Master of Science in Software Engineering
June 2017

Technical debt management in a large-scale distributed project
- An Ericsson case study

Zhixiong Gong
Feng Lyu

Faculty of Computing
Blekinge Institute of Technology
SE-371 79 Karlskrona Sweden

This thesis is submitted to the Faculty of Computing at Blekinge Institute of Technology in partial fulfillment of the requirements for the degree of Master of Science in Software Engineering. The thesis is equivalent to 20 weeks of full time studies.

Contact Information:
Authors:
Zhixiong Gong
E-mail: zhgo15@student.bth.se
Feng Lyu
E-mail: felv15@student.bth.se

University advisor:
Ricardo Britto
Doctoral Student
Department of Software Engineering

Faculty of Computing
Blekinge Institute of Technology
SE-371 79 Karlskrona, Sweden

Internet : www.bth.se
Phone : +46 455 38 50 00
Fax : +46 455 38 50 57

ABSTRACT

Context. Technical debt (TD) is a metaphor reflecting technical compromises that sacrifice long-term health of a software product to achieve short term benefit. TD is a strategy for the development team to obtain business value. TD can do both harm and good to a software based on the situation of TD accumulation. Therefore, it is important to manage TD in order to avoid the accumulated TD across the breaking point. In large-scale distributed projects, development teams located in different sites, technical debt management (TDM) becomes more complex and difficult compared with traditional collocated projects. In recent years, TD metaphor has attracted the attention from academics, but there are few studies in real settings and none in large-scale globally distributed projects.

Objectives. In this study, we aim to explore the factors that have significant impact on TD and how practitioner manage TD in large-scale distributed projects.

Methods. We conducted an exploratory case study to achieve the objectives. The data was collected through archival records and a semi-structured interview. For the archival data, hierarchical multiple regression was used to analyze the relationship between identified factors and TD. For interview data, we used qualitative content analysis method to get a deep understanding of TDM in this studied case.

Results. Based on the results of archival data analysis, we identified three factors that show significant positive correlation with TD. These three factors were task complexity, global distance, and maturity, which were evaluated by the architect during the semi-structured interview. The architect also believed that these factors have strong relationships with TD. TDM in this case includes seven management activities: TD prevention, identification, measurement, documentation, communication, prioritization, and repayment. The tool used for TDM is an internally implemented tool called wiki page. We also summarize the roles involved and approaches used with respect to each TDM activity. Two identified TDM challenges in this case were TD measurement and prioritization.

Conclusions. We conclude that 1) TDM in this case is not complete. Due to the lack of TD monitoring, the measurement of TD is static and lacks an efficient way to track the change of cost and benefit of unresolved TD over time. Therefore, it is difficult to find a proper time point to repay a TD. 2) The wiki page is not enough to support TDM, and some specific tools should be combined with wiki page to manage TD comprehensively. 3) TD measurement and prioritization should get more attention both from practitioners and academics to find a suitable way to solve such challenges in TDM. 4) Factors that make significant contribution to TD should be carefully considered, which increase the accuracy of TD prediction and improve the efficiency of TDM.

Keywords: technical debt management, large-scale distributed project, factors.

ACKNOWLEDGE

First, we would like to thank our supervisor Ricardo Britto for guiding and supporting us. He helped us to contact the interviewee of Ericsson, otherwise we would not have the source to conduct this case study. He gave us a lot of advice to improve the quality of this dissertation and encouraged us when we got confused and lost direction.

Second, we want to thank the architect in Ericsson who provided us with practical knowledge and experience related to our research topic. We also thank the examiners by providing some useful feedback about the thesis to guide us on the right track.

Finally, we thank the support from our families, we would not have the opportunity to study here without their support and understanding.

CONTENTS

LIST OF FIGURE

LIST OF TABLE

TERMINOLOGY

Terms	Explanation
TD principle	The cost of fixing a technical debt.
TD interest	The extra cost by not fixing a technical debt.
TD accumulation	The amount of TD in a system can increase over time due to some intentional or unintentional technical decisions.
Interest possibility	The occurrence probability of the interest of a technical debt.
NIP	Node improvement protocol, which can be the identified TD or other node improvements.

1 INTRODUCTION

1.1 Introduction to research context

In 1992, when studying the development process of a portfolio management system, Ward Cunningham borrowed concepts from financial fields and first introduced the metaphor of Technical Debt (TD). TD reflects technical compromises to achieve short-term benefit at a cost of hurting the long-term health of a software product, which puts future development and maintenance at a high potential risk [1]. TD refers to any incomplete, immature or inadequate artifact in the software development lifecycle affecting subsequent development and maintenance activities, which is treated as a type of debt that the developers owe the system [2].

TD can do both good and harm to a software project [3]. On the one hand, it is common that a software project intentionally incurs some debts in the development process, because small amount of debts can speed up the development and make the company ahead of competitors in the market. On the other hand, TD can also be incurred unintentionally. In this case, neither the project manager nor the development team is aware of the existence, location, and consequences of the TD. If the unintentional TD remains invisible and unresolved, TD can be accumulated incrementally and cause challenges to the maintenance and evolution of a product [3].

To make the accumulated TD under control, technical debt management (TDM) is required throughout the development process. One part of TDM includes activities preventing potential TD from being incurred. Meanwhile, TDM also includes activities dealing with the accumulated TD to make it visible and controllable, and to keep a balance between cost and value of the software project [3]. According to Li et al [3], TDM is basically composed of eight activities: *TD identification, measurement, prioritization, prevention, monitoring, repayment, representation/documentation and communication*, among which *TD identification, measurement, and repayment* are the three fundamental TDM activities. These activities composed a series of management processes for TD. Generally, TD can be classified into ten types based on the causes of TD: *requirements, architectural, design, code, test, build, documentation, infrastructure, versioning and defect TD* [3]. For different TD types, the adopted TDM activities are same, while the approaches used for each activity may vary between TD types.

Recent years, the flourishing discussion and application of TD metaphor in software communities have attracted the attention of academic researchers, and many studies have been conducted on the topic of TD. For example, Fontana, et al. [4] focused on the prioritization of code TD, trying to find the most critical code debt for refactoring. These authors also conducted research into code TD identification by filtering detection results of code smell which is a sub-type of code TD, and removing false positives in the results [5]. However, existing studies focus on activities within TDM rather than investigate the full process in industrial settings [3]. It is important to understand TDM through the whole process because TDM activities are dependent on each other. For example, TD measurement is the premise of TD monitoring, because without input data from TD measurement, it is impossible for development teams to monitor TD. Meanwhile, teams are not able to have any reasoning for other TDM activities due to lack of TD monitoring [6]. So, more studies should be carried out to understand how practitioners manage TD throughout the TDM process.

Furthermore, there is no study investigating the whole TDM process in large-scale distributed projects. Large-scale software development is an activity heavily relying on smooth communication and collaboration, which requires diverse expertise and substantial human resources [7]. But as the size of a software project increases, the size of development team also scales up, it is impractical to concentrate all human resources in a single site. Companies realize that it is beneficial to distribute some tasks to other locations that are closer to customers and necessary resources [7]. With the fast development of information technology, it is more convenient and cost-effective for organizations to distribute their large-scale projects to different positions [7]. However, the coordination between different sites is still problematic due to the intrinsic shortage of distributed collaboration tools. Under such condition, issues that occur during the development and maintenance process are more difficult to solve compared to traditional collocated development, since the relationship between distributed teams is not as tight as traditional teams who coordinate via task organization and team communication [7]. TDM in large-scale distributed projects can be more complex. It needs to guarantee that people in multiple sites share the same understanding of TD and align to TDM [8]. In large-scale distributed projects, TD can be affected by several factors, including common factors like task complexity [9], and factors specifically for the context of large-scale distribution like distance [10]. So before managing TD, it should first be identified what factors affect TD. By controlling factors affecting TD, TDM can be conducted to manage TD sequentially.

Based on the above research gap, we aim to investigate what factors can influence TD under such condition, and how practitioners conduct TDM in large-scale distributed projects. The results from this thesis can contribute to both academia and industry by bridging the gaps between both sides, supplementing literature study with measures in practice and providing practitioners with academic research.

1.2 Aim and objectives

The aim of this research is to explore what factors impact technical debt and identify how practitioners manage TD in large-scale distributed projects in practice.

To achieve the main goal, four sub-objectives are defined:

- To explore the relationship between different factors and TD.
- To identify the tools and methods used for TDM in large-scale distributed projects.
- To explore the roles involved in TDM in large-scale distributed projects.
- To explore the challenges that practitioners face to carry out TDM in large-scale distributed projects.

1.3 Research questions

According to the objective, our research will answer the following questions:

RQ1: What are the factors that impact TD in large-scale distributed projects?

- **RQ1.1:** How do these factors impact TD in large-scale distributed projects?

RQ2: How do practitioners manage TD in large-scale distributed projects?

- **RQ2.1:** What methods and tools are used to manage TD in terms of each management activity in a large-scale distributed project?
- **RQ2.2:** Which roles are involved in each TDM activity in large-scale distributed projects?

- **RQ2.3:** What challenges are associated with TDM in large-scale distributed projects?

In this research, we conducted a case study to figure out how practitioners manage TD in a large-scale distributed project. First, we made a hierarchical multiple regression analysis on the archival data provided by Ericsson to identify several factors which impact TD from all factors in the data (RQ1) and analyze the impact of these factors on TD (RQ1.1). Then, a semi-structured interview was conducted to explore the methods and tools used for TDM (RQ2.1), the roles involved in TDM (RQ2.2), and the challenges associated with TDM (RQ2.3). At the end of the interview, we discussed the hierarchical multiple regression analysis results with the interviewee and tried to find out other new factors (RQ1). The corresponding relationship between methods and research questions is shown in Figure 1.1.

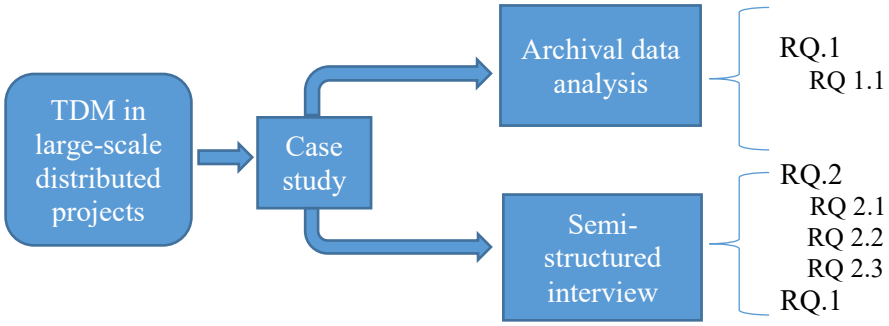

Figure 1.1 Mapping the research method and research questions

1.4 Process of the research

The research was conducted through four steps. First, we reviewed relative papers to get an understanding of TDM and the state of art of TDM. Then, a hierarchical multiple regression was conducted to analyze the archival data provided by Ericsson, in order to figure out the relationship between several factors and TD. In the third step, a semi-structured interview was conducted to explore how practitioners manage TD in a large-scale distributed project, and to discuss the archival data analysis results with interviewee to evaluate our findings. Finally, we got the body of knowledge of TDM in large-scale distributed projects and reported the findings. The process of this research is shown in Figure 1.2.

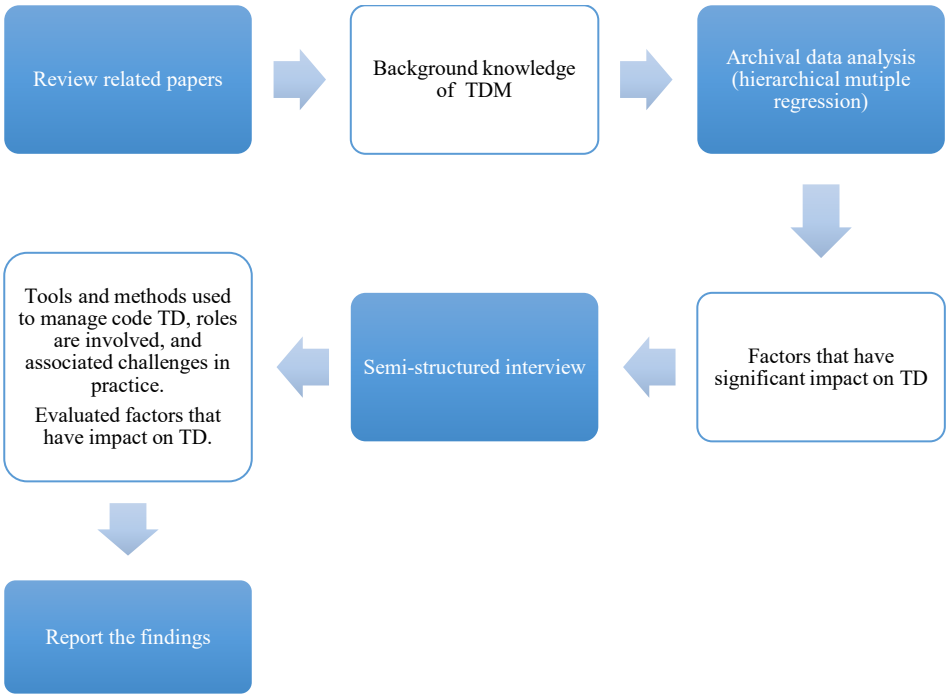

Figure 1.2 The process of this thesis

1.5 Structure of the thesis

The remainder of this thesis is organized as follows: The second chapter presents the background and related work about TDM and large-scale distributed projects. The third chapter presents a detailed description of the research methodology, such as the rationale to select case study and how it is designed as well as executed to achieve the research objectives. The fourth chapter presents and analyzes the results obtained from the case study. The fifth chapter presents a discussion in terms of the research questions. Finally, in chapter six we present our conclusions and vision in the future work. The whole thesis structure is shown in Figure 1.3.

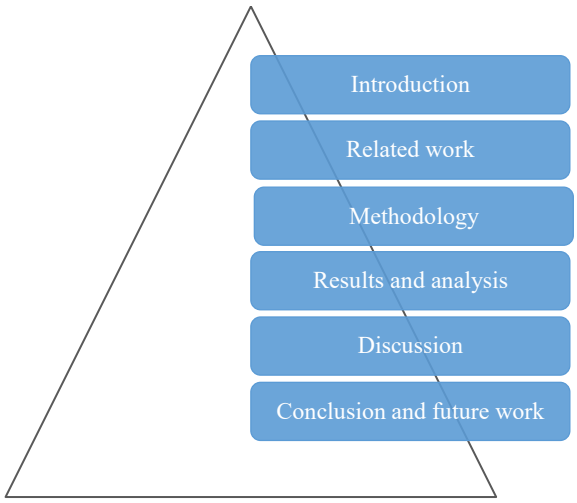

Figure 1.3 The structure of the thesis

2 RELATED WORK

2.1 Large-scale distributed software development

Nowadays, advanced telecommunications and collaboration technologies are providing an added incentive to collaborate with geographically distributed colleagues. Some companies find it strategically important to distribute tasks to other locations that are closer to clients, skilled personnel, and technical resources (e.g., hardware, tools, test labs, etc.) [11]. However, large-scale distributed software development is always associated with a complex network of interconnected tasks, and the interdependence between the various tasks makes system development iterative, leading to development delay which causes extra loss of time without profits. Yao et al. [11] analyzed the network of large-scale software development and found that task networks have properties of sparseness, small world and scaling regimes just like other real-world networks. One strategy to deal with such network is reusing existing modules, since it allows firms to exploit the knowledge embedded in reused modules to reduce the complexity and scope of the software development project, thus significantly reducing the software development time.

