CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Assisting Project Prioritization and Knowledge Management through a Software Measurement System: An Industrial study

Master's thesis in Computer science and engineering

Xiaoyan Shan
Sisi Lai

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2024

Master's thesis 2024

Assisting Project Prioritization and Knowledge Management through a Software Measurement System: An Industrial study

Xiaoyan Shan
Sisi Lai

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2024

Assisting Project Prioritization and Knowledge Management through a Software Measurement System: An Industrial study

Xiaoyan Shan, Sisi Lai

© Xiaoyan Shan, Sisi Lai, 2024.

Supervisor: Miroslaw Staron, Software Engineering and Technology
Advisor: Mattias Nystrom, Volvo GTT
Examiner: Jennifer Horkoff, Software Engineering and Technology

Master's Thesis 2024
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2024

Abstract

Among large organizations managing multiple projects simultaneously, maintaining numerous projects presents significant challenges, particularly in prioritizing projects and managing knowledge erosion.

This study aims to develop and validate indicators for efficient project prioritization and knowledge loss prevention by creating a Software Measurement System (SMS) that offers stakeholders practical information.

This research follows a design science research method, iteratively designed and implemented the SMS in two cycles. In the first cycle the conceptual model of the SMS is designed. Then the SMS is implemented and evaluated in the second cycle.

The two indicators proposed in this study are verified to be relevant, clear, and effective in guiding project prioritization and knowledge management. Stakeholders found the SMS useful, but documentation and transparency in calculation methods were recommended. This research contributes to industrial and academic domains by offering a robust SMS for real-world projects and providing empirical insights into the relationships between various data sources and project metrics.

Keywords: software measurement system, project prioritization, knowledge loss, architectural erosion, measures, indicators.

Acknowledgements

We want to extend our gratitude to Volvo GTT, for their support in topics and stakeholders. Thanks to all the participants in our interview for their valuable insights.

We would also like to thank our supervisor Miroslaw Staron and examiner Jennifer Horkoff for their guidance and advice.

Thanks to Zhuzhu and Alfonzo for their mental support.

Xiaoyan and Sisi, Gothenburg, 2024-06-13

Contents

List of Figures **xi**

List of Tables **xiii**

1 Introduction **1**
1.1 Purpose of the study . . . 2
1.2 Research questions . . . 3
1.3 Significance of the study . . . 4
1.4 Outline . . . 4

2 Background **7**
2.1 Theoretical Background . . . 7
2.1.1 Project Prioritization in Software Engineering . . . 7
2.1.2 Architechtural Erosion and Knowledge Loss . . . 8
2.1.3 Software Measurement Systems . . . 10
2.2 Domain Background . . . 12

3 Related work **13**
3.1 Project Prioritization . . . 13
3.2 Architectural Erosion . . . 14

4 Methods **17**
4.1 Research Methodology . . . 17
4.1.1 Design Science Research . . . 17
4.2 Cycle1: Conceptual model . . . 19
4.2.1 Interview and Thematic Analysis . . . 19
4.2.1.1 Interview guide . . . 19
4.2.1.2 Thematic Analysis . . . 20
4.2.2 Initialization of information model: V-model identifying information need . . . 21
4.2.3 Workshop . . . 21
4.3 Cycle2 . . . 22
4.3.1 Prototyping: defining the objectives of a solution . . . 23
4.3.2 Execution . . . 23
4.3.2.1 Data collection and analysis . . . 24
4.3.3 Evaluation meeting . . . 25

5 Execution and Results **27**
5.1 Cycle 1 . . . 27
5.1.1 Cycle Execution & Results . . . 27
5.1.1.1 Interviews and Thematic Analysis . . . 27
5.1.1.2 Initialization of conceptual information model . . . 31
5.1.1.3 Evaluation: Results from Workshop . . . 34
5.2 Cycle 2 . . . 37
5.2.1 Cycle Execution & Results . . . 37
5.2.1.1 Design for SMS . . . 37
5.2.1.2 Implementation of the SMS . . . 38
5.2.1.3 Evaluation . . . 42

6 Discussion **45**
6.1 Answering Research Questions . . . 46
6.2 Threats to Validity . . . 46

7 Conclusion **49**
7.1 Future Work . . . 49

Bibliography **51**

A Appendix 1 **I**
A.1 Interview 1: process & questions . . . I
A.2 Evaluation meeting: process & questions . . . I
A.3 Consent From . . . II

List of Figures

2.1 Relationships in the measurement information model [25] 11

4.1 DSRM Process Model [43] 18
4.2 Overview of each cycle 18
4.3 V-model: Process of developing measurement systems [10] 21

5.1 Overview of Thematic Analysis 28
5.2 V-model: Process for developing measurement system [10] 32
5.3 The initial conceptual information model for SMS 33
5.4 The conceptual information model for SMS developed after revision of workshop: one indicator was split into two, adding architectural erosion status as a new indicator. Additional data source types were included as attributes. An analysis model for project priority status was also defined. 34
5.5 Prototype for SMS 37
5.6 Final version of the information model for SMS 39
5.7 SMS without history trend for the project 41
5.8 SMS with history trend for the project 42

List of Tables

4.1 Background of interview participants 19
4.2 Mapping of interview questions to research questions 20

1

Introduction

Maintaining numerous projects presents several challenges for organizations, particularly in terms of prioritization [1]. Project prioritization is a process of determining which software project to work on to best meet an organization's long-term goals [2]. By prioritizing, organizations can focus on the most important and impactful project, avoid wasting time and resources on low-value or unnecessary ones, and align teams and stakeholders in the same direction and expectations [2].

Nevertheless, prioritization is a complex task in large organizations especially when a significant amount of projects are running at the same time: most importantly, how do product owners (POs) and developers know if they are effectively prioritizing projects [1]? The answer for many firms is to have a software measurement system where POs and developers can see the status of projects [3]. A software measurement system(SMS) is a software program that collects data about the number of attributes [4, pp. 2], and displays those data to the stakeholders who are monitoring the system and making decisions based on the data. The SMS is very helpful in addressing the problem of lacking quantitative insights into projects [4, pp. 4].

However, the effectiveness of this prioritization is not solely dependent on the selection and execution of these projects [5]. A crucial but often overlooked factor is the team members' familiarity with each project. When stakeholders have various levels of understanding of projects, those with more knowledge about certain projects can prioritize them more effectively due to their deeper understanding of project details [6]. However, this benefit is often lost when knowledgeable stakeholders leave the organization. Over time, many organizations face the issue of knowledge loss, which can lead to possible architectural erosion. This occurs when the initial design and structure of the system degrade due to continuous changes, lack of documentation, and the departure of key employees [7]. This erosion can lead to increased system complexity and reduced maintainability [8].

The study aims to provide stakeholders with insight into two areas: effectively prioritizing projects and identifying potential architectural erosion before it occurs.

To achieve the goal of prioritizing projects effectively, a team needs to share a clear prioritization strategy. We aim to develop a prioritization indicator based on critical base measures such as the number of fixes for security scans, code complexity, and frequency of the project being used. An indicator is a variable that communicates information to stakeholders about the state or trend of one or more system attributes,

reflecting a specific value at a required time [9]. The prioritization indicator will give actionable guidance to the team to focus on the most important projects. The expected outcome is an indicator showing the prioritization status for projects. Ultimately, this will result in a shortened amount of time for decision formulation for project prioritization.

This thesis also aims to reduce the risks of architectural erosion for projects. To achieve this, we worked towards implementing an indicator that tracks inactive file percentage in a project and project modification frequency difference between different developers using git logs and alerts relevant stakeholders for critical but neglected projects, possibly leading to knowledge loss of certain projects. The expected deliverable is an indicator showing the status of knowledge erosion for various projects. The anticipated long-term outcome is reduced architectural erosion risks, leading to faster completion of maintenance tasks.

This study aims to benefit stakeholders and researchers in project prioritization and knowledge erosion. For the research community, this study contributes empirical findings within a specific setting, enhancing the understanding of the relationships between various data sources and indicators for projects. It advances the existing knowledge in project prioritization and architectural erosion, providing actionable insights to inform future research and development practices. For stakeholders, it provides a software measurement system that helps determine project priorities and the need for knowledge-sharing sessions. It also highlights the relationship between git log attributes to architectural erosion, as well as web server logs, source code, SAST data to project prioritization, offering practical insights for managing project indicators.

1.1 Purpose of the study

The purpose of the study is to identify indicators from different types of data, for example: log files, source code, and SAST scans. The data were selected based on the availability from stakeholders. We then evaluated how these indicators could assist in the following scenarios:

- **Project Prioritization** As a technical architecture lead, I want to know among 50 projects, which project I should prioritize first to improve the code quality. So the team can focus on the most important things and know where to improve. Key indicators for this use case could be project prioritization status.
- **Architectural Erosion** As a Product Owner, I want to track project popularity and modification frequency. I also want alerts when a critical project is infrequently updated and only understood by a few developers. This will avoid the potential risks of knowledge loss and architectural erosion when certain maintainers leave. Key indicators for this use case could be architectural erosion status.

Throughout the study, we followed a design science research method. Two itera-

tive cycles were conducted, getting valuable indicators for a software team in Volvo Group Trucks Technology (Volvo GTT), a global leader in manufacturing transportation, mobility, and construction equipment. These indicators formed the basis for developing a measurement information model [10].

The thesis aims to develop two indicators that accelerate the decision-making process for project prioritization and initiate knowledge-sharing sessions. We have a project where we can study different kinds of log files, static application security testing results, and source code. Based on the data, we delivered two indicators, calculated using several base measures, and presented them as a software measurement system.

Indicators are the main measurements generated by software measurement systems. The SMS collects and analyzes data to create indicators, providing stakeholders with clear and actionable insights [11]. The effectiveness of a measurement system is determined by how well its indicators fulfill stakeholder needs quickly and understandably [12].

Our first indicator is the project prioritization status. We anticipated this would help stakeholders identify which projects to prioritize, avoiding wasting time on unnecessary ones. In the long term, this would reduce the time required for decision-making in project prioritization by increasing task management efficiency.

The second indicator is the architectural erosion status. We planned to analyze git logs to understand how frequently different developers interact with a project. This would allow us to deliver an indicator indicating when stakeholders should initiate a knowledge-sharing session. We anticipated that this would help avoid the risks of architectural erosion. In the long term, this would ensure that different developers have a similar level of knowledge about projects within the organization.

1.2 Research questions

Two major research questions will be addressed in this thesis study:

- **RQ1**: *Which indicator can help in quantifying project priority?*
  - **RQ1a**: *What challenge should the indicator indicate?*
  - **RQ1b**: *To what extent can the identified indicator help?*
- **RQ2**: *Which indicator can help in quantifying architectural erosion?*
  - **RQ2a**: *What challenge should the indicator indicate?*
  - **RQ2b**: *To what extent can the identified indicator help?*

These research questions are crucial for this thesis as they address organizations' need for effective project management and knowledge retention. By investigating these questions, this study aims to provide actionable insights that can help improve organizational efficiency and project stability.

We designed two indicators to help the two scenarios. Starting with understanding the essential difficulties stakeholders face, we identified the specifics of how these

obstacles relate to the team's goals in the workplace and explored a solution to these obstacles.

1.3 Significance of the study

This study aims to benefit stakeholders and researchers in project prioritization and knowledge erosion in software design research areas. In this thesis, we focus on the technical aspects of software knowledge.

The contributions of this study are threefold. Firstly, stakeholders from the company will benefit from a software measurement system based on the provided indicators. This system helps answer two crucial questions: "What project should we work on first?" and "Do we need to have a knowledge-sharing session?" Additionally, the system is designed to easily adjust any new measures stakeholders may want to add in the future.

Secondly, the study offers empirical insights into the relationship between various data sources and indicators. Specifically, it explores the connection between git log attributes and the architectural erosion status indicator, as well as the relationship between web server logs and data from SonarQube with project prioritization indicators. These insights enable stakeholders to understand and visualize the statistics generated from these data sources, providing a deeper and broader understanding of project metrics.

Thirdly, this study contributes to the research community by enhancing the existing knowledge of project prioritization and architectural erosion. Existing knowledge in project prioritization includes methodologies such as Agile prioritization, Value-Based prioritization, and Risk-Based prioritization, which emphasize customer feedback, stakeholder value, and potential risks, respectively. In the area of architectural erosion, existing knowledge is techniques for identifying and mitigating erosion, such as consistency-based, evolution-based, defect-based, and decision-based approaches. Studies have highlighted the importance of early detection of erosion symptoms, using metrics and evaluation methods to prevent long-term architectural degradation. By exploring these areas within a specific setting, the study provides actionable insights to inform and improve software development practices, addressing critical industry topics.

1.4 Outline

The thesis is arranged in the following manner:

Background This section presents the background and context information necessary to understand the thesis topics.

Related Work This section covers the current state-of-the-art in prioritization and architectural erosion fields. It also identifies previous work in the field.

Method This section outlines the methodology used to address this thesis's two primary research questions.

Execution and Results This section reports the outcomes and findings of the researchs cycles.

Discussion and Conclusion This section concludes the thesis by discussing the findings and the outlook for further research. It also contains the threats to validity.

2

Background

This chapter presents the necessary background information for this study, including theoretical background and context information. Based on the two scenarios introduced in the introduction section, we extend the basics of prioritization and knowledge erosion in this chapter. Since we have identified software measurement systems as a solution to the two scenarios, we also provide an in-depth knowledge of software measuring systems, comparing various architectures and explaining their use in decision support.

2.1 Theoretical Background

Software maintenance is a large research area. This study focuses on maintenance tasks in an industrial context, as the team we collaborated with works mainly on maintaining existing projects.

This section presents the theoretical background of the thesis, addressing the two main areas of our research questions: project prioritization and software architectural erosion.

2.1.1 Project Prioritization in Software Engineering

Prioritization activities are crucial in software engineering. With limited organization resources, project managers strive to ensure that development efforts align with business goals. For developers in agile teams, prioritization is a dynamic and iterative activity essential for sprint planning and backlog management. Teams regularly review and reorganize backlog items in response to evolving project requirements, business goals, and user input.

In addition to task and requirement prioritization, project prioritization is another critical challenge for project managers. Given the limited resources, allocating more to the most important projects is preferable[13].

In environments with tight development cycles and limited resources, quickly identifying and focusing on the most significant project can significantly reduce the time and effort spent finding and fixing bugs. This involves evaluating factors such as complexity, dependencies, technical risks, and the availability of skilled personnel or tools. Projects that require fewer resources, have lower technical risks or can be

delivered more quickly are often prioritized, especially when resources are limited. However, it's essential to balance short-term gains and long-term technical debt, ensuring that prioritization decisions align with the overall health and sustainability of the software product. Various tools and methodologies support this prioritization, such as static code analysis and change impact analysis. These tools help assess the significance of different files based on their potential impact on the overall system. Effective prioritization not only provides immediate support for current tasks but also contributes to the long-term health of the software architecture, preventing architectural erosion in the later stages of the lifecycle.

