categorization Level	Number of Matching Entries	Percentage of Matching Entries
1	41/45	91%
2	35/45	77%
3	34/45	75%

As can be seen in Table 7 out of validated 45 entries, the percentage of matching entries are decreasing with increasing level of categorization. This shows that when the technical debt categories' granularity increases, the consistency of the decisions made in categorization decreases.

One reason for this decrease at Level 3 is that there is no clear distinction among the categories of this level. For example, in some cases a debt categorized as "Design" could go under "Architecture" or "Code" or even "Requirement" categories as well. Another cause of this decrease is the ambiguities in issue definitions. Since the actual process in the software development life cycle where the debt had been introduced was unknown to researchers and the group of validators, making decisions at Level 3 was more difficult than the other two technical debt categorization levels.

3.3 Findings of the Study

In this section, the results of the three level technical debt categorization of OrgConfession issues are introduced. The categorization results of 300 issue definitions are delivered on the first section of the chapter. Afterwards, the correlation between these categories and the reasons for disagreements in the validation method are specified.

3.3.1 Categorization Results

The categorization of 300 confessions and the results are summarized in Table 8 below**.**

Table 8 – Number of entries in each technical debt category

Level	Technical Debt Type	Number of Entries
1	Intentional	219
	Unintentional	81
2	Reckless-Deliberate	202
	Reckless-Inadvertent	66
	Prudent-Deliberate	11
	Prudent-Inadvertent	21
3	Design	138
	People	51
	Process	41
	Documentation	24
	Code	16
	Test	13
	Requirement	5
	Infrastructure	5
	Service	4
	Architecture	3

Table 8 shows that, out of 300 entries analyzed, most of the issues are classified as Intentional at Level 1 and as reckless – deliberate at Level 2.

Mapping of the categories at Level 1 and Level 2 as shown in
Table 9 revealed that most of the intentional type of technical debt corresponds to reckless-deliberate type debt according to Level 2 categorization and most of the unintentional type of technical debt corresponds to the reckless-inadvertent type debt at Level 2.

Table 9 – Mapping of Level 1 and Level 2 categories

		Level 2				
		R.D	R.I	P.I	P.D	
Level 1	Intentional	200	6	2	11	219
	Unintentional	2	60	19	0	81

As it can be seen in Table 8, “Design” and “People” type categories are the most frequent ones at Level 3. Hence, in Table 10 and Table 11, we present a breakdown of the “Design” and “People” category confessions for Level 1 and Level 2. As shown in Table 10, most of the “Design” type technical debt are categorized as intentional (113) and reckless-deliberate (103). On the other hand, most of the “People” type technical debt are categorized as intentional (27) and reckless-deliberate (25) as well. Additionally, the second frequent category for the “People” technical debt was unintentional and reckless-inadvertent. Hence, we can say that most of the “People” type technical debt can also be classified as a reckless technical debt.

Table 10 – Total number of entries in Level 1 vs Level 2 categorization when Level 3 category is Design

Detailed results for Level 3 = Design	Level 2					
		R.D	R.I	P.I	P.D	
Level 1	Intentional	103	3	2	5	113
	Unintentional	1	19	5	0	25

Table 11 - Total number of entries in Level 1 vs Level 2 categorization when Level 3 category is People

Detailed Results for Level 3 = People	Level 2					
		R.D	R.I	P.I	P.D	
Level 1	Intentional	25	1	0	1	27
	Unintentional	0	19	5	0	24

3.3.2 Relationship Between Categorization Methods

Within the analysis of the results and the number of entries categorized at these three different levels, we observed that the first two levels of technical debt categorization are strongly correlated to each other. The most likely cause of this correlation is that the intention aspect is also covered by the Level 2 categorization defined by Fowler in the Technical Debt Quadrant. Fowler states that "…the moment you realize what the design should have been, you also realize that you have an inadvertent debt.", pointing out that the correlation we mentioned between Level 1 and Level 2 categorizations is valid. By nature, the unintentional debt cannot be known until the moment it is realized, and Fowler's statement supports that. This also explains the same case for the inadvertent debt category.

3.3.3 Inferences on Level 3 Categorization – "Nature" of the TD

After evaluating the first two levels of categorization and the results, the third and most detailed categorization results are discussed for the top three categories: "Design", "People" and "Process".

The "Design" type technical debt in the Salesforce platform can be attributed to requirements errors as well. It is not possible to distinguish design and requirement type errors due to ambiguity in issue definitions. Therefore, a design issue may also suggest an inefficiency in requirement elicitation and specification processes (such as requirements are not well defined or analyzed enough for a specific business need which leads to incorrect solutions in the Salesforce platform). Some of the design issues may be related to replicating an already existing third party package or

features in the Salesforce platform which ends up with a redundant development. Similarly, instead of using built-in components which already exist in the Salesforce platform, introducing new custom-made components or solutions that satisfy the same set of requirements are classified as “Design” debt.

We observed that the “People” type technical debt category is strongly related to faults caused by human errors and lack of user training. The technical debt introduced in this category are also linked to communication errors or lack of communication. People working on the Salesforce platform with different roles are implicitly or explicitly mentioned in the confessions. The following are the entries for representing the different roles causing the “People” type technical debt at Level 3:

- “We can’t move to Lightning because our Dev Team refuses to learn Javascript to write the Lightning components we need.” (confession #114) → The role of the person mentioned in this confession is Developer

- “10 year old Org. Admin didn’t know how customize nav bar. So they went in for each user and customized their tabs.” (confession #186) → The role of the person mentioned in this confession is Administrator

- “All managers insist on having a password that never expires.” (confession #216) → The role of the person mentioned in this confession is Businesspeople

With the study it is understood that the technical debt linked to “Process” type is strongly related to the methodology followed in the development and delivery processes. Process related debts are indicators of not following the software development best practices accepted by the Salesforce community (2020). (i.e. not using the suggested deployment connections for deployment purposes).

3.3.4 Evaluation of the First Case Study

As a result of the first case study based on anonymous and ambiguous issue statements, we were able to comprehend the benefits of different technical debt categorization methods in CRM platforms. We also detected the top three processes that gives some insight on the problems that are experienced heavily in Salesforce platform based on the third level of categorization. However, making these

inferences from ambiguous issue statements prevented us from understanding the root causes of these problems in SaaS applications.

As these evaluations and inferences are made based on predictions from experts, based on the knowledge on the platform, we wanted to verify the TD categorizations based on unambiguous problems that software development teams experience in CRM and SaaS applications. Also the capability of these TD categories to define the problems experienced in these platforms are still open to discussion after conducting the first case study. Therefore we started to design a second case study with developers working with such applications, and the design was based on the problems faced in a project that was created using these services, SaaS, and CRM.

CHAPTER 4

CASE STUDY ON TECHNICAL DEBT AND SDLC PROCESSES WITH A SOFTWARE DEVELOPMENT TEAM

The second case study was conducted to specify the problems related to technical debt in organizations using SaaS and CRM applications, by analyzing unambiguous problems that software organizations experience. As the evaluations made on the first case study was based on ambiguous issues, the second study's purpose was to create a deeper understanding of technical debt by eliminating the ambiguity and providing evidence on an actual project based on SaaS applications.

The main objective of the case study was to specify the relationship of organizations using SaaS and CRM applications and technical debt. The specification of technical debt, the related problems caused by debt, and different options for managing the debt are specified based on the second case study. As a result of the case study, we were able to determine various categories for specifying technical debt in organizations using SaaS applications.

The second case study, like the first, consisted of three main stages: Design, Conduct and Analysis. The main purposes of the second case study were to eliminate the ambiguity in the first case study – due to anonymous confessions – and to focus on the TD activities based on the third level of categorization – nature of the debt – which was related to SDLC activities. Another purpose was to focus on the managerial activities on TD.