In order to manage distributed concurrent development, Aoyama [12] proposed a framework model which consists 3 layers: collaboration layer, information layer, and operation layer. This model can implement the management of the process, product, project, organization and software engineering environment in distributed concurrent development.

Traditional organization theories suggest that teams coordinate explicitly via task organization and team communication. However, asynchronous and distributed collaborators have fewer opportunities to interact than real-time and collocated collaborators. In spite of the abundance of distributed collaboration tools, the fact remains that coordination in distributed, large-scale software development is still problematic for many organizations because working from a distance brings increased coordination overhead, reduced richness in communication media, and more substantial delays [11].

2.2 Technical debt

Technical debt (TD) is a metaphor reflecting technical compromises that can yield short-term benefit but may hurt the long-term health of a software system. This metaphor was initially concerned with software implementation (i.e., at code level), but it has been gradually extended to software architecture, detailed design, and even documentation, requirements, and testing [3].

In order to make a secondary study of TD, Tom et al. [13] conducted a systematic literature review (SLR) to establish a theoretical framework of technical debt. The authors identified two elements of TD, code decay and design debt, and the boundaries of TD. Meanwhile, they discussed the reasons and outcomes of technical debt's arising. Based on this SLR, they complemented with a multivocal literature review (MLR) [14] and improved the framework to consolidate understanding of TD. The updated framework incorporates the different dimensions, attributes, precedents and outcomes of TD.

2.3 Technical debt types

TD was initially concerned with software implementation, namely, it focuses more on the technical issues at code level [1]. During the past decade, it has been extended to other aspects of a software project such as testing, documentation, requirements, design and architecture [15]. A detailed classification of TD type helps development teams to understand TD comprehensively, which also facilitates the management of TD. Generally, TD can be categorized into ten types [3]. The definition of each TD type is shown as below:

- ***Requirements TD*** refers to the distance between the optimal requirements specification and the actual solution.
- ***Architectural TD*** refers to some improper architecture decisions that can harm internal quality aspects.
- ***Design TD*** is about making shortcuts with respect to detailed design.
- ***Code TD*** refers to poorly written code that does not follow best coding standards or coding practices.
- ***Test TD*** refers to taking shortcuts in testing (e.g., lack of unit tests or integration tests due to time budget).
- ***Build TD*** refers to the flaws in the build system or build process of a software system, which make the build overly complex and difficult.
- ***Documentation TD*** refers to any insufficient, incomplete, or outdated documentation of a software project (e.g., out-of-date architecture documentation and lack of code comments).
- ***Infrastructure TD*** refers to sub-optimal configuration of software and hardware that negatively affects the team's ability to produce a product with good quality.
- ***Versioning TD*** is caused by improper source code versioning (e.g., unnecessary code forks).
- ***Defect TD*** refers to defects, bugs, or failures detected in a software product.

2.4 Technical debt management

In order to systematically manage TD, it is necessary to have a clear and thorough understanding on TDM. TDM is composed of a set of activities that prevent potential TD from being incurred or deal with existing TD to keep it under a reasonable level. Generally, there are eight TDM activities which are *TD prevention, identification, measurement, representation/documentation, prioritization, monitoring, and repayment* [3]. The definition of each TDM activity is shown as below:

- ***TD prevention*** aims to prevent potential TD from being incurred.
- ***TD identification*** aims to detect TD caused by intentional or unintentional technical decisions in a software system.
- ***TD measurement*** aims to quantify the cost and benefit of identified TD or estimates the overall accumulation level of TD in a software system.
- ***TD documentation*** aims to provide a uniform way to present and codify TD, which address the concerns of particular stakeholders.
- ***TD communication*** aims to discuss the identified TD so that it can be further managed.

- ***TD prioritization*** aims to prioritize identified TD based on predefined rules or standards so that TD can be repaid through a reasonable order.
- ***TD monitoring*** aims to keep tracking the changes of the cost and benefit of unresolved TD continuously. Through this way, the development team could find a proper time point to repay TD.
- ***TD repayment*** aims to resolve or mitigate TD in a software system through reengineering, refactoring or other techniques.

The eight TDM activities mentioned above cover the main aspects of TD management comprehensively, and are adopted by many other studies. In this study, we also base our research on these eight activities.

2.4.1 Approaches for technical debt management

For each identified TDM activity, there are different approaches which have been used, proposed, and developed for TDM. Li et al. [3] collected and synthesized approaches adopted for each TDM activity, which covered approaches mostly mentioned in the literature. The extracted approaches are shown in Table 2.1.

Table 2.1 Approaches used for each TDM activity

TDM activity	Approach	Description
TD prevention	Development process improvement	Improving existing development processes to prevent certain types of TD from occurring
	Architecture decision making support	Choosing architecture design options with less chance to incur potential TD according to previous experience.
	Lifecycle cost planning	Planning the cost throughout a system's lifecycle to minimize overall TD
	Human factors analysis	Minimizing the unintentional TD caused by human factors (e.g., indifference and ignorance)
TD identification	Code analysis	Identifying violated coding rules and lacked tests by analyzing the source code. It also identifies design or architecture issues by calculating metrics from the source code.
	Dependency analysis	Analyzing dependencies between software elements (e.g., components, modules) to detect potential TD
	Check list	Creating a list to check predefined scenarios about whether TD is incurred. Development teams could check against this list to identify potential TD.
	Solution comparison	Comparing the adopted solution with the optimal one in several aspects (e.g., cost/benefit ratio). TD is incurred if the adopted solution is not the optimal one.
TD measurement	Calculation model	Using mathematical formulas or models to calculate TD
	Code metrics	Using source code metrics to calculate TD
	Human estimation	Manually estimating TD based on experience and expertise of practitioners
	Cost	Estimating the cost of handling the incurred TD by

	categorization	various categories
	Operational metrics	Using quality metrics of product operation to estimate TD
	Solution comparison	Calculating the deviation between the adopted and the optimal solution
TD documentation	Formatted TD item	Identifying each TD item with a detailed description, including various fields of the TD item (e.g., ID, location, Responsible/author, type, etc.)
TD communication	TD dashboard	Creating a dashboard displaying TD items, types, and amounts to inform stakeholders of the existing TD
	Backlog	Putting all identified TD items into the backlog of the software project, together with issues to be resolved in the development. In the backlog, TD items can be treated as important as bugs and unimplemented features
	Dependency visualization	Visualizing the dependencies between software elements (e.g., components and packages) and identifying the undesirable dependencies (e.g., overly complex dependencies)
	Code metrics visualization	Visualizing code metrics generated by tools, and highlighting elements whose measured quality (e.g., code complexity) is bad
	TD list	Making TD items visible to stakeholders by creating a TD list including all identified TD items
	TD propagation visualization	Showing the connections about how a TD item affects and is affected by other TD items
TD prioritization	Cost/benefit analysis	Repaying a TD item first if resolving this TD item can yield a higher benefit than cost.
	High remediation cost first	Repaying TD items that are costlier to resolve first
	Portfolio approach	Considering TD items, new functionality, and bugs as assets with risks and investment opportunities, and selecting the asset set that can maximize the return on investment or minimize the investment risk
	High interest first	Repaying TD items which incur higher interest first
TD monitoring	Threshold-based approach	Defining thresholds for quality metrics related to TD, and showing warnings with thresholds which are not met
	TD propagation tracking	Tracking the impacts of an identified TD on other parts of a system through checking dependencies
	Planned check	Regularly measuring and tracking the change of the identified TD
	TD monitoring with quality attribute focus	Monitoring the change in quality attributes that can deteriorate the status of TD

	TD plot	Plotting various aggregated measures of TD over time and showing the trends indicated by the curve
TD repayment	Refactoring	Making changes to the code, design, or architecture of software to improve the internal quality without affecting current behaviors
	Rewriting	Rewriting the code that contains TD
	Automation	Automating manually-repeated work like manual tests, manual builds and manual deployment
	Reengineering	Evolving existing software to exhibit new features and improve operational quality
	Repackaging	Grouping cohesive modules to simplify the dependencies among the modules so that the maintainability can be increased.
	Bug fixing	Resolving identified bugs
	Fault tolerance	Strategically adding runtime exceptions in the location of TD

2.4.2 Tools for technical debt management

In order to implement TDM approaches, we need to select proper tools to support the approaches. Different tools have their own usage, working in specific environments and focusing on different aspects, therefore the selection of tools is a crucial issue to consider when managing TD. Li et al. [3] also studied the tools used in TDM, most of which deal with code TD. Table 2.2 shows the identified TDM tools.

Table 2.2 Tools for TDM

Tool	**Functionality**	**TD Type**
SIG Software Analysis Toolkit	Calculating code properties	Code TD
Google CodePro Analytix	Calculating code metrics	Design TD
iPlasma	Calculating code metrics	Design TD
Eclipse Metrics	Calculating code metrics	Design TD
Rational AppScan	Identifying security flaws in source code	Code TD
PMD	Looking for potential problems in source code	Code TD
PHPMD	Detecting mess (e.g., over complicated expressions) in PHP code	Code TD
NDepend	Calculating. NET code metrics	Code TD
NCovcr	Analyzing code coverage for .NET	Test TD
FxCop	Analyzing managed code assemblies to identify compliance issues against .NET programming guidelines	Code TD
CodeXpert	Automating PL/SQL code quality and standards reviews	Code TD
Cobertura	Analyzing code coverage for Java	Test TD
Checkstyle	Checking Java code against coding standards	Code TD
Software maps tool*	Visualizing code quality of source code files	Code TD

RE-KOMBINE	Identifying and measuring requirements TD	Require ments TD
Code Christmas Trees	Visualizing code complexity and coverage	Code TD
CAST's Software's Applications Intelligence Platform	Identifying violations in source code and categorizing the violations by quality attributes	Code TD, architect ural TD
Technical Debt Evaluation (SQALE) plugin for SonarQube*	Analyzing, measuring, visualizing, and prioritizing TD based on SQALE quality model	Code TD
STAN	Calculating the structure quality metrics of Java systems	Design TD
Resource Standard Metrics	Calculating source code metrics and analyzing code quality to find style violations and logic problems	Design TD, Code TD
DebtFlag*	Supporting TDM by maintaining an implementation level representation of TD and providing needed information for project level management; providing an Eclipse Plugin to capture TD by using lightweight documentation tool, and a Web application to manage TD.	Code TD
RBML compliance checker	Calculating a distance between a realization of a design pattern and the intended design	Design TD
A tool to identify bad dependencies	Identifying bad intra- and inter- module dependencies	Architect ural TD, Code TD
Sonar TD plugin*	Identifying and measuring TD in the form of low code coverage, design, violations, complexity, comments	Code TD, Test TD, Docume ntation TD
SonarQube	Open platform for managing code quality	Code TD
SonarQube COBOL Plugin	Performing objective and automated COBOL code reviews against coding best practices	Code TD
CLIO	Identifying modularity violations	Architect ural TD
CodeVizard	Identifying code smells	Design TD
FindBugs	Identifying automatic static analysis issues	Code TD

2.4.3 Strategies for managing TD

When managing TD, appropriate strategies are necessary to support decisions about when and to what extend a TD item should be paid off, because there should be a proper balance between improving productivity and ensuring the health of the

product. Alves et al. [16] studied the strategies followed by different researches and found six most cited strategies, which are shown in Table 2.3. However, the drawback of these researches is that they failed to evaluate the applicability of these strategies.

Table 2.3 TDM strategies

Strategy	Definition
Cost-Benefit Analysis	This strategy evaluates whether the expected interest is high enough to support the decision of paying the debt.
Portfolio Approach	This strategy centers on the list of identified TD items, containing the location, the identification time, the responsible person, the reason why it is considered TD, an estimate of the principal, estimates of the expected interest amount (EIA) and interest standard deviation (ISD), and estimates of the correlations of this item with other TD items.
Options	Investment in paying off the debt is similar to paying for the option that facilitates change to the software, but without immediate profits. The value of the investment is in the form of options, which are used to support decisions of repaying TD
Analytic Hierarchy Process (AHP)	AHP helps to structure a problem, compare alternatives with respect to specified criteria, and determine an overall ranking for each alternative. In TDM, AHP will provide a prioritized ranking of TD items for repayment
Calculation of TD Principle	This strategy follows a defined process to calculate the TD Principal and relate the identified issues to different quality attributes.
Marking of Dependencies and Code Issues	This strategy manages problems and dependencies in the source code. It inserts tags in the source code to mark dependencies that may cause TD, so that the development team can visualize where TD is inserted and thus decide when to pay it based on the involved effort and the availability of project time.

Note that some strategies mentioned above like Cost-Benefit Analysis and Portfolio Approach are also mentioned in TD prioritization approaches. What distinguishes strategies from approaches is that strategies need to "support decisions about when and if a TD item should be paid" [16] to be considered as strategies, while approaches are concrete measures after the decision to pay off a TD item has been made.

3 METHODOLOGY

This chapter deals with the selection of research method to answer the research questions. The motivation for selecting the research method and rejecting other alternative methods is discussed. The design and execution process of the research method is also presented. The remainder parts of this chapter are structured as follows: Section 3.1 presents the rationale behind selecting case study and rejecting other alternative research methods. Section 3.2 presents case selection strategy and unit of analysis. Section 3.3 presents subject selection. Section 3.4 presents data collection methods selection and execution process. Section 3.5 presents data analysis methods selection and analysis process.

3.1 Research methods selection

According to [17], four research methodologies are most relevant to software engineering: controlled experiments, case study, survey and action research. Among these four methods, case study was chosen as the most suitable research method for this study. The reasons for selecting case study and rejecting other alternative methods are given below.

Case study is "*an empirical inquiry that investigates a contemporary phenomenon within its real-life context, especially when the boundaries between phenomenon and context are not clearly evident.*" [17] There are four types of case study which are exploratory, descriptive, explanatory, and improving case study [18]. This paper adopted exploratory case study to conduct the research. Exploratory case study aims to "*find out what is happening, seek new insight, and generating ideas and hypothesis for new research.*" [18] Although many studies have been reported with respect to TDM, we still have less knowledge about TDM in large-scale distributed projects. It is necessary to understand the current situation and explore new knowledge of TDM with its real-life context (large-scale distributed project). Furthermore, case study involves a list of different data collection methods (e.g. questionnaire, interview, archival records, etc.). The advantage of this is being able to gather data from different sources, which gives us a comprehensive understanding of TDM and strengthen the validity of the research.

A controlled experiment is "*an investigation of a testable hypothesis where one or more independent variables are manipulated to measure their effect on one or more dependent variables."* [17] It allows researchers to find out whether a cause-effect relationship exists between strictly controlled variables. However, the main objective of this study is to explore how practitioners manage TD in large-scale distributed project. Even though exploring the relationship between related factors and TD is also a sub-objective of this paper, we are not able to control these independent variables strictly. For example, it is not practical to ask a team with a high maturity to develop a simple task to see the changes in the amount of TD. So, we realized that this method was not adequate for our investigation.

Survey research is "*used to identify the characteristics of a broad population of individuals."* [17] It is most closely associated with the use of questionnaires for data collection and can also be conducted by using structured interviews [17]. Researchers have a high risk to get no answer or inaccurate answer, and a low response rate. Survey is a good choice when we are concerned with discovering what is the truth in general [17]. On the contrary, in this case, we wish to gain a deeper insight into how practitioners manage TD in one company with its specific environment. The TDM can

vary from one company to another. So, we realized that this method was not adequate for our investigation.