2.1.2 Architechtural Erosion and Knowledge Loss

Software engineering relies heavily on specialized knowledge. The primary assets in this field are the expertise of the employees and the software they create [14]. Knowledge loss occurs when an organization loses access to previously acquired information. A typical case could be a senior developer leaving the project abruptly and leaving a gap in understanding the code base. The general impact of such knowledge loss on software projects includes decreased quality [7] and productivity [15]. As we explore the impacts of knowledge erosion in software engineering, a critical area to understand is architectural erosion.

Software architectural erosion is the deterioration of a software architecture's structure, leading to increased resistance to change and decreased maintainability [8]. This erosion can be caused by various factors, including poor design and architectural mismatch problems.

The connection between knowledge loss and architectural erosion is clear: lost insights and undocumented decisions result in deviations from the original architectural design, complicating maintenance efforts. Jansen [16] argues that the erosion observed in the evolution of software architecture is partly due to knowledge loss. The current approach often implicitly embeds knowledge and information about design decisions within the architecture without explicit representation. This leads to the disappearance of crucial knowledge in the architecture itself, escalating these issues.

The cause of knowledge loss and software architectural erosion exceeds technical reasons, emphasizing the importance of focusing on both non-technical and technical causes. Software knowledge loss can occur due to staff turnover, lack of knowledge retention strategies, and reliance on informal communication tools like instant messaging. The phenomenon where contributors in the teams join, leave, or change their role is referred to as turnover [7]. The turnover can be catastrophic for the project if a knowledgeable contributor on major parts of the system leaves, reducing the spread of the knowledge and leading to a significant drain of tacit knowledge and expertise. Contributor turnover is rated very high in the software industry, especially in volatile job markets or projects with high stress levels [17]. Also, reliance on informal communication tools like instant messaging hurts the durable or searchable records of crucial decision-making processes and technical discussions, leading to overlooked discussions and decisions, especially in startups and software

companies [18].

Knowledge loss in software engineering projects affects continuity, software quality, and team productivity. It introduces potential delays, increased costs, and reduced quality. Project continuity is compromised as new team members, or replacements may struggle to fill knowledge gaps, leading to delays and increased project costs. Similarly, software quality may decline when the complicated reasons behind specific design and implementation decisions are lost, potentially resulting in errors and bugs.

Multiple methods are being experimented with to mitigate knowledge loss in software engineering, including integrating knowledge management in the software development process [19], using gamification [20], and implementing knowledge sharing framework [21].

Transitioning from the topic of mitigating knowledge loss, it is crucial to understand how retained knowledge impacts the structural integrity of software architecture. There are two types of knowledge: explicit knowledge and tacit knowledge [22]. Explicit knowledge is codified, representable, and transferable using formal language, and tacit knowledge is unarticulated and preconscious. During software development, developers gain knowledge about the design and code, but solutions are often not well-documented. Much of this knowledge remains tacit. Nonaka and Takeuchi claim that knowledge is constantly converted in these two forms [22]. However, these kinds of transfer remain informal and unassessable until a knowledge retention strategy is made to ensure the formal mechanisms to capture and disseminate the accumulated wisdom effectively. Tacit knowledge is often lost due to team turnover. Critical design decisions about software architecture are frequently not explicitly documented but are implicitly embedded within the architecture itself. Over time, as the rationale and details of these design decisions become obscured or forgotten, the architecture deviates from its original design intent. This deviation manifests as architectural erosion, where the structure becomes increasingly resistant to change and less maintainable.

Metrics are widely used in identifying and addressing architectural erosion, with various approaches proposed to detect and handle erosion, including through code review comments. Baabad [23] identified more than 100 metrics for identifying decay and enabling further mitigation.

Current trends in software architectural erosion include using erosion symptoms as an early warning to developers and exploring the correlation between test smells and architectural smells. A "code smell" in computer programming is any characteristic in a program's source code that may indicate a deeper issue. However, De Silva [24] concludes that no single strategy can fully address the erosion problem. The possibility of combining strategies and developing a holistic controlling framework has been explored. This thesis aims to provide a preventive measure for preventing architectural erosion at early stages by identifying knowledge loss in developers.

2.1.3 Software Measurement Systems

A software measurement system(SMS) is a technical system that gathers information, runs calculations and shows stakeholders the results of the calculations. A stakeholder is an individual who requires the outcome of measurement to make decisions that are significant for their work and organization [4]. The measurement process lets developers know whether they are on the right track in their development and has been an essential part of any development activities. ISO/IEC 15939:2007 [25] defines measurement process as "a process for establishing, planning, performing and evaluating measurement within an overall project, enterprise or organizational measurement structure".

Like in all other engineering domains, measurement is essential to software engineering to describe, assess, control, and improve processes, product quality, estimation accuracy, and productivity. Examples of these software entities include processes, products, and resources. In this era, every software system generates vast amounts of data, providing developers with extensive information to measure and analyze; software measurement is ubiquitous.

For large-scale corporations involved in software engineering, having a robust Software Measurement System (SMS) is critical. Within this context, measurement programs are integrated into the measurement process [26]. Such a system enables the quantification and assessment of various aspects of software projects, including software quality, project progress, software complexity, and more [27].

This is where the measurement program comes into play. A measurement program is a socio-technical system that gathers data on various attributes (such as program size and quality) and presents it to stakeholders.

ISO/IEC 15939:2007 [25] gives a thorough definition of the terminologies involved in the measurement process:

- **Entity:** An object that is to be characterized by measuring its attributes.
- **Attribute:** A property or characteristic of an entity that can be distinguished quantitatively or qualitatively by human or automated means.
- **Base Measure:** A measure defined in terms of an attribute and the method for quantifying it.
- **Analysis Model:** Algorithm or calculation combining one or more base and/or derived measures with associated decision criteria.
- **Derived Measure:** A measure that is defined as a function of two or more values of base measures.
- **Indicator:** A measure that provides an estimate or evaluation of specified attributes derived from a model with respect to defined information needs.

Figure 2.1 illustrates a model delineating how specific elements interrelate and outlines the method for quantifying and translating pertinent attributes into indicators, which form the basis for decision-making. In this thesis, we follow the modeling

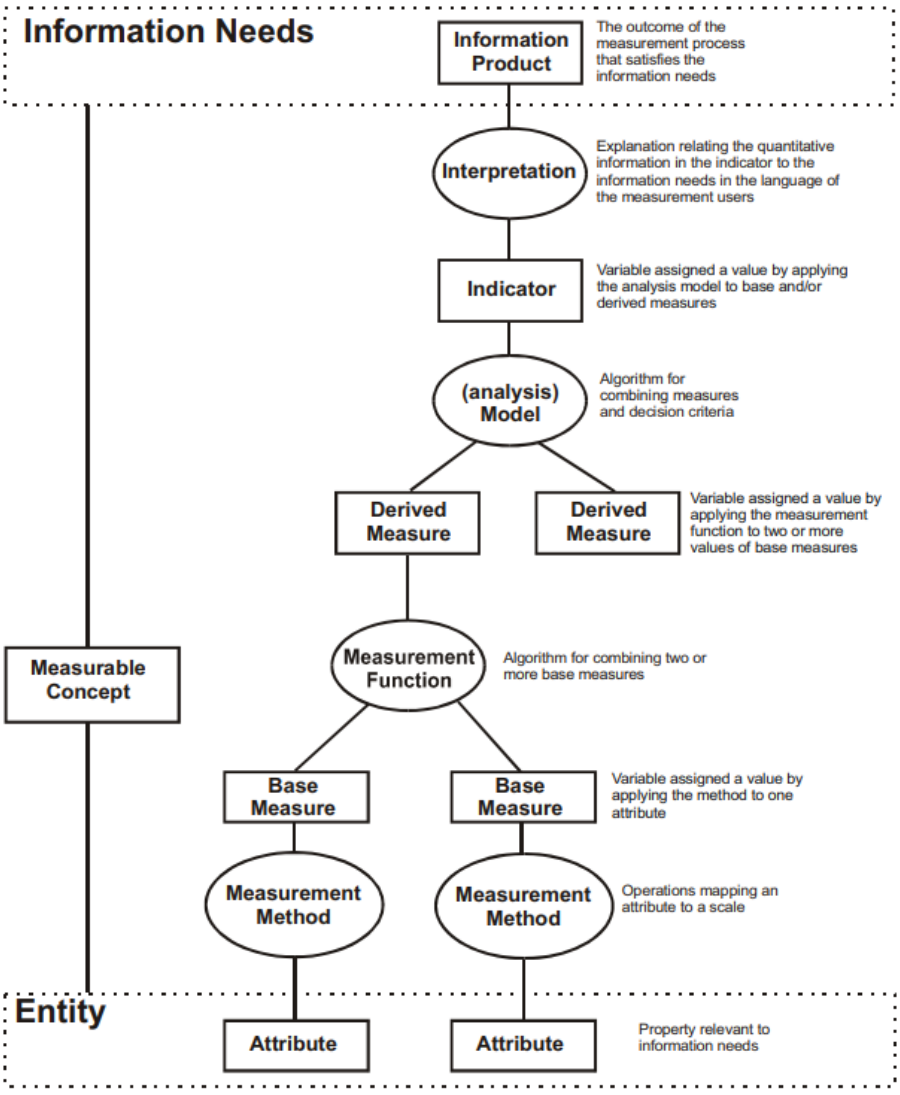

Figure 2.1: Relationships in the measurement information model [25]

language as the ISO 15939 standard shown in the figure, a box represents a measurement, and an oval represents an indicator. A critical distinction is between measures and indicators: indicators are crucial for initiating decisions (metrics push), while measures play a role in overseeing the implementation of those decisions [4].

The SMS must operate on two levels simultaneously: technological measuring systems and social stakeholders and their organizations. The social component of the measurement defines the measurement's meaning and value. The technical portion of the measurement program provides the method of data gathering and management to arrive at the measure of the 'what' [4]. This brings challenges including ensuring the sustainability of measurement methods, establishing clear goals, selecting appropriate measuring technologies, and allocating necessary resources, time, and budget. According to a recent study, the vast majority of MPs do not last more than two years and frequently fail to achieve their goals [27]. The reasons for these failures are typically linked to a mismatch between MPs and larger company goals,

a lack of organizational commitment, and a lack of alignment between measurement results and subsequent action plans.

The SMS is designed to address two critical scenarios: project prioritization and knowledge erosion. By integrating specific indicators, the SMS enables stakeholders to get a straightforward impression of the project's current status, enabling them to identify and prioritize high-impact projects and monitor the risk of architectural erosion due to knowledge loss.

2.2 Domain Background

The thesis received support from Volvo Groups, Trucks, Technology (Volvo GTT), a leading transportation, mobility, and construction equipment manufacturer, headquartered in Gothenburg, Sweden, with a global workforce exceeding 100,000. Specifically, the research was conducted within *Software Development Team A* at Volvo GTT, responsible for engineering database services for internal stakeholders. This team, comprising nine members, including product owners, software architects, developers, and testers, operates with overlapping roles. All team members actively participated in the research, facilitated by the company's provision of financial support, industry supervision, and access to essential resources.

As Staron [4] mentioned, a metrics team should have at least one person dedicated full-time to measurement. In this case, two software engineering students developed this software measurement program. We built the program based on specifications provided by company stakeholders, ensuring timely delivery of our deliverables and thorough evaluation of the software.

One key success factor for measurement programs is clear communication with stakeholders regarding their goals and needs [27]. The POs and developers are identified as external stakeholders in this project. There may be an overlap in the identity as POs also take on some development tasks.

In Agile practices, POs are responsible for evaluating progress, prioritizing backlog, and managing an overall vision for the product. They provide domain knowledge and team-specific information on which the design of the measurement program should take into consideration. They also give instant feedback on our design and guide us in identifying suitable development team members for interviews and workshops.

The developers are the primary users of our measurement program. They use the information model we create and are directly impacted by it. Their input is incorporated at various stages, including prototyping, development, and evaluation.

3

Related work

This section covers the related work for prioritization and architectural erosion, with a minor focus on applying measures.

3.1 Project Prioritization

Project prioritization is an important part of software engineering. Bug prioritization, pull request prioritization, and requirements prioritization are examples of common project prioritization aspects [3].

While direct literature on project prioritization is limited, there is a relevant research topic on requirement prioritization. Requirement prioritization is essential for project success because it helps decide which features to implement first based on factors like risk, time to market, value, and cost. Traditional methods like the Analytical Hierarchy Process (AHP) [28] have issues, such as being difficult to scale and taking too much time with pairwise comparisons. To overcome these problems, a method called Use Case-Based Analytical Hierarchy Process (UC-Based-AHP)" [29] has been proposed. This method uses natural language processing and existing use cases to make the process simpler and more reliable.

For project prioritization, several approaches are usually adopted. Agile prioritization emphasizes customer feedback and iterative development, while Value-Based Prioritization focuses on maximizing stakeholder value through techniques like Cost of Delay (CoD) and Return on Investment (ROI). Risk-based prioritization prioritizes projects by assessing potential risks using methods like Failure Mode and Effects Analysis (FMEA) [30]. Common frameworks include the MoSCoW method [31], which categorizes features into must-haves, should-haves, could-haves, and won't-haves; the Kano model [32], which balances customer satisfaction against implementation cost; and the RICE framework [33], which scores features based on reach, impact, confidence, and effort. These approaches help align project outcomes with strategic goals, improve decision-making, and enhance communication among stakeholders.

Combining different prioritization methods is becoming popular to leverage the strengths of each approach. Hybrid models offer more comprehensive and flexible prioritization frameworks, tailored to specific project needs and organizational contexts.

Software project prioritization trends are shifting significantly. There's a strong emphasis on automation and generative AI to streamline low-value tasks and enhance decision-making processes [34]. Additionally, the integration of data analytics is becoming standard practice for building data-driven project management.

Despite advancements, academic research often faces challenges in providing context-specific solutions, integrating complex systems, and ensuring reliability. Thus, there is a need for empirical studies where the effectiveness of different prioritization methods is validated across diverse industrial contexts. Addressing these needs can improve the practical applicability of academic findings and support the adoption of innovative technologies in the industry.

Our approach uses automation and data analytics to support decision-formulation processes, providing a useful solution that addresses gaps in current research and offers practical, context-specific solutions for diverse industrial applications.

3.2 Architectural Erosion

Software architectural erosion is an important issue invoking problems like declined software quality and increased complexity. Although Lehmans Laws of software evolution [35] has proven that it's inevitable that software quality deteriorates over time and the complexity increases, a lot of research has been done to both identify and mitigate architectural erosion.

According to Jansen [16], knowledge loss contributes to the erosion of software architecture evolution. The common approach of incorporating knowledge and data into architectural design choices implicitly causes problems of crucial knowledge's disappearance. Identifying erosion symptoms early has been ideal for preventing further degradation and ensuring software stability. In this chapter, we discuss both from the knowledge side and from the architecture side.

