CHALMERS | UNIVERSITY OF GOTHENBURG

Identifying Technical Debt Impact on Maintenance Effort

- An Industrial Case Study

Master of Science Thesis in the Programme Software Engineering

ERIC BRITSMAN
ÖZGÜR TANRIVERDI

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
Göteborg, Sweden, June 2015

Identifying Technical Debt Impact on Maintenance Effort
- An Industrial Case Study

ERIC BRITSMAN,
ÖZGÜR TANRIVERDI

Supervisor: ANTONIO MARTINI
Examiner: MIROSLAW STARON

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering
Göteborg, Sweden, June 2015

Abstract

Technical Debt refers to sub-optimal solutions in software development that affect the life cycle properties of a system. Source Code Technical Debt is considered to be a problem for many software projects, as neglecting Technical Debt on actively developed software will increase the required effort to maintain the software as well as to extend it with new features. However, refactoring Technical Debt also requires effort which is why it must be investigated if the Technical Debt is worth resolving. To be able to make such decisions, the existing Technical Debt must be correctly identified and presented in an understandable way. This thesis addresses the problem by conducting a case study at Ericsson, where Technical Debt in a large industrial C/C++ project has been investigated. The investigation was done by designing a measurement system based on ISO standard 15939:2007 and reviewing Technical Debt measurement tools suitable for its construction. The investigation also included correlating the resulting Technical Debt measurements with maintenance effort by applying triangulation of several methods. This allowed the Technical Debt types to be prioritized based on their correlation strength. The chosen Technical Debt measures are presented as a unified indicator that indicates file refactoring priority in order to support decision-making regarding Technical Debt Management, so that additional maintenance effort can be avoided in the future.

Acknowledgements

We would like to give a special thanks to our academic supervisor Antonio Martini for his continuous support and suggestions throughout this thesis work. We would also like to thank our on-site supervisor Patrik and the development team at Ericsson for providing the commits necessary for this thesis work. Additional thanks are also in order to Patrik due to his suggestions for and confirmations of the work through weekly meetings. Furthermore, we are grateful to Per for helping us to get started with the on-site tools used in this study. Last but not least, it has been a great experience to conduct this thesis work at Ericsson, so many thanks goes to Pär for initiating the project.

Özgür Tanriverdi and Eric Britsman, 2015-05-22.

Vocabulary

TD: Technical Debt.

Measurement System: Aggregation of measures into a joint indicator.

Effort: Hour spent to complete a specific commit.

Modified LoC: Modified line(s) of code of a file/commit.

Commit: Code changes and additions submitted to a versioning system.

Commit Difficulty: Modified LoC per hour spent ratio.

On-site: Refers to the work environment at the company.

StDev: Standard Deviation.

Lightweight Tool: A tool that does not require extensive setup/installation.

Table of Contents

1. Introduction

Technical Debt (TD) refers to sub-optimal solutions in software development that affect the life cycle properties of a system. According to Nugroho et al. [5], neglecting TD on actively developed software can cause inefficiency and expansion difficulties in the system, and such overhead cost is considered as the interest of TD. If TD is not properly managed, the growth of TD over time will result in the growth of the interest, which will increase the required effort to maintain the software as well as extending with new features [5]. This is a problem that can cause severe long-term consequences [1]. For example, TD was estimated to cost the global software industry 500 billion dollars in 2010 [1].

A common type of TD is Source Code TD, which refers to short-term solutions in coding. A common cause of Source Code TD is the rapid evolution of software and frequent deadlines present in Agile processes, as they can drive developers to using such short-term solutions [1]. Examples of Source Code TD include Code Duplication and high Cyclomatic Complexity [5]. Another TD type measurable on source code level is Non-Allowed Dependencies between components. Martini et al. [2] claim this is an especially severe type of TD, due to the fact that these dependencies might cause ripple effects when changes in the source code are made.

An important factor to take into account is that TD also requires effort to resolve, referred to as the principal of TD [3]. Therefore, it is important to decide when to pay the principal during TD Management. To be able to make such decisions, the existing TD must be correctly identified and presented in an understandable way. As such, this thesis measures and visualizes source code level TD measures in an industrial context as a joint indicator. This thesis also analyzes correlation strength between the chosen TD measures and commit difficulty, in order to indicate which measures has the strongest impact on maintenance effort within the case context. This combination in turn supports decision-making concerning TD Management. These procedures are based on approaches [3], metrics [5][6][7] and best practices [20][21][22][23] from previous literature. This case study was conducted with an agile development team at an Ericsson site, using suitable TD measurement tools identified during the course of the project. In order to measure TD interest, comparisons have been made between occurrence-levels of measured TD types and levels of maintenance effort, by investigating historical data spanning approximately three months.

1.1 Purpose

The purpose of this study is to identify and apply suitable methods/tools for identifying, quantifying, presenting and prioritizing source code level TD in C/C++ applications. This included correlating these TD measurements with maintenance effort measured as modified LoC per hour in commits, where effort has been manually specified in the accompanying message. This allows the TD types to be prioritized based on their correlation strength to increased maintenance effort. Their correlation strength represents the severity of their interest.

1.2 Scope

This is a holistic case study [4], meaning that the study is delimited to one case (due to the established contract with the company in question). The scope is limited to identifying and visualizing Source Code TD from historical/current data to allow for validation of the found TD, by using a combination of build-integratable tools, ad-hoc parsers and manual processes. This study has measured TD in production code specifically rather than testing code, as requested within the case context. The restriction to build-integratable tools is due to context specific requirements. Estimating future TD accumulation and interest payments is outside the scope of this thesis, due to time restrictions. Focusing on gathering measures from relatively current data has given the opportunity to discuss TD qualitatively, as it is easier for team members to discuss data on their recent work.

The generalizability of the correlation and validation results of this study are limited since the case only covers a single development team from a department of a large company, and mainly one large application out of four. The correlation process, tool recommendations and measurement system should however be of interest to companies in similar contexts and maintainers of C/C++ projects of any size. This includes the method for finding difficult commits, based on specifying effort as hours spent and comparing to modified LoC, which could easily be applied to other programming languages.

1.3 Research Questions

The following research questions have been used to fulfill the purpose of this study:

- **RQ1:** How can an understandable aggregation of multiple source code level TD types be designed and implemented?
- **RQ2:** How can source code level TD be measured in large industrial C/C++ projects?
- **RQ3:** How can interest be used to prioritize source code level TD types?

1.4 Main Contributions

The main contributions of this thesis are:

- A measurement system based on ISO standard 15939:2007 [15], that indicates refactoring priority of files based on levels of several TD types. This answers **RQ1**.
- A mapping of Source Code TD tools for C/C++, with accompanying recommendations. This answers **RQ2**.
- A process for correlating specific TD types and overall TD to commit difficulty, allowing for TD type prioritization. This answers **RQ3**.

1.5 Thesis Outline

The second chapter first presents a thorough explanation of TD and the chosen TD types, followed by a review of related research. The third chapter then provides a detailed description of the processes and accompanying methods that were used to reach the study’s three main contributions, while the fourth chapter presents the results of constructing a realization of the measurement system, reviewing tools, correlating TD with effort and presenting TD types as a unified indicator. These contributions and how they answer the research questions are then discussed in the fifth chapter. Finally, the conclusions of this study and the significance of its contributions are summarized in the sixth chapter.

2. Theoretical Background

This chapter provides further details on TD and related concepts, and details the TD types that were chosen for this study. The technique for measuring the interest of the TD types is also detailed. Finally, a review of related research is presented, and this study's placement with regards to the existing literature is explained.

2.1 TD Overview

As outlined by Li et al. [9], TD can be split into several areas (all including both interest and principal) based on the origin of a compromise. On the other hand, that which is not TD includes unimplemented features as well as runtime properties such as performance, but existing TD may be the underlying cause. This study focuses on the area of Source Code TD while touching upon Architecture TD and Defect TD as well. According to Li et al. [9], Source Code TD includes such types as Code Duplication and over-complex code, while Defect TD refers to unmanaged defects and Architectural TD refers to architectural decisions that reduce maintainability. Architecture TD can be measured through source code, by analyzing dependencies against an intended architecture as well as by analyzing the amount of dependencies in general [9]. Defect TD is also studied through source code, and is essentially covered by standard static verification tools.

The areas that this study focuses on have received plenty of attention in the past. In fact, Li et al. [9] state that over half of their 94 reviewed studies concern Source Code TD to some degree. They emphasize that this is related to the amount of available tools, as well as the fact that most team members work with source code on a daily basis. They also reason that Source Code TD is a form of TD that the team members themselves should be able to resolve. As can be seen in section 2.2 however, major studies on these TD areas have used measurement tools that cannot be used in the context of this case study (due to language incompatibility or tool unavailability), hence why one of this study's goals is to answer **RQ2**.

TD Management (TDM) is also divided, in this case into activities centered on either dealing with existing TD or preventing potential future TD [9]. Out of these categories, this study focuses mainly on TD Identification, as it is the first step to engaging in other TDM activities such as TD Monitoring, Prioritization, Communication and Documentation. This study lacks the statistical sample rate required for estimations of future TD, hence the focus on other activities.

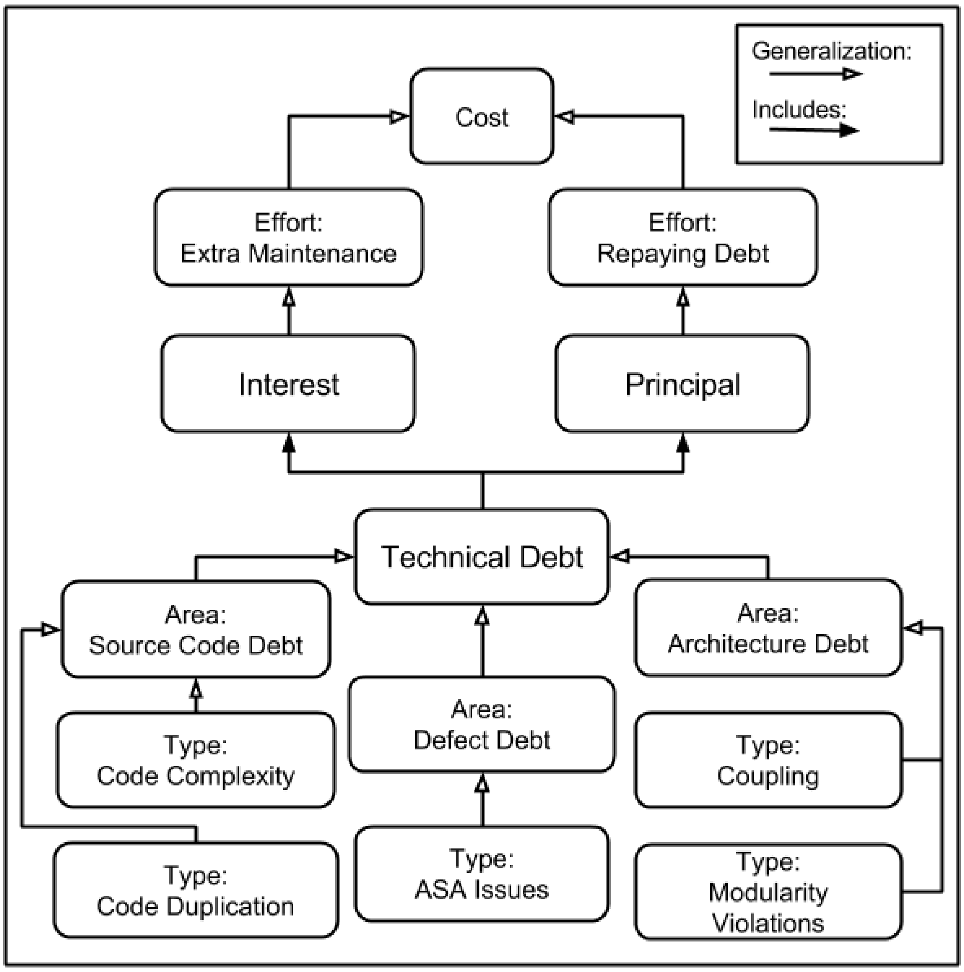

Figure 1: TD overview

2.1.1 Chosen TD Types

The TD types that were chosen for measurement (see Fig. 1) within the case context are:

2.1.1.1 Code Complexity: McCabe Cyclomatic Complexity

Cyclomatic Complexity relies on each function in a source code file being graded with one point for each independent path through the function. When a function is graded >15, that function is considered complex [6]. This measure was chosen based on its frequent use for maintainability predictions in previous research [3][5][6][7][11][13]. Higher complexity potentially increases maintenance cost due to its effect on source code readability, and this extra cost can be interpreted as the interest of this TD type [7]. Additionally, the amount of test cases/complexity of test code required to verify that the method works as intended increases with its Cyclomatic Complexity (essentially one case per point is required for full branch coverage). High Cyclomatic Complexity can also affect the changeability of a method [13], due in part to the reduced readability. As Cyclomatic Complexity is a method based measure, certain steps are required to transform it into an unbiased file-level rating (the principle used by Antinyan et al. [6] was followed), which is explained in detail in section 3.2.1.2.