Action research "*attempt to solve a real-world problem while simultaneously studying the experience of solving the problem*." [17] Action researcher aims to intervene in the studied situations for the explicit purpose of improving the situation [17]. The researchers of the study will not make any interventions to the studied situation or tend to solve real-world problems that they are facing, but will explore how they cope with TDM in practice from an objective perspective. So, we realized that this method was not adequate for our investigation.

3.2 Case selection strategy and unit of analysis

A case is selected to explore the way that practitioners used to manage TD in large-scale distributed project. We choose the case from companies which have complete large-scale distributed projects, and the case itself should be a large-scale distributed project.

There are two main categories of sampling technique which are probabilistic and non-probabilistic [17]. Probabilistic techniques include random stratified sampling and simple random sampling. Non-probabilistic techniques include convenience sampling, quota sampling, judgment sampling and snowball sampling. The case selection strategy we used is convenience sampling, which allows us to choose cases from the industrial collaboration network that our research can gain access to, based on mutual trust [18]. In this case study, the case is a large-scale distributed project of Ericsson as a part of a large system, which was launched in 1997 and development teams are still working on this project now. Ericsson is a global software developing company with many branch offices around the world, and the project in this case has distributed teams in Sweden, India, USA, Italy and Poland. This case is selected because our supervisor has other research with Ericsson, so that he can introduce our research to Ericsson and help us gain access to this project.

3.3 Subjects selection

We also used convenience sampling strategy to select the subjects for the semi-structured interview. Convenience sampling suggests inviting the nearest and most convenient persons to take part in the interview [17]. In order to get a deep understanding of our research topic, the subjects should have experience related to large-scale distributed software development. In this research, it is convenient to select and access the subject in Ericsson who has a working relationship with our supervisor and has experience of large-scale distributed project development.

In the planning phase of case study, we aimed to interview several people in this project including several roles (project managers, architects, developers, etc.) so that we can have a comprehensive understanding of TDM in this project from many aspects. But due to staff availability and time constraints, at last we were only able to interview one subject from Ericsson who is the chief architect of the software architect team (SAT) of this studied project. He is also the team leader of SAT in the last years. The SAT takes requirements for new functionality and is also responsible for keeping the architecture within the node to guarantee maintenance of the product from the long-term perspective. He was responsible for the TDM in the project for about 18 years. Therefore, with a thorough understanding of the TDM process in this project, this architect is the proper subject for the interview and can provide us with explicit and convincing opinions about this project.

3.4 Data collection

3.4.1 Data collection method selection

The most common methods for data collection of case study includes: interview, questionnaire, archival record, focus group and observation [18]. In this research, we plan to conduct semi-structured interviews (a sub-type of interview) and archival records instead of the other three methods due to the following reasons:

- The research questions need to be answered qualitatively, while questionnaire is suitable to answer question quantitatively.
- Interviews have higher response rate than questionnaires and enable more in-depth data collection, since interviewer can observe and ask questions, and generally decrease the number of "do not know" and "no answer".
- The archival records include different kinds of documents which is able to help us to collect data from different perspectives of a project. In this case, we can use the archival documents to analyze the factors that have impacts on TD.
- Focus group is not realistic because we cannot guarantee to meet all the subjects at the same time.
- Observation is not considered mainly due to the fact that it takes a lot of time, effort and money that beyond our budget.

3.4.2 Data collection process

3.4.2.1 Archival records

A spreadsheet that aggregated relevant data was developed as the result of data collection conduct in other investigation [19] [20]. After signing the confidentiality agreement, we were allowed to use the data for academic purposes. The archival data was collected from 33 product customization tasks (PCs) of this studied project between 2013 and 2016. Six variables are included in this document, which are *technical debt, task complexity, lead time, global distance, total developers,* and *maturity*. The 33 values of each variable were collected and calculated by the authors of [19] [20]. Therefore, it is convenient to use these data directly for our research. The variables listed above are described as follows:

Dependent variable:

- *Technical debt*: technical debt in this case was calculated by using SonarQube, which is the amount of dollars needed to fix all problems (*duplications, violations, comments, coverage, complexity, bugs, bad design*) in the code base [21].

Independent variables:

- *Task complexity*: the parameter is used to describe how complex the task is, which is measured by the unit manager and a group of product-level software architects. They selected one PC as the baseline and used planning poker to measure others. each PC was estimated by a positive integer (complexity points) [20].
- *Lead time*: the total time needed to deliver a task, counted by days.
- *Global distance*: the metric that measures the complexity of communication between sites, which represents the overhead of cooperation and coordination when more than one site is involved [22].
- *Maturity*: the parameter to describe the level of how a team can deliver the product independently.

- *Total developers*: the number of developers involved in the development of each task.

3.4.2.2 Semi-structured interview

The interview instrument containing a list of questions was carefully designed before conducting the interview. These questions are open-ended, which gives interviewee less limitation and makes them feel free to express what they think explicitly. The interview questions can be found in Appendix A.

Before the formal interview was performed, we conducted one pilot test with other researchers who had some experience with software development in large-scale distributed project. The pilot test was used to evaluate the understandability of our interview instrument and to decide a proper interview procedure. Based on the feedback of subjects, we modify and polish the interview questions, and delete several unnecessary questions. The interview was planned to start with the general questions and then move to the specific ones.

The interview was conducted by the two authors of this paper. One interviewer was responsible for interviewing and the other was responsible for taking notes. The interview totally lasted about 90 minutes. The whole conversation was recorded using a voice recording equipment and transcribed into text for further analysis. The detailed interview process is shown in Figure 3.1.

As we can see in Figure 3.1, the interview was composed of 7 steps:

- Step 1: We introduced ourselves and asked for the permission to record the whole conversation.
- Step 2: We interpreted the aim and objectives of this research.
- Step 3: We asked the interviewee to tell us some background information in terms of this project and his responsibility in this project. These questions were easy to answer and were bedding for the further discussion of TDM.
- Step 4: The questions in this step were more specific. We firstly let him interpreted his understanding of TD. Then we moved forward to discuss TDM in this project. The TDM activities were not discussed in a fixed order. Instead, we let the interviewee interpret TDM in his own way. There are several reasons for that: 1) to avoid the interviewee feeling annoyed; 2) to avoid giving him any implication. However, the discussion should cover all questions. If we did not find any answer to a specific question (e.g., the people who are responsible for TD measurement are not mentioned), then we explicitly asked the interviewee regarding that part.
- Step 5: We checked if any TDM activities were not mentioned during the discussion. For example, in the interview, TD monitoring was ignored by the interviewee, then we presented the whole TDM process to the architect.
- Step 6: Discussed the ignored TDM activity to figure out the reason why it was overlooked and the consequence of not conducting this TDM activity.
- Step 7: We displayed the hierarchical multiple regression results to the interviewee, and let him interpret his supposed relationship between the identified significantly influential factors and TD from his own perspective.
- Step 8: Expressed thanks to the interviewee.

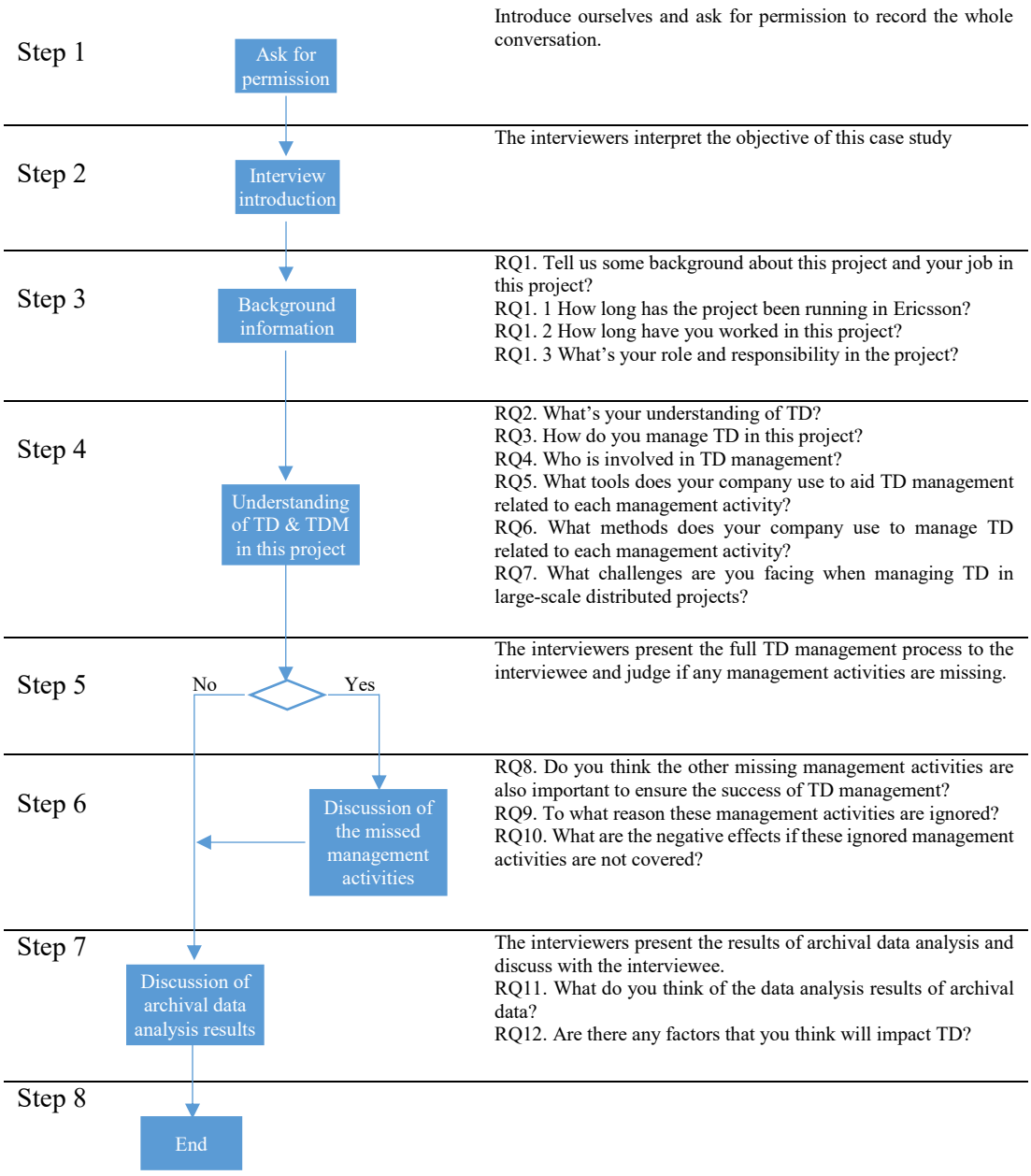

Figure 3.1 Interview process

3.5 Data analysis

3.5.1 Data analysis method selection

The results of the semi-structured interview were analyzed by using content analysis, since it is a systematic and rule-guided technique used for analyzing all sorts of textual data. Particularly, it is useful for analyzing interview transcription [23]. Content analysis provides a way to distill words into a few content-related categories, which provides a condensed and broad description of the phenomenon and allows researchers to enhance the understanding of raw data [24]. Besides, content analysis is also a suitable method when we want to develop an understanding of a communication and identify essential process [25]. It focuses more on the meanings, intentions, consequences, and context [26]. Therefore, content analysis is a proper method to help us analyze the interview transcription and understand TDM in the context of large-scale distributed project, and finally generate a complete process of TDM.

We conducted a hierarchical multiple regression analysis to analyze the archival data. Hierarchical multiple regression is one kind of multiple regression, which aims to assess the contribution of a specific variable or a set of variables to the response variable [27]. That is, identifying independent relationships [27]. It is more flexible and allows researchers to specify the order of the entry of the independent variables in the regression equation [27]. Therefore, each independent variable can be assessed in terms of its additional explanatory power when it is entered into the equation. Hierarchical multiple regression is an improved version of standard multiple regression, while the full models of both methods are the same, and it also can be used to find the best prediction equation for a given set of variables as standards multiple regression does. According to the above description, hierarchical multiple regression is a suitable method to analyze archival data, so we could explore which factor is a significant predictor of TD, and generate the best prediction equation with the five variables (*task complexity, lead time, global distance, maturity, and total developers*) from the archival document.

3.5.2 Data analysis process

3.5.2.1 Hierarchical multiple regression analysis

This section presents the process of conducting hierarchical multiple regression analysis. As we can see in Figure 3.2, it starts with detecting unusual points, and the unusual points are removed after discussion between us. Then, all variables are normalized. After the data is processed, we conduct hierarchical multiple regression in (Statistical Package of Social Science) SPSS. Four regression assumptions (linearity, independence, homoscedasticity, and normality) are tested with the output from SPSS. We also check the multicollinearity between variables and finally report the results.

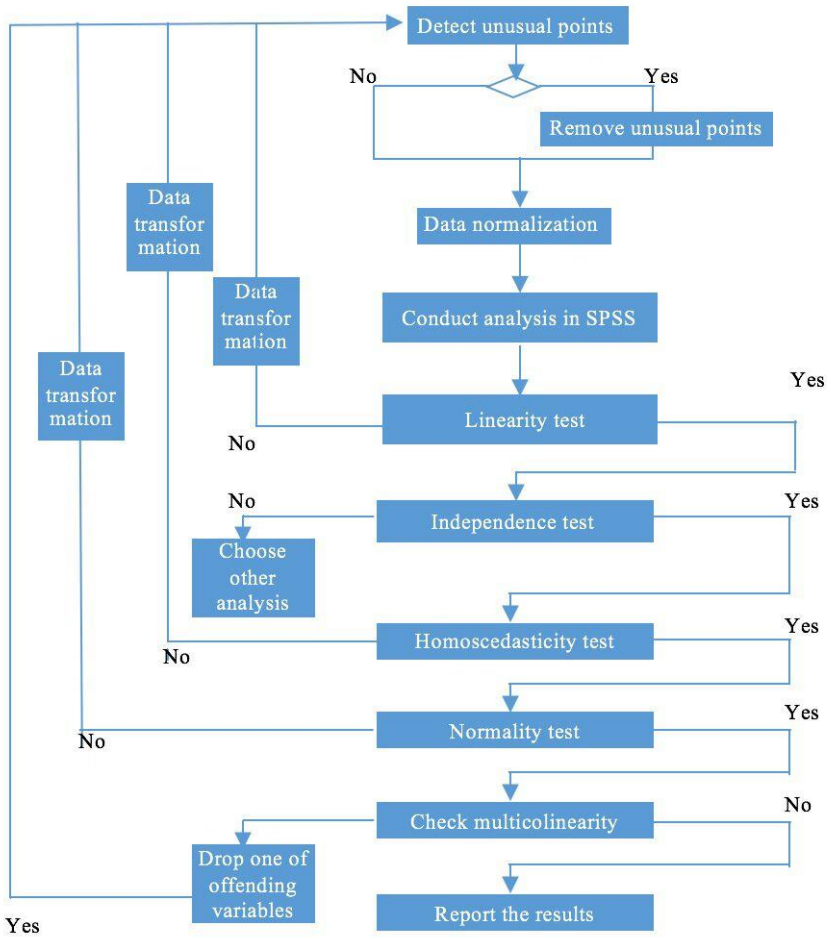

Figure 3.2 Hierarchical multiple regression analysis process

A. Checking unusual points

There are three kinds of unusual points: outliers, high leverage points and highly influential points [28]. We use Studentized deleted residual (SDR), Leverage value (LEV), and Cook's distance (COO) to detect the outliers, high leverage points, and highly influential points separately. After checking, only outliers and high leverage points are identified, and no highly influential points are identified. After discussion, these nine unusual points were removed from the data set and 24 cases were left. The details of unusual points are shown in Table 3.1.