Lima [18] explains how automated approaches can be used to identify lost knowledge in software development projects. The methods used are data-mining techniques and text summarization algorithms. Newton [36] gives an empirical case on how expertise levels and complexity, impacting productivity can help identify knowledge loss in open-source software development. Nonnen [37] proposed a method to identify knowledge loss in software engineering by monitoring vocabulary divergence among developers, tracking active developers' word usage, and analyzing commit history for word understanding discrepancies. Manner [38]investigated how knowledge loss affects the software development process and proposed a Knowledge Loss Risk Model (KLRM). This model identifies and minimizes risks by focusing on documentation, expertise retention, and comprehensive process knowledge. Additionally, Rashid [39] emphasized the impact of knowledge loss in open-source projects, where abandoned code and project instability are common due to turnover and unshared expertise.

Li [40] did comprehensive research covering all aspects of software architectural erosion. Four types of detecting approaches were defined: consistency-based approaches, which depend on the evaluation of architecture consistency; evolution-based ap-

proaches, where different history versions are checked; defect-based approaches, where review and inspections are performed; and decision-based approaches, where important design decisions are captured.

Measures also play an important role in addressing and preventing architectural erosion, including preventive and remedial measures. Preventive measures are widely in the form of monitoring and evaluation. Bouwers [41] applied an architecture evaluation method to periodically assess the implemented architecture at different developing steps so architectural erosion could be effectively identified and prevented. Remedial measures focus more on maintenance. However, Li [40] points out that repairing erosion is not always an option in light of dependency on code quality and architecture knowledge. Building a new system would be more cost-effective.

After building a framework and comparing all existent architecture erosion control techniques, De Silva [24] concluded that none of the techniques in existence is effective in preventing architecture erosion. And Li [40] identified a gap between theoretical frameworks and practice, more empirical studies should be conducted.

We contribute to developing a Software Measurement System (SMS) that explicitly addresses architectural erosion by integrating knowledge retention strategies. This approach helps in ensuring critical knowledge is preserved and architectural integrity is maintained.

4

Methods

This chapter presents the methodological framework of this thesis. The thesis was developed through two iterative cycles. The first cycle focused on identifying challenges the proposed indicator should indicate (**RQ1a and RQ2a**) and identifying stakeholder's information needs. The second cycle focused on illustrating and evaluating the identified indicator's effect (**RQ1b and RQ2b**).

4.1 Research Methodology

We followed a design science research(DSR) method described by Henver [42] and Peffers [43]. The suitability of DSR is justified by its alignment with the research questions, through the structure of three iterative cycles, each cycle consists of the full process of understanding, execution and evaluation. The innovative artefact is an SMS. The first cycle focuses on identifying current challenges, perceiving the situation, and helping stakeholders identify things they are not clear about (RQ1&RQ2), the generated artefact is a conceptual model of the SMS, mapping the aspects of indicators and metrics. Subsequent cycles are dedicated to the creation of a prototype and the final implementation of the SMS, respectively, to address the comprehensive research questions designated as RQ1, RQ2, and RQ3. Later in this chapter, we present the detailed design for every cycle, focusing on how to iteratively develop the SMS.

4.1.1 Design Science Research

Design Science Research Method (DSRM) has grown in popularity and importance particularly in information systems, due to its emphasis on developing and evaluating new artifacts that solve real-world problems and contribute to both practical and theoretical knowledge.

Compared to Action Research, where the focus is on improving specific organizational contexts practically and understanding the change process, DSRM aims at creating new realities by constructing innovative artefacts to solve generic problems [44]. The artefact could be any designed object with an implemented solution to an understood research question [43]. In the field of software engineering, the artefact could be a model in a UML (Unified Modeling Language) or a prototype of a software system. The DSR framework involves three stages: understanding, executing,

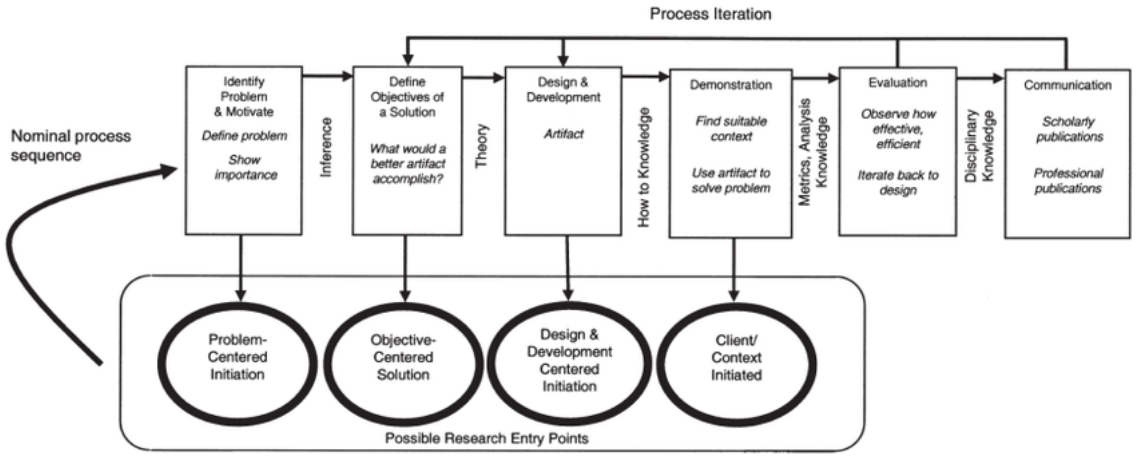

Figure 4.1: DSRM Process Model [43]

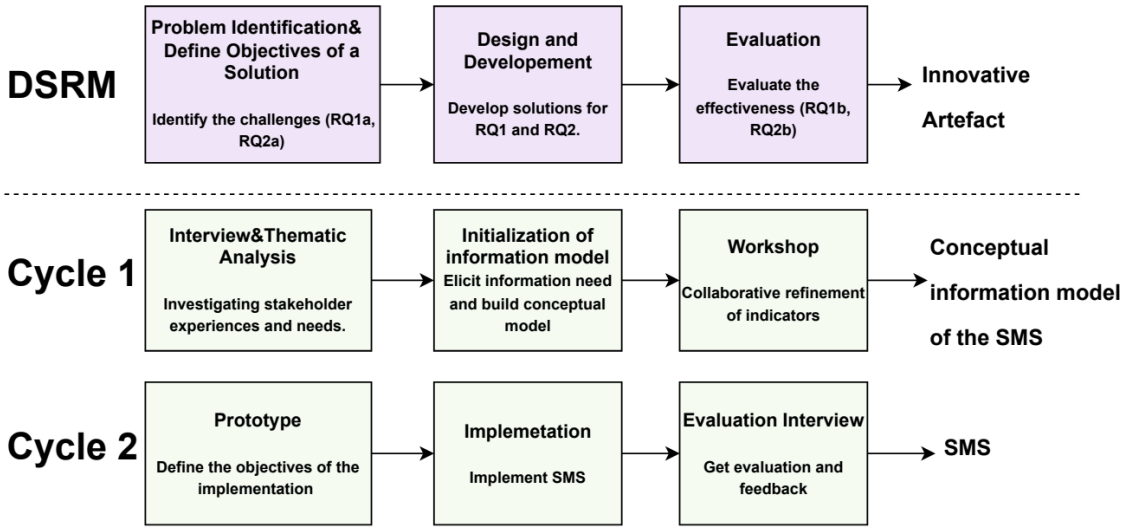

Figure 4.2: Overview of each cycle

and evaluating. Figure 4.1 presents typical design process elements.

Figure 4.2 gives an overview of the process of our two cycles. The boxes at the top are the three main stages from DSRM, and correspondingly, we have our stages in each Cycle 1 and Cycle 2. The description in the purple boxes presents the correlation of the stages and the research questions. The description in the green box gives a brief introduction on the activities in each stage of the cycle. Cycle 1's problem investigation is performed by interviews with stakeholders and thematic analysis, focusing mainly on identifying common challenges stakeholders face in their daily development related to our two scenarios (RQ1a&RQ2a). Through literature review, we developed a conceptual model for our SMS containing indicators needed for the two scenarios (RQ1&RQ2). The evaluation is performed by a workshop where then stakeholder advice is taken into model refinement (RQ1b&RQ2b).

Cycle 2 starts with prototyping, this is the stage of defining the objectives of a solution. This aims to provide concrete examples and data to inform subsequent

design decisions. The execution stage is performed by implementing a measurement system. And the evaluation meeting with a key stakeholder is conducted as the evaluation of the artefact we produced.

4.2 Cycle1: Conceptual model

This section outlines the methodology and development process of the conceptual software measurement information model within the first cycle of our research, figure 4.2 gives a visual workflow. The innovative artifact at this stage is the SMS information model, a conceptual model designed to map the final product.

4.2.1 Interview and Thematic Analysis

A semi-structured interview is ideal for understanding the specific limitations and challenges related to project prioritization and knowledge erosion. Since we are not very familiar with the team's work, it allows flexibility in discussion, enabling a deeper exploration of issues directly from stakeholders' perspectives. We aimed to dig out the qualitative data that is rich and context-specific that could help when identifying practical solutions for RQ1a and RQ2a, guiding the development of an effective software measurement system (SMS). Three participants are selected based on their roles and experience as presented by Figure 4.2.1

Table 4.1: Background of interview participants

Role	Time at EDB	Often make technical decisions
Developer	4	Yes
PO	22	No
PO	24	No

4.2.1.1 Interview guide

Our goal for the interview is to address RQ1a and RQ2a: *What challenge should the indicator indicate?* Starting from this goal, we try to develop our first version of interview questions. We iterated and formalized our interview questions twice with the help of our supervisor to avoid problems that were too vague or too broad. As a result, 8 questions are included in the final draft. A consent form A.3 is also designed for ethical reasons.

The interview was conducted individually in an informal setting and had open-ended questions to guarantee the flexibility of the discussion, allowing the participants to freely express their thoughts and provide insights beyond the scope of predetermined answers. The two participants were selected based on their roles, expertise and involvement in the decision-making process.

Before the interviews, the participants were reminded of the anonymity and confidentiality of the study. We ask interviewees for permission to record and use

Table 4.2: Mapping of interview questions to research questions

Interview Questions	
Q1. How much do you know about SMS?	General
Q2. Can you describe your experience with using metrics to formulate decisions in software development? What metrics have you found most valuable?	General
Q3. How does your team currently prioritize projects? What challenges do you face in this process?	RQ1a
Q4. What strengths and weaknesses do you see in your current approach to project prioritization?	RQ1a
Q5. What specific challenges do you encounter with knowledge management and architectural erosion?	RQ2a
Q6. Can you provide examples of how knowledge loss has impacted your projects?	RQ2
Q7. Can you share examples of the types of log data that you find particularly valuable for your decision-making process for project prioritization and knowledge erosion?	General

their words in our study and have them sign a consent form. The purpose of the study and structured interview questions were then given to participants, following the guideline [45]. This introduction to the interviews can be found in Appendix A. The interview was hosted virtually with Microsoft Teams with recording and auto-transcription using the same tool. All interviews last between 45 minutes to 60 minutes. Table 4.2 shows the connections between our interview questions and research questions.

4.2.1.2 Thematic Analysis

An inductive approach is adopted for qualitative thematic analysis. We used a tool called NVivo to do the thematic analysis for the interview. After creating a project in NVivo, we import transcripts from two interviews. By carefully reading through the transcripts, we find some interesting text and we label them with codes. Then, we categorize the code we identified into themes. Note that the process was conducted separately and concurrently, and then discussed together to reach a consensus on the themes, this also guarantees that the result is more accurate and less biased by personal preference on expression.

This bottom-up approach allowed for a detailed analysis where each quote was coded, leading to a nuanced understanding of the interviewees' perspectives. This process is iterative for more in-depth, specific, and actionable results for our workshop design.

To maintain consistency, we individually categorized the participants' responses into different themes. Following this initial coding phase, we reviewed our separate analyses together, aiming to align our understanding of themes. The outcome of the collaborative process is then documented in a Google document. Furthermore, we constructed a thematic relation diagram to visually represent the interconnections

among the identified themes.

4.2.2 Initialization of information model: V-model identifying information need

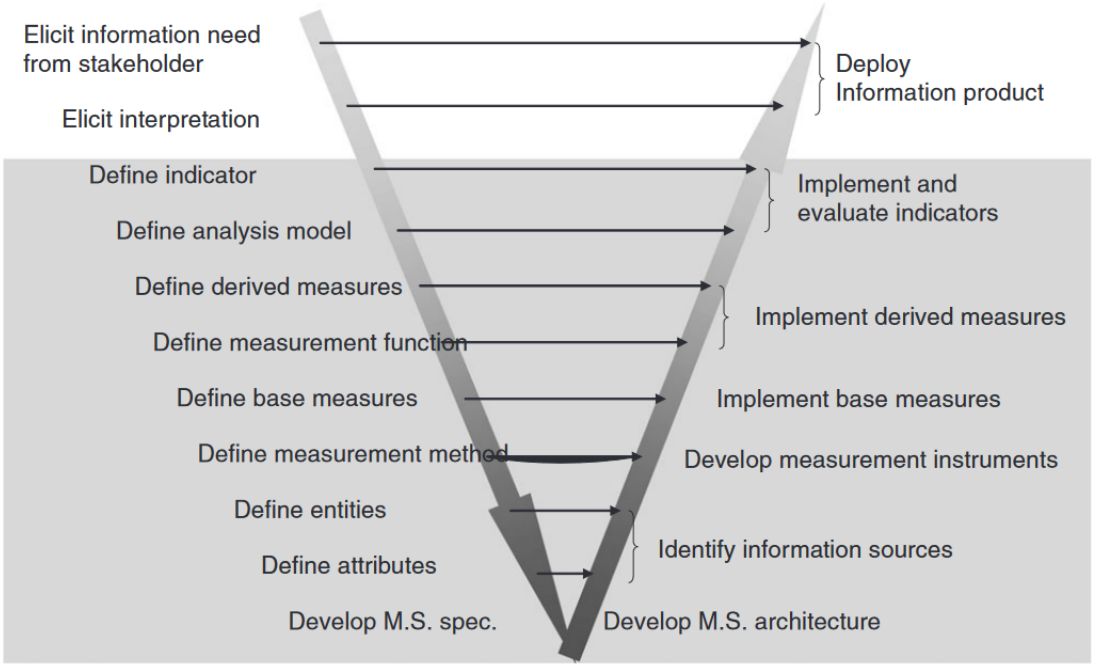

Figure 4.3: V-model: Process of developing measurement systems [10]

The V-model presented by Staron [10] illustrates the development process of measurement systems in an industrial context, a list of activities planning is suggested in a natural way. The model has a structure of a top-down approach in the initial stages followed by a bottom-up approach during implementation. This aligns with the traditional model of software development. This structure ensures that the system meets the information needs of stakeholders while minimizing misunderstandings through iterative refinements in the middle stages of the process.