2.1.1.2 Code Complexity: Halstead Error/Delivered Bugs

Another form of code complexity is Halstead Error (also known as Halstead Delivered bugs). This measure is derived from other measures in the Halstead suite, which at its core is based on the number of unique and total operands, and the number of unique and total operators, in a method [19]. Halstead Error specifically estimates the defect proneness of a method. The measure it is derived from is incorporated into Hewlett-Packard's Maintainability Index [11], but Halstead Error itself has not been used in any of the previous literature reviewed as part of this study. This measure was chosen mainly as a byproduct of it being available in one of the tools used in this study, and it was also the easiest to explain out of the available Halstead measures (which is vital for a measure to be adopted by team members, according to Heitlager et al. [13]). The interest of this measure is the amount of time spent dealing with any impact of these possible defects, as well as increased source code review difficulty due to the complexity required to get a high Halstead Error value. This method level measure also requires certain steps to transform it into a file-level rating, which is explained in detail in section 3.2.1.2.

2.1.1.3 Code Duplication

Code Duplication is commonly created during the development and maintenance of large software systems [12]. Excessive amounts of duplication in a system can potentially influence many quality attributes, while also rendering a system larger than it needs to be [13]. According to sources used by Grundèn & Lexell [1], Code Duplication violates separation of concerns, and can impact the modifiability, testability, reusability and understandability of a system. This risk of high and varied impact is due to the fact that refactoring may need to be applied in all places the code is duplicated, rather than one central file. It can also be quite difficult to find all instances where a certain block is duplicated [12]. Manual investigation is often required to judge the severity of a duplicated block [12], thus duplication is measured in this study to assist in such investigations.

2.1.1.4 Static Analysis Issues

Automatic static analysis tools analyze code looking for issues that might cause faults or might degrade some dimensions of software quality [3]. These issues can be either general or highly language-specific. The interest that these issues can cause is the amount time that was used to identify and deal with the effects of these defects, while refactoring them ensures they have no effect even if the effect would have been trivial. Most ASA tools divide the issues they can detect into separate priority categories by default. As these tools can measure a multitude of issues this study makes a restriction to measuring only the issues that the tool itself considers as the most important, in order to reduce false-positives.

2.1.1.5 Dependency Count

Dependency count ('includes' in C/C++) is a very basic form of coupling (equivalent to ATFD used by Rapu et al. [10]). As changes in a file can lead to required changes in files that depend on it, having many dependencies is a form of TD due to higher risk of change cascade. Izurieta & Bieman [17] also claim that as dependencies increase, the system becomes harder to extend. This measure was selected over more "lower-level" coupling measures that prioritize dependencies based on how they are used, as the initially intended coupling measurement tool did not work on-site, and manually analyzing dispersed/intensive coupling [14] requires quite complex algorithms.

2.1.1.6 Non-Allowed Dependencies

Non-Allowed Dependencies are a form of Modularity Violation, and refer to dependencies that do not follow the original architecture [1]. According to results by Zazworka et al. [3] Modularity Violations, which they measure in the form of Non-Allowed Dependencies, point to change-prone files. As extra refactoring is a typical form of TD interest, Non-

Allowed Dependencies can be interpreted as a form of TD. According to Grundèn & Lexell [1], Non-Allowed Dependencies often occur in rapidly evolving software, where the architecture cannot follow. Another factor that creates Non-Allowed Dependencies is that developers may not be aware of what is allowed [1]. Non-Allowed Dependencies can require extra changes in other files due to the existence of the dependency. If Non-Allowed Dependencies exist that are unknown, this might cause time estimations to be inaccurate and delay releases of features [2]. The reason this would be unknown is that Non-Allowed Dependencies cause inconsistency against existing architecture documentation, meaning the documentation can no longer be trusted. This measure relies on the existence of a defined architecture, which provides the rules that dependencies are validated against [1][2]. One such diagram was available for one large application on-site, allowing this measure to be implemented.

2.1.2 Chosen TD Interest Indicator

As discussed by Grundèn & Lexell [1], producing accurate estimations of TD interest and principal are both difficult and time-consuming, which is why this study does not focus on estimations. As detailed in section 3.3.2, this study instead uses modified LoC divided by hours spent working on a commit to indicate the interest of TD types found in the related files (in order to answer **RQ3**). This method, which is less reliant on estimations even though the effort value is still partially estimated by the team members, has not been used in the related research encountered during the literature review phase. Its purpose was to produce more accurate correlations with raw effort, compared to previous studies using effort surrogates. Compared to change frequency (which is the most similar interest indicator in previous studies), this method instead measures the difficulty of individual changes rather than how often these changes occur. With change frequency change size is ignored while modified LoC per hour instead prioritizes changes based on change size divided by time. A benefit to change frequency however, is that it is a file level measure while modified LoC per hour is limited to commit level. As can be seen by the case context specific correlation results (see section 4.5), the effort indicator has a major weakness in regards to sample size. Enough commits with effort specified need to be produced (while change frequency can be applied naturally to the entire commit history).

2.2 Related Research

Zazworka et al. [3] have studied TD identification using four approaches (code smells, grime, ASA Issues and Modularity Violations). Their results show that different approaches find TD in different parts of the code. Their results also indicate that certain types of TD (from Source Code or Architecture area) are of higher priority based on their likelihood to cause defects and/or require changes. Their study is however limited to a single (large) open source project written in Java, which has also affected their choice of tools for each of the four approaches. Even so, their findings were of great interest to apply in the context of this study such as their assessments of God Classes (files with high Cyclomatic Complexity and number of dependencies [10]), Modularity Violations, and Dispersed Coupling. Their prioritization is based on correlation strength, which is calculated by applying Pearson’s Correlation, Conditional Probability, Chance Agreement and Cohen’s Kappa to the two phenomena being studied, and is replicated in this study.

Zazworka et al. [7] have also evaluated human identification of TD and compared it to automated identification. For human identification a TD template accompanied by a short questionnaire was used, whose questions have affected how this study intends to extract similar information during qualitative investigation of team members’ work. Their results indicate that human identification finds different forms of TD compared to tool usage, which is why this study plans to do both to make sure it is focusing on the right forms. The tools used in this study are not applicable in the case context, but they highlight that the so called "priority 1"-issues in the tool FindBugs was especially helpful. This study in turn also focuses on measuring for defects that fall into the highest category when using the identified FindBugs alternatives. Zazworka et al. [7] also achieve similar success with code smells related to coupling as they did in their previous work [3].

Another study was partially conducted at Ericsson by Antinyan et al. [6], and focuses on identifying risk areas in code. They define risky code as "files that are fault prone, difficult-to-manage or difficult to maintain" which aligns well with definitions of (Source Code) TD such as the one given by Zazworka et al. [3]. Their aim was to enable systematic tool-assisted identification and prioritization of risks during Agile development. Their results are quite useful for this study as well, especially due to the similarities in case context. Specifically, their study provided an aggregated measure to take file-size into account when measuring McCabe's complexity, which allows code complexity to be measured at file instead of function level. Their refactoring prioritization method based on file complexity and change frequency within a specific time frame was however not used in this project, as change frequency was not measured. The tool they developed was not usable in this case context due to a change in code versioning system.

Rieger et al. [12] propose visualization strategies for better communication and prioritization of Code Duplication based on research on both industrial and open source projects. As part of their study they also provide insight into identification of Code Duplication. They reason that refactoring of duplication is non-trivial due to required decisions on were shared blocks shall be allowed to remain. Refactoring of Code Duplication leans more towards manual investigation rather than automation [12]. They also present several metrics for duplication from which this study specifically uses what they call LCC (the amount of Code Duplication in a file). This metric tracks the number of lines whose clones can be found in the same or other files for each file in the system. Their polymetric views are quite cluttered however, and thus they were not reused in this study.

Heitlager et al. [13] have provided an alternative maintenance measurement model to replace an older one (the Maintainability Index found in [11]). They disapprove of this older model due to obfuscation of which measures contributed to the derived index value, which in turn makes it difficult for team members to understand how to improve said value. They reason that providing measures that team members can easily influence improve the team members' acceptance of the model. To replace MI, they suggest a set of minimal requirements for a practical maintainability model based on source code analysis, by mapping source code metrics to sub-characteristics of maintainability from an ISO standard. This model has strong parallels to how Source Code TD is measured, as it shares metrics with TD-oriented studies such as [3][5][7]. As such, this model was kept in mind when choosing which derived measures this study would focus on. Heitlager et al. [13] also suggest threshold values for their derived measures, which they claim to be language and context independent based on experience and expert opinion from "dozens of industrial projects". This argument is clearly lacking in validity since results from these projects are not presented, but these thresholds at least provide a starting point which then can be adjusted based on the case context.

From an industrial perspective, a master level thesis that aimed at finding methods for how to handle Non-Allowed Dependencies (a type of Modularity Violation) was recently written by Grunden & Lexell [1]. In order to find such methods, knowledge about how the problem behaves was explored in a real life context at Ericsson Radio base stations department. They successfully connected source level elements to components and found Modularity Violations between them through static analysis [1]. However, the tool they developed was not fully automatic, and they limited themselves to measuring Non-Allowed Dependencies only rather than including other TD types. This study also analyzes existing dependencies against a set of allowed dependencies, but uses a different tool for this purpose with an improved feature set. Grunden & Lexell [1] also derive an indicator for prioritizing Modularity Violations, following the steps of ISO standard 15939:2007 [15]. This standard was deemed suitable for defining the indicator created from measures used in this project as well.

Nugroho et al. [5] focus on empirically estimating TD through estimation of Repair Effort (RE) as well as estimation of (extra) maintenance effort. An ideal quality level is extracted, and the gap between the current and ideal quality level is said to represent the current TD level. The estimated extra maintenance effort (interest) is based on historical maintenance costs, current quality level and current TD level. Similarly to Zazworka et al. [7], they advocate the usage of historical data as grounds for future TD estimations. Metrics were measured on a file/function level and threshold values were chosen based on previous statistical studies, which made them suitable for this study as well. The tool they have used to extract these measurements however is commercial and thus not usable for this case study, as the only commercial tools used were already purchased since previously on-site, or had free trials available.

From the data Nugroho et al. [5] extract from applications they estimate RE which ties the existing TD both to the percentage of source code that needs changing as well as the amount of man-months required. Specifically the man-month component requires feedback from technology experts tied to the application in question in order to set an accurate value. Due to the expert knowledge required, as well as the difficulty in identifying ideal quality levels and historical maintenance costs, their methodologies have not been used in this study. Their strategy of making estimations based on historical data is quite suitable for the case context, and would allow for a more quantitative approach. There is however a major limitation in the fact that the study was only performed on Java projects. Potentially a lot of rework would be needed in order to adapt their process to other languages.

2.3 Study Placement

While there are several relevant studies in both the areas of Source Code TD and Architecture TD, concerning the TDM activities of Identification/Monitoring/Prioritization/ Communication, these studies are also quite different, so there was an opportunity to combine/compare several successful methods while also further validating them. The method for measuring interest (hours spent per modified LoC in a commit) is also potentially less complex and more accurate than interest indicators used in previous studies such as [3][5][6]. Certain methods encountered in previous research where only partially used ([6][12][13]) or not used at all ([5][11]) in this study, but the tool alternatives presented as part of the results (see section 4.3.1) should assist those who wish to apply unused TD processes on C/C++ projects as well.

3. Methods

In this chapter the research methodology of this study is described. First, a visual overview of the research process is presented followed by details on the research setting and the involved stakeholders. The procedures used for constructing the measurement system, selecting tools and measuring TD and effort are then described in detail. Finally, the procedures for analyzing commit difficulty, correlating TD levels with maintenance effort and for validating results are detailed.

Fig. 2 shows an overview of the research process, which visually explains in which order each action was performed. The actions in the figure will be explained in detail throughout this chapter. Fig. 2 also shows which action(s) are prerequisites for other actions to be started. Prerequisites that involve multiple actions are represented as join nodes, showing which actions are required before the next steps in the process can be performed. A prerequisite that connects to multiple actions are represented as fork nodes. Conditional prerequisites are presented as decision nodes, using their conditions as labels. At the decision node, the process can only move in the direction where the condition is fulfilled.