Table 3.1 Checking for unusual points

Category	Exclusive criteria	Case ID	Value
Outliers	SDR>3 or SDR<-3	13	5.15621
High leverage points	LEV>0.2	2	.77965
		31	.75911
		1	.37636
		9	.34120
		30	.32983
		25	.31498
		24	.26779
		27	.24127

B. Data normalization

Data normalization is necessary when the scale of data is not at the same level. In this multiple regression analysis, it is obvious that the scale is different among different variables (e.g. the values of *technical debt* variable were mostly more than 1000, while the value of *maturity* was ranging from 0 to 4). Therefore, we normalized the data range from 0 to 1 by using min-max normalization method which also preserved the relationship among the original data values [29]. The formula of this min-max method as shown below:

$$X_{new} = (X - X_{min}) / (X_{max} - X_{min})$$

Where X means the variable under transformed. X_{new} means the variable that after transformed. X_{min} and X_{max} represents the smallest and biggest value of this variable in the data set.

C. Conduct hierarchical multiple regression analysis in SPSS

These data of each variable were inputted into SPSS version 23 and the template formula of final regression model for this analysis is given as below:

$$Y = B_0 + B_1X_1 + B_2X_2 + B_3X_3 + B_4X_4 + B_5X_5$$

The independent variables were divided into five groups (group1: *task complexity, group2: lead time, group3: global distance, group4: total developers, group5: maturity*). Each regression model is a different combination of independent variables to be used for one multiple regression. In the beginning, the independent variable in the first group was entered into the regression model. Then a second multiple regression was done with a new group of independent variable together with the independent variable(s) in the previous step. This process was iterated until all the remaining independent variables were entered into the regression model.

D. Testing linearity

The partial regression plots are used to check the linear relationship between dependent variable and each independent variable [30]. Visually, from these regression plots in Figure 3.3, *task complexity, global distance, maturity, and total developers* shows a significant linear relationship with *technical debt. Lead time* shows some kind of linear relationship with *technical debt.*

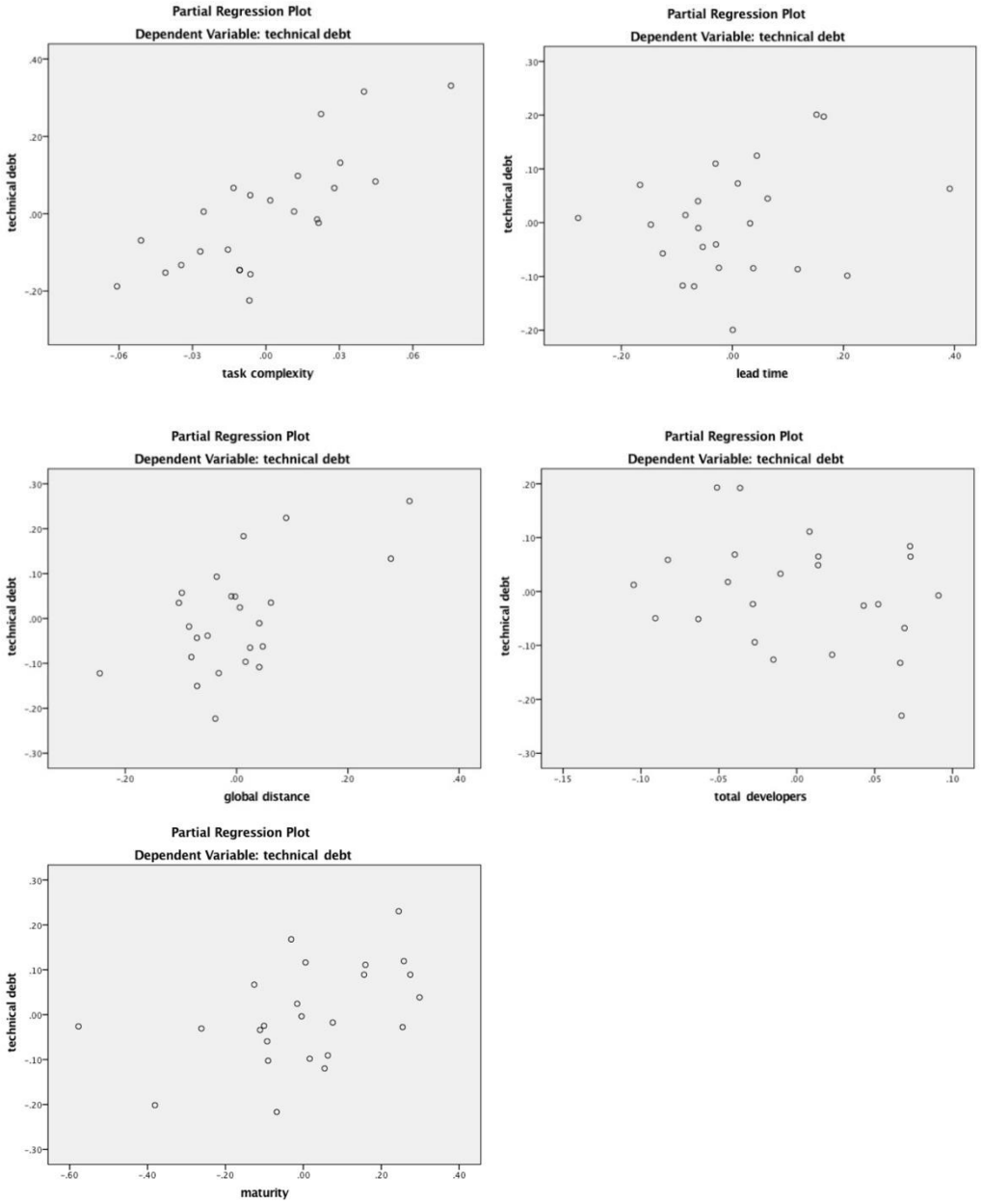

Figure 3.3 Regression partial plots of independent variables against technical debt

E. Testing independence

It is assumed that observations between and within groups are independent. Independence of residuals was checked by analyzing Durbin-Watson statistics. If the value of Durbin-Watson test is between 1.5 and 2.5, there is no linear autocorrelation in the data [27]. The Durbin-Watson value in our test is 2.332 (seeing Table 3.2), which is acceptable. So, there is independence of residuals in our data.

Table 3.2 Durbin-Watson test

Durbin-Watson test
2.332

F. Testing homoscedasticity

The assumption of homoscedasticity is that the residuals are equal for all values of the predicted dependent variable. To check for heteroscedasticity, we

conducted the Breusch-Pagan and the Koenker test [31]. The results of the tests are shown in Table 3.3. As we can see, the sig-value of both tests are not less than 0.05. So, we cannot reject the null hypothesis (heteroskedasticity not present), which means that the assumption of homoscedasticity has been met.

Table 3.3 Breusch-Pagan and Koenker test

Breusch-Pagan and Koenker test statistics and sig-values		
	LM	Sig
Breusch-Pagan	6.813	.235
Koenker	10.160	.071

G. Testing normality

The normal distribution of residuals is tested by visually checking the normal P-P plot [27]. In Figure 3.4, the points on the plot remain close to the diagonal line, which means residuals are normally distributed. So, we do not violate the assumption of normality.

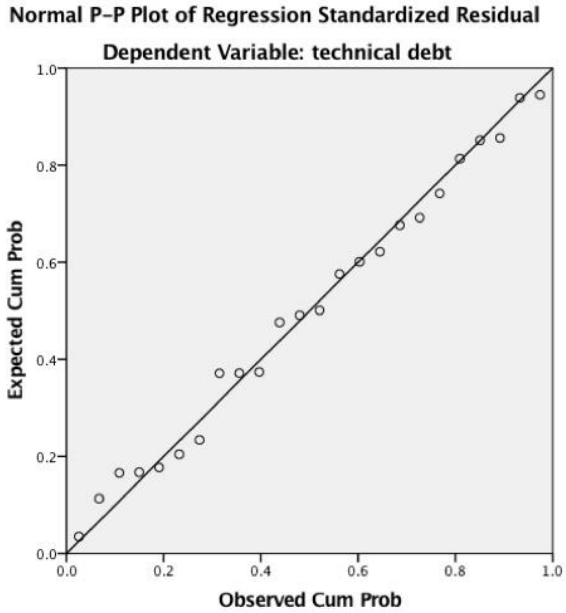

Figure 3.4 Normal P-P plot of regression standardized residual

H. Testing multicollinearity

We used Tolerance/VIF values to check multicollinearity [32]. The tolerance of independent variables should be greater than 0.1 for there to be no multicollinearity. In addition, the VIF should be less than 10. As we can see in Table 3.4, the tolerance values in our study are all greater than 0.1 and the VIF values all less than 10. Therefore, there was no multicollinearity issue in this analysis.

Table 3.4 Tolerance/VIF values of the variables in five regression models

Variables	Technical debt									
	Model 1		Model 2		Model 3		Model 4		Model 5	
	T	VIF	T	VIF	T	VIF	T	VIF	T	VIF
Task complexity	1.000	1.000	.681	1.469	.671	1.490	.640	1.562	.333	3.006
Lead time			.681	1.469	.620	1.613	.524	1.909	.521	1.919
Global distance					.907	1.103	.893	1.120	.697	1.434
Total developers							.665	1.505	.360	2.780

Maturity									.362	2.760

Note: T means tolerance

3.5.2.2 Content analysis

According to Elo and Kyngäs [33], content analysis is conducted through three phases which include preparing, organizing and reporting. Each phase consists of several sub-activities. The detailed analysis process is shown in Figure 3.5. Based on the research questions, we divided the content analysis into two independent parts. The first part deals with the text related to TDM and the second part deals with the text about the factors that have impacts on TD.

In the first part, the analysis started with the definition of unit analysis. We treated each TDM activity as a recording unit. Then, we made sense of the raw data and defined five coding categories: TDM activity, tools, methods, roles, and challenges. Before coding the text, we coded a sample text to test the clarity and consistency of the category definitions, and got an inter-coder agreement. Next, we coded the whole text, totally eight TDM activities were identified. The tools, methods, involved roles, and challenges associated with each TDM activity were listed in Table 4.3. After the above steps, we rechecked the coding consistency. These five coding categories were suitable for coding our interview transcript, and all necessary data was labeled with corresponding categories. Finally, we reported the analysis results.

The second part followed the same content analysis process except that the definition of unit analysis and categories are different. The unit of analysis is each factor that has impacts on TD and the categories were defined as factor and impact.

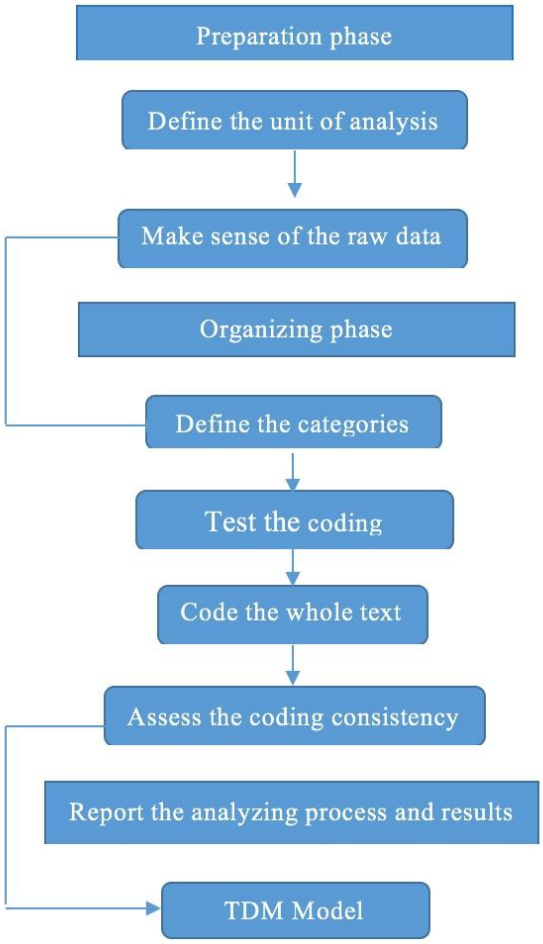

Figure 3.5 The process of content analysis

4 RESULTS AND ANALYSIS

This chapter deals with results presentation and analysis. The results were generated from hierarchical multiple regression analysis and content analysis. The results of hierarchical multiple regression analysis are used to answer RQ1. The content analysis results are divided into two parts to answer RQ1 and RQ2 separately. The remainder parts of this chapter are structured as follows: Section 4.1 presents and analyzes the results related to factors that have impacts on TD to answer RQ1. Section 4.2 presents and analyzes the results regarding TDM in large-scale distributed project to answer RQ2.

4.1 Results for RQ1

To analyze the relationship between relative factors and TD in large-scale distributed project. We firstly analyzed the archival records, five factors were entered into regression model sequentially. The factor included in each model can be seen in the left column of Table 4.1. We also list the relative statistics used for evaluating the goodness of each model to predict *technical debt*, as well as the statistics for analyzing the statistical significance of each factor for predicting *technical debt*. The identified significant factors (*task complexity, global distance, maturity*) were discussed during the semi-structured interview and recognized by the interviewee. We did not get any new factors that may have significant impacts on TD. The results of identified factors and their impacts are listed in Table 4.2. We will give a detailed analysis of these results in the following sub-section.

Table 4.1 Summary of hierarchical regression analysis

Variables	Technical debt														
	Model 1			Model 2			Model 3			Model 4			Model 5		
	B	Beta	t	B	Beta	t	B	Beta	t	B	Beta	t	B	Beta	t
Constant	-.014		-.354	-.037*		-.738	-.303*		-2.294	-.306*		-2.267	-.605*		-3.379
Task complexity	2.962**	.799	6.224	2.720**	.734	4.667	2.582**	.696	4.763	2.524**	.681	4.459	3.648**	.984	5.135
Lead time				.124	.115	.732	.230	.213	1.398	.195	.181	1.070	.168	.155	1.012
Global distance							.409*	.270	2.150	.397*	.263	2.033	.611*	.404	3.054
Total developers										.155	.072	.482	-.457	-.213	-1.154
Maturity													.248*	.419	2.283
R	.799			.804			.844			.847			.883		
R^2	.638			.647			.713			.717			.780		
Adjust R^2	.621			.613			.670			.657			.719		
ΔR^2	.638			.009			.066			.003			.064		
ΔF	38.737**			.536			4.624*			.232			5.214*		
df_1	1			1			1			1			1		
df_2	22			21			20			19			18		
F	38.737**			19.228**			16.572**			12.010**			12.782**		

Note: * $p < 0.05$, ** $p < 0.01$

Table 4.2 Results of content analysis with respect to factors that have impact on TD

ID	Factor	Impact
1	Task complexity	Positive
2	Global distance	Positive
3	Maturity	Negative

4.1.1.1 Relationship between factors and TD

Looking at Table 4.1, since we divided the five factors into five groups, so we totally got five regression models (the details about how the factor in each group was entered into the regression model has been described in section 3.5.2.1). These fives model can be used to predict TD and the best predicting model will be achieved through the following analysis.

In **Model 1** with *task complexity* as the predictor of TD, the R value is 0.799, thus a strong relationship exists between predictor variable and TD. The R^2 (0.638) is significant at F (1, 22) = 38.737, $p < 0.01$, since it could account for 63.8% of the variance. The results show that *task complexity* is a significant predictor of TD.