The information and interpretation stage is performed by the interview and thematic analysis. Later we followed the first top-down approach to define the indicator and the corresponding analysis mode, derived measures(DM) and base measures(BM). Staron [10] also provided example questions for each activity in the v-model, when defining the specific measures, we started with answering these example questions. Thus the first draft of the information model was built. The v-shape also allowed us to demonstrate the transition from high abstraction levels to detailed specifications. The information model was then used to guide the follow-up implementation of the measurement system using the bottom-up approach in Cycle 2.

4.2.3 Workshop

The workshop was meticulously designed to validate and refine the proposed indicators within our measurement information model. The primary goal was to ensure

these indicators accurately reflect the practical needs and can be effectively applied in real-world scenarios. Encourage participants to provide case studies or examples illustrating the use of these indicators in decision-making processes. We selected 3 stakeholders for this workshop, with no overlap on participants with our interviewees in the interviews.

The workshop was structured as follows:

- Introduction (10 minutes): Briefly introduce our research work and the objectives of the workshop. This included an explanation of the terms and definitions, ensuring all participants have a uniform understanding of the key concepts.
- Workshop Session (45 minutes): Dive into the core activities of the workshop. This involved presenting an initial illustration of four indicators derived from our current research findings. Participants were then invited to discuss, verify, and refine these indicators, focusing on their relevance and applicability.
- Closing (5 minutes): Summarize the discussions, acknowledge the contributions of the participants, and outline the next steps following the workshop.

The Key Questions and Activities ensured a thorough evaluation and refinement of the proposed indicators and measures.

- Verification of Information Need: We asked participants to verify the goal of refactoring code/files based on the information provided by the indicators.
- Elicitation of Interpretations: Participants were encouraged to share how they would act based on the indicator information, providing examples through a Post-it activity. This helped in verifying if the anticipated prioritization actions aligned with practical actions taken by the participants.
- Verification of Indicators: We specifically sought feedback on the "File_Priority_Status" indicator, categorized by color codes (Red, Yellow, Green) to determine the urgency of working on the code.
- Defining Analysis Models and Measurement Functions: Through discussions and Post-it activities, we explored the BM used to calculate DM, including establishing formulas and thresholds.
- Verification of DM: This included discussing the importance, decommission points, file performance points, and knowledge erosion possibility, aiming to understand how these DMs contribute to the overall prioritization index.

4.3 Cycle2

This section describes the approach and development process of the software measurement system during the second cycle of our research. Figure 4.2 gives a visual workflow. The innovative artifact at this stage is the software measurement system.

4.3.1 Prototyping: defining the objectives of a solution

Prototyping is a key method to explore and express designs for interactive computer artefacts [46]. The prototype was designed with a functional focus, aiming to visually and practically demonstrate the system's core functionalities. The primary goal was to create a practical representation of the system to facilitate understanding and feedback. We adopted an evolutionary prototyping approach, incrementally refining the design based on stakeholder feedback. The initial design established the basic framework, focusing on core functions. Subsequent iterations added features like adjustable attribute weights to enhance flexibility.

Cycle 2 starts with prototyping to define the objectives of a solution. This stage provides concrete examples and data to inform subsequent design decisions. The prototype provided a conceptual design of the information dashboard and created an adaption of the information model to illustrate the planned system. The prototype demonstrates the system's functionalities, aiming to provide a practical and functional representation to facilitate stakeholder feedback.

The key activity is to decide which information to show and in which form should we present it to the stakeholders. Staron pointed out that the result of a measurement could be signs (e.g., an arrow pointing up), a diagram, or just a color (e.g., red, indicating a problem)[4, pp.51]. The interpretation and valuation of the numbers provided by the measurement systems are conveyed through these signs, diagrams, and colors [4, pp.51]. In this study, we adopt the color coding strategy since color is straightforward to give a highlight on a status. The prototype includes two main indicators with attributes that determine their values. The status of these indicators follows the color coding method and is illustrated using three colors: red (immediate action required), yellow (no immediate action needed), and green (no action needed). Stakeholders' feedback confirmed the effectiveness of the prototype, with several functions receiving high recommendations.

4.3.2 Execution

To develop a robust SMS, we began by exploring a variety of existing models and practices documented in academic and industry literature. Frameworks such as Goal Question Metrics (GQM), Goals Questions Indicators Measures (GQIM), and the ISO/IEC 15939:2007 Measurement Information Model are examined. We also reviewed adaptations like GQM+ Strategies and Goal Argument Metrics (GAM) to understand their structured approaches and identify potential areas for improvement, particularly addressing operational shortcomings such as process rigidity and inconsistent terminology [27].

Following Fenton's foundational principles[47], our approach started with defining the measurement initiative with clear objectives. We iterated the information model defined in Cycle 1 and performed data analysis on real data to ensure the system was grounded in practical, actionable insights. After distinguished whether the goal was assessment-oriented or predictive, we identified specific entities and attributes to be assessed to tailor the system to our project needs, avoiding a one-size-fits-all

approach. Then, a thorough analysis to distinguish between internal and external attributes critical to our metrics is conducted. By employing this methodical approach, we developed a measurement system that is precisely adapted to our project goals, supporting informed decision-making on the two identified scenarios.

During the design and development phase, we transformed the prototype into a real system. To quickly build a minimal viable product, we used MySQL as the database, Node.js for the backend, and Vue for the frontend.

4.3.2.1 Data collection and analysis

Before implementing the software measurement system (SMS), a data collection and analysis phase was conducted. Data was gathered from various sources, including git logs, web server logs, and SonarQube. We acknowledge that in Cycle 1, the connections between attributes are rather ambiguous, the links between BMs, DMs and indicators were not perfectly clear. This section analyzes the collected data and clarifies how these connections are established.

Git logs were obtained using a script, web logs were downloaded by a cron job on the production server, and SonarQube data were collected through Web API. Some attributes are internal which can be measured purely in terms of it's own without considering its behavior, like git log. Other attributes are external and must be measured by their interaction with the environment, where behavior is crucial, such as web server logs.

- Example internal attribute: git log

```
commit d815a (HEAD -> master, origin/master, origin/HEAD, origin
/Jira-8848)
Author: Developer <developer@se.com>
Date:   Thu Feb 1 14:51:05 2024 +0100
    Add test of BindVar default constructor (#2271)
    * Add test of BindVar default constructor
    * Add test of BindVar default constructor
```

- External attribute: web server log

```
[1/Feb/2024:02:15:27 +0100] 192.168.1.101 b528971 TLSv1.3 ECDHE-
ECDSA-CHACHA20-POLY1305 "GET /api/test/envstatus.cgi HTTP/1.1"
5021 200
[1/Feb/2024:02:15:31 +0100] 192.168.2.102 - TLSv1.3 ECDHE-ECDSA-
AES256-GCM-SHA384 "POST /api/info/dashboard.php HTTP/1.1"
2450 200
```

Following Staron, we use measurement instruments to assign a value to base measure from an attribute[10]. Measurement instruments are used to quantify specific properties of one type of entity. An example of a widely used type of measurement instrument is a command line script measuring the number of physical lines of code in a program. An example of our instruments is a script that analyzes Git logs to

calculate, for each file within a specified time, how many different people modified it and the number of modifications made by each person. The calculation from base measure to derived measure is defined as measurement functions in ISO/IEC 15939, measurement functions are formulas that combine BMs and get a derived measure, in theory, the BMs' quantity could be infinite, but more BMs will lead to low interpretability. Then the final goal is to define indicators that allow stakeholders to interpret.

4.3.3 Evaluation meeting

In this section, we evaluate the measures defined in terms of the information need and the implemented SMS. This activity also aims to answer our RQ1b and RQ2b.

Software measurement validation is always an ongoing topic raising practitioner's awareness, Fenton [47] proposes that a measure must be viewed in the context in which it will be used. A valid measure is if it can accurately characterize the attribute it claims to measure. A validation process for a software measure ensures the measure is a proper numerical characterization of the claimed attributed by showing that the representation condition is satisfied. We use validation to make sure the measures are defined properly and are consistent with the entity's real-world behavior.

One participant was selected. He is the key stakeholder as he is the initializer for this project in the company, he has experience in the team for over 20 years. The evaluation meeting lasted for 30 minutes, with a 10-minute opening session aligning our understanding and showing the system, and a 20-minute session for evaluation. The evaluation meeting provided significant insights for answering RQ1b and RQ2b. A list of Likert scale questions are designed, followed up by requests for further elaboration on the rating. We asked the stakeholder to assess the effectiveness and clarity of specific indicators used in project prioritization and knowledge loss. For RQa, we asked stakeholders to rate the complexity and clarity of these indicators on a scale of 1-10. We also evaluated how accurately these indicators captured challenges in assessing project priority and identified any limitations. For RQb, we examined the usefulness of these indicators in providing actionable data for decision-making. Stakeholders rated their effectiveness on the same scale and described ways in which the indicators aid in prioritization. The detailed meeting question can be found in Appendix A.2.

5

Execution and Results

5.1 Cycle 1

In Cycle 1, we aimed to get insights into RQ1a & RQ2a by conducting interviews, doing a literature review, and hosting a workshop.

Initially, a thematic analysis of the interviews was performed. Following this, we conducted a literature review, which informed the early development of a conceptual information model. The model was then evaluated and revised during the workshop.

5.1.1 Cycle Execution & Results

5.1.1.1 Interviews and Thematic Analysis

The topics discussed during the interviews were what are the challenges and limitations of project prioritization and architectural erosion identification process with the existing approaches, as well as how the indicator can indicate those challenges. These topics were discussed as part of investigating RQ1a and RQ2a.

A metric is a standardized measure to evaluate the extent to which a software system or process exhibits a specific property [48]. Indicators, on the other hand, are metrics accompanied by their associated interpretations [4]. We want to emphasize the meaning of metrics here since they will be pivotal in the subsequent themes discussed in this section.

Figure 5.1 presents a combined overview of the themes identified from thematic analysis. Each theme was also described in further detail, combined with support from interview quotes.

Theme: The use of current quantifiable indicator for making decisions

The first challenge participants mentioned was although there were already some monitoring tools in use within the team, such as Grafana and SonarQube, their usage in decision-making processes remains limited. Grafana, a platform for monitoring and observability, is a dashboards that display real-time data about developers' operations. SonarQube, on the other hand, is a static analysis tool that helps in continuously inspecting the code quality and detecting bugs and security vulnerabilities.

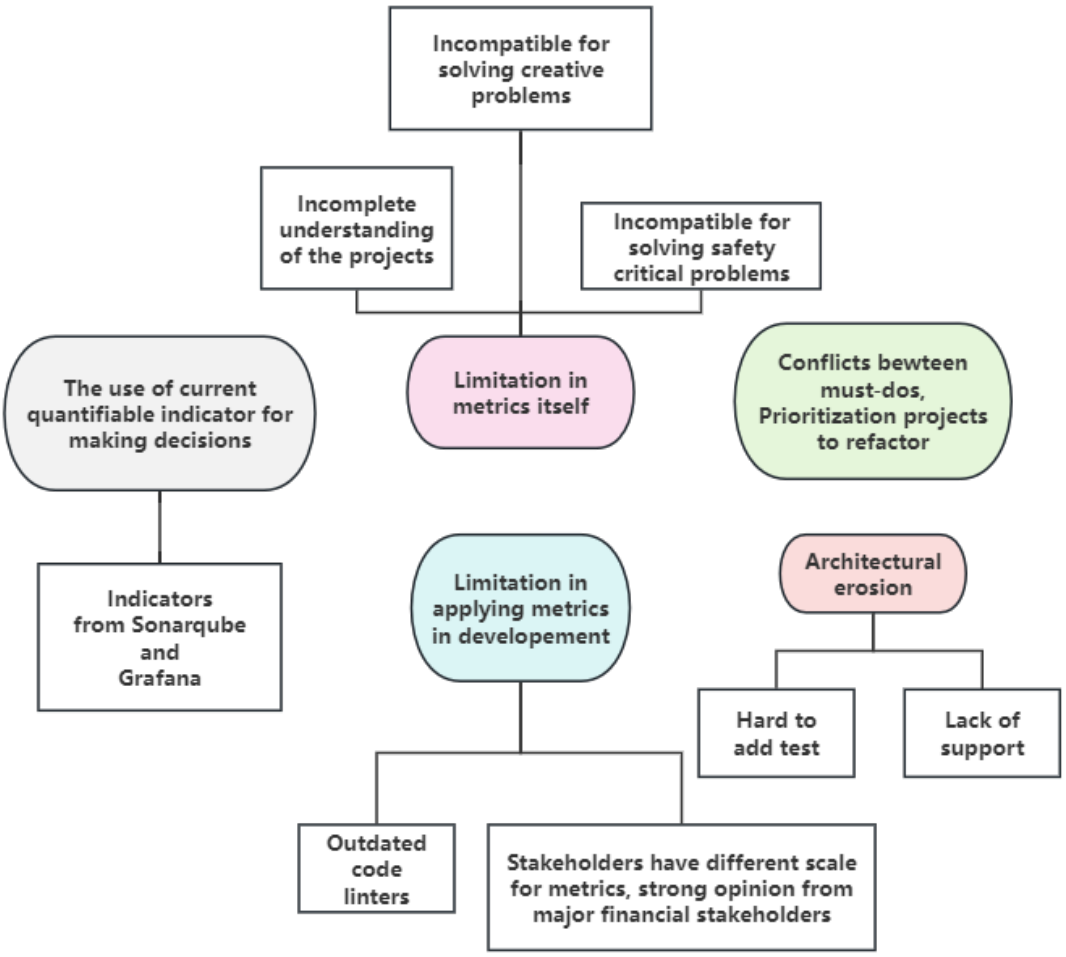

Figure 5.1: Overview of Thematic Analysis

The fact they have monitoring tools that contain several indicators but they were not in used shows a gap between the current resources and how well they have been used in practical decision-making scenarios.

> "I mean, I haven't been much involved in that. But we monitor a lot of services on the platform called Grafana. So, there are a lot of monitoring of services on that one" -P2

> "I wouldn't say metrics do much in this decision-making process, I guess. I mean you can say it's like we have a lot of tests in this program or we do not have a lot of tests in this program. So, it's kind of maybe like black and white, if tests are not enough, then you write it. And I guess until then it's good enough at least." - P2

Theme: Limitations in metrics itself

sub-theme: Incomplete understanding of the projects

Followed by the challenges that existing tools are not often used for the decision-making process. Participants expressed frustration over their inability to obtain a

full picture of the project using the existing tools. For example, while SonarQube provides metrics such as the number of lines of code and code complexity for specific files, participants pointed out that it lacks a big picture of the project's overall functionality. Consequently, they find it challenging to determine whether changes to the code base are beneficial or if the changes complicate the system without adding clear value to SonarQube.