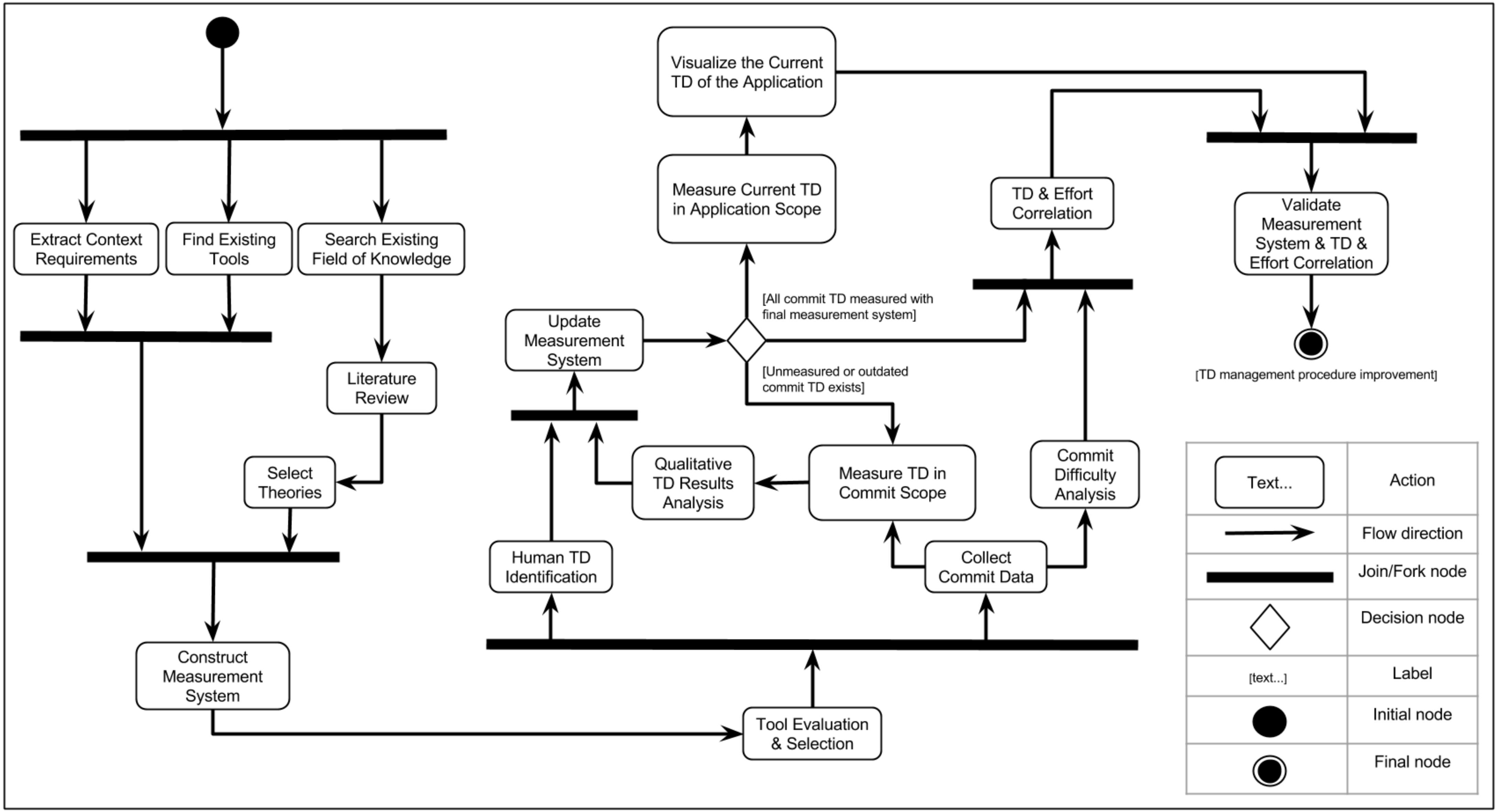
Visualize the Current TD of the Application
Measure Current TD in Application Scope
TD & Effort Correlation
Validate Measurement System & TD & Effort Correlation
Extract Context Requirements
Find Existing Tools
Search Existing Field of Knowledge
[All commit TD measured with final measurement system]
Update Measurement System
[Unmeasured or outdated commit TD exists]
[TD management procedure improvement]
Literature Review
Select Theories
Qualitative TD Results Analysis
Measure TD in Commit Scope
Commit Difficulty Analysis
Human TD Identification
Collect Commit Data
Construct Measurement System
Tool Evaluation & Selection
Text...
Action
Flow direction
Join/Fork node
Decision node
[text...]
Label
Initial node
Final node

Figure 2: Research process activity diagram

3.1 Research Setting

This study was conducted at a large telecom company in Gothenburg, Ericsson. This site provided an actively updated C/C++ project for analysis with the selected TD measurement tools through the implemented measurement system. The site also affected who was interviewed concerning human identification of TD as well as the tool result/correlation result and process validation. Although the results are limited to this specific context, such as the criteria for the tool selection and which TD types were prioritized, the design of the measurement system and the method for TD prioritization can be applied to other contexts.

3.1.1 Stakeholders

This section describes the stakeholders that have affected this study.

Product Guard

The product guard was one of the initiators of this thesis work. This stakeholder impacted this study by deciding that the scope should focus on production code, and also affected which agile development team participated in the study.

Software Expert

The software expert was the on-site supervisor of the thesis work, and a senior member of the assigned development team. Several suggestions were received through weekly status meetings as well as confirmation of the work that has been done to that current point. The first and most significant of these suggestions was to use effort in commits as an interest indicator, which allowed this study to differentiate itself from previous research. Besides the weekly meetings, the software expert was most of the time also available for further guidance. Especially his information regarding on-site procedures was used during work environment setup and additional configurations. This stakeholder was also involved in both human identification of TD and validation of the chosen TD measures, tools and the final correlation results.

Tool Expert

The tool expert was from a different team at the same site, and configured, provided instructions and directed to documentation for two proprietary tools available on-site. This included semi-frequent email correspondence. The tool expert also participated and provided feedback at many of the weekly meetings, including validation of chosen measures and on the TD measurement & visualization process.

Team Members

The members of the assigned development team. They have provided the commits with effort specified in the commit message. It is their rate of work that is compared to TD level before the work was done.

3.2 Data Collection

To guide the data collection towards answering the research questions, related literature was reviewed to gain increased domain knowledge regarding TD Management and to find applicable theories and accompanying measures for which TD types should be measured, visualized and evaluated for maintenance effort impact. The resulting data collection methods are described in this section. The section starts with describing the measurement system and which TD types are used with it. Next, it is explained how human TD identification was conducted and how it affected the measurement system. This is followed by an explanation of the evaluation criteria that were used to select tools for the measurement system implementation, and descriptions of how they were used to collect TD measurements. Finally, the process for collecting commit data is detailed.

3.2.1 Measurement System

Based on the steps of ISO standard 15939:2007 [15], guidelines from Staron & Meding [20][21] and from Staron et al. [22][23] a measurement system was defined to combine TD measures in order to answer **RQ1**. Measurement systems are used to combine measurements from multiple measures and to evaluate them based on system-specific criteria, which results in joint indicator(s) [20]. These indicators simplify the presentation of results to stakeholders, and are more efficient to manage than the separate measures they are built from. According to Staron & Meding [20], a measurement system needs to be designed around satisfying information needs connected to a specific stakeholder. This is because there needs to be a person or a group of people that is interested in the information that the measurement system provides [22]. This is necessary to ensure that the measures are suitable, and that enough of them have been identified [21]. Staron & Meding [21] also stress that measurement systems should not be based around what base measures are technically feasible, as they see it as the opposite of their top-down approach. In the case of this study however, the information need from the product guard and software expert was: “Is the existing (source code) TD causing noticeable interest?” which can be only be analyzed for the types of TD that are in fact technically feasible to measure within the case context.

To analyze TD impact based on the TD visible at source code level, the indicator: “How many/which TD types does the source code file have too much of?” was chosen. The purpose of this indicator is to find files with high TD levels and compare if they were modified in commits that were more difficult. The TD types used for this indicator are based on the following information needs identified from the initial project discussion with the product guard and software expert:

- Is the code in a certain file too complex?
- Is the code in a certain file duplicated in the same or another file?
- Does a certain file contain any high priority issues?
- Does a certain file interact with a lot of other files?
- Does a certain file violate predefined rules for interaction with other files?

An additional possible information need was discovered when reviewing tool capabilities, namely:

- Is a certain file estimated to contain defects?

Out of the stakeholders with information needs, the software expert has had the largest impact on the resulting measurement system due to participation in weekly progress meetings. The indicator was thus designed to assist the team members in knowing which files could cause their work to take longer than expected (see Fig. 3). This points out which files are prime candidates for refactoring tasks.

The measures that make up the indicator were chosen based on recommendations from previous literature. Clarity to team members was also prioritized, based on the concepts of each measure and on how their measurement results can be improved. The indicator was also supposed to be updated to prioritize the measures that were proven to be important through correlation with commit difficulty. This would have led to certain measures being excluded. As no such correlations could be proven however (see Table 2 in section 4.5), the indicator was not updated. Possible changes based on qualitative validation from the software expert are discussed in section 5.1.3.

3.2.1.1 Measurement System Design

The following list provided by Staron & Meding [20] explains the elements in a measurement system. Fig. 3 displays how these elements have been mapped in this study's measurement system.

Measurement System Element Definitions from Staron & Meding [20]

- **Interpretation**: How the indicator addresses the information need.
- **Indicator**: The indicator that is used to address the information need.
- **Analysis model**: Thresholds, targets, or patterns that are used to determine the need for action or further investigation, or to describe the level of confidence in a given result.
- **Derived measure**: A measure that is defined as a function of two or more values of base measures.
- **Measurement function**: Algorithm or calculation performed to combine two or more base measures.
- **Base measure**: A measure defined in terms of an attribute and the method for quantifying it. In this study values that are "pre-derived" by tools are treated as base measures, as other measures are in turn derived from them. This allows the graphical model to be shown at a higher level of abstraction.
- **Measurement method:** Logical sequence or operations, described generically, used in quantifying an attribute with respect to a specified scale.
- **Attribute**: Property or characteristics of an entity that can be distinguished quantitatively or qualitatively by human or automated means.
- **Entity**: Object that is to be characterized by measuring its attributes.