In **Model 2** with two predictor variables (*task complexity and lead time*), the R value is 0.804 and the R^2 is 0.647, thus 64.7 % of the variance have been accounted for. The change in R^2 is not significant with F (1, 21) = 0.536, $p > 0.05$. This means that entry of the lead time does not increase the explained variance in TD.

In **Model 3** with three predictor variables (*task complexity, lead time and global distance*), the R value is 0.844 and the R^2 is 0.713, thus 71.3% of the variance is accounted for. The R^2 change is significant at F (1, 20) = 4.624, $p < 0.05$. This means that entry of *global distance* increases the explained variance in TD by 6.6% to a total of 71.3%.

In **Model 4** with four predictor variables (*task complexity, lead time, global distance and total developers*), the R value is 0.847 and the R^2 is 0.717, thus 71.7% of the variance is accounted for. The R^2 change is not significant by the F change test, with F (1, 19) = 0.232, $p > 0.05$. This means that entry of the total developers does not increase the explained variance in TD.

Model 5 is the final model with five predictor variables (*task complexity, lead time, global distance, total developers, and maturity*), in which the R value is 0.883 and the R^2 is 0.780, thus 78% of the variance is accounted for. The R^2 change is significant at F (1, 18) = 5.214, $p < 0.05$, which means that entry of the maturity increases the explained variance in TD.

We also list the significance of five models respectively (seeing Table 4.3). As we can see, all five models are significant with $p < 0.01$. Through checking the F value which represents the overall predictive effects, the first model has the largest F value.

From Table 4.2, the coefficients for the constant and all five predictors of TD are as follows: Constant = - 0.605, t = - 3.379, $p < 0.05$ (significant); *task complexity*, $\beta = 3.648$, t = 5.135, $p < 0.01$(significant); *lead time*, $\beta = 0.168$, t=1.012, $p > 0.05$ (not significant); *global distance*, $\beta = 0.611$, t = 3.054, $p < 0.05$ (significant); *total developers*, $\beta = -0.02$, t = - 1.154, $p > 0.05$ (not significant); *maturity*, $\beta = 0.62$, t = 2.283, $p < 0.05$ (significant). Therefore, the best fitting model for predicting TD from the analysis above would be the linear combination of the constant, *task complexity*, *global distance* and *maturity*. The model is shown as below:

$$Y \text{ (technical debt)} = \beta_0 + \beta_1 \text{ (task complexity)} + \beta_2 \text{ (global distance)} + \beta_3 \text{ (maturity)}$$

Where, $\beta_0, \beta_1, \beta_2, \beta_3$ are respectively – 0.605, 3.648, 0.611, 0.248.

As we can see, all these three factors show a significant positive correlation with TD. Thus, the more complicated the task is, the more TD may be incurred during

the development process. The increase of *global distance* will also lead to higher TD. It is surprising that data shows that teams with higher *maturity* tend to create more TD. This finding needs further discussion.

During the interview, when we showed the hierarchical multiple regression results to the interviewee, he believed that among all five factors, *task complexity*, *global distance and maturity* have significant relationships with TD.

For *task complexity*, the architect suggested that *task complexity* has a strong relation to TD, complicated tasks tend to associate with higher debts. However, it was hard for the architect to judge exactly what can be seen as a complicated task, since the boundary is not clear for him. For example, the architect expressed his confusion as: "*what is a complex task is hard to say, when a task contains a lot of lines of code, but from the functional perspective, it is very easy to build and will not create any debts at all, do we still think it has low complexity?*" So, the metrics used for measuring *task complexity* can vary between different people. Actually, the *task complexity* in the archival records was also a subjectively rough estimation based on experience. However, the architect believed the amount of code will contribute to *task complexity,* and the percentage of debts will increase correspondently as the amount of code increases.

For *global distance,* the architect thought it is possible that more TD will be incurred when a project is developed by distributed teams, because the cooperation between teams can be more difficult in such condition. Due to different time zones, it is hard to guarantee that people in different sites follow the same schedule, which also creates communication barriers among distributed teams. As he said: "*The worst case is that people working with the same functionality are sitting in different places and doing different phases of the work.*" On the contrary, if the product designer and developers sit close to each other, it will be easier for them to keep smooth communication and establish a tight relationship, and the amount of TD can be decreased efficiently.

For *maturity,* the architect emphasized that *maturity* can affect TD significantly. A mature team tends to have higher performance, and the team members care more about the project from long-term perspective instead of short-term interests. They normally have a better technical competence and stronger awareness of self-control. A higher quality product can be produced with less TD in such teams. However, the response from the architect is against what we found in the hierarchical multiple regression analysis. After matching the data of these factors, we find that the data of *maturity* in this data set may be not fair enough, since most complex tasks were carried out by teams with high maturity, while complex tasks are normally associated with more TDs. This is the reason why we got the opposite conclusion related to *maturity*. So more comprehensive data is needed to verify the impact of *maturity* on TD. Another reason may be as that teams with high maturity may be able to identify more TDs, since they are more skilled and have high ability. That is, the amount of debts hidden in the product may not increase, but higher proportion of TD can be found by people from mature team.

4.2 Results for RQ2

This section presents the results obtained from the semi-structured interview regarding TDM in large-scale distributed projects. According to the defined categories of content analysis. In Table 4.3, we list the TDM activities, corresponding tools, and approach used for each activity in the studied case. We also list the roles involved in

each TDM activity and the associated challenge. These results will be analyzed to answer RQ1 in following sub-sections.

Table 4.3 Results of content analysis with respect to TDM

ID	TDM activity	Tool	Approach	Role	Challenge
1	TD prevention	X	Coding standards, code review	Developers, architects	X
2	TD identification	X	Expert judgment	Everyone involved in the project	X
3	TD measurement	X	Expert judgment	The one who identified the TD	It is difficult to quantify the cost and benefit of fixing TD
4	TD documentation	Wiki-page	Formats of TD items	The one who identified the TD	X
5	TD communication	Wiki-page	Backlog, TD list	Product community	X
6	TD prioritization	X	Expert judgment	Chief architect	The conflict between TD and other things like new functionality.
7	TD monitoring	N	N	N	N
8	TD repayment	X	Refactoring	SAT, FBT, DMT, CT	X

Note: N means overlooked by the practitioner.
X in **Tool** column means the corresponding TDM activity was not supported by the tool.
X in **Challenge** column means there is no challenge with respect to this TDM activity.
Acronyms: SAT - Software Architect Team; FBT - Functionality Building Team;
DMT - Design Maintenance Team; CT - Customization Team.

4.2.1 Results analysis

This section deals with the analysis of the results in Table 4.3. We give a brief description of the whole TDM process in section 4.2.1.1, then each TDM activities will be deeply analyzed in section 4.2.1.2. Finally, a full TDM concept model of this studied case is summarized in section 4.2.1.3.

4.2.1.1 Brief overview of TDM process

Based on the subject's description, the whole TDM process in this project can be summarized as the process in Figure 4.1. It starts with TD prevention, which strives to avoid TD issues at the early phase. The next step is TD identification. People involved in the project identify the potential TD based on their experience. After identifying TD, they write a node improvement proposal (NIP) about how to fix the TD and submit it to a wiki page system which is an internally implemented tool used for TDM. The new NIPs are discussed every second week within the product community steered by the chief architect (the interviewee). The community members evaluate the new NIPs and decide which one would be put into the TD priority list or rejected due to some reasons. TDs are prioritized by the chief architect after the community meeting. Then the TDs would be repaid by the teams that have free space.

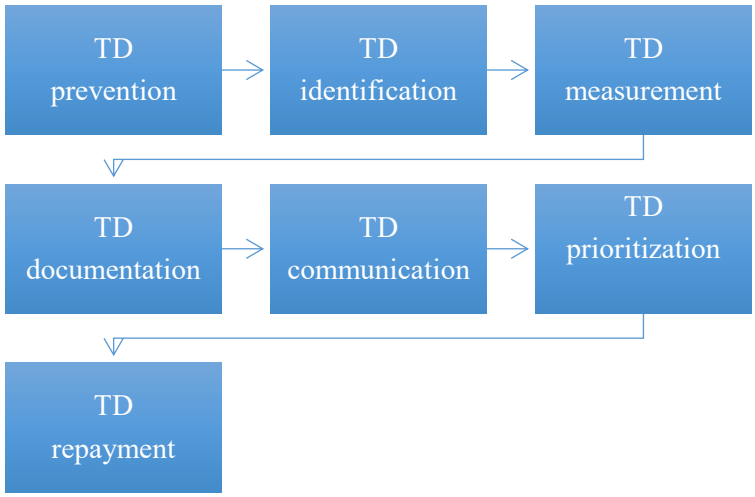

Figure 4.1 The whole process of TD management in Ericsson

In Figure 4.2, we matched the identified TDM activities from literature with what we got in this interview. TDM activities in the blue rectangles are the ones that explicitly included in this case, and the activity in the gray rectangle means it is overlooked during management. It is obvious that they did not cover all the eight identified TD management activities. *TD prevention, identification, measurement, documentation, communication, prioritization,* and *repayment* were mentioned by the interviewee. TD monitoring did not get any attention during the whole TDM process.

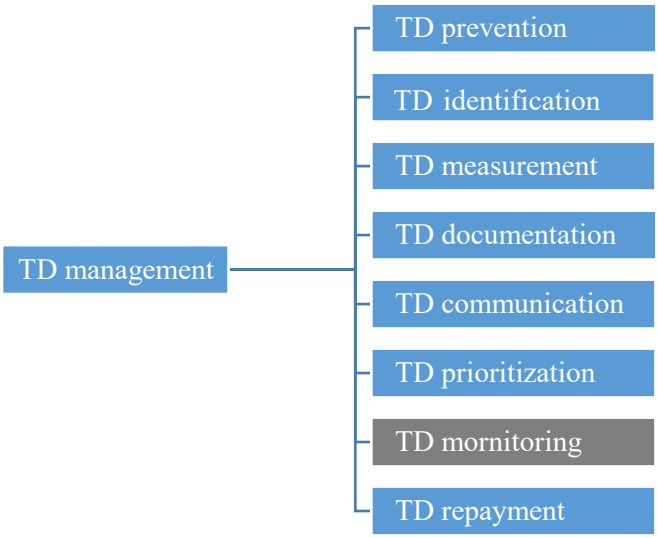

Figure 4.2 Matching the identified TDM activities from practice and literature

4.2.1.2 TDM activity

In this section, each TDM activity is analyzed through several aspects which include the description, roles involved, tools and approaches used, and associated challenges.

A. TD prevention

Description	Prevent potential TD from being incurred
Role	Developers, architects
Tool	Not supported by tool

Approach	Coding standards, code review
Challenge	No challenge

TD prevention is about taking actions to prevent potential TD being incurred. As the architect said: "*From my point of view, we need to do the things right at the beginning to avoid potential debts.*" In this case, TD was prevented by adopting coding standards and code review, coding standards were not always followed strictly by developers. The degree of developers following the coding standards depends on the program push. When the left time was very short before the release deadline, it was difficult to follow a specific rule and time-to-market would take a higher priority. Code review occasionally happens if the author were not confident enough regarding some parts of the code. The review can be done by another skilled developer who has free space. Architects will also review the code to ensure the product within its designed architecture. Especially, for immature teams, architects spent more efforts on code review to detect TD. Tools were not adopted for TD prevention. The architect also did not mention any challenges associated with TD prevention in this project.

B. TD identification

Description	Detect TD caused by intentional or unintentional technical decisions in a software system
Role	Everyone involved in this project
Tool	Not supported by tool
Approach	Expert judgment
Challenge	No challenge

TD identification is about detecting the debt caused by intentional or unintentional technical decisions. Normally, it was triggered by line-manager and program push. Sometimes, line-manager will motivate development teams to write some NIPs. Then team members will start to think about the potential debts that they owed the project. Program push means when some tasks must be finished before a specific deadline, then developers make shortcuts during the development, at this stage, they realized that this technical decision will cause a debt here and wrote a NIP immediately. Besides, people also identify some TDs triggered by themselves, when they think a technical issue will be risky if it continuously gets bigger, then they will document a NIP to let the manager know there was a TD in this part. Everyone in this project can identify TD and write a NIP. However, it was mainly done by the developers during the development. That is, when the code base was developed, developers will check the potential TDs associated with the code, e.g., code complexity.

There are no tools being used for TD identification. Therefore, people identify TD based on their experience and personal preference. They reported a TD that they thought it should be. Generally, a small TD will be solved without writing a formal NIP, so the manager even did not know the existence of that TD. Another thing which should be emphasized is that they did not have a clear classification of TD. Different types of TD are put together and not labeled with TD type (seeing Table 4.2). When we showed the TD classification tree to the architect, he clearly emphasized that they did not manage TD in such details. The architect did not mention any challenges

associated with TD identification. However, lack of a clear classification of TD type is an obvious shortage regarding TD identification.

C. TD measurement

Description	Quantify the benefit and cost of identified TD in a software or estimate the overall accumulation level of TD
Role	TD was measured by the one who identified it
Tool	Not supported by tool
Approach	Expert judgment
Challenge	It is difficult to quantify the cost and benefit, especially transform TD into economic consequence

TD measurement is about quantifying the cost and benefit of identified TD as well as the overall level of TD accumulation in a project. In this case, TD was measured by the one who identified it. Generally, it is measured in man-days and the cost is just a guess based on experience or intuition. Tools were not adopted for TD measurement. The architect stressed that it is difficult to give an accurate estimation of the cost until the TD was almost fixed. It is even more difficult to transform technical problems into economic consequence. This is a clear challenge mentioned by the architect. Besides, the benefit of fixing or the loss of not fixing a TD was not measured. Instead, they only described the advantages and disadvantages semantically of solving a TD. For example, "*the advantage is increasing the maintainability of the product and the disadvantage is that it costs a lot of resources to fix this TD."* They also lack measuring the overall level of TD accumulation. That is, adding up the cost of fixing all types of TD to get a sum, this number shows the overall health conditions of the product and reminds manager to take actions on TD. Managers should pay more attention to these overlooked measurement aspects.

C. TD documentation/presentation

Description	Provide a way to represent and codify TD in a uniform manner addressing the concerns of particular stakeholders.
Role	TD was documented by the one who identified it
Tool	Wiki page, an internally implemented tool for TDM
Approach	Format of TD items
Challenge	Not challenge

TD documentation provides a way to codify and represent TD through a uniform format, which makes TD easy to track and further management. In this case, TD was documented by the one who identified it. To make it visible to relevant stakeholders, they document the TD through a uniform format supported by wiki page. Each TD item was represented with several fields such as ID, name, location, etc. (seeing Table 4.4). These properties are useful inputs for other activities. When a developer chose a TD from the priority list to fix, s/he could quickly get an overview of this debt and know exactly where it was, then took actions to mitigate it according to the suggested improvement solution. The architect was optimistic about the way

they used for TD documentation. Therefore, no challenge was mentioned respect to this activity.