In the design stage, the different purposes of the study were recapped, and different steps were carried out to fulfill each purpose. After the determination of an organization working with SaaS applications, it was decided that one-on-one interviews should be conducted with the software practitioners working in the organization, in order to reduce the ambiguity that was present in the first case study. As the practitioners were to discuss the problems that they experience in software development based on their current project, they were more precise about the root causes of some of these problems. While choosing the interviewees, the roles and experiences of the software practitioners were taken into consideration to provide in-detail analysis based on different levels of expertise. Further actions taken in the design stage was to conduct a literature study on TD management, and the categorization method on the nature of the debt. The nature level categorization – third level in the first case study – and the different categories were mapped with

ISO/IEC 12207 software life cycle processes to carry out the interviews based on a standard. In the light of the information gathered in the design stage, the interview questionnaire was created, and the design stage of the study was terminated. The interview questionnaire can be seen in Appendix C and D for English and Turkish versions respectively.

During the conduct stage, structured, one-one-one interviews were conducted with the software practitioners. During these interviews, the questionnaire prepared with open-ended questions was reviewed, and interviews were made based on the roles and responsibilities of the interviewee. During the interviews, the observations, and the information that each interviewee shared were recorded, and they were listed at the end of each interview by the interviewer. When all interviews were completed, the conduct phase of the study was terminated with the coding of the recorded findings.

As a result of the first analysis in the recorded interview findings, TD indicators and TD problems were determined from the perspective of software practitioners working with SaaS applications. Later, with the coding phase, the TD categories for SaaS were determined in line with these problems and indicators. 24 TD indicators, 10 TD problems, 9 TD categories were determined, answering the second and third research questions of the thesis. Along with these, the last research question of the thesis was answered by analyzing the managerial activities for TD mentioned by practitioners.

Figure 3 presents the flow of the second case study based on the design and conduct processes and the analysis with respect to research questions. Details of these stages are presented in the following sections of the study.

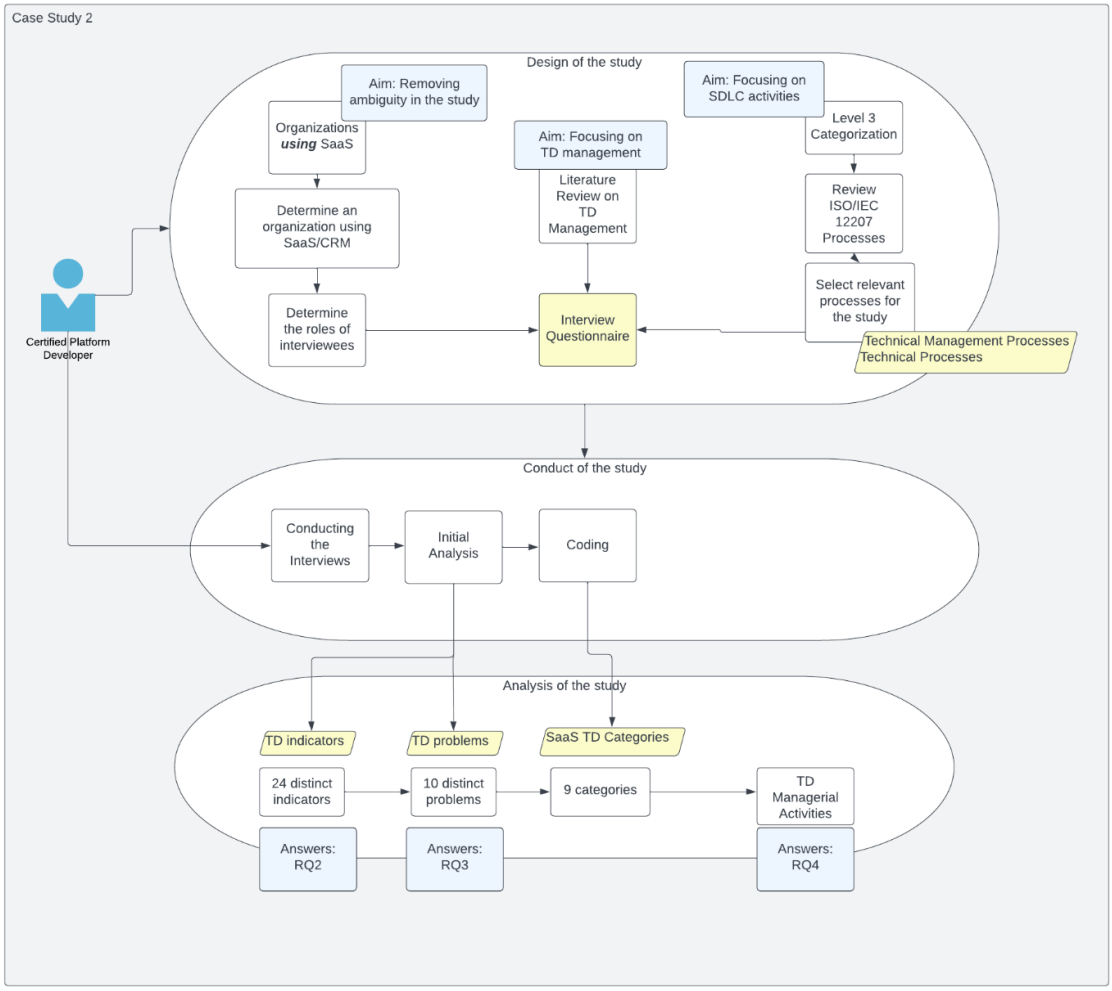

Figure 3 - Data collection process for Case Study 2

4.1 Design of the Study

As understood from the first case study in identifying TD, we concluded that Level 3 categorization has higher importance, when it comes to understanding the nature/root cause of the debt. This categorization provided a higher level of granularity, and the debt categories were similar to software development life cycle processes. However, the level 3 categories for TD can be manipulated specifically for SaaS and CRM applications for a better understanding of TD in these systems.

4.1.1 Software Life Cycle Processes

In order to understand the nature of the TD in SaaS applications, the first step was to design an interview questionnaire to illuminate the SDLC processes defined for software development teams working with SaaS applications. The main target of the interview questionnaire was the different technical debt issues experienced by people working in such organizations, having various experiences and roles in the project. Afterwards, the software development team that participates in similar SDLC activities in SaaS applications was detected.

To identify these roles and responsibilities of the participants, the ISO/IEC 12207 (ISO/IEC 12207, 2008) software life cycle process international standard was used. The standard was used to specify the SDLC activities and conduct the study with a solid foundation. Table 12 shows the processes that have been utilized in the study.

Table 12 - ISO/IEC 12207 processes utilized in the case study

Technical Management Processes	Project Planning Process	6.3.1
	Project Assessment and Control Process	6.3.2
	Configuration Management Process	6.3.5
Technical Processes	Stakeholder Needs and Requirements Definition Process	6.4.2
	Systems/Software Requirements Definition Process	6.4.3
	Architecture Definition Process	6.4.4
	Design Definition Process	6.4.5
	Implementation Process	6.4.7
	Integration Process	6.4.8
	Verification Process	6.4.9
	Transition Process	6.4.10
	Validation Process	6.4.11

After being informed about the structure of the organization and the level of experience of the participants, the main goal was to understand the degree of achievement on tasks and activities of each SDLC process defined. For this reason, we have designed some questions in the interview questionnaire to identify the roles and responsibilities of each participant, by specifying their titles and the SDLC processes that they are responsible from.

On the next steps, without mentioning about the TD metaphor, we created questions to point out the specify the problems they have been experiencing in the project, by also focusing on the SaaS aspect. After specifying the common problems

experienced in all software development projects, we tried pointing out the process related problems for each participant. Afterwards, the TD concept was introduced and the participants knowledge on TD was tried to be measured.

4.1.2 TD Management

After preparing the initial TD specification questions, we determined managerial options from different sources regarding technical debt management for organizations. The options are listed in Table 13. Questions related to management options are also included in the questionnaire, to specify the managerial activities that software practitioners follow on their project. We also created separate questions for understanding the participant's preferences regarding the management of TD specific to SaaS applications.

Table 13 - TD Management Options

TD Management Options	Definition
Recognizing TD	Identification of TD, conducting analysis on architecture, design, development and management artifacts
Making TD visible	Communication, Tracking, Involvement
Planning TD	Decision making on options: Prevention, Monitoring, Repayment, Prioritization of TD
Living with TD	Strategic TD, Intentions, Avoidance, Accepting TD

We tried to unify different TD management options that are accepted in TD management. We created questions for further understanding on participants' preferences and experiences on TD management (Ozkaya, 2021).