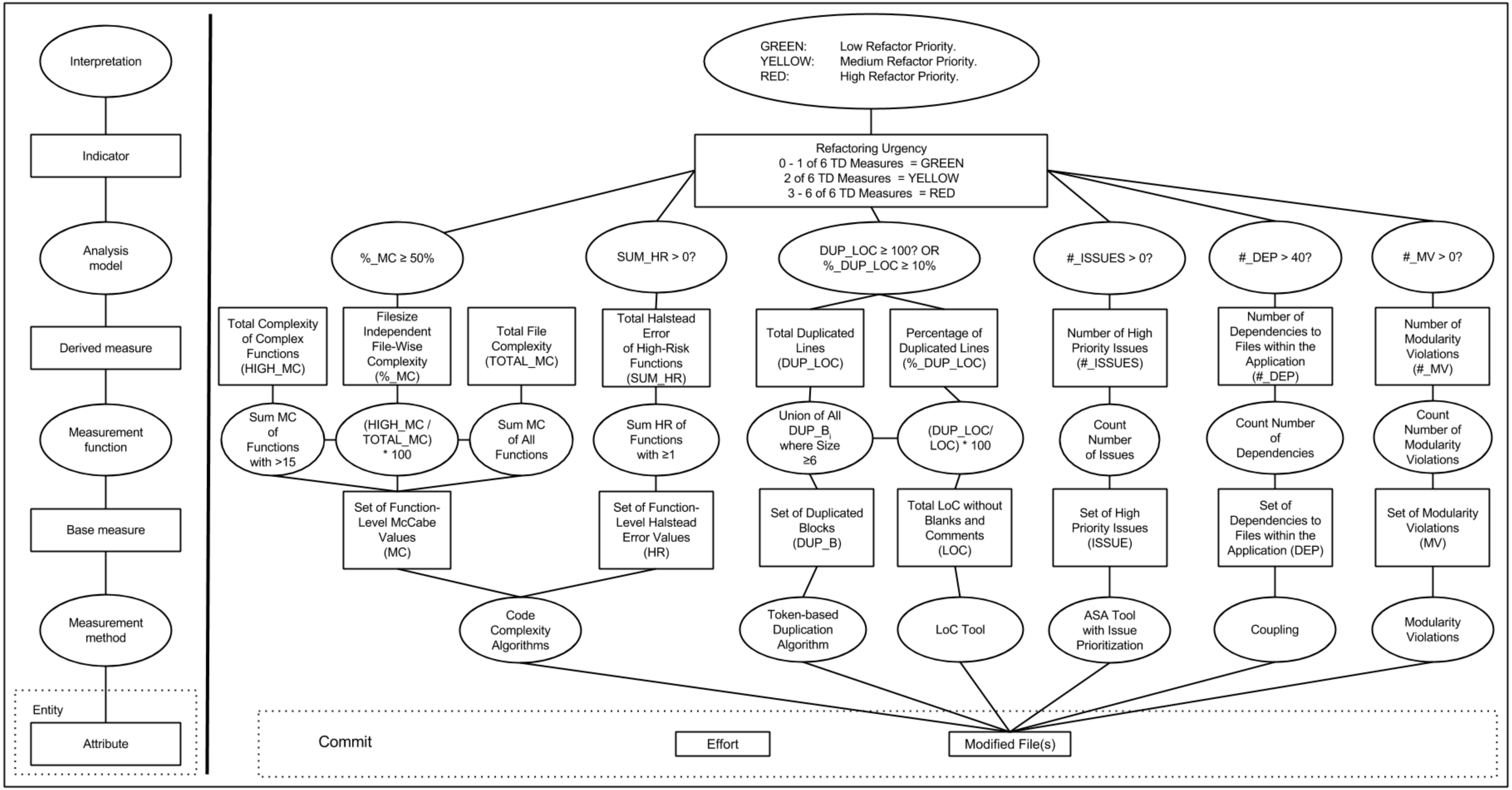
Interpretation
Indicator
Analysis model
Derived measure
Measurement function
Base measure
Measurement method
Entity
Attribute
GREEN: Low Refactor Priority.
YELLOW: Medium Refactor Priority.
RED: High Refactor Priority.
Refactoring Urgency
0 - 1 of 6 TD Measures = GREEN
2 of 6 TD Measures = YELLOW
3 - 6 of 6 TD Measures = RED
%_MC ≥ 50%
SUM_HR > 0?
DUP_LOC ≥ 100? OR %_DUP_LOC ≥ 10%
#_ISSUES > 0?
#_DEP > 40?
#_MV > 0?
Total Complexity of Complex Functions (HIGH_MC)
Filesize Independent File-Wise Complexity (%_MC)
Total File Complexity (TOTAL_MC)
Total Halstead Error of High-Risk Functions (SUM_HR)
Total Duplicated Lines (DUP_LOC)
Percentage of Duplicated Lines (%_DUP_LOC)
Number of High Priority Issues (#_ISSUES)
Number of Dependencies to Files within the Application (#_DEP)
Number of Modularity Violations (#_MV)
Sum MC of Functions with >15
(HIGH_MC / TOTAL_MC) * 100
Sum MC of All Functions
Sum HR of Functions with ≥1
Union of All DUP_B$_i$ where Size ≥6
(DUP_LOC/ LOC) * 100
Count Number of Issues
Count Number of Dependencies
Count Number of Modularity Violations
Set of Function-Level McCabe Values (MC)
Set of Function-Level Halstead Error Values (HR)
Set of Duplicated Blocks (DUP_B)
Total LoC without Blanks and Comments (LOC)
Set of High Priority Issues (ISSUE)
Set of Dependencies to Files within the Application (DEP)
Set of Modularity Violations (MV)
Code Complexity Algorithms
Token-based Duplication Algorithm
LoC Tool
ASA Tool with Issue Prioritization
Coupling
Modularity Violations
Commit
Effort
Modified File(s)

Figure 3: Measurement System design based on ISO standard 15939:2007

3.2.1.2 TD Measure Definitions & Calculations

The TD measures, measurement functions and analysis models seen in Fig. 3 are described in detail in this section (including the relevant equations). The TD types themselves are explained in section 2.1.1.

Type: Cyclomatic Complexity

- **Base measure:** The score of each function *i* in a file (MC_i).
- **Derived Measure:** The sum of the file's function scores (TOTAL_MC), the score sum of the file's methods with complexity >15 (HIGH_MC), and %_MC which stands for the percentage of TOTAL_MC that is made up of HIGH_MC, i.e. the ratio of the file's complexity that comes from too complex methods. This measure was originally used by Antinyan et al. [6], who claim that this measure was accepted by stakeholders at both Volvo and Ericsson in said study. The threshold of 15 complexity for a file to be complex is used from previous studies [5][6], and it is also the default value in the tested McCabe tools.
- **Analysis Model:** Is %_MC ≥ 50? This threshold is based on a file-wise McCabe threshold specified by Heitlager et al. [13]. 15 Cyclomatic Complexity falls within their definition of moderate complexity rather than high, hence the large percentage. While %_MC is slightly different from how Heitlager et al. [13] calculate file-wise McCabe, the tests performed to compare the two file-wise indicators resulted in highly similar percentages.
- **Equations:**

$$\text{HIGH\_MC} = \sum_{i=1}^{n} MC_i \text{ for all } MC_i > 15 \text{ in a file} \quad \text{(Eq. 1)}$$

$$\text{\%\_MC} = (\text{HIGH\_MC}/\text{TOTAL\_MC})*100 \quad \text{(Eq. 2)}$$

Type: Halstead Error

- **Base measure:** The predicted error for each function *i* in a file (HR_i).
- **Derived Measure:** The sum of predicted errors for the file using only functions with HR ≥ 1 (SUM_HR). This threshold was chosen based on the goal of only showing intuitive measures to team members [13]. As such, a method is only counted as too complex if it is predicted to contain at least one error. There are two ways to calculate Halstead Error (resulting in different values), one based on Halstead Effort and one based on Halstead Volume. Coleman et al. [11] consider Halstead Volume to be a more accurate indicator of maintainability than Halstead effort. The documentation for the Halstead effort based tool also mentions that their Halstead Error value is often lower than the true amount of errors. Due to those arguments the Halstead Error calculation based on Halstead volume was chosen. Details on how to measure Halstead measurements in C/C++ code can be found in the documentation of the tool CMT++ [19], and to calculate Errors(delivered bugs) using Volume $\text{Effort}^{2/3}$ is replaced with Volume in the delivered bugs equation.
- **Analysis Model:** Is there at least one function in the file with HR ≥ 1 (SUM_HR ≥ 1)? Assuming zero-tolerance for predicted errors.
- **Equations:**

$$\text{SUM\_HR} = \sum_{i=1}^{n} HR_i \text{ for all } HR_i \geq 1 \text{ in a file} \quad \text{(Eq. 3)}$$

Type: Code Duplication

- **Base measure:** Set of duplicated blocks (DUP_B), where the line number it starts on and size is known for (DUP_B_i). The Code Duplication tool used in this study is token-based, which means it can also detect duplication within a line of source code rather than just if the entire line is duplicated (as a side-effect it also ignores indentation). Severity is specified by how many characters are allowed to be identical in sequence on each line. The threshold used for severity is the same as the threshold used in the tool's official documentation (100).

- **Derived Measure:** The number of lines containing Code Duplication in the file after eliminating block overlap and blocks with DUP_B < 6 (DUP_LOC) and the ratio between DUP_LOC and the file's total LoC (%_DUP_LOC). The restriction to blocks that contain six lines or more is based on a recommendation for avoiding false-positives and flooding of trivial data from Heitlager et al. [13].
- **Analysis Model:** Is DUP_LOC ≥ 100? OR Is %_DUP_LOC ≥ 10%? The percentage threshold taken from Heitlager et al.'s work [13], while the LoC based threshold is used to find large duplication blocks in exceptionally large files (%_DUP_LOC does not account well for file-size).
- **Equations:**

$$\text{DUP\_LOC} = \sum_{i=1}^{n-1} \text{DUP\_B}_i \cup \text{DUP\_B}_{i+1} \qquad \text{(Eq. 4)}$$

$$\text{\%\_DUP\_LOC} = (\text{DUP\_LOC} / \text{TOTAL\_LOC})*100 \qquad \text{(Eq. 5)}$$

Type: Static Analysis Issues

- **Base measure:** Each individual High Priority Issue in a file (ISSUE_i) (filtering out types that have been labeled as false-positive by team members).
- **Derived Measure:** Total High Priority Issues count in a file (#_ISSUES)
- **Analysis Model:** Is #_ISSUES > 0? (assuming zero-tolerance policy on issues).
- **Equations:**

$$\text{\#\_ISSUES} = \sum_{i=1}^{n} \text{ISSUE}_i \qquad \text{(Eq. 6)}$$

Type: Dependency Count

- **Base measure:** Each individual dependency to files within the application that a file has (DEP_i).
- **Derived Measure:** Total dependency count to files within the application that a file has (#_DEP).
- **Analysis Model:** Is #_DEP ≥ 40? (This is the threshold used for ATFD by Rapu et al. [10]).
- **Equations:**

$$\text{\#\_DEP} = \sum_{i=1}^{n} \text{DEP}_i \qquad \text{(Eq. 7)}$$

Type: Non-Allowed Dependencies

- **Base measure:** Each individual Modularity Violation a file contains (MV_i) (based only on components visible in the available diagram).
- **Derived Measure:** Total Modularity Violations count a file contains (#_MV).
- **Analysis Model:** Is #_MV > 0? (assuming zero-tolerance on violations).
- **Equations:**

$$\text{\#\_MV} = \sum_{i=1}^{n} \text{MV}_i \qquad \text{(Eq. 8)}$$

3.2.2 Human TD Identification

As part of answering **RQ1**, TD identification from team members was performed with the intention of helping to discover any additional factors affecting maintenance effort not found through the measurement system. This was done through a semi-structured group interview with three of the team members including the software expert. The fact that this TD identification was carried out as a group interview was due to how team members expressed that filling in TD templates/questionnaires as outlined in by Zazworka et al. [7] was less optimal for them. They also wanted to reach a joint answer to the questions through discussion, rather than producing separate answers. This was motivated by claiming that the quality of the answers would be improved. This group interview was recorded and transcribed per recommendation of Runeson & Höst [4], so that the results could be further analyzed. The questions used can be seen in Appendix B. Said questionnaire also contains definitions based on material by Li et al. [9], which were used to introduce team members to TD concepts/types in a similar fashion as by Zazworka et al. [7] in order to align them with TD terminology.

3.2.3 Tool Evaluation & Selection

In order to answer **RQ2** and to provide the TD measurements necessary for **RQ1** and **RQ3**, a tool search was conducted to find and evaluate software analysis tools that might provide the measures outlined in designed measurement system (see Fig. 3). Several searching strategies have been used to find the tools that were evaluated. As mentioned previously, tools used in the reviewed previous literature were either incompatible with C/C++ or otherwise unavailable, and thus have not been evaluated. Instead, the current set of tools was discovered through searches based on these key sentences:

- Measuring *"TD type"* in C/C++
- C/C++ alternative to *"tool from literature"*
- C/C++ technical debt tool
- C/C++ static analysis tool
- C/C++ source code analysis tool

These tools were evaluated based not only on their ability to provide the necessary measures/the data that those measures can be derived from, but also on context-specific requirements regarding licensing, build integration and automation. Specifically, proprietary tools were considered unsuitable unless they were already licensed on-site. The requirements with regards to integratability were that tools should not require manual manipulation of a GUI in order to take measurements. It was considered acceptable however to view the final results in a GUI similar to the "radiators" already used on-site. These "radiators" refer to colour-coded status pages, which has been used for the visualization of the measurement system as well (see Table 4). The output formats of the tools were also examined to assess if the results could be accessed outside the tools themselves. The higher priority tools based on these criteria were tested on-site on the production code, in order to arrive at a selection of tools covering all TD types. The evaluation results have been mapped to a table (see Table 1), and motivations have been provided for the tool selections that became part of the measurement system implementation. The TD types where ad-hoc parsers are required in order to avoid proprietary tools are also highlighted, and rough outlines on the required functionality of such code has been determined.

3.2.4 Collection of TD Measures

Measurements for the measures outlined in Fig. 3 were gathered by using the selected tools combined with ad-hoc derived measure parsers and indicator parser. All source code written during the study follows the throwaway prototype agile practice [24], as the time available for development was limited. As the accuracy of the derived measures is essential, the speed of the calculation was compromised instead. These measurements were taken both within the scope of each commit (to provide the TD measurements that are compared to commit difficulty as part of **RQ3**), and within the full application scope (to provide the measurements that are aggregated and visualized and as part of **RQ1** and the final validation of results).