> "I don't know if that exists yet, but basically, SonarQube is working like that, and you can do all the stuff, but it is not on a high level. This tends to be more on the details. They don't understand the big picture of what the system does, so they still don't really give a full picture of whether this makes sense." - P1

> "They don't know at the end of the day if it contributes in a good way or if it's over-complicated because it doesn't do any global analysis." - P1

sub-theme: Incompatible for solving creative problems

Our participants pointed out that existing measurement tools have limitations when it comes to addressing creative problems. Participants mentioned that when faced with such challenges, the metrics and indicators provided by these tools proved insufficient. The tools failed to offer the necessary insights or guidance needed to solve creative problems, leading stakeholders to rely on intuition or other methods instead of these metrics.

> "That was when people tried to find the solutions with statistics. But it's not really applicable. I mean, for statistics, they only capture one aspect at a time. I mean it can still be the wrong thing to focus on because it's a kind of micro-optimization." - P1

sub-theme: Incompatible for solving safety critical problems

As highlighted in the previous sub-theme, the metrics are incompatible with creative problems and prove insufficient in safety-critical problems. Participant 1 noted that while SonarQube provides valuable metrics, their experience in safety-critical systems was less than satisfactory. In the safety-critical context, where resolving 100% of security issues is mandatory, they found themselves spending an excessive amount of time addressing what often turned out to be false positives. This inefficiency shows the metrics are not suitable in situations where speed and accuracy are important.

> "With SonarQube I think they provide good metrics. But there I have a bit of a bad experience when you work in safety critical systems where you need to solve 100% of all these rules that you know you have to spend a disproportionate amount of time on on the arrows that might be for false positives." - P1

Theme: Limitations in applying metrics in development

sub-theme: Outdated code linters

One significant challenge in applying metrics is current code linters are outdated, which provides inaccurate information to developers in the team. This misinformation can lead to the implementation of incorrect features, potentially causing

subsequent issues in the development process.

> "About measurement system outputs, old code linters are bad to use. That's not updated. That gives false positives and also performance benchmarks that measure CPU counts when the network latency is the dominating factor. Then maybe you don't understand this management system so well. Otherwise, it's very good." - P1

sub-theme: Stakeholders have different scales for metrics, strong opinion from major financial stakeholders

Another challenge encountered in applying metrics arises from the diverse range of stakeholders involved. Stakeholders who contribute more financially often have more influence, potentially making final decisions. This issue is particularly pronounced when such stakeholders interpret metrics differently from others, leading to decisions that may not align with broader consensus or perspectives.

> "Then you also have strong individuals that can decide the budget, so we have 10 stakeholders, we have a stakeholder that pays more. He has to answer his managers about what he gets for the money so then you have to mix that thing. About the metrics, we put values on things, we put estimations on how long time it takes, how high priority this is all on a very high level and according to the safe and agile principles" - P1

Theme: Conflicts between must-dos and code quality improvement: need to prioritize important projects to refactor

Numerous projects have been developed over 20 years since the team has been established. Even though they remain functional, there has been a tendency to prioritize the initiation of new projects over the improvement of the code quality in these older applications. Currently, there is a growing focus on enhancing the maintenance of these critical yet code-deficient legacy projects.

> "Overall the goal is to know if we should work on the code, it is just different reasons for doing it" - P3

> "So you tend to have a lot of items that are binary decision-making in the sense that they are must DOs and they expel others, they take precedence before other items like improve quality, removing technical work long term with things. So you have a software prioritization situation where you have a problem working long-term to do things that they are not the most and that is I think in the industry, in general, is a problem." - P1

Theme : Architectural erosion

As indicated by the stakeholders, there are amount of legacy code within the organization that requires updating and refactoring to enhance code quality. However, many of the earliest applications, developed by developers who have since transitioned to other roles or organizations, lack clear documentation or explanations for their code. Despite this, numerous legacy projects continue to be operational and essential. Furthermore, these older systems often serve as a foundation for newer projects, creating dependencies that add difficulty to modifying the legacy code. This situation pronounced a knowledge gap for those applications, complicating the

maintenance of these critical systems.

> "I mean, we have a lot of old code. It's not very up-to-date." - P2

> "Yeah, you have to do it quite manually to pin the problem down and sometimes you need to get the knowledge from someone else who has worked on it before." - P2

> "We have a lot of services, so you can't really be an expert on everything. It's really important to have logs because you can't be an expert on everything and you need to be able to get information from logs to pin down things."" - P2

> "Yeah. I mean you can do that, or you can do it yourself with just analyzing logs, of course. But maybe it takes more time." - P2

The themes "Conflicts between must-dos and code quality improvement: need to prioritize important projects for refactoring" and "Architectural erosion" is highly relevant to our RQ1a and RQ2a. These themes provide insights into the question, *What challenge should the indicator indicate?* and were used in subsequent steps of our analysis.

5.1.1.2 Initialization of conceptual information model

In the development of our conceptual measurement systems, we used the V-model, a framework well-documented by Staron et al [10]. In their work, the V-model was used to structure the development process which involves identifying stakeholder needs, defining measures, and developing and validating. The process was depicted in Figure 5.2. This model guided us in identifying specific information needs from the thematic analysis. This section is divided into two parts: the process of how we conducted the initial conceptual model and a detailed explanation of the model.

Process of initialization of the conceptual information model

To initialize a conceptual information model, we first elicited information needs through thematic analysis. The participants we interviewed, as primary users, pointed out that they wanted to get an idea of which file should be prioritized first to work on from the SMS. This need helps speed up decision-making in file prioritization and improves the overall code quality of projects.

> "Overall the goal is to know if we should work on the code, it is just different reasons for doing it" - P3

> "So you tend to have a lot of items that are binary decision-making in the sense that they are must DOs and they expel others, they take precedence before other items like improve quality, removing technical work long term with things. So you have a software prioritization situation where you have a problem working long-term to do things that they are not the most and that is I think in the industry, in general, is a problem." - P1

Next, we focused on understanding how participants interpret this information need. They indicated a requirement for metrics that would help them decide

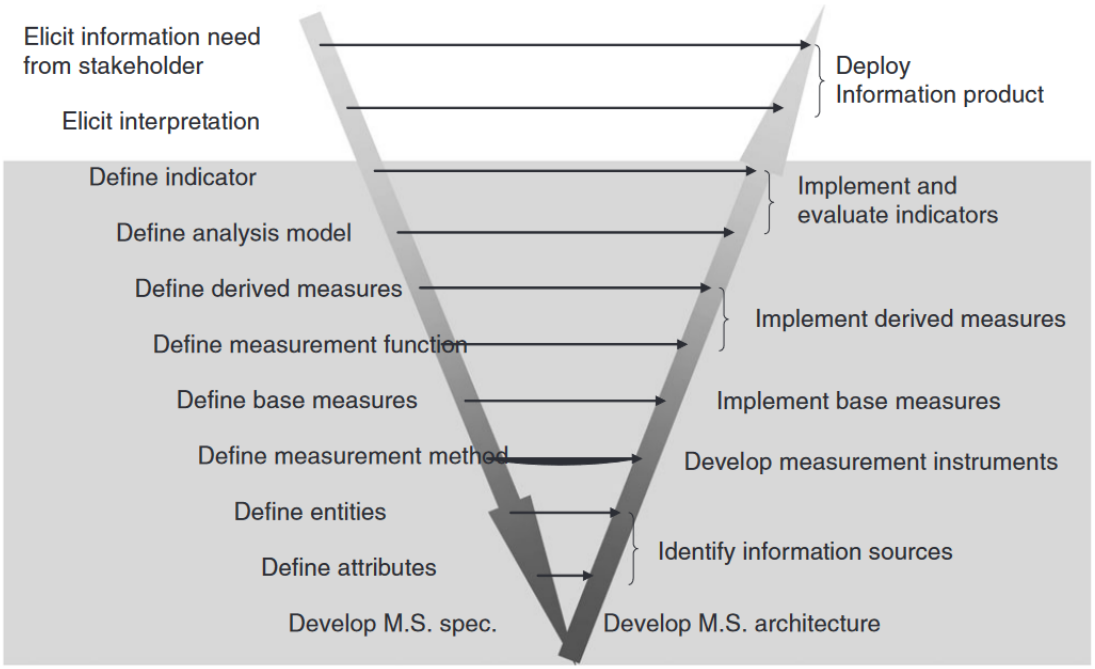

Figure 5.2: V-model: Process for developing measurement system [10]

whether to work on files when faced with conflicting maintenance tasks. This understanding guided us in defining relevant indicators, with the primary one being the file_priority_status.

To define the indicator and the corresponding analysis model, decision measures (DMs) and base measures (BMs), we followed a top-down approach. Staron [10] provided example questions for each activity in the V-model. When defining the specific measures, we started by answering these example questions.

Initially, we did not obtain any information on how stakeholders could define the analysis model based on literature review, so we left it as combined with different kinds of measures. However, during a later workshop with stakeholders, the analysis model was defined and subsequently documented in Chapter 5.1.13.

The top-down approach led us to ask ourselves questions, which revealed that participants did not suggest any specific measures to include in the formulas for defining decision measures (DM). However, we proceeded to define base measures (BM). The necessary BMs identified included the frequency of file touches, file-related URL requests, and the time taken to complete these requests.

We then defined measurement methods to describe how values are assigned to BMs, such as using git logs to count the number of changes and web server logs to count URL requests and measure the time to complete these requests.

Lastly, we aimed to define the entities and their attributes, which are critical as they form the main sources of information. At this stage, specific entities and their relationships were not fully detailed by the participants. The primary attributes identified for measurement were derived from git logs and web server logs.

Explanation for Conceptual Information Model

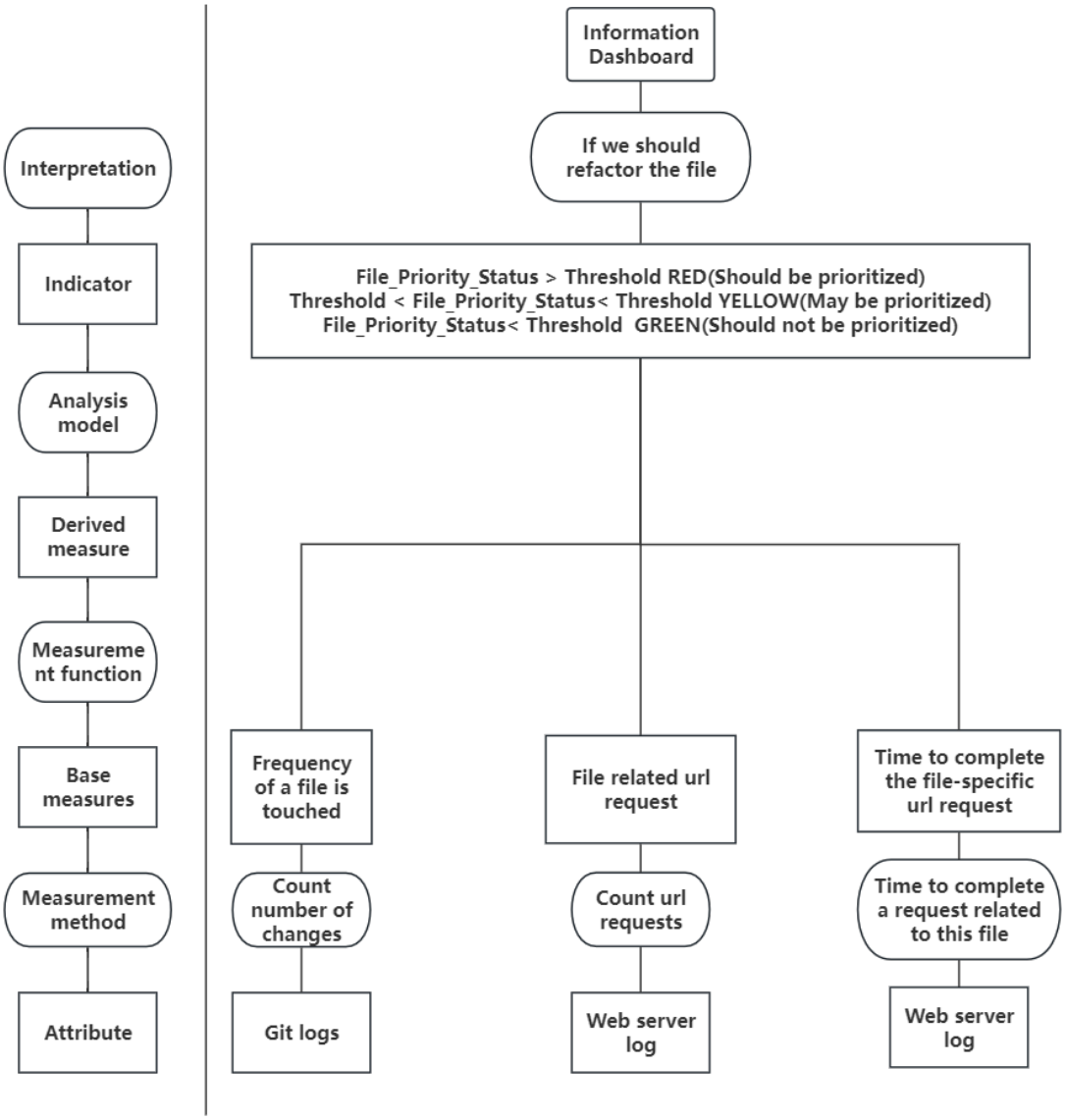

Figure 5.3: The initial conceptual information model for SMS

The results of are summarized in Figure 5.3. The figure was explained from top to bottom. As the figure presents, on the left-hand side is the conceptual information model based on the theoretical framework from the literature, while the right-hand side represents the model developed from thematic analysis and literature review. The main indicator identified is the file_priority_status, which is divided into three thresholds: RED, indicating high urgency and files that should be prioritized; YELLOW, indicating moderate urgency and files that may be prioritized; and GREEN, indicating low urgency and files that should not be prioritized. the indicator helps participants make informed decisions about file prioritization.

Base measures provide the raw data needed for the analysis model. The identified base measures include the frequency of a file being touched, file-related URL requests,

and the time to complete file-specific URL requests. These are tracked using git logs for file changes and web server logs for URL requests and response times.

The attributes define specific characteristics of the entities being measured. In this model, the attributes include data from git logs and web server logs, which provide detailed information on file changes and URL requests.

The final output is presented in an information dashboard that helps stakeholders decide if a file needs refactoring based on the file_priority_status. This dashboard visualizes the derived measures and provides a clear indication of which files should be prioritized for maintenance or improvement.

This stage played a crucial role in informing the early development of a conceptual information model. The review helped us identify key concepts, frameworks, and methodologies that are essential for constructing a robust conceptual information model. This model served as the foundation for the workshop preparation, ensuring that our measurement strategies were not only grounded in theoretical frameworks but also finely tuned to address specific organizational contexts. This step primarily aimed to deepen our understanding of Research Questions 1a and 2a.