4.1.3 Coding – Categorization Practice

When the interviews were concluded, we evaluated the interview results by applying coding steps in the grounded theory method. The first step was open coding, where the interview records were broken down into discrete parts. On the second step the connections between these parts were defined. At the final step, the connections between the codes were made into categories, where we tried to capture the essence of the case study and tried finding out TD categorization for software development

using SaaS applications. The categorization and coding were made based on the level 3 categorization and the ISO/IEC 12207 processes.

In this way, the design of the case study was completed, and it can be summarized as follows:

1. Determining the software development team to conduct the case study, working with SaaS and CRM applications
2. Analyzing the results from first case study and finding out that third level of categorization can be enhanced
3. Identifying the ISO/IEC 12207 processes and reviewing each activity and task defined for the processes
4. Creating the first version of the interview questionnaire by combining both contexts on SDLC
5. Creating questions related to specifying problems in SaaS applications
6. Identifying TD management methods
7. Advancing questions in accordance with the management methods and options

4.2 Conduct of the Study

4.2.1 Organizational Structure and Software Team Experience

Based on the design for the case study, one of the important steps was to determine the software development company that the interview took place. When determining the software company, our expectation was for the company to fulfill some SDLC activities, such as analysis, design, implementation, and testing, to a certain level. Apart from this, it was important that there was an active project being worked on by the company. The most important characteristic of the company was to work with SaaS applications.

There were three separate teams working with SaaS applications in the company. However, one of the teams was dealing with product development and software services delivery, and all implementation effort was based on a CRM platform. The team can also be identified as an ISV in the specific CRM ecosystem.

The competencies, experiences, and the roles of the people in the team varied. Considering the experience level as years of study and developing in CRM platform, the variations were 1, 2, 3, 5, 10 years of experience. The titles of the people in the team were:

- Consultant
- Developer
- Quality Assurance (QA) Engineer
- Technical Team Lead
- Release/Project Manager
- Customer Support Agent

4.2.2 Software Life Cycle Processes and Level 3 TD Mapping

As a result of the first case study the technical debt categories were found to be mostly related with the Design, People, and Process categories. These categories can be paired with the following ISO/IEC 12207 processes in Table 14.

Table 14 - TD Level 3 vs ISO/IEC 12207 Processes

TD Categories for Level 3	Definitions	ISO/IEC 12207 Process
Design Debt	Refers to debt that can be discovered by analyzing the source code and identifying violations of the principles of good object-oriented design.	Architecture Definition Process Design Definition Process
People Debt	Refers to people issues that can delay or hinder some development activities.	Project Planning Process Project Assessment and Control Process

We excluded the process debt from this pairing, as the process debt refers to inefficient processes, or not maintaining the processes that cannot handle their responsibilities appropriately, and therefore accommodates all processes that are not fully achieved by the organization.

For the design debt, as it refers to the debt that can be discovered by identifying the violations of solid design principles, the pairing was made with architecture and design planning processes and their tasks and activities, where the purpose is to

develop a detailed architectural view and prepare a detailed design, including all software elements.

For the people debt, as it refers to issues related to people delaying the activities or create difficulties for development tasks, the pairing was made with respect to ISO/IEC 12207 technical planning processes. In these processes the plans for executing the project are determined and the roles and responsibilities for stakeholders are defined. Besides from these, the technical progress reviews are performed for achieving to project objectives. For the conduct of the study another area that people debt was used was on including the decision makings for the managerial activities on TD.

For each ISO/IEC 12207 process mentioned in the study, we've tried to summarize the tasks and activities defined in each process in SaaS applications context. We also considered this context during the grounded theory coding iterations.

4.2.3 SDLC Process Involvement

After the interview questionnaire was prepared, we conducted 45 minute interviews with each person in the software development team. The interview questions were asked in English and repeated in Turkish, and the rest of the interviews were held mostly in Turkish. The software development life cycle activities were recapped in English. Two people out of all interviewees preferred to conduct the interviews in English. Each person in the team was responsible for at least one activity or task in the given process list in ISO/IEC 12207. The SDLC processes experienced in the organization with respect to number of participants taking responsibilities for the respective process is shared in Table 15.

Table 15 - SDLC processes practiced in the organization and the number of participants taking responsibility in each process

ISO/IEC 12207 processes	Process involvement score based on participant count
Project Planning Process	3
Project Assessment and Control Process	1
Configuration Management Process	0
Stakeholder Needs and Requirements Definition Process	6

Table 15 (cont.)

Systems/Software Requirements Definition Process	5
Architecture Definition Process	3
Design Definition Process	5
Implementation Process	6
Integration Process	5
Verification Process	6
Transition Process	0
Validation Process	4

The processes for which the participant took responsibility were asked. To fulfill this purpose, the names of each process were listed to the participant. If the participant requested examples for any process, these examples were given by referring to the tasks and activities defined for these processes in ISO/IEC 12207 standard. The following table presents the SDLC process involvement for each participant of the interviews. The configuration management and transition processes are not presented in the table as there aren't any participant stating involvement in these processes.

Table 16 – SDLC process involvement for each interviewee

Interviewee #	1	2	3	4	5	6	7	8	9
Project Planning Process					X			X	X
Project Assessment and Control Process								X	
Stakeholder Needs and Requirements Definition Process			X	X	X		X	X	X
Systems/Software Requirements Definition Process			X	X	X	X	X		
Architecture Definition Process			X	X	X				

Table 16 (cont.)

Design Definition Process			X	X	X	X	X		
Implementation Process	X		X	X	X	X	X		
Integration Process	X		X	X	X	X			
Verification Process		X		X	X	X	X		X
Validation Process		X				X	X		X

4.2.4 TD Awareness

We chose not to discuss the technical debt concept in the first part of the interviews. First, we questioned the problems that participant had with each of these processes, for the tasks and activities that they participate in, without creating a biased opinion on TD. When the TD metaphor was introduced to the interviewees, 10% of the participants stated that they didn't know about this metaphor at all, even though they knew about the indicators of TD issues and experienced it in their project.

The following table presents the experience levels of participants based on years in software development and in SaaS organizations and their TD awareness. The TD Awareness score is given based on the scale below:

- **Low**: the participant doesn't know the TD metaphor completely but gives broad examples of TD throughout the interview OR knows the metaphor but doesn't know what is considered as TD in the processes that they are involved in.

- **Intermediate**: the participant knows the TD metaphor, gives examples not in detail, does not express impact on process or opinion on managing TD.

- **High**: the participant knows the TD metaphor, gives proper examples of TD, knows the indicators and problems caused by TD, expresses opinion to some extent to overcome/manage TD.

Table 17 – Experience Level and TD Awareness for each interviewee

#	Role	Experience Level in Software Development (in years)	Experience Level in SaaS (in years)	TD Awareness
1	Consultant	3	3	Low
2	QA Engineer	5	5	High
3	Developer	4	4	High
4	Developer	5	2	Intermediate
5	Team Lead	13	3	High
6	Developer	5	2.5	Low
7	Developer	3	3	Intermediate
8	Release Manager	2	2	Low
9	Customer Support Specialist	5	2.5	Intermediate

4.2.5 Coding and Categorization

After each interview was completed, the observations and answers of each interviewee was recorded. The interview data was coded into different sections with respect to level of detail and their relation to TD, where the options were: indicators, problems, managerial options initially. The coding of the interviews was evaluated using grounded theory coding. After the initial step, the connections between each code were formed based on the nature/process levels and the keywords that each participant mentioned throughout the interviews. The TD categories that are mentioned in the findings section are formed using this axial coding step, by eliminating and reforming some of the categories based on resemblance to one another. The following table presents the example coding and categorization process for an interview with participant #4.