3.2.4.1 Measure TD in Commit Scope

The commit scope measurements were performed on the parent versions of files modified in commits with effort specified. This was done to evaluate the status of file(s) before the commit was made, to analyze how much TD the team members may have had to deal with. Rather than analyzing all files in a commit, only enough files to cover the sources of at least 90% of the modified source code is analyzed. This exclusion of files was due to commits often containing files with only one to three modified LoC, where said modification consists of "informing" that something new was added in another part of the code. It was decided that it was unreasonable to expect that TD levels in files with such minor changes would have had any noticeable impact. When discussing this with one team member, they indicated that they generally saw modified LoC as equal value within a specific commit (outside of the described "one to three lines"-case).

Two special cases of change can also be encountered in commits; file creation and file deletion. When creation and deletion is the result of a filename change, the lines modified through deletion and the lines modified through addition should remain uncounted. For file deletion it was decided to not count the associated LoC, as deleting a file rarely requires much work. For file creation however, the associated LoC are counted as part of the total modified LoC, but the TD levels of the new file are not evaluated. In essence, TD in related files may have affected the difficulty of creating the new file, but TD incurred while creating the file should not have affected the commit's ratio negatively.

3.2.4.2 Measure TD in Application Scope

The application scope measurements were taken in order to discuss the current TD state of the system with the software expert, while also providing the opportunity to validate the usefulness of the measures (and thus how well they answer **RQ2**) using strategically selected examples of large measurements found in the production code. The presentation of the measures also provided validation on how well the goals of aggregation and understandability from **RQ1** are achieved. The measurement examples were chosen based on the following criteria:

- **%_MC:** A function with high Cyclomatic Complexity but low Halstead Error was chosen to evaluate Cyclomatic Complexity in isolation.
- **SUM_HR:** A function with high Halstead Error but low Cyclomatic Complexity was chosen to evaluate Halstead Error in isolation.
- **DUP_LOC:** A large block of duplicated lines that occurred twice in the same file was chosen, in order to extract information on why such duplication exists.
- **#_ISSUES:** An example for each discovered issue category was chosen, in order to evaluate if there were still false-positive categories left to filter out.
- **#_DEP:** The smallest file with over-threshold #_DEP was chosen, in order to extract information on why it needs to have so many dependencies.
- **#_MV:** A file with many Non-Allowed Dependencies in the opposite direction of what the architecture diagram shows was chosen in order to illustrate that cyclic dependencies exist on the component level of the application.

3.2.5 Collection of Commit Data

In order to measure the difficulty of the commits, the effort specified in the commit message and the modified LoC was collected. These measurements were used as part of answering **RQ3**.

3.2.5.1 Effort Measurement

In this study, effort was retrieved in the form of hours spent on commits. This effort value was manually specified in each commit's accompanying message by the team members involved in this study. These commits with effort specified are this study's unit of analysis. The team members decided together with the software expert that such effort should only be specified in production code commits rather than testing code commits. This was due to there being more interest in improving the system rather than the test code applied to it. Another consideration was that looking at both types of commits together would cause too much variation in the data. The downside of this restriction was its effect on the amount of commits available to analyze during the course of this study, as the testing code commits were about as frequent as production code commits.

3.2.5.2 Modified LoC Measurement

An amount of modified LoC can also be extracted from these commits, and it is through calculating the ratio of modified LoC per hour for each commit that they can be compared. The resulting ratio becomes a difficulty grade for the commit, where a lower ratio indicates that the modified LoC in the commit were more expensive to produce. This extra effort may be a manifestation of TD interest, and is treated as such in comparisons with TD levels in order to answer **RQ3**. Likewise, high ratio commits could indicate lack of TD. In this study pure modified LoC was used, which means blanks, whitespace and comments are ignored when counting. This was chosen rather than the modified LoC value provided by Git, as it was discovered that even indentation changes were counted by Git as a modified LoC. This change led to much more accurate modified LoC values and calculated ratios.

3.3 Data Analysis

This section describes how the collected data was analyzed. The first subsection explains how two separate approaches, MAD-Median-rule and 20% Trimmed Mean, were used to determine a representative mean for the set of modified LoC per hour ratios. The next subsection details how theory triangulation was applied through three correlation methods, Pearson's Correlation & Conditional Probability & Cohen's Kappa, in order to reduce the risk of correlations being found due to chance. Two of the methods used also required that values be transformed to "1" and "0" based on a chosen condition. These conditions are explained in detail following the method equations. The final subsection describes the two forms of validation that were conducted during this study in order to evaluate the quality of analysis results with regards to the research questions and stakeholder expectations.

3.3.1 Commit Difficulty Analysis

The set of modified LoC and the set of hours spent values gathered from the available commits were examined for two purposes. The first purpose was to investigate the commits that appeared to have a large amount of extra effort per LoC (based on their modified LoC per hour ratio) by contacting the commit author and attempting to elicitate a reason behind the difficulty. Such reasons may be relatable to established TD types. None of these commits had a statistically large amount of extra effort however, as no such commits exist in this study's sample population of commits with effort.

The second purpose was to calculate the correlation strength between sets of modified LoC values and sets of effort values. Two such sets have been used; one using the MAD-Median-rule [18], and one based on trimming the dataset by 20% at both ends. Both of these methods deal with finding the true mean, and are among the methods recommended by Wilcox [18]. The MAD-Median-rule was specifically chosen over other alternatives that were recommended in the same book [18], as it identified a larger number of commits as unusually easy. This was considered preferable as it allowed for a larger contrast in correlations. The 20% Trimmed method was chosen due to it being described as a good compromise between the mean and the median (which is the same as 50% trimming) [18]. For the MAD-Median-rule based dataset, the commits with the four largest ratios were identified as outliers and removed, while the commits with the four largest and four smallest ratios were removed in the 20% Trimmed dataset. The equations used are detailed below:

The Mad-Median-rule:

$$\frac{|\,X - M\,|}{MADN} > 2.24 \qquad \text{(Eq. 9)}$$

$$MADN = \frac{MAD}{0.6745} \qquad \text{(Eq. 10)}$$

where

MAD, is the Median Absolute Deviation
MADN, is the Median Absolute Deviation Normalized

Median Absolute Deviation:

$$|\,X_1 - M\,|, \ldots, |\,X_n - M\,| \qquad \text{(Eq. 11)}$$

where

$X_1, \ldots, X_n$, is the sample value
M, is the sample median

20% Trimmed:

$$\overline{X}_t = \frac{1}{n - 2g}\left(X_{(g+1)} + X_{(g+2)} + \ldots + X_{(n-g)}\right) \qquad \text{(Eq. 12)}$$

where

n, is the sample size
g, is the amount of trimmed values

The correlation algorithms that are applied between modified LoC and maintenance effort are detailed in section 3.3.2. If the two datasets can be correlated, the average modified LoC per hour ratio could possibly be used to increase the amount of available commits for effort correlation, which would help increase the small sample size (which is the main weakness of the “effort in commit message”-method). More specifically, the average ratio could be used to investigate recent commits from team members outside of the current team, by contacting them and asking if more (or less) time was spent than what the ratio ± two Standard Deviations (StDev) expected. These commits need to be recent, due to the difficulty of discussing time spent on commits older than one to two weeks. A normally distributed commit would have an effort value between modified LoC divided by ratio – two StDev and modified LoC divided by ratio + two StDev. If the commit is instead a negative outlier it could also be qualitatively investigated for reason(s) behind its difficulty. During this study however, the correlation between modified LoC and effort did not become strong enough to consider contacting team members outside the team until the very end of the project.

3.3.2 TD & Effort Correlation Process

An important factor in TD Management is knowing the interest of existing TD, as it allows refactoring to be prioritized for the TD types that have a more noticeable impact. To achieve this goal (**RQ3**), effort spent on commits were specified as hours by team members, and Zazworka et al.’s [3] process to find significant correlations has been applied between the commits’ TD occurrences and the commits’ modified LoC per hour ratios. This process combines Pearson’s Correlation, Conditional Probability, Chance Agreement and Cohen’s Kappa to compare if the relation is detected by several different algorithms. A strong correlation between a TD type and the maintenance effort indicates that the TD type’s interest is large. The choice to use Zazworka et al.’s [3] combined process was due to the risk of biased correlation results if for example only Pearson’s is used. By having more than one viewpoint on the correlation, theory triangulation [4] can be achieved, thus improving the certainty that the relation in fact exists.

Through the equations in this section, three indicators for correlation strength are calculated, and any relation where at least two out of three indicators result in a “1” in their corresponding Ω(...) function is considered as high correlation strength. The dataset values need to be transformed into “1” and “0” (representing true and false for dataset-specific conditions) for all equations in this section except Pearson’s Correlation. These transformations are explained in section 3.3.2.4.

3.3.2.1 Pearson’s Correlation

The Pearson’s Correlation calculation is used to check if two datasets increase together or if one dataset increases while the other decreases in a linear pattern. This type of calculation is widely used in defect prediction models as well as for maintainability predictions [3]. The calculation itself can be represented by the Pearson’s Correlation Coefficient, r. The equation, Eq. 13, requires two datasets $\{ x_1, ..., x_n \}$ and $\{ y_1, ..., y_n \}$ where n is the total amount of data points. The result of Eq. 14 is the dataset mean value.

$$r = r_{x,y} = \frac{\sum_{i=1}^{n} (x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\sum_{i=1}^{n} (x_i - \overline{x})^2} \sqrt{\sum_{i=1}^{n} (y_i - \overline{y})^2}} \qquad \text{(Eq. 13)}$$

where

$$\overline{x} = \frac{1}{n} \sum_{i=1}^{n} x_i \qquad \text{and} \qquad \overline{y} = \frac{1}{n} \sum_{i=1}^{n} y_i \qquad \text{(Eq. 14)}$$

The r-value will be between -1 and 1, where an r-value closer to 1 reflects a positive linear correlation and an r-value closer to -1 reflects a negative linear correlation. To be able to use the Pearson r-value as a significance indicator, the significant Pearson functions Ω(PearsonPos) or Ω(PearsonNeg) are used to determine if the correlation is strong enough. The threshold r-value depends on if it is modified LoC & effort or TD occurrences & commit difficulty that are being compared. When comparing modified LoC with effort a threshold of 0.6 is used, just as by Zazworka et al. [3], as these variables are expected to have a positive correlation. When comparing TD occurrences with commit difficulty however, the r-value threshold is -0.6, as a negative correlation is expected. This is due to commit difficulty being represented by the commit’s modified LoC per hour ratio, which means that a **decreasing** difficulty is represented by an **increasing** ratio. As it is expected that less TD occurrences will cause the ratio to increase, a negative threshold is thus chosen. The r-values of -0.6 and 0.6 have the same significance; it is only the type of correlation they describe that differs.

Following Zazworka et al.’s [3] technique, the significance level for the Pearson’s result must also be at least 0.05 (same as 95%, can be checked in the following table in [25]). This is used to make it more difficult to achieve significance at low sample sizes, as a guard against bias. The significance functions (Eq. 15 & Eq. 16) will result in either “0” or “1”, where “1” equates to strong correlation and “0” interprets to no correlation.

$$\Omega(PearsonPos) = \begin{cases} 1, if\ r_{x,y} \geq 0.6 \land 95\%\ significant \land (x \neq y) \\ 0, otherwise \end{cases} \qquad \text{(Eq. 15)}$$

$$\Omega(PearsonNeg) = \begin{cases} 1, if\ r_{x,y} \leq -0.6 \land 95\%\ significant \land (x \neq y) \\ 0, otherwise \end{cases} \qquad \text{(Eq. 16)}$$

3.3.2.2 Conditional Probability and Chance Agreement

Conditional Probability is used in pairwise comparisons to understand the probability of event A occurring given that event B has already occurred, and vice versa [18][3]. Conditional Probability can be used for software defects and maintainability predictions [3], but first it requires that all values are transformed into “1” and “0” (representing true & false for a specific condition per dataset). The equation for the Conditional Probability, P(A | B) or P(B | A), is calculated in this manner:

$$P(\,A\,|\,B\,) \;=\; \frac{P(\,A \cap B\,)}{P(\,B\,)} \cdot 100 \qquad \text{(Eq. 17)}$$

$$P(\,B\,|\,A\,) \;=\; \frac{P(\,A \cap B\,)}{P(\,A\,)} \cdot 100 \qquad \text{(Eq. 18)}$$

where

P(A), is the probability that event A occurs.
P(B), is the probability that event B occurs.
P(A ⋂ B), is when the event A and B occur simultaneously.