5.1.1.3 Evaluation: Results from Workshop

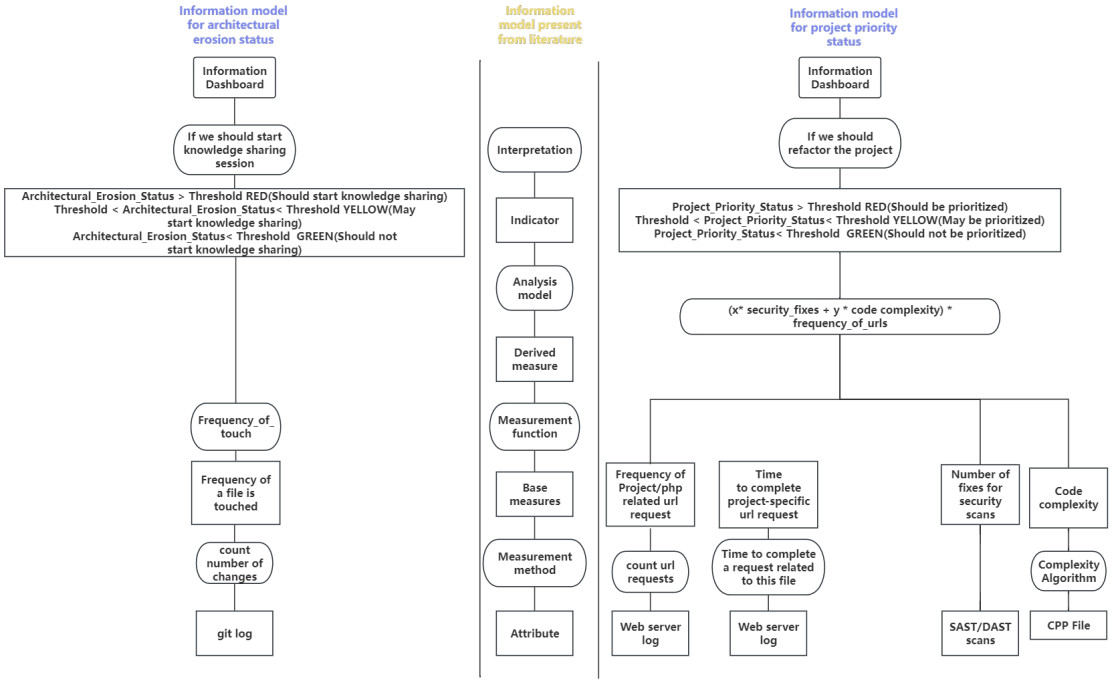

Figure 5.4: The conceptual information model for SMS developed after revision of workshop: one indicator was split into two, adding architectural erosion status as a new indicator. Additional data source types were included as attributes. An analysis model for project priority status was also defined.

The goal of this workshop was to validate and refine the conceptual information model with stakeholders. Three participants attended the workshop.

This section presents the process of how the workshop was been conducted and the results of the revised conceptual information model, organized into several main areas: topics and focus areas, areas for improvement and adjustments made, and a summary for the workshop.

Workshop topics and focus areas

The workshop focused on two key goals to ensure the applicability of our information model. First, we aimed to validate how participants act on the information provided by the indicator. i.e. file_priority_status. Participants were encouraged to refine the actionable steps based on the indicator. For example, if action should be taken when certain thresholds are met, such as deciding whether to refactor the code or file.

Second, we aimed to verify the validity of the indicators. Participants were asked to validate the indicators previously defined, specifically the file_priority_status indicator. The indicator was demonstrated to participants with the following examples: Red indicates that the team should work on the code (when it exceeds the threshold), Yellow suggests that the team can work on the code if they have extra time, and Green means there is no need to work on the code. This discussion was crucial to ensure that the indicators met the stakeholders' needs.

Areas of Improvement and Adjustments Made:

- **Diverse Data Sources**: During the workshop, participants identified the need for incorporating diverse data sources, as the current model relies heavily on log files. Expanding the range of data sources could significantly enhance the comprehensiveness of the information model. This would provide a broader and more accurate picture, capturing nuances that log files alone might miss. By integrating data from multiple sources, the model can become more robust and reflective of real-world scenarios.
  - **Adjustment**: To address this issue, we incorporated extra attributes beyond those found in log files. Specifically, this includes source code and SAST scans. This enhancement ensures that the information model includes a wider variety of data, leading to a more complete and accurate representation. The additional attributes enable the model to account for different aspects of the data, improving its overall quality and usefulness.
- **Classification of Indicators**: Another area of improvement identified was the classification of indicators. Participants suggested that the current model's indicators are too limited, with "file_prioritization_status" being the primary example. They recommended categorizing indicators based on their attributes to facilitate more effective analysis and interpretation. A more detailed classification system would make it easier to derive meaningful insights from the data, as it would allow for a finer granularity of analysis.
  - **Adjustment**: In response to this feedback, we refined the classification system by separating a single indicator into two distinct indicators. From file_priority_status to project_priority_status and architec-

tural_erosion_status. This change allows for more precise measurement and analysis, enabling users to gain deeper insights from the data. By having more specific indicators, the model can better capture the complexities of the data, leading to more accurate and actionable results.

- **Development of Analysis Models**: The final area of improvement discussed was the absence of analysis models for indicators. Participants noted that without these models, it is challenging to fully use the information provided by the indicators. The development of robust analysis models was crucial for making the most of the data and deriving valuable insights. Since the current team does not have established thresholds for security and complexity values, the stakeholder would like an adjustable parameter, ranging from 0 to 1 for both values. This will help them make adjustments once they have defined the thresholds for these values. Parameters are presented as x and y in the analysis model for project priority status indicator. These models would provide the necessary framework for interpreting the indicators and understanding their implications.
  - **Adjustment**: To address this gap, we collaborated with workshop participants to formulate a preliminary analysis model. This initial model serves as a foundation for further refinement and validation in subsequent project cycles.
  - **Project Priority Status**:

    Project Priority Status = $(x \times \text{Security} + y \times \text{Complexity}) \times \text{Frequency of URLs}$

  - **Knowledge Erosion Status**:

    Knowledge Erosion Status depends solely on the frequency of file touches.

Summary for the workshop

During the workshop, stakeholders iteratively refined the indicator, transitioning it from file_priority_status to project_priority_status. This change resulted from the recognition that monitoring file_priority was infeasible due to existing limitations within our development and logging systems.

Additionally, the workshop identified the need for different kinds of data sources. Consequently, we adjusted our attributes to include SAST scans, CPP files, web server logs, and git logs. A prioritization session conducted during the workshop determined that SAST scans should be given the highest priority when deciding whether the project needs refactoring. This contributes to the development of the analysis model for project prioritization.

In subsequent discussions, consensus was reached on the architectural_erosion_status indicator derived from git log data, leading us to retain this indicator in our analysis.

The workshop conducted was a pivotal element in achieving the overarching goals of this thesis, which aims to validate and refine the initial information model. This step primarily aimed to verify our understanding of Research Questions 1a and 2a.

5.2 Cycle 2

Initially, Cycle 2 aimed to develop a software measurement system to illustrate the conceptual model created in Cycle 1. The process was divided into three stages: design, development, and evaluation.

During the design stage, we designed and demonstrated the potential SMS of a solution to a stakeholder. The feedback we received from the stakeholder ensured that we were aligning their needs with our understanding and staying on the right track. This stage was crucial for aligning the system's features with stakeholder needs (RQ1a & RQ2a). Next, the SMS was developed targeted to our specific organization. The analyzed data was entered into the SMS. Finally, the evaluation stage was conducted through an interview with a company stakeholder (RQ1b & RQ2b).

5.2.1 Cycle Execution & Results

5.2.1.1 Design for SMS

Figure 5.5: Prototype for SMS

A prototype was drawn to illustrate the function of the SMS. The prototype's design was primarily functional, aiming to demonstrate the system we were planning to build. It focused on illustrating the functionalities. The main goal was to provide a practical and functional representation of the system. Figure 5.5 shows the prototype of the SMS.

The system initially lists project names, followed by measures represented in blue cubes. These cubes denote various base measures that contribute to two critical indicators: Knowledge Erosion Status and Project Prioritization Status. Knowledge Erosion Status was intrinsically tied to the touch frequency of files, indicating how often files were modified, which could suggest potential knowledge loss. Project Prioritization Status uses multiple base measures, including the number of fixes for security scans, code complexity, and frequency of URL requests. These indicators were assessed using an analysis model, with their statuses represented by three colors: red indicates immediate action is required, yellow signifies that action is not required

immediately, and green means no action is needed at all. The analysis model defines how these derived measures are calculated from the base measures.

The prototype also has a feature of adjustable weights under base measures, enabling users to modify the importance of different measures. This offers flexibility in how the data is interpreted and applied.

During the development process, these features were presented to a key stakeholder and received confirmation. Several functions within the prototype were highly recommended by stakeholders for their effectiveness and utility.

The contribution of the prototype was to define objects as solutions for RQ1a & RQ2a.

5.2.1.2 Implementation of the SMS

Data collection and analysis Before implementing the SMS, a comprehensive data collection and analysis phase was undertaken. This phase involved gathering data from various sources, including git logs, web server logs, and SAST scans. The purpose of this data collection was to extract relevant insights needed to address Research Questions 1b and 2b.

Data collection

The data was collected from several sources:

- **Project Repository on github**: source code from EDB team repo and related log files were gathered from the project repository to ensure accuracy and relevance.
- **Production Server**: Logs from the production server provided critical operational insights.
- **Static Application Security Testing (SAST) Scans**: Data from SonarQube contributed to the security and code quality analysis.

Data Analysis

During the implementation of the SMS, we applied a significant change due to the lack of direct connections between some base measures and indicators when analyzing real data. Specifically, we could not establish a direct link between the frequency of file touches and the architectural erosion status. Consequently, we carefully re-examined the data and modified the base measures from the frequency of file touches to inactive file percentage and ownership overlap.

Inactive file percentage refers to the proportion of files in a project that have not been modified for a long time. A low inactive file percentage indicates that many files have not been touched for a significant period, suggesting a high risk of knowledge erosion within the project.

Ownership overlap measures how many different developers have worked on a project over a given period. If ownership overlap is zero, it means that only one developer has touched the project within that time frame, indicating a high risk of knowledge

loss. Conversely, a high ownership overlap value means multiple developers have contributed to the project, reducing the risk of knowledge loss.

Another change is we add a feature that provides historical trends for two base measures derived from git logs. By analyzing the git logs, we can extract data over specific time periods, such as week-by-week comparisons. These adjustments were necessary to better reflect the real-world dynamics of knowledge erosion and to improve the accuracy of our indicators. The refined model incorporating these changes is presented in the figure 5.6.

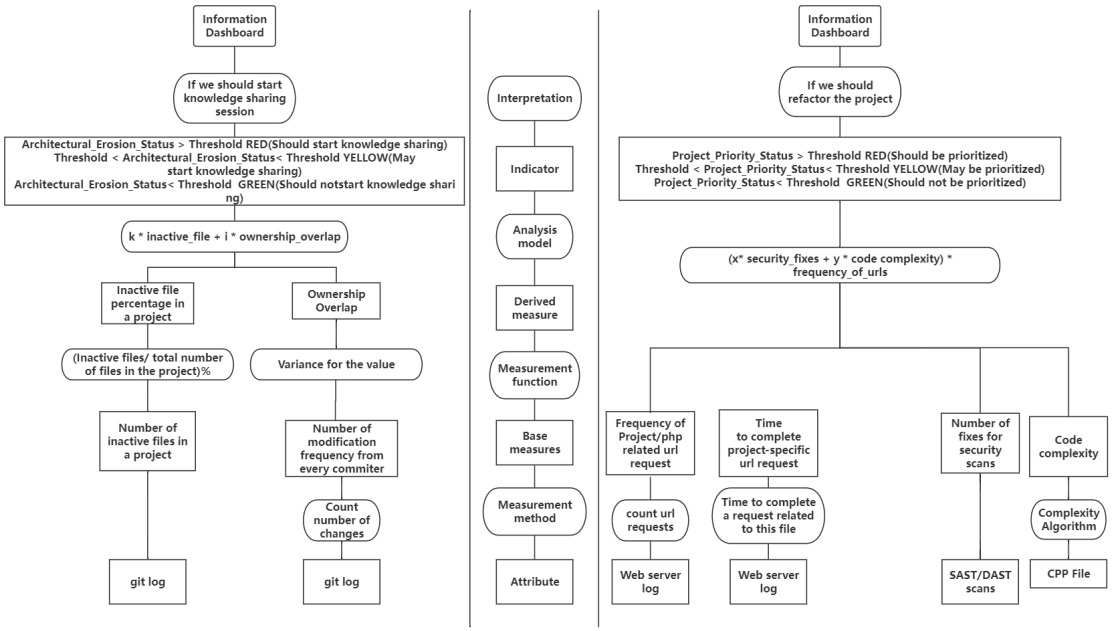

Figure 5.6: Final version of the information model for SMS

The collected data was analyzed using Python scripts. The analysis was structured as follows:

- **Git Logs**:
  - *Inactive File Percentage Calculation*: This was calculated to understand the proportion of files that had not been modified over a specified period.
  - *Ownership Overlap*: Analysis of contributions by various developers helped in assessing the distribution of work and identifying key contributors.
  - *Time Frame*: The analysis of git logs covered the period from January 1, 2024, to February 1, 2024.
- **Web Server Logs**:

  Below are two example entries of the web server log:

  ```
  [1/Feb/2024:02:15:27 +0100] 192.168.1.101 b528971 TLSv1.3 ECDHE-
  ECDSA-CHACHA20-POLY1305 "GET /api/test/envstatus.cgi HTTP/1.1" 5021
  ```

```
200
[1/Feb/2024:02:15:31 +0100] 192.168.2.102 - TLSv1.3 ECDHE-ECDSA-
AES256-GCM-SHA384 "POST /api/info/dashboard.php HTTP/1.1" 2450 200
```

Checking the web server logs revealed critical information, including timestamps, client IP addresses, unique session identifiers, TLS protocol versions, HTTP methods, specific URIs requested, data size transmitted, and HTTP response codes. Understanding how often different web pages (URLs) were visited helped address prioritization challenges in decision-making processes.

- *Frequency of a project-related URL requests:* This measure tracked how often different projects were accessed, providing insights into project popularity and usage patterns.

  > " Frequency of URLs is nice to have to monitor code quality." - P3

- *Time to Complete Requests*: Stakeholders indicated the importance of tracking how long it takes to complete each request in the web server logs. This information could be used to determine the time efficiency for different programs/files and identify key areas for improvement.

  > "Time to complete request could be useful, I will ask a developer to deploy a function to production environment." - P3

  > "I mean time information is of course important. " - P2

- *SAST Scans*: We got data directly from SnoarCube
- *Time Frame*: The web server logs analyzed were on February 1, 2024.

Implementation

The SMS is shown as figure 5.7 and 5.8.