Table 18 – Coding and Categorization Process for Case Study 2 – Participant #4

Sample Interview Data	R.Q.2: TD issues	R.Q.3: TD problems	Categories
… Sometimes when we can't speak to the customer directly… It is not our concern, but the requirements are not gathered completely... We may have to generalize our product for multiple customers, that prevents us from coming up with simple designs at some points... Environment setup and SaaS configurations takes time, and sometimes an issue is only reproducible on the customer's org. …the number of issues on the customer's end may be higher than our expectation... We may have to act based on changes on SaaS application, which ends in unexpected development effort… We record some of these issues but sometimes we have to do this ASAP...	Communication issues when we can't speak to customers directly	Incomplete architecture / work	Miscommunication
	Not understanding requirements completely	Decrease in product quality	Not following standards
	Overengineering	Testing takes forever	Time constraints
	Troubleshooting is difficult - SaaS	Spending more time to find workarounds to pass by the limits by SaaS provider	SaaS Related Limitations
	Designs not being approved by customers	Spending more time on analysis on problems	Design flaws
	Compatibility issues due to SaaS	Having to remove/deprecate a feature not working anymore	SaaS Related Limitations

The next section will state the findings of the case study, specifying the TD issues and problems, managerial preferences of the participants, based on the results of the coding processes.

4.3 Findings of the Study

As a result of this case study, we have examined the problems related to technical debt in the context of SaaS applications from the perspective of software development practitioners, that has been developing products and providing professional services in a CRM platform. After our analysis, the first findings that emerges from the interviews can be examined in the following contexts:

1. Technical debt indicators
2. Problems caused by technical debt
3. Technical debt management methods

4.3.1 TD Indicators and TD Problems

For the first and second contexts, the indicators and the problems caused by the technical debt, it was found out that most participants were affected by the design flaws present in their project, as well as realizing that they are not always following the best practices and standards that should be in a software development project. In the following tables some of these ideas and discourses are shared.

Table 19 - TD indicators

Non-informative issue descriptions
No test steps/reproduce steps defined for bugs found
No definition of stakeholder needs and requirements (sometimes)
Known issues on SaaS provider
Harder to replicate a production/live system on SaaS provider
No way to overrun the limits by SaaS provider
Quick fixes per customer request with suboptimal design / low performance code
Lack of information by possible impact on certain functionality by SaaS provider
No consensus with the customer on certain requirements / performance
Definition of done is not specified
Features being deprecated by SaaS provider
Using “hacky” methods (sometimes)
Unit tests with no assertion

Table 19 (cont.)

Not following well designed coding conventions
Time constraints
Refactor/rework cycles with developer initiative
Not following the changes happening on SaaS provider side
Incomplete design
Issues found by customers but not in verification processes
Failing in making features generic for all SaaS users / customers
Compatibility issues due to SaaS
Using partial solutions for some problems
Lack of technical support from SaaS provider
Lots of options for customization/configuration on SaaS application side

After evaluating these samples, it was understood that the TD indicators were present in the project, across different software development life cycle processes, such as design, validation, and implementation. It was also obvious that there were some indicators on SaaS applications side as well. When interviewed on the problems that was caused to experiencing these TD indicators, all the interviewees mentioned customer dissatisfaction and a decrease in their motivation. Besides from these, the technical and managerial problems are shared in the following table.

Table 20 - TD problems

Spending more time to find workarounds to pass by the limits by SaaS provider
Spending more time on analysis on problems
Troubleshooting takes much more time
Decrease in product quality
Cannot create new features due to constant rework cycles
Having to remove/deprecate a feature not working anymore
Spending more time on documentation in order to provide workarounds for customers

Table 20 (cont.)

Conflict on requirements and design
Issues usually not being fixed completely/perfectly
Testing takes forever

4.3.2 TD Categories and Analysis

Based on these ideas and samples, we have completed the coding process, and obtained some categories. You can see the explanation for each of these categories in Table 21.

Table 21 – Categories found by coding

#	Category	Explanations with examples
1	Miscommunication	Issues related to miscommunication between developers, other team members such as QA, customer success agent, etc. Also misunderstood requirements from customer side or upper management
2	Incomplete work	Design or architectural decisions not being completed, missing design definition and documentation, missing test steps and definition of done not being defined
3	Design flaws	Non-matching requirements and design, design not being applicable to SaaS application, design that misses some scenarios on SaaS usage and customization
4	Not following standards	Not following good coding conventions, creating highly coupled classes, not following best practices provided by SaaS application, not completing the unit testing, non-defined test steps
5	Time constraints	Deadlines for product releases, trying to keep up with
6	Testing failures	Non-descriptive test cases, impossible to define every usage scenario for each customer, no testing strategy for some features, not measuring the success/failure rate for tests, unit tests with no assertions
7	Finding workarounds	Using partial solutions, using “hacky” methods, quick fixes per customer request, not creating the perfect solution for a bug

Table 21 (cont.)

8	Unawareness	Not all team members have the same expertise level, delayed training on some features on SaaS application, not knowing all coding conventions that the seniors are following, developers only following the design that is provided, but not adding up to the design, checking the architectural requirements, etc.
9	SaaS Related Limitations	Features being deprecated by SaaS provider, lack of documentation on some features on SaaS application, lack of technical support on features that require technical expertise

After creating the categories, for the next step, an analysis was made on the categories and the interview results, to find the ones that were mentioned the most by the practitioners. We made this analysis by counting the indicators and problems referred the most by the interviewees.

The results showed that the top three categories were found are as follows:

1. SaaS Related Limitations
2. Not following standards
3. Design Flaws

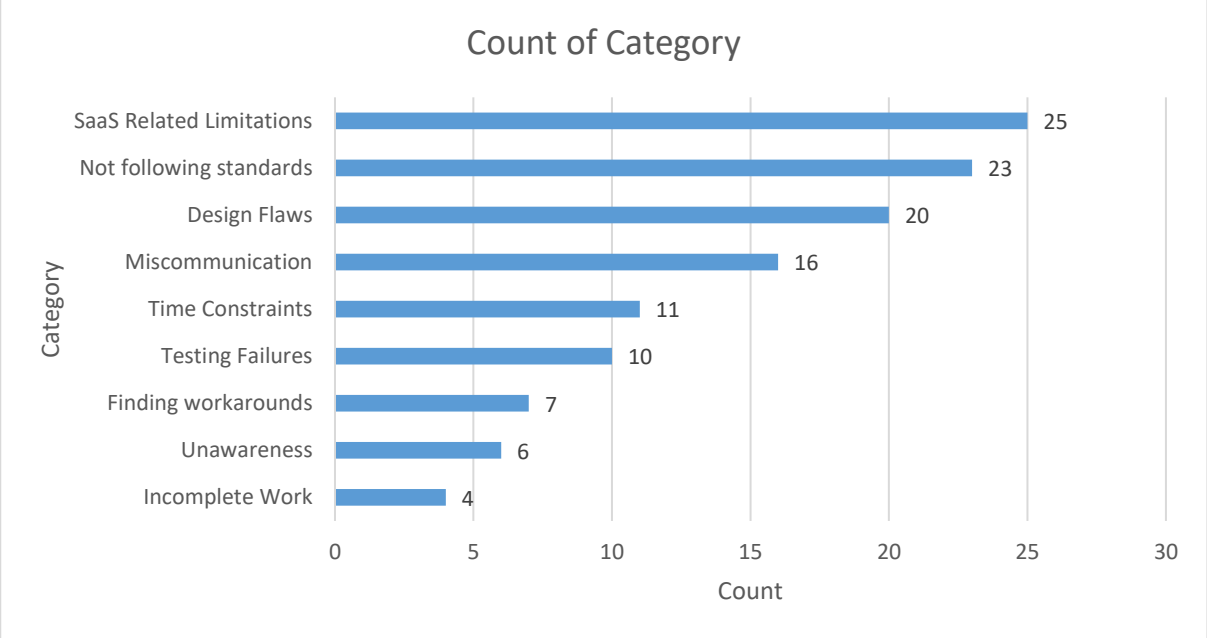

Figure 4 – Categories mentioned the most by interview participants

4.3.3 SaaS Related Limitations

To examine the first category – SaaS Related Limitations – which is a problem mentioned by every interview participant, we present the list all problems and indicators mentioned during the study in Table 22.