The result of the equations Eq. 17 & Eq. 18 is a probability of 0 to 100%. To be able to use this calculation as a significance indicator the Chance Agreement needs to be calculated as well. The Chance Agreement is used to understand the probability of event A and event B occurring simultaneously by chance.

$$P(\,Chance\,) \;=\; (\,P(\,A\,)\,\cdot\,P(\,B\,)\,+\,(\,1\,-\,P(\,A\,)\,)\,\cdot\,(\,1\,-\,P(\,B\,)\,)\,)\,\cdot\,100 \qquad \text{(Eq. 19)}$$

The Chance Agreement probability will also range between 0 to 100%. The significant Conditional Probability functions Ω(ConditionalAB) or Ω(ConditionalBA) are used to determine if the correlation strength is high enough to be counted as an indicator. The threshold for the Conditional Probability value is 60% [3], as seen in Eq. 20 & Eq. 21.

$$\Omega(ConditionalAB) = \begin{cases} 1, if\ P(\,A\,|\,B\,) > P(\,Chance\,) \wedge P(\,A\,|\,B\,)\ \geq\ 60\%\ \wedge\ (\,A\ \neq\ B\,) \\ 0, otherwise \end{cases} \qquad \text{(Eq. 20)}$$

$$\Omega(ConditionalBA) = \begin{cases} 1, if\ P(\,B\,|\,A\,) > P(\,Chance\,) \wedge P(\,B\,|\,A\,)\ \geq\ 60\%\ \wedge\ (\,A\ \neq\ B\,) \\ 0, otherwise \end{cases} \qquad \text{(Eq. 21)}$$

3.3.2.3 Cohen's Kappa

Cohen's Kappa is used to determine the strength of the agreement and disagreement of two datasets transformed into "1" and "0" values [3]. Once the datasets' values have been transformed, Cohen's Kappa can be calculated by counting where the two datasets (x, y) values are in agreement by both being "1" or "0", while in any other case they will be in disagreement.

$$Pr(\,a\,) \;=\; \frac{\sum_{i=1}^{n} a_i}{n}, where \begin{cases} a_i \;=\; 1, if\ x_i \;=\; y_i \\ a_i \;=\; 0, otherwise \end{cases} \qquad \text{(Eq. 22)}$$

where

Pr(a), is the proportionate agreement between the two datasets divided by the total number of dataset values, n.

Once Pr(a) is calculated, the Chance Agreement, P(Chance), is also needed since it is used as a variable in the Cohen's Kappa calculation.

$$K = \frac{Pr(\,a\,) - P(\,Chance\,)}{1 - P(\,Chance\,)} \qquad \text{(Eq. 23)}$$

To use the Cohen's Kappa as a correlation indicator the following significance function Ω(Kappa) is used, where the threshold value has been set to 0.60 [3].

$$\Omega(Kappa) = \begin{cases} 1, if\ K > 0.60 \wedge (\,x \neq y\,) \\ 0, otherwise \end{cases} \qquad \text{(Eq. 24)}$$

3.3.2.4 Dataset Transformation Strategies

The goal of the correlation process is to analyze the strength of relation between specific over-threshold TD types/all over-threshold TD combined and commit difficulty, as well as between modified LoC and effort. As mentioned previously, to apply all methods besides Pearson's Correlation, this requires the values of the datasets to be transformed into sets of "1" and "0" [3].

Transformations of Specific Over-Threshold TD and Commit Difficulty

For analyzing the relation between specific over-threshold TD types and commit difficulty, comparisons are made between if over-threshold levels of a TD type was found in at least one file in a commit (over all commits with effort) to two separate transformations of the modified LoC per hour ratios. For the over-threshold TD transformations, a "1" signifies that at least one TD occurrence high enough to cross the threshold was found in the commit; while "0" represents that no such occurrences were found. One of the modified LoC per hour ratio transformations that this is compared against is if the ratio was pointed out as normal or as a positive outlier (unusually high ratio) by performing the MAD-Median-rule based equation. For the MAD-Median-rule based modified LoC per hour ratio transformation, a "1" signifies that the ratio was within a normal distribution, while a "0" signifies that it was a positive outlier. The choice to focus on positive rather than negative outliers (unusually low ratio) was due to how the dataset only contained outliers in that direction. Relational strength in this comparison is based on if the TD occurs when the ratio is normal and doesn't occur when the ratio is a positive outlier.

The other modified LoC per hour ratio transformation was based on if the commit was one Standard Deviation below the 20% Trimmed mean [18]. For the 20% Trimmed based modified LoC per hour transformation, a "1" indicates that the commit was more difficult than usual compared to the 20% Trimmed mean. Based on recommendations by Wilcox [18], this conversion should have been based on two Standard Deviations, but the sample in this study contained no data points that were that far below the 20% Trimmed mean, so one Standard Deviation was used instead. This means that the Conditional Probability/Chance Agreement/Cohen's Kappa values for all comparisons related to 20% Trimmed are too lenient (The MAD-Median-rule based calculations were conducted to replace them). As such, only the results related to the MAD-Median-rule based method are shown in sections 4.4-4.5, while the results related to 20% Trimmed are shown in Appendix A. The Conditional Probability, Chance Agreement and Cohen's Kappa results are used together with the Pearson Correlation results between each specific TD type occurrence datasets to each of the two commit datasets, where the comparisons to the first commit set has had the specific TD occurrence values belonging to the outlier commits identified via MAD-Median-rule removed, and the comparisons to the other commit set has had the specific TD occurrence values belonging to the trimmed commits removed.

Transformations of Overall Over-Threshold TD and Commit Difficulty

For analyzing the relation between overall over-threshold TD occurrences and commit difficulty, the set of over-threshold TD occurrences per commit was transformed in two ways. One transformation assigns a commit as "1" if the total over-threshold occurrences were two Standard Deviations above the total over-threshold occurrences mean for all commits with effort. This is then compared to the MAD-Median-rule based transformation of the modified LoC per hour ratios. The other transformation is based on total over-threshold TD occurrences being one Standard Deviation above the total over-threshold occurrences mean and is compared to the 20% Trimmed based transformation of the modified LoC per hour ratios. These results are used together with the Pearson's Correlation results of the two datasets, where the comparisons to the first commit set has had the total over-threshold TD occurrence values belonging to the outlier commits identified via MAD-Median-rule removed, and the comparisons to the other commit set has had the total over-threshold TD occurrence values belonging to the trimmed commits removed.

Transformations of Modified LoC and Effort

For analyzing the relation between modified LoC and effort, the sets of modified LoC and effort were both transformed in two ways. One comparison converts values to "1" if they are two Standard Deviations above their respective sample means, while the other transformation is based on one Standard Deviation. These results are used together with the Pearson's Correlation results of the two dataset pairs, where the first pair has had the modified LoC and effort values belonging to the outlier commits identified via MAD-Median-rule removed, and the other pair has the modified LoC and effort values belonging to the trimmed commits removed.

3.3.3 Validation

Two forms of validation have been conducted during this study; qualitative TD results analysis and validation of the measurement system as well as the maintenance effort correlations. The first validation form was done at several points during the course of this study, where the software expert was shown TD tool results in order to identify false-positives. The tool results have also been compared between similar tools and to manual calculations, in order to verify their correctness.

The second validation form was conducted through a semi-structured interview with the software expert in order to validate the measurement system, the measures it is made of and the correlation results between these measures and maintenance effort. The questions that were used during this interview can be seen in Appendix C. A summarizing transcript of this interview was also written and later confirmed to be accurate by the software expert. A short interview with a tool expert was also conducted to validate how well the measurement system implementation conformed to the on-site requirements for automation. The purpose of all this validation was to gain stakeholder feedback regarding how well the research questions were answered based on the software expert's comments on the related contributions.

4. Results

This section presents the analysis results of this study. First, the final implementation of the measurement system is shown, which details how TD tools were combined with ad-hoc parsers and manual processes in order to implement the design previously seen in Fig. 3. This is followed by an explanation of how the human TD identification through the semi-structured group interview resulted in changes to the visualization of the measurement system's indicator. The final version of the measurement system implementation is used to answer **RQ1.**

The measurement system implementation is followed by showing the results of mapping C/C++ compatible TD measurement tools capable of measuring the intended measures. The motivations for why certain tools were ultimately selected for the measurement system implementation are also detailed. These results provide an answer to **RQ2.**

After going through the two first contributions, the results of commit difficulty analysis and the correlations to TD types that were made possible through it are presented, showing how **RQ3** was answered.

Finally, the results of the validation interview with the software expert are detailed, which shows how well the research questions were answered from this stakeholder's point of view. This includes validation of the measurement system and the way its results are presented (**RQ1**), validation of the measures and the examples of them found in production code using the selected tools (**RQ2**) and validation of the correlation results including the process to evaluate correlational strength (**RQ3**).

4.1 Measurement System Implementation

In this section a visual outline of the measurement system implementation is provided (see Fig. 4). The process contains the selected tools and ad-hoc parsers that are further detailed in sections 4.3.2.1-4.3.2.6. The overview details how the base measures are extracted, how the derived measures are calculated and how the joint indicator is constructed and visualized. The output formats of file-creation actions are also specified. While this implementation uses two proprietary tools (Coverity SA and Coverity AA), the measures they extract could potentially also be covered with a combination of open source alternatives and ad-hoc parsers.

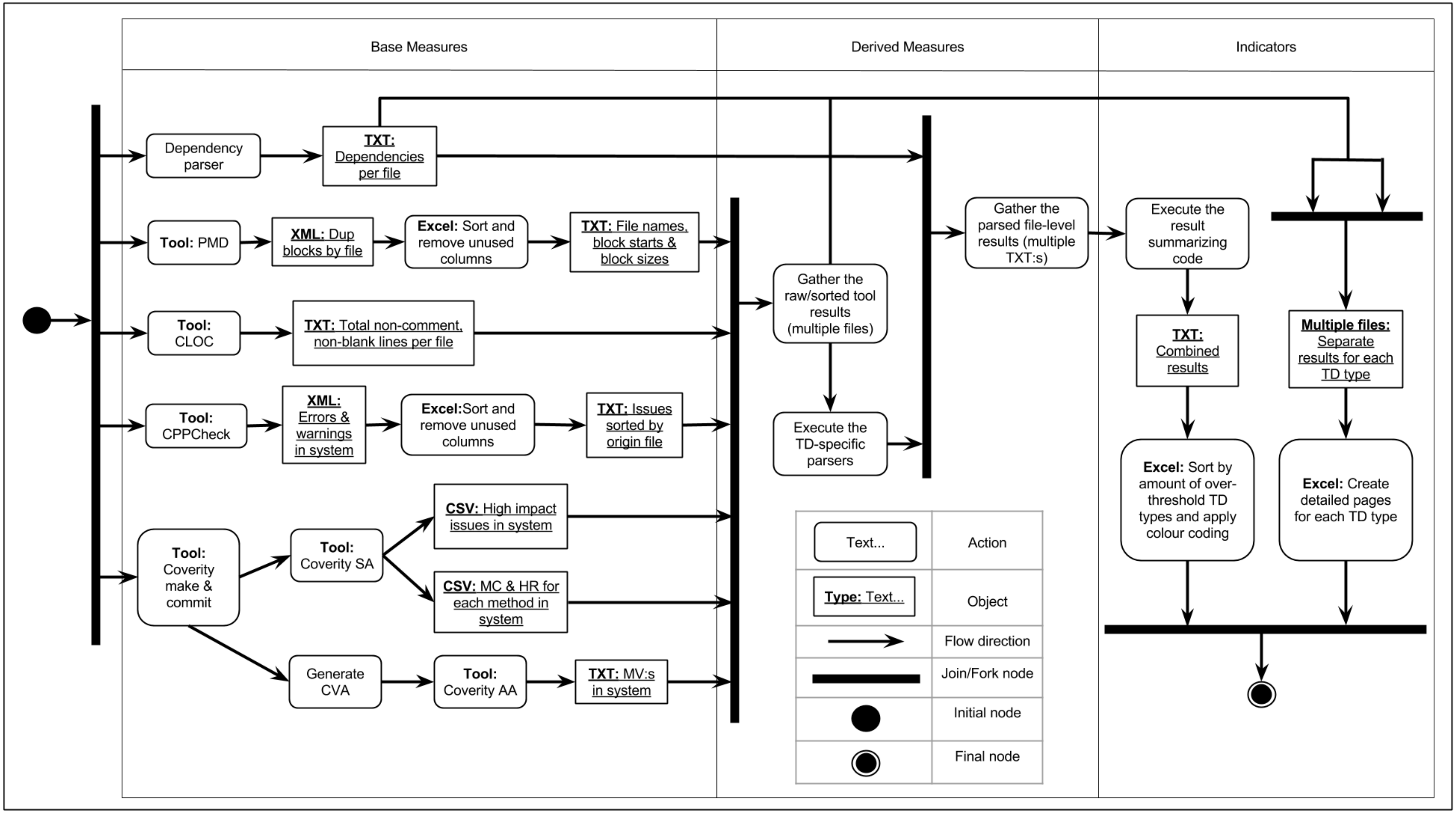
Base Measures
Derived Measures
Indicators
Dependency parser
TXT: Dependencies per file
Tool: PMD
XML: Dup blocks by file
Excel: Sort and remove unused columns
TXT: File names, block starts & block sizes
Tool: CLOC
TXT: Total non-comment, non-blank lines per file
Tool: CPPCheck
XML: Errors & warnings in system
Excel:Sort and remove unused columns
TXT: Issues sorted by origin file
Tool: Coverity make & commit
Tool: Coverity SA
CSV: High impact issues in system
CSV: MC & HR for each method in system
Generate CVA
Tool: Coverity AA
TXT: MV:s in system
Gather the raw/sorted tool results (multiple files)
Execute the TD-specific parsers
Gather the parsed file-level results (multiple TXT:s)
Execute the result summarizing code
TXT: Combined results
Excel: Sort by amount of over-threshold TD types and apply colour coding
Multiple files: Separate results for each TD type
Excel: Create detailed pages for each TD type
Text...
Action
Type: Text...
Object
Flow direction
Join/Fork node
Initial node
Final node