Table 4.4 TD documentation format

Field	Description
ID	A unique identifier for a TD item
Slogan	The name of NIP which fixes the TD
Author	The author of the NIP
Improvement Cost	The cost to conduct TD fixing
Last changed	The time point when the NIP is last changed, indicating the freshness of TD
Status	The status of the NIP (Category: finished, ready, rejected, not done, updated)
Priority	The priority decided after discussion
Source	The location of the identified TD item in source code
Planned Release(s)	The planned release time of the task which contains the TD item
Description	The details of the TD item and its improvement solution
Pros/cons	The advantages and disadvantages of conducting the improvement
Decision log	The decision of whether the NIP is rejected. If rejected, there will be the reason for rejection here

E. TD communication

Description	Make identified TD visible to stakeholders so that it can be discussed and further managed
Role	Software architect, program manager, line manager
Tool	Wiki page, an internally implemented tool for TDM
Approach	Backlog, TD list
Challenge	Not challenge

After TD was documented into a NIP, it can be discussed and further managed by particular stakeholders. TD communication was conducted within the product community every second week. This community was composed of software architects, program manager, and line manager (seeing Figure 4.3). The chief architect steered the product community and facilitated communication among multiple stakeholders goes smoothly. There were two central topics with respect to TDM in this meeting. First, choose real TDs from the backlog in the wiki page, since not all NIPs stored in the backlog were TDs. For example, a NIP regarding improving the usability of the product would not be treated as a TD. Then these TDs were discussed separately based on the documented information such as *description*, *pros/cons, priority, and improvement cost.* Product community evaluated the importance and urgency of the identified TDs. So, not every TD can be put into the TD list (a TD list is stored into a spread sheet and only includes approved TD), the rejection reason will be shown in the *Decision log* field. All stakeholders must reach an agreement upon the results to avoid any personal preference that may lead to wrong decisions. TD communication was conducted very well in this case and no challenges were associated with this TDM activity.

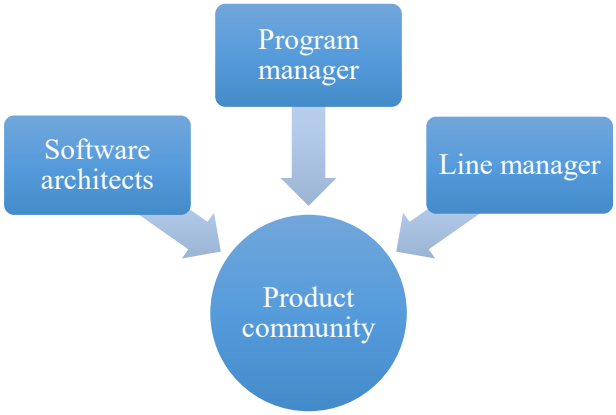

Figure 4.3 The construction of product community

F. TD prioritization

Description	Rank identified TD according to certain predefined rules to support deciding which TD items should be repaid first and which TD items can be tolerated until later releases
Role	Chief architect
Tool	Not supported by tool
Approach	Expert judgment
Challenge	The conflict between TD and new functionality or other things

TD prioritization is about ranking the identified TDs and deciding which one should be repaid first. In this case, a TD list was generated after the discussion within the product community. Then the TD list was stored into a spread sheet individually and prioritized by the chief architect. TD was prioritized according to the importance of each TD. However, the importance was mostly estimated based on experience or intuition. Meanwhile, the chief architect showed more confidence to people's opinion rather than specific numbers, so the knowledge gained from product community was also an important input to support him to make decisions. For TD prioritization, the biggest challenge was the conflicts between TD and other things like new functionalities. TD was always assigned with the lowest priority, so some TDs were not able to be repaid timely before it gets worse.

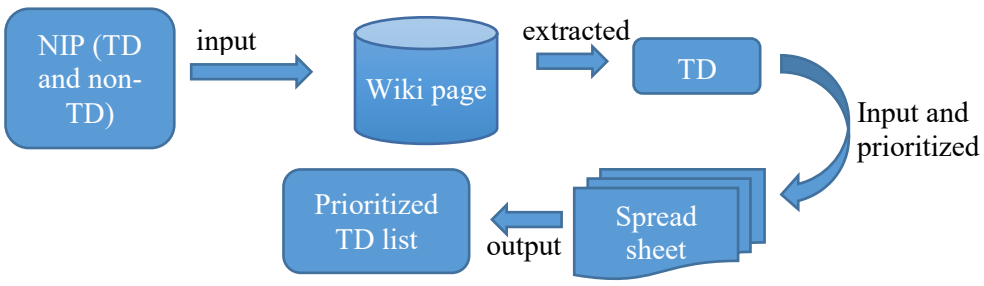

Figure 4.4 TD prioritization process

G. TD monitoring

Description	Help managers to see the changes in the cost and benefit of unresolved TD over time
Role	Not mentioned
Tool	Not mentioned
Approach	Not mentioned
Challenge	Not mentioned

TD monitoring helps managers to see the changes in the cost and benefit of unsolved TD over time. However, TD monitoring was not conducted in this project, because the cost was measured only once after a TD was identified. Then the approved TDs were assigned with priority and stored into a spread sheet, the cost of each TD will not be measured again before it was completely removed. The reason was that the chief architect lacked the awareness of monitoring TD, and development teams would care more about which TD was within their ability rather than monitor the continuous change of cost, because the TD priority list was kept by the chief architect, they were not able to differentiate TD and non-TD in the backlog by themselves. Therefore, team members did not have clear monitoring targets.

H. TD repayment

Description	Resolve or mitigate TD in a software system
Role	SAT, FBT, DMT, CT,
Tool	Not supported by tool
Approach	Refactoring
Challenge	No challenge

TD repayment is about resolving or mitigating TD in a software product. In this case, TD was repaid through refactoring the code base by teams with free space. The free space was a time gap before new functionality came into the project. The team members chose a TD from the priority list to fix that within their ability. Therefore, there was not a fixed time reserved for fixing TD. Generally, SAT and FBT take the most responsibility of TD repayment. However, when the two teams were trapped in other tasks, then DMT and CT will take care of the TD list. This mechanism spreads across the whole organization. The architect stressed that sometimes they did not get time to fix TD, because new features were already waiting in the queue with higher priority than fixing TD. Actually, the developers also fix small technical issues during the development. As the architect said: "*whether they documented a formal NIP mainly based on the size of the debt, a small TD will be solved during the development, there was no need to write a NIP about this.*" Tools were not used for TD repayment, so TD was solved manually in most cases and no obvious challenge is associated with this activity.

4.2.1.3 The concept model of TDM in this case

According to the above analysis of each TDM activity, we summarize a concept model of TDM followed by this large-scale distributed project (see Figure 4.5), which includes the TDM activities, tools, approaches, and the people involved in each activity. The model provides a clear picture about the whole TDM process in this case.

We note that this model is just in terms of one node, but the TDM process of other nodes also follows the same rule. As we can see, wiki page plays a key role in the whole management process, almost all TDM activities have a direct or indirect connection with wiki page except TD prevention (*Direct connection means a TDM activity is supported by the wiki page; indirect connection means a TDM activity is not supported by the wiki page, but still has relationship with the wiki page. For example, TD was not repaid through the wiki page, but people need to check some basic information of TD stored in the wiki page*). In large-scale distributed project, wiki page can work for people in different locations since it is an online system which can be accessed and edited on the internet by all team members of the project, regardless of differences in distance and time. The wiki page system will collect TDs from all nodes and put them into one backlog. So, it is convenient for the manager to have an overview of current TDs that developers owe the project. Normally, people just handle the debts within his/her own node.

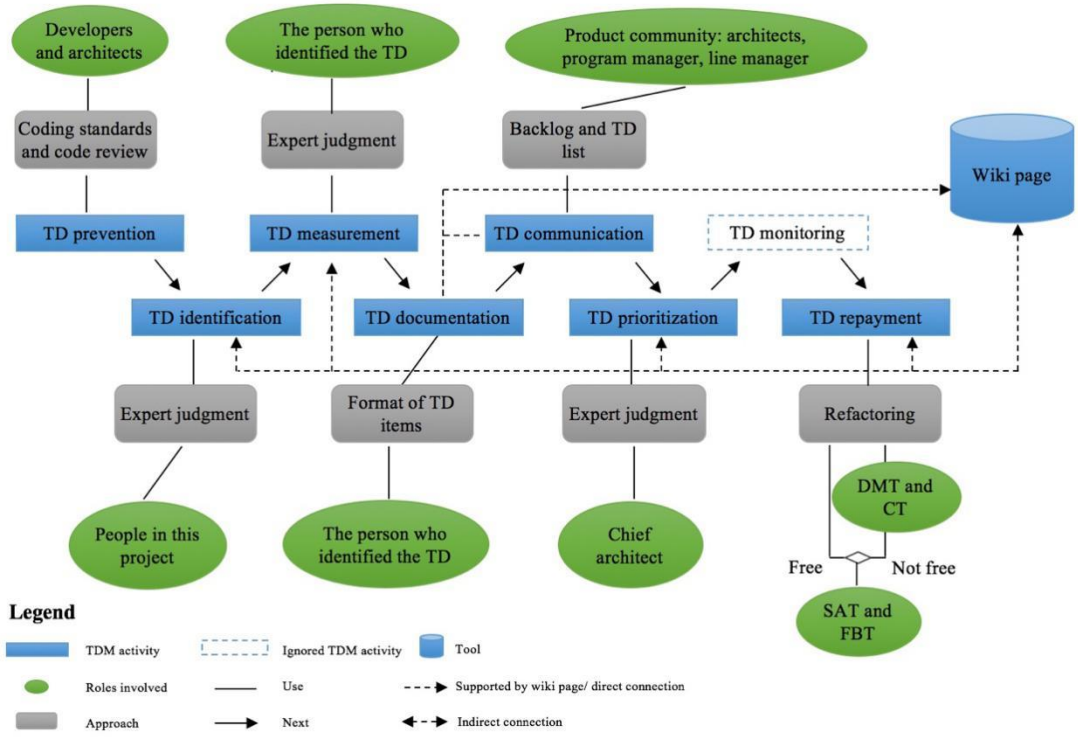

Figure 4.5 TDM model of this studied case

5 DISCUSSION

This chapter presents discussions for the results of our study. We try to extend our views by including findings from the literature and comparing them with the situation in the studied case, in order to find new possibilities of TDM. Section 5.1 discusses the results related to our research questions. In section 5.2 and 5.3, we try to extract some implications for researchers and practitioners separately based on our research.

5.1 Discussions in terms of research questions

RQ1: *What are the factors that impact TD in large-scale distributed projects?*
RQ1.1: How do these factors impact TD in large-scale distributed projects?

A. Task complexity

Task complexity in the archival data of this study refers to how complex the task is. As complexity increases with changes in structure, the dependencies among the constituent services will also change, which may cause potential extra work to maintain the software. Alzaghoul et al [9] studied a cloud-based architecture in order to manage TD in the structure, and found that higher complexity may indicate higher rework cost and lower complexity may indicate otherwise. As a result, increased complexity will lead to an increase in TD. Therefore, managers should pay more attention to *task complexity,* and assign more effort to complicated tasks from the TDM perspective. That is, tasks with high complexity need more regular tests to check the status timely, and development teams need to focus more on the parts with high complexity when they identify TD. During development of the software, developers should follow the coding standards strictly to avoid incurring unnecessary TD.

B. Global distance

Global distance measures the overhead of cooperation and coordination in communication between several sites, which is highly related to the distance between sites. Effective communication between distributed sites is crucial for a successful distributed project, since it ensures that the information sender and receiver can reach a common understanding, avoiding misunderstanding in coordination and controlling [10]. However, distance negatively affects communication, which in turn reduces coordination effectiveness, increasing the risk of incurring TD. To reduce the impact of distance, regular synchronous communication like telephone and audio/video conferencing is necessary to reduce misunderstandings, preventing small problems from becoming more serious problems. Another way to deal with distance is sharing people's work and let other people asynchronously track the activities and comment on their work items. This way is beneficial for finding the potential TD as early as possible [10].

A common understanding of TD and its management is vital when a project is carried out across multiple sites. It is necessary for team members to be aligned to TDM instead of being merely aware of what constitute technical debt. Teams sharing a common understanding of TD are able to differentiate between trivial code quality issues and TD, and recognize TD efficiently. On the contrary, when teams are not aligned, there is no guarantee that they will systematically pay off technical debt [8].

C. Maturity

Maturity is used to describe how a team can deliver the product independently. High maturity implies an increased capability of controlling and managing TD, with important historical data available for development teams to quantitatively manage and control key project as well as organizational processes [34]. Some studies also show that higher level of process maturity is related to higher product quality, and meanwhile reducing cycle time and development effort suggests a lower TD [35]. From this perspective, development teams with high maturity are beneficial for TDM, since historical data is available for team members to estimate and measure TD. Therefore, TD can be managed through a more systematic way, and the good TDM strategy can be realized under such environment. So, the debts can be controlled under the risk level effectively. It needs to be clarified that a mature team does not mean there is no TD in the product they develop. In order to achieve profitable time-to-market goals, TD may be acceptable even in a highly mature team [35]. But the mature team has a good ability to make the TD under their control. Therefore, we should not correlate TD with team maturity separately, since mature teams tend to be assigned with complex tasks which may also cause TD issues.

RQ2: *How do practitioners manage TD in large-scale distributed projects?*
RQ2.1: What methods and tools are used to manage TD in terms of each management activity in large-scale distributed projects?
RQ2.2: Which roles are involved in each TDM activity in large-scale distributed projects?

A. TD Prevention

TD prevention was conducted through adopting coding standards and code review. Development teams followed the specific coding standards to avoid creating any unnecessary debts. There are also a lot of other approaches which can be used for TD prevention such as training, education, test-driven development etc. [36]. Among these approaches, coding standard is the mostly mentioned and used approach both in literature and practice. However, developers are not able to follow such rule strictly during daily development. The reason is that organizations always face with competition from other competitors, guaranteeing time-to-market and achieving more business value become the most important thing [37]. So, following a rigid coding standard will slow down development speed and consume more resources in short term [38]. Sometimes, such rule will also make developers feel annoying and affect their productivity. Code review was only occasionally conducted by the development teams, since it is time-consuming to review every line of code. The code will be reviewed by another developer if the author did not feel confident about a certain part. Besides, architects also review the code generated by immature teams and provide improvement recommendations to help them improve the code quality.

TD prevention is useful to reduce the pressure of TDM, since good prevention is beneficial to other subsequent TDM activities. When the development teams obey a coding standard, it is possible that the amount of TD related to code base will decrease [39]. The saved resources for fixing TD could be assigned to other practices like building new features.

B. TD Identification

TD identification is the first step before development teams try to fix TD, because a TD can be resolved only after the development teams know where it is. [40]. TD identification was conducted when it was pushed by project or motivated by line-

manager. TD can be detected manually or automatically. In our case, it seems that the development teams identify TDs manually during the development. A set of tools have been developed for TDM and most of them facilitate TD identification. Code TD is the main target of these tools, while only a few of them can be used to detect other TD types (e.g., design TD and architectural TD) [3]. So, development teams lack a powerful tool to help them solve such problems. Tools like SonarQube are suitable to find small errors in the code base [18]. However, the higher-level TD like architectural TD is still a challenge faced by practitioners [41].

Meanwhile, a clear map of TD types is also important for practitioners to identify TD systemically. Such TD type map serves as a checklist, reminding the development teams of what technical issue should be detected as a TD and keeping it under control. Otherwise, it is possible that some TD types may be overlooked. They have the same problem in the case since TD was identified objectively based on experience, so different people may have different rules to judge what should be treated as TD or non-TD.

C. TD Measurement

TD measurement is the most fundamental TDM activity. TD can be measured into man-days or money. For technical side, through checking cost (man-days), project managers could understand the risk level of a TD and assign reasonable resources to it [34]. For business side, benefit (money) is a strong evidence to convince business stakeholders to invest in fixing TD, since most business stakeholders are more sensitive to benefit that they can get after fixing a technical issue or the money they will lose if not mitigating it [34].