Table 22 – Examples related to SaaS Related Limitations

Known issues on SaaS provider
Harder to replicate a production/live system on SaaS provider
No way to overrun the limits by SaaS provider
Lack of information by possible impact on certain functionality by SaaS provider
Features being deprecated by SaaS provider
Not following the changes happening on SaaS provider side
Failing in making features generic for all SaaS users / customers
Compatibility issues due to SaaS
Lack of technical support from SaaS provider
Lots of options for customization/configuration on SaaS application side

If we review the results in this field, it is indicated that the problems experienced in the SaaS applications have their own sub-categories. These can be further discussed with respect to ISO/IEC 12207 processes. The discussion around the sub-categories will be introduced in detail in the next chapter of the thesis.

CHAPTER 5

RESULTS AND DISCUSSION

In this chapter the answers to the research questions introduced in the first chapter of thesis will be presented and will be discussed in the light of the multiple case studies conducted and described in chapters 3 and 4, respectively.

Three different categorization levels that were used in conducting the categorization help tremendously in analyzing the debt. The first two levels summarize the intentions and the behavior in terms of carefulness of people and organizations, which is helpful in detecting the debt in terms of the company's culture. The third level of categorization is helpful on the identification and the causes of technical debt in a more detailed manner, since it gives a lower level understanding of the debt, in terms of where it was introduced in the first place. To make it more useful, we need more clear distinctions between categories, and we should consider multiple categories for each issue especially when we deal with ambiguous issue definitions.

In line with these answers, we made the following inferences:

- It was understood that the intentions of the decision makers in software delivery processes are very important in analyzing the technical debt incurred when delivering and maintaining software, especially for the debts introduced in the “Design” stage of the project/product.
- The most beneficial and clear technical debt categorization could be performed by the people who were involved in the customization and development of the Salesforce platform.

  Each technical debt may be related to one or more issues involving several different development activities or aspects. Hence, we may consider multiple categories for each issue rather than mapping each issue to just one category.
- There is also a need to define each category in the CRM context for the third level of categorization which offers the higher granularity. It may also be possible to bring new levels of categories to cover different aspects of software development. Moreover, it may be necessary to customize categories to better support development in different domains.

With the conduction of the second case study the ambiguity that was present in the first case study due to anonymous issue statements was inhibited. With the interviews and the categorization results, it can be concluded that the technical debt issues in organizations using SaaS applications in CRM platforms, are not very different from the technical debt issues experienced in other software development companies. The top three technical debt issues are:

1. SaaS related limitations
2. Not following standards
3. Design flaws

"Not following standards" and "Design flaws" issues are already a reflection of the technical debts foreseen in previous studies and in other software development companies (Alves et al., 2014). It is possible to find a relevance between the indicators obtained as a result of the interviews, and the technical debt categories previously determined in the ontology of terms for technical debt. Based on the answers given by the practitioners, it is possible to relate these issues to the following technical debt types:

Table 23 – TD issues experienced in SaaS applications vs TD types

TD issues experienced in SaaS applications	TD types in Ontology of Terms
SaaS related limitations	Build Debt, Infrastructure Debt, Test Debt
Not following standards	Code Debt, Test Debt, Requirement Debt
Design flaws	Architecture Debt, Design Debt

The mapping here is also related to the top 5 frequent TD types in Ramač et al.'s study, listed as: Design, Test, Code, Architecture, Documentation (2022). The issues of design flaws and not being able to follow standards in the code, are tried to be addressed in various techniques described in different studies (Albuquerque et al., 2022). It can be stated that these TD issues, which are also experienced in the field of SaaS applications, are effective in different fields in software development, and different solution techniques and management methods are being studied to address these issues.

These categories which were obtained from the interview data coding in the second case study, vary according to the experience level and the role of the interviewees as well. The different perspectives and awareness towards different TD categories are presented in the following table.

Table 24 – TD awareness based on roles

Role	Cons.	Dev.	Lead	QA Eng.	Release Mng.	Customer Sup.
Miscommunication	X	X	X	X	X	X
Incomplete work			X			
Design flaws		X				X
Not following standards		X		X		
Time constraints		X		X		
Testing failures		X		X	X	X
Finding workarounds		X		X		
Unawareness			X			
SaaS Related Limitations	X	X	X	X	X	X

Apart from the roles, the experience levels of the practitioners in the software development and in SaaS applications created a different level of awareness towards TD. When the categories are listed based on frequency of mention based on participant experience, it is possible to state that the categories of “SaaS Related Limitations” and “Not following standards” are mentioned by each participant, whereas the “Unawareness” and “Incomplete work” categories are mentioned in a non-frequent manner, and explicitly by the participant who is the team lead. The rest of the categories can be considered as commonly mentioned during the interviews.

As one of the commonly mentioned categories, the “Miscommunication” category has different kind of indicators based on the interviewee experience and role. For practitioners having roles that are more involved with customer requests and customer support, the miscommunication was mentioned to be caused by indicators such as “Not understanding requirements completely” or “Lack of requirements elicitation with customer”, whereas from a developers perspective, the miscommunication category indicators were more likely to be related to overengineering problems – which can also be considered in design flaws – but actually caused by “lack of requirements elicitation with upper management” or technical decision makers. Also one of the important indicators on miscommunication is “Customers can’t keep up with our releases”, which was stated by the customer support role. This indicator is different from others in the same category, because although the root cause behind this indicator is creating many versions and releases for customers to fix different problems in the product, the customer support role put emphasis on the communication problems with the customer due to this indicator.

When SaaS related limitations are considered as a highly mentioned issue in a company where SaaS applications are heavily used, it is a necessity to explain the sub-categories on this indicator.

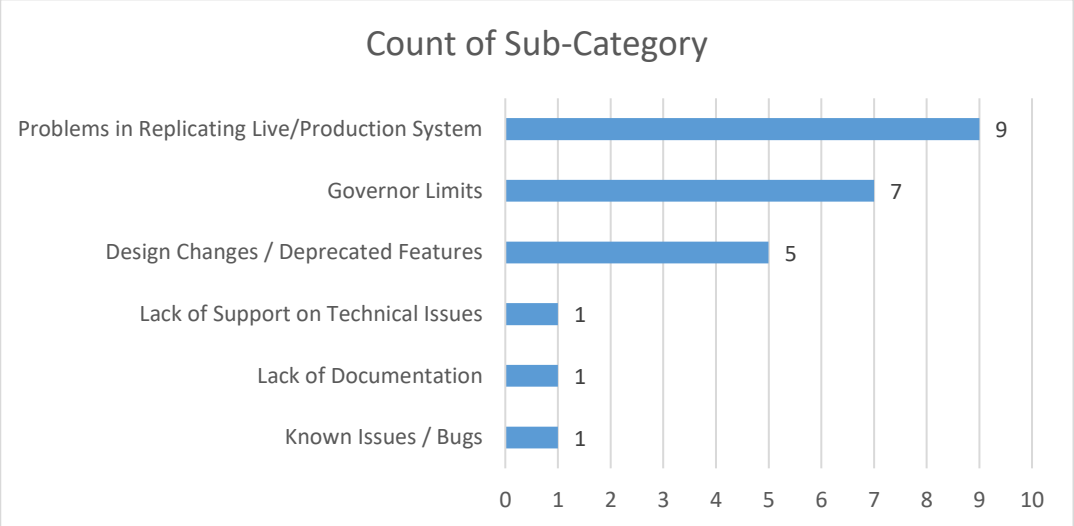

Figure 5 – Sub-categories on SaaS related limitations

In Figure 5, a lower level categorization for SaaS related issues is provided. Depending on these sub-categories for TD issues experienced, it is possible to say that there can be platform-dependent TD.

In other traditional or on-premise solutions, these TD issues do not occur, because the maintenance, repairs, updates, and the entire infrastructure are fully customized for a specific company. Therefore the services or the software delivered on these circumstances, doesn't have difficulties as in replicating live or production systems as in SaaS, or on-demand solutions. Also, based on the sources being selected, the limitations could easily be avoided. In the terms for SaaS overcoming these problems are not always possible, but there should be some workarounds that are found by the software practitioners using SaaS applications in their services.