Figure 4: Measurement system implementation

4.2 Human TD Identification

A questionnaire for assisting in human identification of TD was designed (see appendix B), which was then used as part of a semi-structured group interview. One such group interview was performed, with a length of 90 minutes. This interview helped to point out which parts of the application the team commonly modified, which parts the team considered to be more complex, and which parts of the application the team believed likely to remain unchanged. It also helped indicate which parts of the application the team were most interested in improving. When comparing the parts of the source code that team members considered complicated to where files with over-threshold amounts of TD were discovered, a discrepancy was detected. This was due to the fact that these files were occasionally outside the complicated part of the application. This implies that either the team members' view of the application does not reflect its actual state, or otherwise the chosen TD measures are not what are causing significant maintenance effort. The former would prove the need for the measurement system, but as the correlation results show (see section 4.5) it may well be the latter that is the true reason.

Besides making comparisons, the results from the group interview also resulted in a filter for which files are visible in the visualization of the measurement system, as the team members stressed the fact that they were not interested in seeing TD in unit testing and stub files. This filtering prevents information overflow by hiding these uninteresting files, which contributed to answering **RQ1** as it is easier for team members to interpret results that are limited to the relevant parts of the application.

4.3 Tool Evaluation & Selection Results

As part of this study multiple source code analysis tools were reviewed to find the most suitable tools for implementing the designed measurement system (see Fig. 3) in the case context. The tools evaluated have been mapped in a table according to evaluation criteria outlined in section 3.2.3, which include integratability, functionality and output format support. Recommendations for which of these tools to use as well as which to avoid are also provided, based on testing the tools within the case context.

4.3.1 Tool Mapping

The results of the tool review (see Table 1) and the necessary details to understand the mapping are detailed below:

4.3.1.1 Table Details

Language Support (ignoring languages irrelevant for the context)
Name = Supports both C/C++
Name = Supports C++ only

Integratability Grade (context specific)
A = Commandline interface (should work "out-of-the-box")
B = Eclipse plugin OR commandline interface + some setup/libraries required
C = Eclipse plugin + some setup/libraries required
D = Anything more complicated with commandline/plugin OR standalone GUI

Integratability Clarifications

Commandline is preferable due to possibilities for automation. Eclipse plugins can be used to detect TD during development rather than at commit time (several plugins can be installed on the same eclipse version, so a variety of measures can be measured in one place). Standalone GUI tools are unsuitable due to limited measure scope or limited interaction with other tools (or plugins). They also lack the ability for automation.

Licenses
For links to the definitions of the licences listed in Table 1, see Appendix D.
CMR = commercial off-the-shelf software.

Supported Measures
COM = Code complexity related metrics.
DUP = Code Duplication related metrics.
LOC = Lines of code related metrics.
ASA = Common general/language specific issues.
MV = Modularity Violations (Non-Allowed Dependencies).
COUP = Coupling related metrics.

Output Formats
TXT = Raw text file or similar.
XML = File using xml tag system.
CSV = Comma separated values file.
HTML = Results displayable in browser, which is often the most readable format.
OUT = Output directly in eclipse/tool GUI.

4.3.1.2 Tool Table

Table 1: Tool capabilities & ratings

GENERAL			**TD TYPES**	**OUTPUT**
TOOL	**INTEGRAT ABILITY**	**LICENSE**	**COM/DUP/LOC/COUP/ ASA/MV**	**TXT/XML/CSV/HTML/ OUT**
PMD	**A**	BSD-4	DUP	TXT/CSV/XML
SonarQube	**D**	CMR	COM/DUP/LOC/COUP	OUT
OCLint	**B**	BSD-3	COM/ASA	TXT/XML/HTML
GitInspector	**B**	GPLv3	LOC	TXT/XML/HTML
CppCheck	**B**	GPLv3	ASA	XML/HTML
CCCC	**B**	GPLv2	COM/LOC/COUP	XML/HTML
Dependometer	**D**	VPL	MV/COUP	HTML
CONQAT	**C**	ALv2	DUP/MV/ASA	XML/HTML
YASCA	**A**	GPLv2	DUP/ASA	HTML
CodeAnalyzer	**D**	GPLv2	LOC	TXT/CSV/HTML/OUT
Metriculator	**C**	EPL	COM/LOC/COUP	TXT/XML/HTML/OUT
UCC	**A**	USC	COM/LOC	TXT/CSV
Lizard	**B**	MIT	COM/LOC	XML
Complexity	**B**	GPLv2	COM/LOC	TXT
Pmccabe	**B**	GPLv2	COM/LOC	TXT
CLOC	**A**	GPLv2	LOC	TXT/CSV/XML
Coverity SA	**D**	CMR	COM/LOC/ASA	CSV/XML/OUT
Coverity AA	**D**	CMR	MV/COUP	TXT/OUT
Structure101	**D**	CMR	MV/COUP	TXT/OUT
CMT++	**A**	CMR	COM/LOC	TXT/XML/HTML

4.3.2 Tool Selection Motivations

Through attempting to install and use most of the tools listed above on-site, a set of suitable tools to cover all chosen TD types were selected. Gaps that can be covered by ad-hoc parsers in order to avoid proprietary tools were also identified. Only the tools ranked A or B and the tools SonarQube, Coverity SA, Coverity AA and Structure101 have been tested. Sections 4.3.2.1-4.3.2.6 detail the final tool selections and ad-hoc parsers for each TD category. SonarQube appeared to be highly suitable for visualizing several of the TD types together, but the commercial C/C++ plugin takes several hours to complete its analysis. This made SonarQube unusable in this study as SonarQube could not support analyzing several commits per day.

4.3.2.1 Category: COM

For this category, UCC proved to be most suitable non-proprietary tool for McCabe Cyclomatic Complexity, due to its ease of use and potential for integration. A major flaw that separates it from the proprietary solutions Coverity SA and CMT++ is the lack of direct link to source code. UCC still provides function/file name and function McCabe value, but it is then required to find the method manually for further investigation. The HTML output of CCCC provides such source code links, but this tool did not work on-site (even though it worked when testing on a random C++ project retrieved from GitHub).

For other forms of code complexity from the identified tools, Halstead metrics are measurable by the proprietary tools Coverity SA and CMT++. These two tools also support calculating both versions of Halstead Error (Halstead Effort based and Halstead Volume based). In order to measure Halstead without proprietary tools, one would need to write code that counts operators and operands in source code files based on the descriptions in CMT++:s documentation [19].

After considering the many steps required to get Coverity SA complexity results and the transparency on how CMT++:s Halstead metrics are calculated [19], CMT++ stands out as the best choice for measuring the COM category of measures, as it is both highly integratable and automatable. Its biggest flaw is its lack of the “link-to-source-code”-functionality previously described. However in this case context Coverity SA was used instead, due to their existing licenses. From the tool results, %_MC and SUM_HR is programmatically derived by code written during the project.

4.3.2.2 Category: DUP

For this category, PMD was chosen due to its commandline interface and many output formats. It is only the Copy-Paste Detector (CPD) module of PMD that is used in this study. The other identified DUP tools are built on top of PMD, and YASCA:s HTML output is useful for improving result readability and visual link to source code. However PMD:s eclipse plugin is not very useful, as it does not provide any syntax highlighting of the duplicated code. The tradeoff for these alternate visualizations of PMD results is the negative effect on integratability, which is why the tool’s default commandline interface is used instead. The (XML) output of the tool is currently fed into excel in order to perform a custom sort (FIRST BY filename THEN BY block_start THEN BY block_size). The sorted results are parsed programmatically to combine overlapping blocks, before summing the remaining blocks to derive the DUP_LOC measure for each file. The parser source code was written manually during the project.

4.3.2.3 Category: LoC

For this category, CLOC was the best alternative both for measuring total LoC of a file as well as comparing differences between two versions of the same file. This is due to how it can ignore whitespace differences (such as indentation changes), while also clearly separating its results between blank lines, comment lines and code lines. This made it easy to extract the true value of modified LoC in a commit, which was often quite different from what the Git diff reported.

4.3.2.4 Category: ASA

For this category, CppCheck appears to be the most suitable non-proprietary alternative (when comparing functional similarity to the ASA Issues tool used by Zazworka et al. [3], which is for Java only). It should be configured to only look for errors and warnings, in order to avoid trivial issues. This study was able to use CppCheck together with the High Impact Issue categories from Coverity SA, thanks to the existing licence and build integration of Coverity on-site. CPPCheck does not find the same issues as Coverity SA, and also lacks the source code link functionality. The issues found by Coverity SA also contained less false-positives according to the software expert than validated them. The found issues from each tool are programmatically parsed by source code written during the project, to count #_ISSUES for each individual file after filtering out the false-positive types.

4.3.2.5 Category: Coup

The tool CCCC would have been an obvious choice for this category if it had worked on the on-site code, as it can measure some of the more specific method-level coupling measures mentioned by D’Ambros et al. [14]. File-level dependency count is used instead, more specifically the number of dependencies to other files from the same application rather than to language-specific system libraries. This can be analyzed manually through the GUI:s of Coverity AA and Structure101, or by looking directly in the source code. To fulfill the automation requirements, an ad-hoc parser is used to parse out the number of dependencies per file automatically.

4.3.2.6 Category: MV

For this category, Dependometer was not a suitable alternative due to it not supporting C code. However, both Coverity AA and Structure101 were usable for this measure. Both those tools are similar in functionality to the tool developed by Grundèn & Lexell[1], in that they require a model to verify against to detect Non-Allowed Dependencies. Coverity AA reads package/file structure from this existing source code, while Structure101 requires a Doxygen file generated from the code. From the input both tools construct a full model of the system. In these models a subsection of the chosen abstraction level can easily be selected and a new diagram generated from it. The resulting diagram can then be modified with dependency arrows, and any dependencies that do not follow a manually added path are registered as a violation. As the entire source code has been parsed, a diagram abstracted to component level still reports the files inside the components that participate in the violation (significantly reducing the amount of arrows required). The MV file extracted from Coverity AA or Structure101 is then parsed with an ad-hoc derived measure parser to calculate #_MV per file. The dependency treeviews in these tools also allow for manual analysis of how the Non-Allowed Dependencies are used, which can be used to prioritize them.

As Structure101 has the same functionality as Coverity AA but requires running Doxygen on source code with the required annotations (to map the structure), Coverity AA is the better choice. To measure Non-Allowed Dependencies without proprietary tools, a text parser program that analyzes includes based on the component the file belongs to would need to be written.

4.4 Commit Difficulty Analysis

Before performing correlations on the commit modified LoC and effort values, the commits with the lowest ratios (in this case the three most difficult commits) were qualitatively investigated to learn about potential causes. One of these “difficult” commits was pointed out to be due to a build configuration change that required changes in the source code for compatibility, which while arguably being a form of TD is also a highly specific and rare occurrence. The remaining two commits could have been affected more strongly by general TD levels in the modified files. To compare commit TD levels to commit difficulty however, it had to be determined which commits were unusually difficult (or easy).