In this case, TD was measured mainly based on experience. One problem of human estimation is that the experience is various between team members. Especially, when the TD is not fixed by the one who measures it, the actual cost may deviate from the original estimation. For example, the one who identifies the TD is a highly skilled developer, while the one who fixes the TD may be less skilled. So, the actual cost has a high possibility to exceed the estimated cost. The cost can also change if a specific developer leaves the project [34]. Some researchers point out that using a single value as an estimation of the cost would lead to a +/- 80% deviation [34]. They suggested using a range or best case and worst case (e.g., fixing this TD will cost less than 4 hours) to estimate the needed effort instead of a single value.

We find that benefit and overall TD accumulation were not measured in this case. As the architect mentioned, it is difficult to translate technical issues into economic consequence. A good way to convert TD into money is to estimate the required effort and then translate it into money [34]. The overall TD accumulation is also essential in TDM, since the cost of a single TD item may be small, but when we sum up the cost of all identified TDs, the result may be surprisingly high. So, the overall TD accumulation can give the manager an impression of impacts caused by existing TDs, and more time may be given for TD repayment.

Due to lack of using tools, human estimation is the only choice for the development team to measure TD. But here we are not judging that measurement through tools is better than measurement based on experience, because automated estimation also has its shortage, such as not be able to cover all types of TD and without considering the influential factors (e.g., team competence). However, the most significant advantages of automated estimation are fast and consecutive. Especially in this case, the architect did not have enough confidence to the results of measurements calculated by the tool (SonarQube), but he still showed interest to such tool since it

could help them save a lot of time. Therefore, a good way to measure TD is combining both expert opinion and automated estimation [34].

D. TD Documentation

When development teams realize a debt, documenting the debt will help a lot in systematically managing it. In this case, TD documentation is conducted very well. TD is documented after it is identified, and documentation is mandatory before a TD is put into the priority list. Therefore, TD documentation is not treated as meaningless and a waste of time, but as a part of their job or responsibility. Falessi, et al. [34] called such phenomenon as *organizational constraints* which means an activity is conducted only because it is mandated by the organization.

Efficient documentation is vital for further TDM. Improper or lack of documentation will cause negative impacts on other TDM actives, such *TD communication, prioritization, monitoring and repayment* [42]. Managers cannot monitor an undocumented TD since it is oral and invisible. Moreover, if the undocumented debts are not repaid immediately, it is possible that they will be forgotten or ignored at some point. A formal documentation format can make the TD traceable and increase the effectiveness of TDM.

In Figure 5.1, we make a comparison between the documentation format in this case and in the literature [3]. It was clear that both formats share some common features such as *ID, name/slogan, author, principle, location,* and *description*. Meanwhile, five new fields (*last changed, status, planned release, decision log,* and *priority*) were identified in this case study, which is a supplement of current literature. Fields like *interest amount, interest possibility, interest standards deviation* are important notions in TDM. However, they were not documented in this case. The *interest possibility* means the probability of occurrence that the interest of a TD item needs to be repaid [34]. If there is no probability of occurrence, then it is unreasonable to assign a lot of resources to such TD [34]. So, interest possibility and interest amount should be both estimated when dealing with interest estimation [34]. Other information such as the *context, propagation rules* and *correlations with other debt items* was also helpful for the development team when they decide to repay a TD.

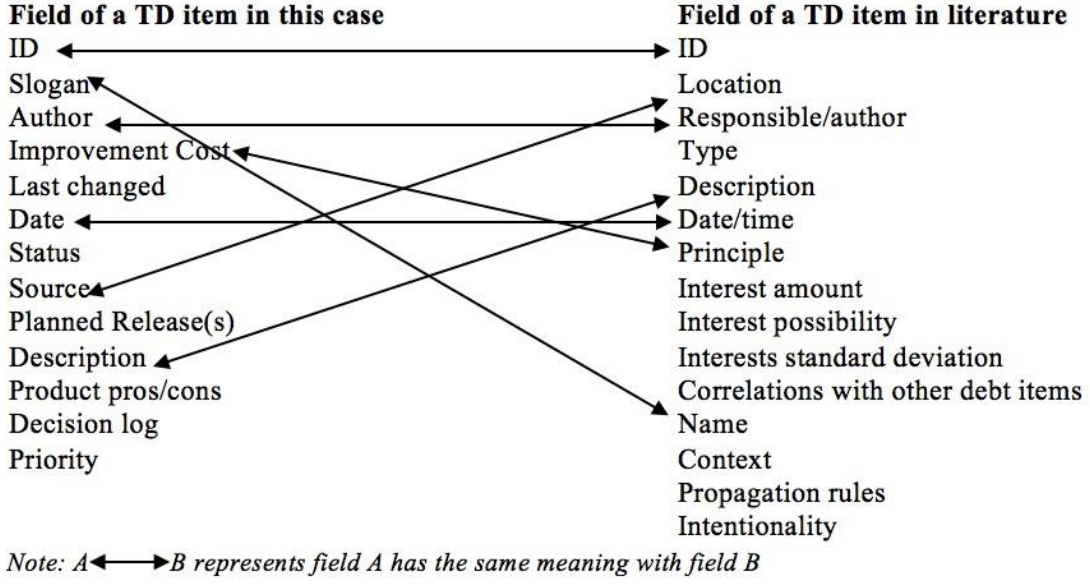

Figure 5.1 Comparison of documentation format between practice and literature

These properties should be considered when documenting a TD. We note that not all these properties must be documented, practitioners can choose the ones that are essential according to their specific context.

E. TD Communication

TD communication is the most usual TDM activity in this studied case. Every second week a meeting was held by the product community to discuss the TDs stored in backlog. The biggest issue associated with TD communication is the gap between technical and non-technical stakeholders [43]. The objective of TD communication is to reach an agreement between two sides. The chief architect steered the meeting and played the role as a bridge to make different stakeholders to share a common understanding of TD, choose the right ones to fix and assign reasonable resources to each TD. From this perspective, TD communication is about doing the right thing rather than doing things right. TD communication is in the middle of the whole TDM process, good communication of TD is the premise of the success of subsequent TDM activities. Without TD communication, the development teams could not realize the TDM strategy successfully.

F.TD Prioritization

TD was prioritized based on the importance of each TD. However, the definition of importance was not clear in this case. We could concern the importance from both the technical and business side [44]. For example, one TD may be urgent and beneficial to the technical side but may create less business value, while some debts may be more important to the business side with less contribution to the technical improvement. From this point of view, both technical and business side should be evaluated when conducting TD prioritization.

There are several issues should be considered during TDM, especially must make tradeoff during TD prioritization. That is balancing: 1) number of features to implement; 2) user perceived quality; 3) time to market; 4) the maintainability of the product in the future [34]. Therefore, TD prioritization is a complicated task, it is not reasonable to prioritize TD separately from other things like releasing new features. When fixing a TD can get more benefit than releasing a new feature, then removing TD should take priority over scheduled delivery [34].

In this case, the interviewee admitted that the priority of TD was always lower than other things, therefore TD was usually handled by teams that had free space. They did not strictly follow the order of the priority list, instead, teams chose the one that they were able to fix. Further, they did not have the figure of the risk level of the TD that was going to be repaid, which means maybe some fixed TD did not cross the breaking point and was not problematic. Whereas, some TDs may be very risky but still not repaid due to the inability of the free team, while teams that have the ability were still developing other new functionality and did not have time to take care of the TD. So, it was highly possible that some TDs with lower priority were solved before the ones with higher priority, which makes the prioritization process less effective. A method called portfolio approach was suggested to solve the conflicts between TD and other things. Portfolio approach considers TD items together with other new features or bugs as assets and chooses the best asset set to achieve maximum benefit with minimum investment risk [45]. Another problem in this case was that they prioritized TD statically instead of a dynamic way. Because the accumulation speed may vary a lot between different TDs. For example, a small code TD could increase quickly within a period of time due to developers did not follow the coding standards, then it should

be given a higher priority than before. Therefore, the priority should be adjusted regularly according to the accumulation degree of each TD. But, this was also impossible to conduct without good monitoring.

G. TD Monitoring

TD *monitoring* is one of the most important TDM activities, which helps managers to see the changes in the cost and benefit of unresolved TD over time [3]. Seaman and Guo [2] suggest that through monitoring, development teams could find a proper time point to fix TD, which is when the amount of interest is higher than the amount of principle. Without monitoring, the development teams are not able to have any reasoning for other TDM activities [46]. However, TD monitoring was not conducted in this case. There can be several reasons in terms of ignoring TD monitoring in this case. First, TD monitoring is tightly related to TD measurement, but TD was only measured (a guess by the people who identify the TD) once after it was identified. Without any continuous input data from daily measurement, it is impossible to monitor TD. Second, historical data was not saved, which can be used to track the increasing and decreasing tendency of TD accumulation. Consequently, without TD monitoring, the team members did not know when was the optimal point to mitigate or remove TD.

However, we should stress that the chief architect thought they manage TD very well without monitoring TD. This can be explained from several aspects. Firstly, as we talked above, TDM was treated as a formal activity in this case, the manager paid enough attention to TD issues and the team members were aligned to TDM. Secondly, they have a relatively systematical TDM process with the use of wiki page system. The TDs were documented very well and discussed regularly. The development teams also have the awareness of repaying TD. Thirdly, TD monitoring is not an indispensable activity to the TDM process, but it is important because it can significantly improve the efficiency of maintenance work and save time and cost for other works. Therefore, there was not deadly threat to the future maintainability of the product without TD monitoring. But, the "well" talked by the chief architect only focus on the technical side, and is only a temporary status because the tasks are in a limited amount. With a big increase in number of tasks and a long running time of the project, it is increasingly difficult for maintainers to keep track of the changes in TD manually and find the proper time point to repay TD. In this condition, TD monitoring will make a difference and play an important role in managing TD within a large amount of tasks. From the financial perspective, it is possible that they do not use the resources through a proper way. If they do not know the up-to-date status of TD accumulation, they cannot take actions timely and may cause extra costs. So, the chief architect should consider adopting TD monitoring more in this project, which will be beneficial to their TDM in the future.

H. TD Repayment

TD repayment has a strong relationship with TD measurement and monitoring, because to repay the debt, the team should check how urgent the debt is and decide when to repay the debt. As mentioned earlier, TD also has beneficial side to the project, it is not problematic if we can get more benefit than cost to retain it [47], so it is important to find the right time to repay TD, which could maximize benefit with a given risk level and minimize the risk with a given level of benefit [47]. But since TD monitoring was overlooked in this project, it was difficult to find the proper point to repay TD.

In this case, development teams repaid TD through refactoring the code base, which is the most used approach for TD repayment mentioned in the literature. Refactoring is about improving the code structure without changing the functionality to customer side [3]. However, the percentage of refactoring need more consideration by practitioners, whether a TD should be removed completely or just mitigate it under the threshold should be decided based on current needs. A reasonable repayment strategy is useful for saving resources and guaranteeing the TDM quality. Besides, automation is also a good approach to ease the repayment process through automating manually-repeated work like manual tests, manual builds and manual deployment [48]. This approach liberates team members from heavy and repetitive repayment tasks and avoids potential human errors [48].

RQ2.3: What challenges are associated with TDM in large-scale distributed projects?

A. Measurement issues

It is difficult to quantify and estimate the cost and benefit of fixing TD since the impacts of TD are not uniform [2]. As mentioned by the architect, the cost of each TD was guessed by the original author. There was not an underlying theory or model to aid TD measurement. Further, TD was measured statically, which means once the cost was decided, it would not be modified during the subsequent process. However, the principle and interest of a TD will always change and will not keep the same level before it was completely solved [34], because the accumulation and the impacts of TD are changing over time. Meanwhile, other factors will also impact the measurement of TD, such as team competence/maturity, which should be considered carefully by practitioners when measuring TD. However, it is not realistic for people to calculate the cost of the same TD every day manually, since it is time-consuming and boring. Tooling is an effective way to resolve such challenges, but current tools are mostly suitable for code TD. However, architectural issues are the largest source of TD and are difficult to deal with, and development teams were struggling to measure such TD [46].

B. Priority issues

TD prioritization is another challenge of TDM. Currently, less out-off-box prioritization methods can be used by practitioners to make a tradeoff between technical and business perspectives. Even though, approaches like portfolio were developed to solve such problem, but few industrial cases can be found to evaluate the efficiency of this method. In this case, the conflicts between TD and new features or functionality was really clear. TD was always assigned with the lowest priority compared to other things. Further, the conflicts also exist among TDs. A TD may not be given a high priority because it creates less business value. So, the customer side took more weight when the chief architect considering the priority issues. This may be caused by the fact that the development teams were facing with continuous delivering with a limited schedule, while the success criterion of a project is creating more business value for customers. Therefore, the balance of different aspects (e.g., time to market, quality of the product, etc.) to make decisions is the key issue when designing TD prioritization solutions. Bavani [8] found that if TD is not managed properly, it can harm the predictability, which can in turn impact market capitalization. Therefore, practitioners should stay a clear mind when making TDM decisions. An unbalanced decision may push the debt to a higher risk level.

5.2 Implications for research

The results of the case study point out some implications for research:

- The subjective approaches for TD measurement should get more attention from researchers. Since the measurement through TDM tools are not convincing to practitioners, more studies should be conducted to compare the results estimated by expert's experience and calculated by TDM tools. Such studies may give practitioners more confidence to use TDM tools.
- There should be more exploration of the relationship between TD and influential factors in large-scale distributed project, as well as the dependence between factors. Such factors should be considered when predicting and managing TD.

5.3 Implications for practice

The results of the case study also point out some implications for practice:

- Many tools have been developed for TDM. Some tools are devoted to TDM and some only support one or several TDM activities. Thus, practitioners can use these tools (see Table 2.2) to manage TD, or combine different tools to improve their TDM process.
- A combination of expert opinions and TDM tools is suitable for those who does not have enough confidence in the results generated by tools.
- It is important to cover all identified TDM activities to manage TD systematically. TDM activities are dependent on each other, overlooking one activity may influence other related activities (e.g., repayment has a tight relationship with monitoring). Specifically, in this case, TD monitoring is an essential part of TDM and should be adopted and paid more attention to.
- Practitioners should make a tradeoff between technical and business aspects when prioritizing TD rather than thinking about each aspect separately.
- A common understanding of TD and its management in distributed project could improve the efficiency of TDM.

6 CONCLUSION AND FUTURE WORK

In this chapter, we draw the conclusion of the study, answer the research questions and discuss future work. The remainder of this chapter is structured as follows: section 6.1 presents the conclusion and the answers to the research questions. Section 6.2 presents the limitation and threats to validity for the study. In section 6.3 we present the contribution of our research. Future work is discussed in section 6.4.

6.1 Conclusion

TDM is a crucial activity to help increase the maintainability of a software product. Researchers have done a lot of studies about TDM. However, little research was related to TDM in large-scale distributed projects, which is a more complex management compared to traditional collocated management. In large-scale distributed projects, with the size of the development team scaling up, it is impractical to concentrate all human resources in one location. Therefore, the communication and cooperation between teams in different locations are facing more barriers, which may create extra TD during this process. We did an exploratory case study about TDM in a large-scale distributed project of Ericsson to understand how the TDM is conducted in practice on the daily basis. Meanwhile, we explored the relationship between several influential factors and TD. According to the objectives, we formulated two main research questions to perform this study. We collected data through archival records and semi-structured interview to answer our research questions. The answer to each research question is shown as below:

RQ1: *What are the factors that impact TD in large-scale distributed projects?*

We totally got five factors (*task complexity*, *lead time*, *global distance, total developers,* and *maturity*) from archival data to make a hierarchical multiple regression analysis in order to explore their impacts on TD. Finally, we found three factors that have statistically significant impacts on TD, which are *task complexity, global distance, and maturity. Task complexity* is the strongest predictor of TD. That means complicated tasks could create more TD compared with simple tasks. The results also showed a positive correlation relationship between the other two factors and TD. The findings were evaluated by the architect during the interview. One divergence was that he thought tasks carried out by mature teams will create less TD. The reason of this divergence has been discussed in this thesis.