The problems that are caused by the SaaS applications are summarized in chapter 4.3 of the thesis on Table 20 - TD problems. Most of these problems are related to the problems caused by TD in other software development activities. An increase in time spent on development activities, an increased cost for the project or services to be delivered due to refactoring cycles, and the decreases in overall product quality are examples to these problems. Other problems in organizations using SaaS can be inferred as follows:

- Troubleshooting taking much more time due to customizations on SaaS application, depending on the platform requirements, limitations, and the level of customizations that can be made in the system.
- Spending more time on refactoring cycles, due to SaaS application changing periodically, resulting in having to reduce the new features being released in the product or services side.

- As SaaS applications can deprecate/remove features, the problems addressed in the earlier versions of the software that is already provided should be re-visited. Therefore some issues are not usually fixed completely for the organization using SaaS in product delivery.

- With each SaaS application release, which can be defined as changes in the infrastructure, some resources in the project are forced to be tracking these changes and the possible impact on the software.

The common options for TD management were shared on section **Error! Reference s ource not found.**. These were defined as follows:

- Recognizing TD
- Making TD visible
- Planning TD
- Living with TD (Ozkaya, 2021)

The analysis on the second case study showed that, most of the TD issues found by software practitioners were mostly being recorded in issue tracking products. Therefore the first step in managing TD, which is the recognition and identification should be achieved and is an important step in managing TD in organizations using SaaS applications as well.

Although the TD awareness levels were differing based on experience level and role, most of the interview participants mentioned that TD was visible across the development team and the higher management. Therefore the communication, tracking and involvement on TD side was achieved in this organization.

The TD awareness results of the participants in the second case study were in line with the "familiarity with TD concept" findings in the study of Ramač et.al (2022). In Ramač' et al.'s study, 69% of the participants were aware of TD while the rest mentioned that they had not heard of the concept before. In the second case study, the participants' experience in software development and their TD awareness were examined, in chapter 4.2.4. According to the findings here, TD awareness in 3 out of 9 participants was at "Low" level. In Ramač et al.'s study, it was also among the findings that experience level was related to TD awareness. It has been mentioned that more experienced members in the teams are more likely to define the TD concept and have practical experience (2022). Considering the experience of the practitioners in this study in software development, TD awareness levels remained at a directly proportional level. Of course, it is worth noting that roles of the software practitioners have a significance. While experience level and TD awareness find clearer answers in the QA, Dev and Lead roles in the participants of the study, the practitioners that has involvement with managerial or customer support related roles do not have much practical experience with TD.

Another important point is that if TD explanations in the first case study shared in Appendix B are reviewed, it can be observed that the group of experts acting as validators were mostly in consensus on the TD definition and explanations based on the validated issues. Considering the roles and experiences of the validators, it was

possible to detect the importance of awareness and identification of TD in the first case study as well.

The communication and involvement methods in managing TD is also significant in terms of SaaS applications. It is important to make TD visible and present it to all stakeholders. While familiarizing with TD, all stakeholders should be aware of the causes of TD, either caused by development team by not following the best practices or caused by SaaS limitations.

Some of the practitioners that participated in the second case study also mentioned about the TD that they introduced in the project and to the product, due to various reasons such as time constraints, and governor limits on the SaaS limitations. The practitioners that work as developers mentioned that they reflect some of this debt in their commit messages and/or comments on the source code of the project. This improves the TD awareness among practitioners that are involved with the coding processes and that has responsibilities in the design and architectural processes of the software development life cycle. This concept is addressed as self-admitted TD in literature, and recognized by other software practitioners, and eventually has a significance on the TD management efforts (Zampetti et al., 2021).

One possible obstacle in making TD visible is to measure it based on different metrics. The practitioners of the organization were not using or aware of any metrics for managing TD. This obstacle makes it difficult to manage TD in terms of planning. The decision making on different options, such as prevention, monitoring, or repayment can be made in an easier fashion, if TD is somehow measurable across all types on level 3 categorization, such as design debt, infrastructure debt, and test debt. As stated in the study from Curtis et al., quantifying the TD is important since the decision makers of the organizations take their actions on the information that is visible to them based on costs and risks (2012). Therefore the importance of TD measurement should be emphasized while mentioning the obstacles towards TD awareness. The developers tend to measure “code” debt with the help of different static code analysis tools, but in this case of SaaS usage, there isn’t a way to measure the impact TD caused by SaaS provider. Therefore, the lack of documentation or support from SaaS provider becomes more significant in this process, but the main objective of software practitioners should be tracking down changes and modifications on SaaS side to track the TD. Apart from these, there is also another step in TD management which is lack of TD prevention. Being able to provide options for preventing TD is important for services and products running in an ever-changing application such as the field of SaaS.

CHAPTER 6

CONCLUSION

In this thesis study, we conducted a study on the specification and categorization of technical debt for organizations dealing with CRM specific development on SaaS applications. By making use of different categorization methods and ISO/IEC 12207 SDLC processes, we tried to reveal what benefits categorization can determine in technical debt specification.

As a result of the study, the effectiveness of three technical debt categorization methods having different levels of granularity was examined. Also, the technical debt issues arising, such as design flaws, SaaS related limitations and not-following best practices were found out. The problems caused by such issues were also considered in SaaS application usage, appearing as inability to meet customer satisfaction, decrease in product quality, and problems in capacity and resource planning. Also, different options on TD management were also evaluated specifically for SaaS.

Since the study was conducted specifically on software companies developing with SaaS applications, we hope that the findings of the study will raise awareness among the organizations that will use such applications. Through this awareness, companies can shape their preferences according to what kind of problems they may encounter when they choose SaaS applications. In addition, being aware that technical debt can be inevitable in SaaS applications, as in on-premise solutions, start-up and small scale organizations may take action to prevent the technical debt.

For future work, we hope to facilitate the management of TD in this area, and to create a set of metrics for software practitioners using SaaS who will just start managing TD in their organization.

REFERENCES

Agarwal, P. (2011, February). Continuous SCRUM: agile management of SAAS products. In Proceedings of the 4th India Software Engineering Conference (pp. 51-60).

Albuquerque, D., Guimarães, E., Tonin, G., Rodriguez, P., Perkusich, M., Almeida, H., ... & Chagas, F. (2022). Managing Technical Debt Using Intelligent Techniques-A Systematic Mapping Study. IEEE Transactions on Software Engineering.

Alves, N. S., Ribeiro, L. F., Caires, V., Mendes, T. S., & Spínola, R. O. (2014, September). Towards an ontology of terms on technical debt. In 2014 Sixth International Workshop on Managing Technical Debt (pp. 1-7). IEEE.

Alzaghoul, E., & Bahsoon, R. (2013, May). CloudMTD: Using real options to manage technical debt in cloud-based service selection. In 2013 4th International Workshop on Managing Technical Debt (MTD) (pp. 55-62).IEEE.

Ascendix. (2022). Top 10 CRM Software Companies. Available from: https://ascendixtech.com/top-crm-software-companies/.Accessed 15.8.2022.

Bryant, M. (2018, Jul). The History of Act! CRM. Available from: https://www.acttoday.com.au/blog/the-history-of-act-crm.Accessed 8.6.2022.

Codabux, Z., Williams, B. J., Bradshaw, G. L., & Cantor, M. (2017). An empirical assessment of technical debt practices in industry. Journal of software: Evolution and Process, 29(10), e1894.

Cunningham, W. (1992). The WyCash portfolio management system. ACM SIGPLAN OOPS Messenger, 4(2), 29-30.

Curtis, B., Sappidi, J., & Szynkarski, A. (2012, June). Estimating the size, cost, and types of technical debt. In 2012 Third International Workshop on Managing Technical Debt (MTD) (pp. 49-53). IEEE.

Customer Relationship Management (CRM). (2022) Salesforce. Available from: https://www.salesforce.com/ap/definition/crm/ Accessed 19.05.2022.