The features are described as follows:

- **Adjustable Weight Sliders**
  - *Security Fixes*: Allows users to adjust the weight given to security fixes.
  - *Code Complexity*: Adjusts the weight given to code complexity.
  - *Inactive File Percentage*: Adjusts the weight given to the percentage of inactive files.
  - *Ownership Overlap*: Adjusts the weight given to the overlap of code ownership among developers.
- **Data Display: Measures**
  - *ID*: Unique identifier for each project.
  - *Security Fixes*: Number of fixes for security scans.
  - *Code Complexity*: Complexity score of the code.
  - *Urls Frequency*: Frequency of URL requests related to the project.

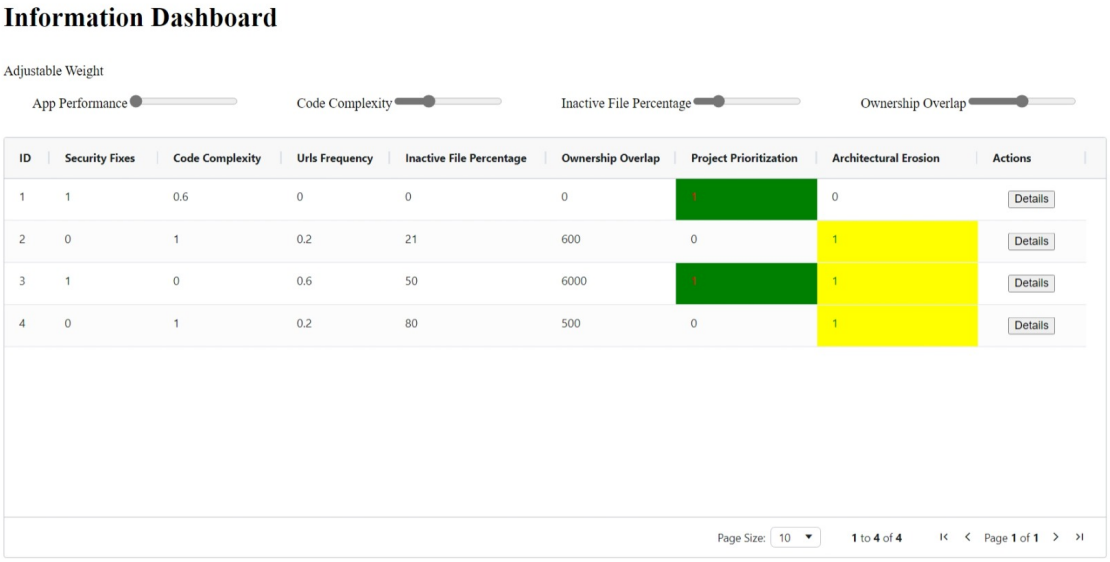

Figure 5.7: SMS without history trend for the project

  - *Inactive File Percentage*: Percentage of files that have not been modified over a specific period.
  - *Ownership Overlap*: Degree of overlap in contributions from different developers.
  - *Actions*: Provides options to view detailed information for each entry. The "Click to see details" button expands to show the detailed history and trends for the specific project.

- **Data Display: Indicators**
  - *Project Prioritization*: Indicates project prioritization status. This indicator uses different colors to show priority levels:
    * **Green**: Low priority
    * **Yellow**: Medium priority
    * **Red**: High priority
  - *Architectural Erosion*: Shows the level of architectural erosion, with colors indicating the severity:
    * **Green**: No erosion
    * **Yellow**: Moderate erosion
    * **Red**: High erosion

- **History Trend**
  - *Inactive File Percentage (Blue Line)*: Tracks the percentage of inactive files over weeks.

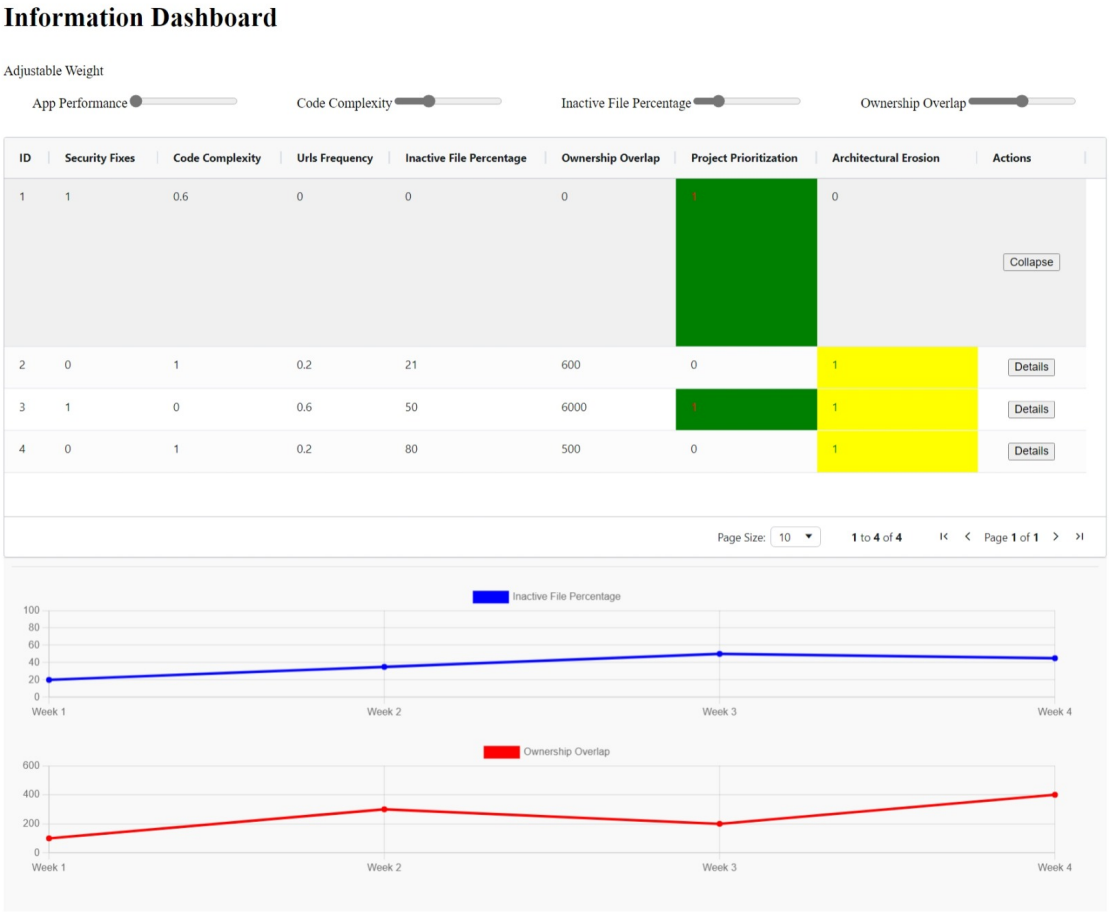

Figure 5.8: SMS with history trend for the project

– *Ownership Overlap (Red Line)*: Tracks the overlap in code ownership among developers over weeks.

Summary

This software system provides multiple measures to assess different aspects of software projects. The ability to adjust the weights of various factors allows users to tailor the prioritization and evaluation criteria according to their specific needs. The analysis of both git and web server logs offers valuable insights into code maintenance, developer contributions, and project usage patterns.

5.2.1.3 Evaluation

The evaluation meeting provided valuable feedback on the effectiveness and clarity of the indicators and the implemented software system.

The stakeholder was presented with two indicators: knowledge erosion and project prioritization. For knowledge erosion, the stakeholder was asked to rate the complexity and clarity of the indicators on a scale of 1-10, where 1 is easy to understand and 10 is hard. He rated the complexity of the knowledge erosion indicator as 3,

adding that it was straightforward to understand once the calculation method was explained. The clarity was also found satisfactory. For the project prioritization indicator, the stakeholder found it equally easily understandable, rating it similar to the knowledge erosion indicator.

The stakeholder, with over 20 years of experience, rated the complexity and clarity of the project prioritization indicator as straightforward and understandable. On a scale of 1 (very inaccurate) to 10 (very accurate), scoring it a 3 on a scale of 1-10. This indicator was effective in capturing the key challenges of project prioritization and was rated 7(very effective) in terms of accuracy. Similarly, the architectural erosion indicator was also rated positively for its clarity and relevance. The stakeholder emphasized that both indicators provided actionable data that assisted decision-making processes, effectively identifying the challenges they were designed to indicate. He acknowledged their relevance and suggested the potential addition of more indicators for a comprehensive evaluation. However, the stakeholder suggested that further improvements and additional features could improve the comprehensiveness of the evaluation. The stakeholder expressed a positive attitude towards the indicators' ability to provide valuable reference points for the team. He particularly noted the benefit of projects with priorities lower than "most important" but still significant. Furthermore, he advised considering future features such as an indicator reflecting the current alignment with the decided strategy.

On the evaluation of the SMS, the stakeholder emphasized the importance of documentation and transparency in the calculation of these indicators. He suggested providing detailed documentation or a help page to explain the values and their derivation, which would enhance understanding and trust among team members. He also suggested adding a tool that allows users to go deeper into the data underlying every indicator for better clarity.

The evaluation results indicate that the current indicators are on the right track in terms of complexity, clarity, relevance, and accuracy. However, adding documentation and features is necessary. The stakeholder's feedback provides a clear direction for future improvements, ensuring that the SMS remains relevant and useful for project prioritization and knowledge loss assessment. This ongoing refinement process will help in achieving a more robust and comprehensive measurement system.

The evaluation meeting provided important insights on answering RQ1b and RQ2b. The stakeholder, with over 20 years of experience, rated the complexity and clarity of the project prioritization indicator as straightforward and understandable. This indicator was effective in capturing the key challenges of project prioritization and was rated 7 in terms of accuracy. Similarly, the architectural erosion indicator was also rated positively for its clarity and relevance. The stakeholder emphasized that both indicators provided actionable data that assisted decision-making processes, effectively identifying the challenges they were designed to indicate. However, the stakeholder did suggest that further improvements in comprehensiveness and additional indicators could be implemented in the future.

6

Discussion

This chapter discusses the methods of the study and how they helped to answer the main research questions and their sub-research questions outlined in the introduction section. The discussion will then move on to the threat of validity and future work.

In our project, addressing the challenge of project prioritization proved difficult due to the limited amount of literature specifically focused on this topic. However, industry practices often involve using software measurement systems (SMS) to address similar issues [49]. Our approach uses SMS to define a project prioritization indicator, providing stakeholders with actionable insights on which projects to prioritize. This indicator bridges a significant gap in the literature by adapting project prioritization into software development.

For our approach, we developed an indicator that helps stakeholders make informed decisions on project prioritization, ensuring that the most critical projects are maintained first.

Our study introduces an indicator that tracks the percentage of inactive files and modification frequency differences among developers using git logs. This indicator can give stakeholders actionable guidance on critical but neglected projects, initializing knowledge-sharing sessions to prevent knowledge loss. By integrating data from git logs, we provide a comprehensive view of project architectural erosion status.

Unlike previous models like KLRM, which focus broadly on risk assessment, our approach specifically uses real-time data analysis to provide actionable insights. While KLRM is adaptable to various software development processes, our indicator is designed to continuously monitor and alert stakeholders about potential knowledge loss, thereby facilitating immediate actions.

To forecast architectural erosion from evolutionary history, Gracia [50] proposed architectural quality prediction models using an effective set of prediction metrics. One aspect is architectural co-change metrics. The number of changes and the number of co-changed files are two examples of co-change metrics. Then, they combine file-level metrics with cross-module level metrics and do a regression analysis without transforming the raw format of data. Our study presents a similar DM called ownership overlap, which measures the extent to which different developers contribute to the same files over time. It focused on file level and was inferred from multiple BMs. This gives a specialized view of the collaborative patterns within the codebase, allowing us to identify potential hotspots of architectural erosion.

This thesis contributes significant insights for industrial stakeholders as well as the research community. It offers empirical evidence for research from a specific setting, emphasizing the relationships between various data sources and project indicators. This study presents a useful software measurement method that helps detect the need for knowledge-sharing sessions and helps determine project priorities for industry stakeholders.

contribution jennifer flagged: This thesis aims to provide a preventive measure for preventing architectural erosion at early stages by identifying knowledge loss in developers.

6.1 Answering Research Questions

RQ1 The indicator identified for quantifying project priority is the Project Priority Status. This status is derived from multiple DMs, including the frequency of project-specific URL requests, the time to complete these requests, the number of security fixes, and code complexity, aggregating them to calculate the overall priority status. The evaluation meeting revealed that the project prioritization indicator is effective and well-received in terms of complexity, clarity, and accuracy. The stakeholder noted that the indicator provides valuable reference points for the team, aiding in distinguishing the relative importance of projects beyond the "most important" category.

RQ2 The indicator identified for quantifying architectural erosion is the Architectural Erosion Status. This status is calculated based on the percentage of inactive files in a project and the ownership overlap, which measures the extent of contributions from different developers to the same files over time. It could identify the challenge of maintaining code integrity and avoiding deterioration due to infrequent updates or excessive contributions from multiple developers, pointing to areas of potential architectural weakening. This indicator received a rating of 7 out of 10 in terms of accuracy in reflecting real-world challenges. The stakeholder emphasized that the indicator provided data on the file level, aiding in initializing knowledge-sharing sessions by highlighting the extent of architectural erosion.

6.2 Threats to Validity

Wohlin presented four main types of validity threats: conclusion validity, internal validity, construct validity, and external validity [51].

External validity refers to the degree to which study findings can be generalized to other contexts or organizational departments. Our initial obligation was to demonstrate that the proposed measures are valid in a narrow sense, which presents a validity threat. Without a stated hypothesis and planned experiment, it is difficult to show that these measures are valid in a broader sense.

We proposed solutions for two specific scenarios, but the generalizability of these scenarios to other contexts remains uncertain. This could be an issue for both

practitioners and academics seeking to draw inspiration from this research. Furthermore, since the research was conducted in an industrial setting in collaboration with a company inside one department, it is challenging to determine whether the Software Measurement System (SMS) is applicable to other departments or companies, it is also challenging to provide detailed information about the technical aspects of the toolchain due to confidentiality constraints.

Internal validity refers to the degree to which a research study develops a reliable cause-and-effect link. The data collected through interviews with participants could threaten the internal validity. The participants' subjective experiences and biases might influence their perception of the two scenarios. This is important given the constrained perspectives provided by the three participating interviewees, all of whom have similar demographic characteristics.

Construct validity refers to how well a test or tool measures the concept it is intended to measure. When it comes to the validation for our proposed indicator by the evaluation meeting, the validity we argued was solely based on the credibility of our stakeholders, rather than on a rigorous validation procedure. Also, we made the connection between the number of issues and the priority in the repo, this validation was made by showing our measures are related to some well-known, existing measures, this implies an assumption that the existing measures are valid.