RQ2: *How do practitioners manage TD in large-scale distributed projects?*

We conducted a semi-structured interview to answer RQ1. Seven TDM activities were mentioned by the architect, which are *TD prevention, identification, measurement, documentation, communication, prioritization,* and *repayment. TD monitoring* was not covered in this case.

TD was **prevented** by adopting coding standard and code review. Developers followed specific standards to build the software. However, they did not follow the rule strictly. Especially, when time-to-market was urgent to the product. The code was reviewed by another developer when the author was not confident in terms of specific parts, besides, the architect will also review the new and modified code to prevent TD accumulation and increase the product's maintainability. Tools were not used for *TD prevention.*

TD was **identified** based on experience (expert judgment), everyone in this project could write a NIP of an identified TD and submit it to the wiki page. But most

of the TDs were identified by the developers during the development. Tools were not used for *TD identification*.

TD was **measured** by the one who identified it based on experience (expert judgment). Tools were not used for *TD measurement*.

TD was **documented** by the one who identified it through a uniform format supported by wiki page.

TD was **communicated** within the product community which consists of software architects, program manager and line manager. The relative stakeholders discussed the NIPs in the backlog (wiki page), then a TD list was generated and stored into a spread sheet so that TD can be further managed.

TD was **prioritized** by the chief architect who estimate the importance of each TD based on his experience or intuition. Tools were not used for TD prioritization.

TD was **repaid** through refactoring the code base. The SAT and FBT took the most responsibility to fix the TD, when the two teams are trapped in other tasks, then DMT and CT will take care of the TD list.

Two **challenges** mentioned by the architect were TD measurement and prioritization.

6.2 Limitations and Threats to validity

Internal validity is concerned with causal relations being examined. This threat is happening when an investigated factor is also affected by a third factor which is not identified [49]. One limitation is about the hierarchical multiple regression analysis. Conducting such analysis extremely depends on the experience of analysts, different analysts may get different results with the same data because of different actions on the data, such as the strategy of data processing. The authors in this study are not professional analysts, so it is difficult to guarantee the results with high accuracy. Some unwanted variations may be introduced during the analysis process. This threat is mitigated by following some existing guides of conducting hierarchical multiple regression.

External validity is concerned with generalizing the results of this study outside of the scope of study [49]. One limitation of the study is the number of interviewee and case. We were only able to be in contact with one interviewee in this single case study. The results may be generalized out of this case by adding more interviewees from different companies. However, the objective of this study was not to generate a generalizable TDM model or framework, but to understand how people manage TD within its specific environment. So, generalization was not necessary to this study.

Construct validity is to what extent the adopted operational measures really represent what the researcher has in mind and what is investigated according to the research questions [49]. This type of threat is mitigated by following measures: 1) The interview instruments was reviewed and evaluated by other researchers, and then we modified questions that may cause misunderstanding. 2) Making sure the interviewee understood the questions correctly before further discussion. 3) Clarifying the purpose of this study to make the interviewee answer the questions without any misgivings. 4) The interview transcript and thesis draft were also sent to the interviewee to verify if we misinterpreted anything.

Reliability validity is concerned with to what extent the data and the analysis are dependent on the specific researchers [49]. To mitigate this threat, we carefully described the process of data collection and analysis to make the study repeatable.

6.3 Contribution

This research and the thesis has some contributions in the following aspects:

For industry, we helped the team in the studied case to identify shortage of their TDM. From this case, we found that they lack TD monitoring which is an important part of TDM and can improve the efficiency of TDM in this case. Before the interview, they even did not realize that TD should be monitored. Meanwhile, TD should be classified into several categories so that they can adopt different strategies for different kinds of TD. But as we showed in the documentation format table, they did not classify TD, so different people may treat different technical problems as TD, and some TD type may be ignored. So we helped the team to understand the different types pf TD. In addition, with our data analysis, we identified several influential factors of TD. Such factors can be referred to when practitioners try to predict TD and increase the TDM quality for this project.

For academia, we summarized and synthesized a TDM concept model reflecting the whole TDM process in this project. Current studies generally answer how to conduct each TDM activity. Some study tried to explain the TDM process, but they only covered several fundamental TDM activities like TD measurement, monitoring, and repayment. In this research, we synthesized the whole TDM process in this project, and visualized the process into a complete concept model, which has not been achieved before within the team. Even though our aim is not to generalize the result out of the case, the activities identified in case were mostly the same as those talked in literature, so the TDM concept model may be referred to by other companies and researchers.

6.4 Future work

We plan to conduct more case studies on this topic to supplement the research. In the next studies, first, we will collect more archival data to re-evaluate the impact of these factors on TD, since the archival data size in this case was relatively small. Second, we aim to interview multiple roles involved in large-scale distributed projects (e.g., developer and tester) to get a more comprehensive understanding of TDM in large-scale distributed projects.

REFERENCES

[1] Cunningham, Ward. "The WyCash portfolio management system." *ACM SIGPLAN OOPS Messenger* 4.2 (1993): 29-30.

[2] Seaman, Carolyn, and Yuepu Guo. "Measuring and monitoring technical debt." *Advances in Computers* 82.25-46 (2011): 44.

[3] Li, Zengyang, Paris Avgeriou, and Peng Liang. "A systematic mapping study on technical debt and its management." *Journal of Systems and Software* 101 (2015): 193-220.

[4] Fontana, Francesca Arcelli, et al. "Towards a prioritization of code debt: A code smell intensity index." Managing Technical Debt (MTD), 2015 IEEE 7th International Workshop on. IEEE, 2015

[5] Fontana, Francesca Arcelli, Vincenzo Ferme, and Marco Zanoni. "Filtering code smells detection results." Proceedings of the 37th International Conference on Software Engineering-Volume 2. IEEE Press, 2015.

[6] Klinger, Tim, et al. "An enterprise perspective on technical debt." Proceedings of the 2nd Workshop on managing technical debt. ACM, 2011.

[7] Espinosa, Alberto, et al. "Shared mental models and coordination in large-scale, distributed software development." *ICIS 2001 Proceedings* (2001): 64.

[8] Bavani, Raja. "Distributed agile, agile testing, and technical debt." *IEEE software* 29.6 (2012): 28-33.

[9] Alzaghoul, Esra, and Rami Bahsoon. "Evaluating technical debt in cloud-based architectures using real options." Software Engineering Conference (ASWEC), 2014 23rd Australian. IEEE, 2014.

[10] Carmel, Erran, and Ritu Agarwal. "Tactical approaches for alleviating distance in global software development." IEEE software 18.2 (2001): 22-29.

[11] Yao, Yi, et al. "Structural characteristic of large-scale software development network." Computer Engineering and Technology (ICCET), 2010 2nd International Conference on. Vol. 3. IEEE, 2010.

[12] Aoyama, Mikio. "Management of distributed concurrent development for large scale software systems." Software Engineering Conference, 1995. Proceedings., 1995 Asia Pacific. IEEE, 1995.

[13] Tom, Edith, Aybüke Aurum, and Richard T. Vidgen. "A Consolidated Understanding of Technical debt." ECIS. 2012.

[14] Tom, Edith, AybüKe Aurum, and Richard Vidgen. "An exploration of technical debt." Journal of Systems and Software 86.6 (2013): 1498-1516.

[15] Brown, Nanette, et al. "Managing technical debt in software-reliant systems." Proceedings of the FSE/SDP workshop on Future of software engineering research. ACM, 2010.

[16] Alves, Nicolli SR, et al. "Identification and management of technical debt: A systematic mapping study." Information and Software Technology 70 (2016): 100-121.

[17] Wohlin, Claes, et al. Experimentation in software engineering. Springer Science & Business Media, 2012.

[18] Runeson, Per, et al. Case study research in software engineering: Guidelines and examples. John Wiley & Sons, 2012.

[19] Britto, Ricardo, Darja Šmite, and Lars-Ola Damm. "Experiences from Measuring Learning and Performance in Large-Scale Distributed Software Development." Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement. ACM, 2016.

[20] Britto, Ricardo, Darja Smite, and Lars-Ola Damm. "Software Architects in Large-Scale Distributed Projects: An Ericsson Case Study." IEEE Software 33.6 (2016): 48-55.

[21] Campbell G, Papapetrou P P. SonarQube in action[M]. Manning Publications Co., 2013.

[22] Avritzer, Alberto, et al. "Extending survivability models for global software development with media synchronicity theory." Global Software Engineering (ICGSE), 2015 IEEE 10th International Conference on. IEEE, 2015.

[23] Mayring, Philipp. "Qualitative content analysis: theoretical foundation, basic procedures and software solution." (2014): 143.
[24] Cavanagh, Stephen. "Content analysis: concepts, methods and applications." Nurse researcher 4.3 (1997): 5-13.
[25] Lederman, Regina P. "Content analysis of word texts." MCN: The American Journal of Maternal/Child Nursing 16.3 (1991): 169.
[26] Downe-Wamboldt, Barbara. "Content analysis: method, applications, and issues." Health care for women international 13.3 (1992): 313-321.
[27] Ho, Robert. Handbook of univariate and multivariate data analysis with IBM SPSS. CRC press, 2013.
[28] Chatterjee, Samprit, and Ali S. Hadi. "Influential observations, high leverage points, and outliers in linear regression." Statistical Science (1986): 379-393.
[29] Jain, Y. Kumar, and Santosh Kumar Bhandare. "Min max normalization based data perturbation method for privacy protection." International Journal of Computer & Communication Technology (IJCCT) 2.8 (2011): 45-50.
[30] Fox, John. Applied regression analysis and generalized linear models. Sage Publications, 2015.
[31] Koenker, Roger, and Gilbert Bassett Jr. "Robust tests for heteroscedasticity based on regression quantiles." Econometrica: Journal of the Econometric Society (1982): 43-61.
[32] Alin, Aylin. "Multicollinearity." Wiley Interdisciplinary Reviews: Computational Statistics 2.3 (2010): 370-374.
[33] Elo, Satu, and Helvi Kyngäs. "The qualitative content analysis process." Journal of advanced nursing 62.1 (2008): 107-115.
[34] Falessi, Davide, et al. "Practical considerations, challenges, and requirements of tool-support for managing technical debt." Managing Technical Debt (MTD), 2013 4th International Workshop on. IEEE, 2013.
[35] Harter, DE, Krishnan, MS, & Slaughter, SA 2000, 'Effects of Process Maturity on Quality, Cycle Time, and Effort in Software Product Development', Management Science, vol. 46, no. 4, p. 451.
[36] Codabux, Zadia, Byron J. Williams, and Nan Niu. "A quality assurance approach to technical debt." Proceedings of the International Conference on Software Engineering Research and Practice (SERP). The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), 2014.
[37] Lehtola, Laura, and Marjo Kauppinen. "Suitability of requirements prioritization methods for market-driven software product development." Software Process: Improvement and Practice 11.1 (2006): 7-19.
[38] Bacchelli, Alberto, and Christian Bird. "Expectations, outcomes, and challenges of modern code review." Proceedings of the 2013 international conference on software engineering. IEEE Press, 2013.
[39] Davis, Noopur. "Driving quality improvement and reducing technical debt with the definition of done." Agile Conference (AGILE), 2013. IEEE, 2013.
[40] Lim, Erin, Nitin Taksande, and Carolyn Seaman. "A balancing act: what software practitioners have to say about technical debt." IEEE software 29.6 (2012): 22-27.
[41] Kruchten, Philippe, Robert L. Nord, and Ipek Ozkaya. "Technical debt: From metaphor to theory and practice." Ieee software 29.6 (2012): 18-21.
[42] Das, Sumita, Wayne G. Lutters, and Carolyn B. Seaman. "Understanding documentation value in software maintenance." Proceedings of the 2007 Symposium on Computer human interaction for the management of information technology. ACM, 2007.
[43] Klinger, Tim, et al. "An enterprise perspective on technical debt." Proceedings of the 2nd Workshop on managing technical debt. ACM, 2011.
[44] Codabux, Zadia, and Byron Williams. "Managing technical debt: An industrial case study." Managing Technical Debt (MTD), 2013 4th International Workshop on. IEEE, 2013.
[45] Guo, Yuepu, and Carolyn Seaman. "A portfolio approach to technical debt management." Proceedings of the 2nd Workshop on Managing Technical Debt. ACM, 2011.

[46] Ernst, Neil A., et al. "Measure it? Manage it? Ignore it? software practitioners and technical debt." Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering. ACM, 2015.
[47] Ampatzoglou, Areti, et al. "The financial aspect of managing technical debt: A systematic literature review." Information and Software Technology 64 (2015): 52-73.
[48] Clay Shafer, Andrew. "Infrastructure Debt: Revisiting the Foundation." Cutter IT Journal 23.10 (2010): 36.
[49] Runeson, Per, and Martin Höst. "Guidelines for conducting and reporting case study research in software engineering." Empirical software engineering 14.2 (2009): 131.

APPENDIX

Appendix A

Question ID	Interview question
1	RQ1. Tell us some background about this project and your job in this project?
1.1	RQ1. 1 How long has the project been running in Ericsson?
1.2	RQ1. 2 How long have you worked in this project?
1.3	RQ1. 3 What's your role and responsibility in the project?
2	RQ2. What's your understanding of TD?
3	RQ3. How do you manage TD in this project?
4	RQ4. Who is involved in TD management?
5	RQ5. What tools does your company use to aid TD management related to each management activity?
6	RQ6. What methods does your company use to manage TD related to each management activity?
7	RQ7. What challenges you are facing when managing TD in large-scale distributed projects?
8	RQ8. Do you think the other missing management activities are also important to ensure the success of TD management?
9	RQ9. To what reason these management activities are ignored?
10	RQ10. What are the negative effects if not cover these ignored management activities?
11	RQ11. What do you think of the data analysis results of archival data?
12	RQ12. Are there any factors that you think will impact TD?

Appendix B

Participation request letter

We (Zhixiong Gong and Feng Lyu) are master students majoring Software Engineering at Blekinge Institute of Technology (BTH). We are conducting a research into Technical Debt (TD) management in large-scale distributed projects. As a part of the research, we plan to perform an interview within Ericsson. We want to investigate how TD is managed in the large-scale distributed project in Ericsson, regarding activities, approaches and tools involved in TD management in practice. We also want to discuss some analysis of archival data collected within Ericsson to compare with the literature.

Ricardo Britto is our supervisor and can be contacted at ricardo.britto@bth.se.

We would like to emphasize that:

- Your participation is entirely voluntary
- You are free to refuse to answer any question
- You are free to withdraw at any time

The interview will be kept strictly confidential and will be made available only to members of the research team of the study, or in case external quality assessment takes place, to assessors under the same confidentiality conditions. Excerpts from the interview may be part of a final research report, but under no circumstances will your name or any identifying characteristic be included in the report.

Thank you for your attention! We look forward to your participation.

Feb. 13, 2017

Zhixiong Gong and Feng Lyu,

in Karlskrona