Cusumano, M. (2010). Cloud computing and SaaS as new computing platforms. Communications of the ACM, 53(4), 27-29.

Elements.cloud, Org Confessions, Elements.cloud. Available from: https://elements.cloud/confessions/. Accessed 9.2.2020.

Falessi, D., & Kazman, R. (2021, May). Worst smells and their worst reasons. In 2021 IEEE/ACM International Conference on Technical Debt (TechDebt) (pp. 45-54). IEEE.

Fowler,M. (2009) martinfowler.com. Available from: http://martinfowler.com/bliki/TechnicalDebtQuadrant.html. Accessed 12.05.2020.

ISO, I. (2008). IEC 12207 Systems and software engineering-software life cycle processes. International Organization for Standardization: Geneva.

Iuliia, G. (2017). Technical Debt Management In Russion Software Development Companies. Master's Thesis. St. Petersburg University Graduate School of Management.

Klinger, T., Tarr, P., Wagstrom, P., & Williams, C. (2011, May). An enterprise perspective on technical debt. In Proceedings of the 2nd Workshop on managing technical debt (pp. 35-38).

Kruchten, P., Nord, R. L., & Ozkaya, I. (2012). Technical debt: From metaphor to theory and practice. Ieee software, 29(6), 18-21.

Kumar, S., Bahsoon, R., Chen, T., & Buyya, R. (2019, July). Identifying and estimating technical debt for service composition in SaaS cloud. In 2019 IEEE International Conference on Web Services (ICWS) (pp. 121-125). IEEE.

Kumar, S. (2021). Technical debt-aware and evolutionary adaptation for service composition in SaaS clouds (Doctoral dissertation, University of Birmingham).

Liu, F., Tong, J., Mao, J., Bohn, R., Messina, J., Badger, L., & Leaf, D. (2011). NIST cloud computing reference architecture. NIST special publication, 500(2011), 1-28.

Mazalon, L. (2021). 12 Salesforce ISVs Who've Raised Millions (Totalling $1.35Billion), SalesforceBen. Available from: https://www.salesforceben.com/salesforce-isvs-whove-raised-millions/. Accessed 20.5.2022.

Mcconnell, S. (2021). Construx, 1 January 2013. Available from: https://www.construx.com/resources/whitepaper-managing-technical-debt/. Accessed 5.2.2020.

Ozkaya, I. (2021). Managing technical debt. CARNEGIE-MELLON UNIV PITTSBURGH PA.

Ramač, R., Mandić, V., Taušan, N., Rios, N., Freire, S., Pérez, B., ... & Spinola, R. (2022). Prevalence, common causes and effects of technical debt: Results from a family of surveys with the IT industry. Journal of Systems and Software, 184, 111114.

Ramasubbu, N., & Kemerer, C. F. (2016). Technical debt and the reliability of enterprise software systems: A competing risks analysis. Management Science, 62(5), 1487-1510.

Rigby, D. K., & Ledingham, D. (2004). CRM done right. Harvard business review, 82(11), 118-130.

Salesforce, Deploy Enhancements from Sandboxes, Salesforce. Available from: https://help.salesforce.com/articleView?id=changesets_about_connection.htm&type=5. Accessed 10.5.2020.

Salesforce, ISVforce Guide, Salesforce. Available from: https://developer.salesforce.com/docs/atlas.enus.packagingGuide.meta/packagingGuide/appexchange_intro.htm. Accessed 9.2.2020.

Salesforce, What is Salesforce? Salesforce. Available from: https://www.salesforce.com/eu/products/what-is-salesforce/. Accessed

12.5.2020.

Salesforce, Worlds Number One CRM, Salesforce. Available from: https://www.salesforce.com/campaign/worlds-number-one-CRM/ Accessed 12.6.2022.

Skourletopoulos, G., Mavromoustakis, C. X., Bahsoon, R., Mastorakis, G., & Pallis, E. (2014, December). Predicting and quantifying the technical debt in cloud software engineering. In 2014 IEEE 19th international workshop on computer aided modeling and design of communication links and networks (CAMAD) (pp. 36-40). IEEE.

Suryanarayana, G., Samarthyam, G., & Sharma, T. (2014). Refactoring for software design smells: managing technical debt. Morgan Kaufmann.

Tohidi, H., & Jabbari, M. M. (2012). The necessity of using CRM. Procedia Technology, 1, 514-516.

Youseff, L., Butrico, M., & Da Silva, D. (2008). Grid computing environments workshop, 2008. GCE '08, Austin, Texas, 12-16.

Zampetti, F., Fucci, G., Serebrenik, A., & Di Penta, M. (2021). Self-admitted technical debt practices: a comparison between industry and open-source. Empirical Software Engineering, 26(6), 1-32.

Zazworka, N., Spínola, R. O., Vetro', A., Shull, F., & Seaman, C. (2013, April). A case study on effectively identifying technical debt. In Proceedings of the 17th International Conference on Evaluation and Assessment in Software Engineering (pp. 42-47).

Zazworka, N., Vetro, A., Izurieta, C., Wong, S., Cai, Y., Seaman, C., & Shull, F. (2014). Comparing four approaches for technical debt identification. Software Quality Journal, 22(3), 403-426.

APPENDIX A

SET OF CONFESSIONS FOR VALIDATION FOR CASE STUDY 1

Entry #	Entry
50	A single apex class with 13,000 lines of code!
51	Added 20 dummy classes (8000 lines code) and test classes to boost code coverage.
52	My Manager removed the ability to export reports. Without telling anybody. 2 days before month end reporting was due.
53	Using i++ to increase code coverage of test classes unethically.
54	Custom address fields on Contact and Lead objects.
55	35 Users, 25 Profiles. Every custom field on every object starts with z_ . Different suppliers, created different fields for same purpose.
56	Multiple Orgs. They couldn't work out what was configured so they start again, and again, and again. We see this a LOT in the nonprofit space.
147	Multiple fields with same label.
148	Process consolidation – 16 PBW on a single object and each PPW has single action and criteria.
149	Renamed every metadata API name to the business unit that needed it. The developers needed to access lookup sheet to be able to decipher.
150	Created a custom fields for every standard field because admin couldn't see them in searches.
151	1 process includes 18 PBW, 2 flows, 12 workflow rules, 30 custom formula fields. Plus maybe some code. No one knows how or why it exists.
152	A managed package that used Translation workbench to swap the labels on two fields so they are now the opposites of their API names.
288	Over 150,000 reports….
294	Role names 'hard coded' within Apex Classes.

APPENDIX B

VALIDATION FORM PROVIDED TO CRM PRACTITIONERS FOR CASE STUDY 1

Analyzing Technical Debt by Categorizing Anonymous Developer Confessions of a CRM Application by Yasemin Doğancı

The study is conducted to evaluate anonymous developer confessions about different Salesforce organizations and categorize them according to three different categorization options on technical debt. After you read the definitions of three different categorization options, please fill in the form and categorize each entry.

Technical Debt Explanation

The term "technical debt" was created as a metaphor by Ward Cunningham, to describe all the code defects, and design and architectural mistakes made by developers, and to summarize these to "non-technical" people in software projects. The idea behind this term is that, when an action is made that adds value at that point in time, will have consequences in the future which should be paid with an extra cost, which is described as debt.

Technical Debt Category Definitions

Option 1 = Intentions: According only to the intension of the people who introduced the debt.

Possible Values: Intentional, Unintentional

Option 2 = Technical Debt Quadrant: According to the intensions and knowledge of the people who introduced the debt.

Possible Values: Reckless – Inadvertent, Reckless – Deliberate, Prudent – Inadvertent, Prudent – Deliberate

Turkish Translation for each:

- *Reckless: Umarsız*
- *Deliberate: Kasten*
- *Inadvertent: Yanlışlıkla Yapılan*
- *Prudent: İhtiyatlı*

Option 3 = Introduction of the Debt in Software Development Life Cycle: Exactly when the debt is introduced in the software development process.

Possible Values: Architecture, Build, Code, Defect, Design, Documentation, Infrastructure, People, Process, Requirement, Service, Test Automation, Test

Please fill this form before starting categorization.