7

Conclusion

This thesis finished a comprehensive study on the instantiation and implementation of a Software Measurement System (SMS) designed to assist in project prioritization and the prevention of architectural erosion within the automotive industry. Two critical indicators were discovered by the research and validated through practical application and stakeholder feedback: one for project priority and another for architectural erosion. The results showed that these indicators are useful, relevant, and effective at giving stakeholders useful information to help them make decisions on knowledge retention and project management. However, further formal validation will be required in the future to guarantee that extending the concept is worthwhile. The complexity of the scenarios, the volume of data, and the trade-offs between short- and long-term objectives were identified to be the main challenges to the validation of these metrics.

7.1 Future Work

This study primarily focuses on the automotive industry. However, in the future, the scope can be expanded to other fields, as the software measurement system is highly flexible and adaptable to various types of software organizations.

One limitation of our study is the time frame, which was defined as one month. Future studies could extend this period to a longer duration, thereby reducing internal threats to validity. Longer time frames would allow for a deeper analysis of historical trends or patterns.

During cycle 2's evaluation meeting, stakeholders expressed they wanted a help page in the SMS that explains the meaning of the data. Due to time constraints, we were unable to implement this feature in the current study. However, this could be an important addition in future work, enhancing user understanding of the system.

Additionally, future research could involve more teams using these indicators to prioritize projects and minimize knowledge erosion risks. By involving a larger number of teams, we can gather more feedback and adjust indicators to have more generality.

In summary, these future directions aim to enhance the applicability, validity, and user-friendliness of the software measurement system.

Bibliography

[1] H. Jafarzadeh, J. Heidary-Dahooie, P. Akbari, and A. Qorbani, "A project prioritization approach considering uncertainty, reliability, criteria prioritization, and robustness," *Decision Support Systems*, vol. 156, p. 113 731, 2022.

[2] H. Alaidaros, A. Bakodah, and S. F. Bamsaoud, "A review on requirements prioritization approaches of software project management," in *2022 International Conference on Intelligent Technology, System and Service for Internet of Everything (ITSS-IoE)*, 2022, pp. 1–5. DOI: 10.1109/ITSS-IoE56359.2022.9990930.

[3] Y. Bugayenko, A. Bakare, A. Cheverda, *et al.*, "Prioritizing tasks in software development: A systematic literature review," *Plos one*, vol. 18, no. 4, e0283838, 2023.

[4] M. Staron and W. Meding, "Software development measurement programs," *Springer. https://doi. org/10.1007/978-3-319-91836-5*, vol. 10, p. 3 281 333, 2018.

[5] A. Purnus and C.-N. Bodea, "Project prioritization and portfolio performance measurement in project oriented organizations," *Procedia-Social and Behavioral Sciences*, vol. 119, pp. 339–348, 2014.

[6] M. Daneva, E. Van Der Veen, C. Amrit, *et al.*, "Agile requirements prioritization in large-scale outsourced system projects: An empirical study," *Journal of systems and software*, vol. 86, no. 5, pp. 1333–1353, 2013.

[7] M. Foucault, M. Palyart, X. Blanc, G. C. Murphy, and J.-R. Falleri, "Impact of developer turnover on quality in open-source software," in *Proceedings of the 2015 10th joint meeting on foundations of software engineering*, 2015, pp. 829–841.

[8] C. B. Jaktman, J. Leaney, and M. Liu, "Structural analysis of the software architecturea maintenance assessment case study," in *Software Architecture: TC2 First Working IFIP Conference on Software Architecture (WICSA1) 22–24 February 1999, San Antonio, Texas, USA*, Springer, 1999, pp. 455–470.

[9] Y. Schwartzman and A. Borning, "The indicator browser: A web-based interface for visualizing urbansim simulation results," in *2007 40th annual Hawaii international conference on system sciences (HICSS'07)*, IEEE, 2007, pp. 92–92.

[10] M. Staron, W. Meding, G. Karlsson, and C. Nilsson, "Developing measurement systems: An industrial case study," *Journal of Software Maintenance and Evolution: Research and Practice*, vol. 23, no. 2, pp. 89–107, 2011.

[11] K. Pandazo, A. Shollo, M. Staron, and W. Meding, "Presenting software metrics indicators-a case study," in *Proceedings of the 20th international conference on Software Product and Process Measurement (MENSURA)*, vol. 20, 2010.

[12] L. Johansson, W. Meding, and M. Staron, "An industrial case study on visualization of dependencies between software measurements," *Software Engineering Research and Practice in Sweden*, vol. 21, p. 23, 2007.

[13] F. Kaymaz, "Prioritisation and selection of the right business and it requirements in the software engineering process," 2013.

[14] F. O. Bjørnson and T. Dingsøyr, "Knowledge management in software engineering: A systematic review of studied concepts, findings and research methods used," *Information and software technology*, vol. 50, no. 11, pp. 1055–1068, 2008.

[15] M. Rashid, P. M. Clarke, and R. V. OConnor, "A systematic examination of knowledge loss in open source software projects," *International Journal of Information Management*, vol. 46, pp. 104–123, 2019.

[16] A. Jansen and J. Bosch, "Software architecture as a set of architectural design decisions," in *5th Working IEEE/IFIP Conference on Software Architecture (WICSA'05)*, IEEE, 2005, pp. 109–120.

[17] Q. Zhou, "The impact of job satisfaction affect on turnover intention: An empirical study based on the circumstances of china," in *2009 Second International Conference on Education Technology and Training*, IEEE, 2009, pp. 220–223.

[18] M. Lima, I. Ahmed, T. Conte, E. Nascimento, E. Oliveira, and B. Gadelha, "Land of lost knowledge: An initial investigation into projects lost knowledge," in *2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, IEEE, 2019, pp. 1–6.

[19] S. B. D. Dakhli and M. B. Chouikha, "The knowledge-gap reduction in software engineering," in *2009 Third International Conference on Research Challenges in Information Science*, IEEE, 2009, pp. 287–294.

[20] J. Nonnen, "Attacking software knowledge erosion with gamification," in *International Workshop on Software Knowledge*, SCITEPRESS, vol. 2, 2012, pp. 3–10.

[21] S. Vesiluoma, "Making knowledge sharing visible in software engineering," 2007.

[22] I. Nonaka, "The knowledge-creating company," in *The economic impact of knowledge*, Routledge, 2009, pp. 175–187.

[23] A. Baabad, H. B. Zulzalil, S. B. Baharom, *et al.*, "Characterizing the architectural erosion metrics: A systematic mapping study," *IEEE Access*, vol. 10, pp. 22 915–22 940, 2022.

[24] L. De Silva and D. Balasubramaniam, "Controlling software architecture erosion: A survey," *Journal of Systems and Software*, vol. 85, no. 1, pp. 132–151, 2012.

[25] Software, S. E. S. Committee, *et al.*, "Ieee standard adoption of iso/iec 15939: 2007 systems and software engineering measurement process," *Electronic book]. IEEE Computer Society*, 2008.

[26] M. Berry and M. F. Vandenbroek, "A targeted assessment of the software measurement process," in *Proceedings Seventh International Software Metrics Symposium*, IEEE, 2001, pp. 222–235.

[27] T. Tahir, G. Rasool, and C. Gencel, "A systematic literature review on software measurement programs," *Information and Software Technology*, vol. 73, pp. 101–121, 2016.

[28] A. Shaygan and Ö. Testik, "A fuzzy ahp-based methodology for project prioritization and selection," *Soft Computing*, vol. 23, pp. 1–11, Feb. 2019. DOI: 10.1007/s00500-017-2851-9.

[29] J. Karlsson and K. Ryan, "A cost-value approach for prioritizing requirements," *Software, IEEE*, vol. 14, pp. 67–74, Oct. 1997. DOI: 10.1109/52.605933.

[30] Wikipedia contributors, *Failure mode and effects analysis — Wikipedia, the free encyclopedia*, [Online; accessed 22-May-2024], 2024. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Failure_mode_and_effects_analysis&oldid=1218951234.

[31] Wikipedia contributors, *Moscow method — Wikipedia, the free encyclopedia*, [Online; accessed 22-May-2024], 2024. [Online]. Available: https://en.wikipedia.org/w/index.php?title=MoSCoW_method&oldid=1223939235.

[32] Wikipedia contributors, *Kano model — Wikipedia, the free encyclopedia*, [Online; accessed 22-May-2024], 2024. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Kano_model&oldid=1217409946.

[33] *RICE Scoring Model — productplan.com*, https://www.productplan.com/glossary/rice-scoring-model/, [Accessed 22-05-2024].

[34] Dec. 2023. [Online]. Available: https://praxie.com/artificial-intelligence-ai-digital-manufacturing-project-prioritization/.

[35] M. M. Lehman, "On understanding laws, evolution, and conservation in the large-program life cycle," *Journal of Systems and Software*, vol. 1, pp. 213–221, 1979.

[36] O. B. Newton, S. M. Fiore, and J. Song, "Expertise and complexity as mediators of knowledge loss in open source software development," in *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, SAGE Publications Sage CA: Los Angeles, CA, vol. 63, 2019, pp. 1580–1584.

[37] J. Nonnen and P. Imhoff, "Identifying knowledge divergence by vocabulary monitoring in software projects," in *2012 16th European Conference on Software Maintenance and Reengineering*, IEEE, 2012, pp. 441–446.

[38] V. Manner, "Knowledge loss in software development," M.S. thesis, 2023.

[39] M. P. Robillard, "Turnover-induced knowledge loss in practice," in *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2021, pp. 1292–1302.

[40] R. Li, "Understanding, analysis, and handling of software architecture erosion," 2023.

[41] E. Bouwers and A. Van Deursen, "A lightweight sanity check for implemented architectures," *IEEE software*, vol. 27, no. 4, pp. 44–50, 2010.

[42] A. R. Hevner, S. T. March, J. Park, and S. Ram, "Design science in information systems research," *Management Information Systems Quarterly*, vol. 28, no. 1, p. 6, 2008.

[43] K. Peffers, T. Tuunanen, M. A. Rothenberger, and S. Chatterjee, "A design science research methodology for information systems research," *Journal of management information systems*, vol. 24, no. 3, pp. 45–77, 2007.

[44] J. Iivari and J. R. Venable, "Action research and design science research-seemingly similar but decisively dissimilar," 2009.

[45] E. Knauss, "Constructive master's thesis work in industry: Guidelines for applying design science research," in *2021 ieee/acm 43rd international conference on software engineering: Software engineering education and training (icse-seet)*, IEEE, 2021, pp. 110–121.

[46] S. Houde and C. Hill, "What do prototypes prototype?" In *Handbook of human-computer interaction*, Elsevier, 1997, pp. 367–381.

[47] N. Fenton and J. Bieman, *Software metrics: a rigorous and practical approach*. CRC press, 2014.

[48] Oct. 2023. [Online]. Available: `https://en.wikipedia.org/wiki/Software_metric`.

[49] B. Pettichord, J. Bach, and C. Kaner, *Lessons learned in software testing: A context-driven approach*. Wiley, 2013.

[50] J. Garcia, E. Kouroshfar, N. Ghorbani, and S. Malek, "Forecasting architectural decay from evolutionary history," *IEEE Transactions on Software Engineering*, vol. 48, no. 7, pp. 2439–2454, 2021.

[51] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in software engineering*. Springer Science & Business Media, 2012.

A

Appendix 1

A.1 Interview 1: process & questions

Current process and challenges(RQ1a and RQ2a)

- How much do you know about software measurement system?
  - Do you use metrics in your work? Any experience in interacting metrics?
- Can you describe your experience with using metrics to formulate decisions in software development? What metrics have you found most valuable?
- How does your team currently prioritize projects? What challenges do you face in this process?
- What strengths and weaknesses do you see in your current approach to project prioritization?
- What specific challenges do you encounter with knowledge management and architectural erosion?
- Can you provide examples of how knowledge loss has impacted your projects?
- Can you share examples of the types of log data that you find particularly valuable for your decision-making process for project prioritization and knowledge erosion?
- Are there some parts of the system that would provide metrics that trigger those decisions frequently? Are there some parts that dont trigger very often? Would you describe their characteristics?

A.2 Evaluation meeting: process & questions

RQ1a: What challenge should the indicator indicate?

- On a scale of 1-10, where 1 is Not complex and 10 is Very complex, how clear and understandable are the indicators to stakeholders?
- On a scale of 1-10, where 1 is Not a lot and 10 is A lot, how accurate is the indicator capturing challenges in assessing project priority?

- Are there any limitations in the indicator's ability to capture project prioritization challenges?

RQ1b: To what extent can the identified indicator help?

- On a scale of 1-10, where 1 is Not a lot and 10 is A lot, how effective is the indicator in providing actionable data on project priority?
- In what ways will the indicator be useful in decision-making for project prioritization?

RQ2a: What challenge should the indicator indicate?

- On a scale of 1-10, where 1 is Not complex and 10 is Very complex, how clear and understandable are the indicators to stakeholders?
- On a scale of 1-10, where 1 is Not a lot and 10 is A lot, how accurate is the indicator capturing challenges in assessing knowledge loss?
- Are there any limitations in the indicator's ability to capture knowledge loss challenges?

RQ2b: To what extent can the identified indicator help?

- On a scale of 1-10, where 1 is Not a lot and 10 is A lot, how effective is the indicator in tracking and quantifying knowledge loss?
- How has the indicator been utilized to mitigate knowledge loss within the program?
- On a scale of 1-10, where 1 is Not a lot and 10 is A lot, to what extent will the indicator contribute to strategies aimed at preventing knowledge loss?

A.3 Consent From

Below is the consent form used for individual interviews in this study.

CHALMERS

Consent and information about processing of personal data in student thesis

I agree to my personal data in the form of:
name, job, audio / video recording or other information
may be treated by Chalmers University of Technology for the study:
A master thesis conducted by Chalmers students Xiaoyan Shan and Sisi Lai. The purpose of the study is to explore the potential of log file analysis for decision-making in the industry.

Information

Your personal data will be handled as follows:
The data will be stored within Volvo Group system, discussed with school supervisor and examiners, analyzed and published as a part of a master thesis.
Your consent is valid until further notice. You have the right to withdraw your consent at any time. You do this through contacting *Xiaoyan Shan* or xiaoyans@chalmers.se.
If you withdraw your consent, we will cease processing personal data we have collected with the support of your consent. Some information may be saved due to Chalmers obligations under Swedish archive legislation.
Chalmers University of Technology, org. No. 556479-5598 is personal data controller. You can find Chalmers privacy policy at www.chalmers.se.
As a participant you have the right to receive information about how your personal data is processed. You have the right to have incorrect information corrected, redundant data deleted, request that processing shall be restricted and data transferred to another actor. You also have the right to submit a complaint to the Swedish Authority for Privacy Protection (Integritetsskyddsmyndigheten). Do you have any questions about Chalmers's processing of personal data contact Chalmers's data protection officer at dataskydd@chalmers.se.

I agree that Chalmers University of Technology processes personal data about me in accordance with the above.

Place:	Signature
Date:	Name clarification

The form is drawn up in duplicate.