Name:	
Role *:	
Years of Experience in Software Development *:	
Years of Experience in Salesforce Platform *:	

Option1 Possible Values:

- Intentional
- Unintentional

Option 2 Possible Values:

- Reckless – Inadvertent
- Reckless – Deliberate
- Prudent – Inadvertent
- Prudent – Deliberate

Option 3 Possible Values:

- Architecture	The debt is caused by Salesforce's architectural flaws or deficiency
- Build	Errors in the implementation of the solution or software
- Code	Lack of coding standards, hard coding user data
- Defect	Bugs, errors that are not traceable
- Design	Building without thinking through the data model, poor business analysis
- Documentation	Undocumented work, uncertain/outdated documents
- Infrastructure	Lack of tools needed for the operations, mis-usage of the tools
- People	Insufficient training, human error, lack of understanding on platform
- Process	No implementation methodology, development in Production
- Requirement	Changing requirements, not understandable requirements
- Service	Errors in the maintenance stages
- Test Automation	Errors in test automation process, lack of automated tests
- Test	Lack of unit tests, low code coverage

APPENDIX C

INTERVIEW QUESTIONS

1. Can you tell me about the organization, its size and domain of service?
2. What is your role in the organization, and do you have any responsibilities in any of the projects?
3. Can you describe the project and the technologies being used?
4. Do you participate in any of the following Software Life Cycle Processes?
    a. Project Planning
    b. Project Assessment and Control
    c. Configuration Management
    d. Stakeholder Needs and Requirements Definition
    e. Systems/Software Requirements Definition
    f. Architecture Definition
    g. Design Definition
    h. Implementation
    i. Integration
    j. Verification
    k. Transition
    l. Validation
5. Do you experience any issues with the processes/practices that you participate in the project?
6. Do you practice the tasks/activities defined for each process?
7. Do you think there are tasks that you fail or don't achieve completely?
8. What kind of problems occur in the project due to these issues?
9. How did you realize these problems? Do you have any further observations on the problems?
10. Do you know if these problems are caused by the SaaS provider or the development team that you belong to?

11. Do you have any recommendations or a way to solve/prevent these problems?

12. Do you experience any further challenges regarding the usage of SaaS applications in the project?

13. Do you sometimes choose easy (limited) solutions to problems in your project, instead of using the better approach?

14. Do you have rework or refactoring cycles for your source code?

15. Do you have any problems that you choose not to fix/solve and only monitor the course/trend over time?

16. Are you familiar with the technical debt metaphor?

17. Do you measure the technical debt that is present in your project, using any tools or processes?

18. Do you mostly monitor or take further action on technical debt issues?

19. How do you decide the managerial activities on the technical debt issues?

20. Do you have any metrics or scale to prioritize the technical debt issues?

21. Do you have any technical debt that is expected to become worse in the future?

22. To what extent does technical debt affect your motivation?

23. Is your management aware of technical debt and are they taking any action?

24. Do you have additional issues that you would like to mention/highlight?

25. Do you think that there’s room for improvement for your project/organization?

APPENDIX D

GÖRÜŞME SORULARI

1. Çalıştığınız organizasyondan, boyutundan ve hizmet alanından bahsedebilir misiniz?
2. Organizasyondaki rolünüz nedir ve herhangi bir projede sorumluluğunuz var mı?
3. Projeyi ve kullanılan teknolojileri anlatabilir misiniz?
4. Aşağıdaki Yazılım Yaşam Döngüsü Süreçlerinden (Software Life Cycle Processes) herhangi birinde rol alıyor musunuz?
    a. Proje Planlama – (İng. Project Planning)
    b. Proje Doğrulama ve Kontrol – (İng. Project Assessment and Control)
    c. Konfigürasyon Yönetimi – (İng. Configuration Management)
    d. Paydaş İhtiyaçları ve Gereksinimleri Tanımı – (İng. Stakeholder Needs and Requirements Definition)
    e. Sistem/Yazılım Gereksinimleri Tanımı – (İng. Systems/Software Requirements Definition)
    f. Mimari Tanım – (İng. Architecture Definition)
    g. Tasarım Tanımı – (İng. Design Definition)
    h. Uygulama – (İng. Implementation)
    i. Entegrasyon – (İng. Integration)
    j. Doğrulama – (İng. Verification)
    k. Geçiş – (İng. Transition)
    l. Geçerleme – (İng. Validation)
5. Projeye katıldığınız süreçlerde/uygulamalarda herhangi bir sorun yaşıyor musunuz?
6. Her süreç için tanımlanan görevleri/etkinlikleri uyguluyor musunuz?

7. Başarısız olduğunuz veya tam olarak başaramadığınız görevler olduğunu düşünüyor musunuz?
8. Bunlar nedeniyle projede ne gibi sorunlar yaşanıyor?
9. Bu sorunları nasıl fark ettiniz? Sorunlarla ilgili başka gözlemleriniz var mı?
10. Bu sorunların SaaS (Servis Olarak Yazılım Uygulamaları) sağlayıcısından mı yoksa ait olduğunuz geliştirme ekibinden mi kaynaklandığını biliyor musunuz?
11. Bu sorunları çözmek/önlemek için herhangi bir öneriniz veya yönteminiz var mı?
12. Projede SaaS (Servis Olarak Yazılım Uygulamaları) uygulamalarının kullanımıyla ilgili başka zorluklarla karşılaşıyor musunuz?
13. Bazen projenizdeki sorunlar için daha iyi yaklaşımı kullanmak yerine kolay (sınırlı) çözümler mi seçiyorsunuz?
14. Kaynak kodunuz için yeniden işleme (rework) veya yeniden düzenleme (refactoring) döngüleriniz var mı?
15. Düzeltmeyi/çözmeyi tercih ettiğiniz ve sadece zaman içindeki gidişatı/eğilimi takip ettiğiniz herhangi bir sorununuz var mı?
16. Teknik borç metaforuna aşina mısınız?
17. Projenizde mevcut olan teknik borcu herhangi bir araç veya süreç kullanarak ölçüyor musunuz?
18. Teknik borç sorunlarını daha çok izliyor veya daha fazla önlem alıyor musunuz?
19. Teknik borç sorunlarını yönetsel faaliyetlere nasıl karar veriyorsunuz?
20. Teknik borç sorunlarını önceliklendirmek için herhangi bir ölçütünüz veya ölçeğiniz var mı?
21. Gelecekte daha da kötüleşmesi beklenen herhangi bir teknik borcunuz var mı?
22. Teknik borç motivasyonunuzu ne ölçüde etkiliyor?
23. Yönetiminiz teknik borcun farkında mı ve önlem alıyor mu?
24. Belirtmek/vurgulamak istediğiniz ek sorunlarınız var mı?

Projeniz/bulunduğunuz organizasyon için iyileştirmeye yer olduğunu düşünüyor musunuz?

UYGULAMALI ETİK ARAŞTIRMA MERKEZİ
APPLIED ETHICS RESEARCH CENTER

ORTA DOĞU TEKNİK ÜNİVERSİTESİ
MIDDLE EAST TECHNICAL UNIVERSITY

DUMLUPINAR BULVARI 06800
ÇANKAYA ANKARA/TURKEY
T: +90 312 210 22 91
F: +90 312 210 79 59
ueam@metu.edu.tr
www.ueam.metu.edu.tr

Sayı: 28620816 /

15 ŞUBAT 2022

Konu : Değerlendirme Sonucu

Gönderen: ODTÜ İnsan Araştırmaları Etik Kurulu (İAEK)

İlgi : İnsan Araştırmaları Etik Kurulu Başvurusu

Sayın

"Çevik Yazılım Geliştirme Pratiklerindeki Uyarlama Hatalarının Teknik Borçlanmaya Etkisini Gösteren Bir Model ve Çözümleri Destekleyen Bir Karar Destek Aracı" başlıklı araştırmanız İnsan Araştırmaları Etik Kurulu tarafından uygun görülmüş ve **0112 ODTUİAEK-2022** protokol numarası ile onaylanmıştır.

Saygılarımızla bilgilerinize sunarız.