4.4.1 Outliers via MAD-Median-rule

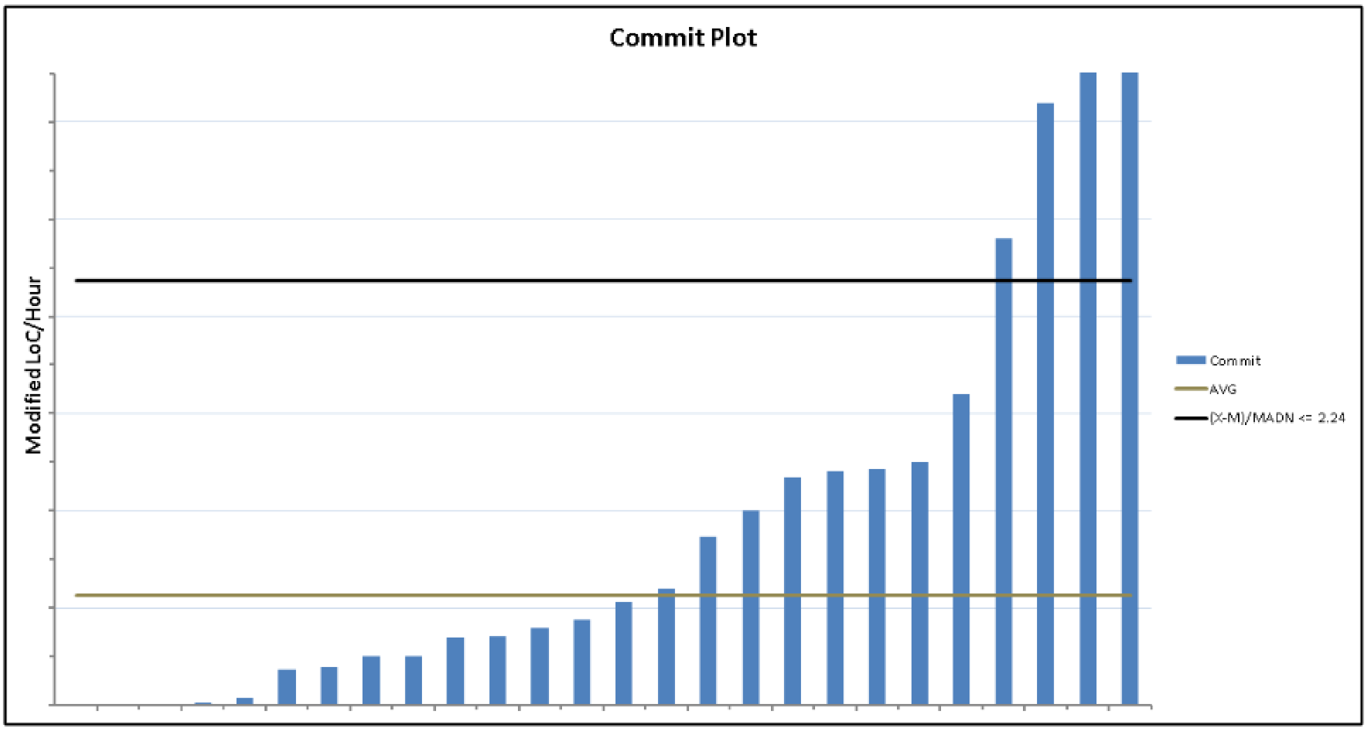

Figure 5: Modified LoC per hour ratio for each commit

The average modified LoC per hour for the set of commits with effort (after removing outliers based on applying the MAD-Median-rule [18] to the modified LoC per hour ratios) was calculated in order to produce a line representing the mean of the normal values in the sample (see Fig. 5). A line representing the MAD-Median-rule is also drawn, to show which commits were branded as positive outliers (unusually easy). Each bar represents a single commit and as can be seen in Fig. 5 four commits were identified as outliers. The diagram based on 20% Trimmed [18] (see Fig. 6) can be seen in Appendix A.

4.4.2 Correlation between Modified LoC & Effort

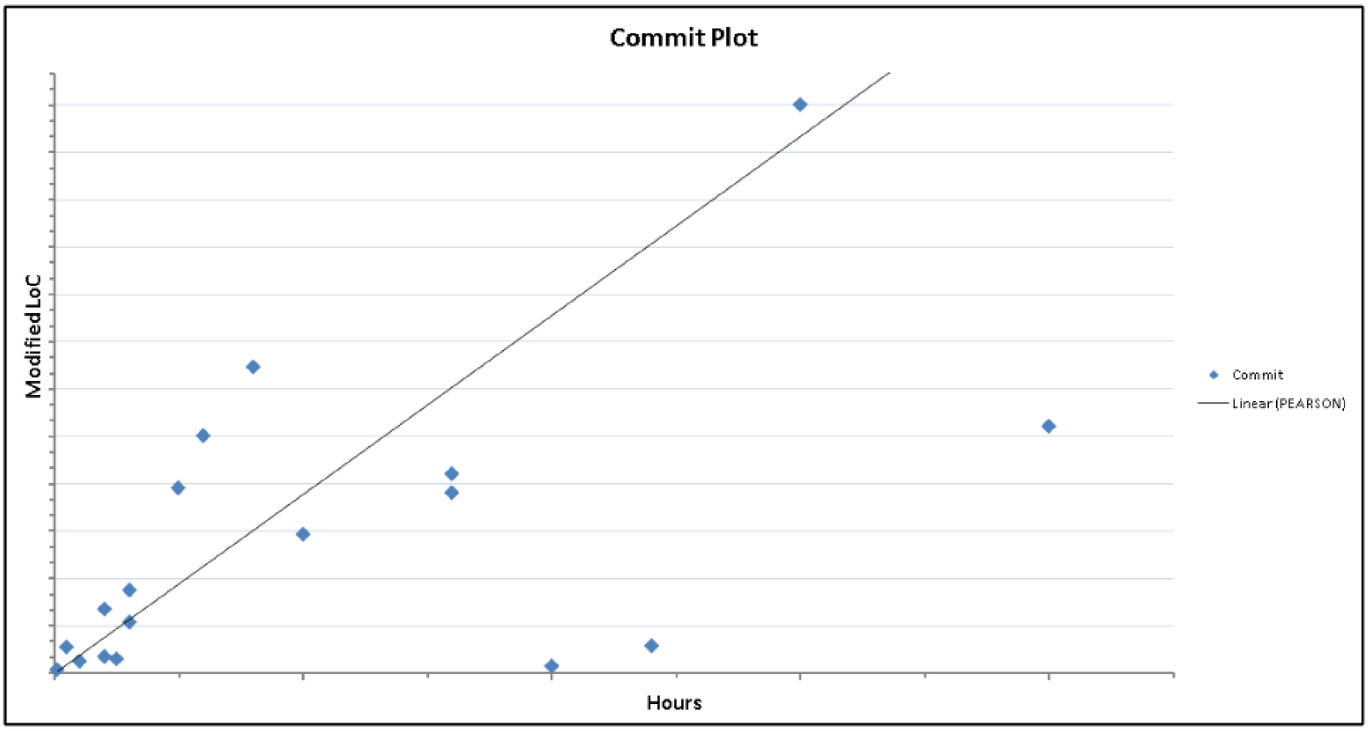

Figure 6: Chart showing distribution of commit modified LoC/effort pairs

The current result of Pearson's Correlation on the set of commit modified LoC and the set of commit effort, after excluding positive outliers found via the MAD-Median-rule, is 0.76. The manually added line in Fig. 6 represents Pearson's Correlation and helps visualize the result of the equation (the further the commits are from the line the less normal the distribution is). These results show that the average ratio is a fair indicator of how many LoC can be produced per hour when taking into account the range between one Standard Deviation below/above this average. This was strengthened by the fact that the final, 25th commit had a ratio that fell inside the range for the average ratio of the previous 24 commits.

4.5 TD & Effort Correlation Results

In order to analyze the impact existing TD has had on commits with effort, over-threshold TD occurrences in files before they were modified has been compared to the modified LoC per hour ratio of the related commits following the proccess described in section 3.3.2. Table 2 shows the modified LoC per hour ratio of each commit, as well as the occurrences of files with too high quantities of a TD type, both for each type separately and for all of them combined. As the diagram used for measuring Non-Allowed Dependencies only covered one application, the #_MV measure was not applicable (n/a) to several commits. The colour coding represents the commits that were unusually easy (green) and if the TD occurrences were enough to be transformed to "1" (red) when performing the Conditional Probability, Chance Agreement and Cohen's Kappa calculations. From Table 2 it is easily visible that commit difficulty and over-threshold TD occurrences is not perfectly related, as the most difficult commit is not the one with the most occurrences. It also shows that certain TD types were more/less frequently found at above threshold levels. Rather than the TD type actually being more/less frequent, this could instead be an indication that the related thresholds are too low/high.

Table 2: Modified LoC per hour & TD occurrences for each commit with effort

Commit	MLoC/h Ratio	McCabe	Halstead	Issues	Duplication	High Dep	MV	Total TD
1	0.013	0	0	0	0	0	n/a	0
2	0.086	0	0	0	1	0	n/a	1
3	0.2	0	0	1	0	0	n/a	1
4	0.708	2	3	0	2	0	n/a	7
5	3.6	0	2	0	1	1	0	4
6	3.9	1	2	1	1	2	1	8
7	5	0	0	0	0	0	0	0
8	5.04	8	7	2	17	3	4	37
9	7	4	4	1	4	1	2	16
10	7.125	2	1	1	1	1	1	7
11	7.875	2	2	1	1	1	1	8
12	8.8	1	2	0	1	0	n/a	4
13	10.67	1	5	2	5	1	2	16
14	12	0	3	1	2	1	3	10
15	17.33	0	1	0	1	0	0	2
16	20	1	1	1	1	1	2	7
17	23.4	0	1	0	2	1	n/a	4
18	24.1	1	0	0	1	0	n/a	2
19	24.25	2	3	0	3	2	2	12
20	25	0	0	0	0	0	1	1
21	32	0	0	0	0	0	n/a	0
22	47.95	2	7	0	6	0	n/a	15
23	62	0	1	0	1	1	n/a	3
24	114.9	3	6	0	3	0	1	13
25	121.3	0	1	0	0	0	n/a	1

While certain inferences can be made by examining Table 2, the correlations needed to be calculated to reach more detailed results. Table 3 shows the results of the comparisons made between occurrences of each TD type separately and the modified LoC per hour ratios, occurrences of all the TD types combined and the modified LoC per hour ratios, and finally between modified LoC and effort, through applying the process from section 3.3.2. More specifically, the high total TD row shows the results for comparing the sum of indicator values for all analyzed files in each commit to the corresponding commit ratios. These are the results based on the MAD-Median-rule [18]. The results based on 20% Trimmed mean [18] can be seen in Appendix A.

The correlational strength columns refer to if high enough values were reached in the three correlation algorithms (Pearson's, Conditional and Cohen's). There are two of these columns as the Conditional Probability strength depends on which condition was initially filled. For example, the probability of a commit being normally distributed given that it has at least one occurrence of %_MC is higher than the probability of a normally distributed commit having an %_MC occurrence (84.62% vs 52.38%). It is possible for the Conditional Probability to only be strong enough in one direction. As previously mentioned, ≥2/3 correlational strength is the threshold for significant correlation. This was only achieved between modified LoC and hours spent, which indicates that the work pace of the participating team is quite consistent. The multiple relations with 1/3 strength are as of yet inconclusive, as they could potentially become stronger or weaker if the amount of commits with effort were to increase. Note that the negative Pearson result between specific/combined TD types and commit difficulty shows that both the specific- and the combination of all TD types have had an effect (even if it's small), as decreasing difficulty is represented by an increasing modified LoC per hour ratio.

Table 3: Correlations based on MAD-Median-rule Dataset

TD Type	Pearsons	Conditional Probability commit|TD & TD|commit	Chance Agreement	Cohen's Kappa	Correlation strength commit|TD (x/3)	Correlation strength TD|commit (x/3)
McCabe	-0.18	84.62% & 52.38%	51%	0.35	1/3	0/3
Halstead	-0.20	77.78% & 66.67%	65%	0.31	1/3	1/3
Issues	-0.26	100% & 42.86%	40%	0.39	1/3	0/3
Duplication	-0.13	84.21% & 76.19%	68%	0.41	1/3	1/3
High Dep Count	-0.04	91.67% & 52.38%	49%	0.40	1/3	0/3
Modularity Violations	-0.09	90.91% & 76.92%	74%	0.39	1/3	1/3
High Total TD	-0.15	100% & 4.76%	19%	0.17	1/3	0/3
Modified LOC / Effort	0.76	100% & 33.33%	85%	0.63	3/3	2/3

4.6 Validation of the Results

The tool results for the TD types have been successfully validated for correctness manually by comparing them to either similar tools or to manual calculations. This validation was considered successful as these comparisons showed the same or highly similar values. The indicator and measures used by the measurement system as well as the tools that were used to produce measurements have also been evaluated by the software expert on several occasions. The last of these occasions consisted of an extensive semi-structured interview towards the end of the project, where the final correlation results were discussed in addition to the measurement system and tool results. The following two subsections provide more details on the tool results analysis and the final validation.

4.6.1 Qualitative TD Results Analysis

By gathering the chosen TD measures from a few of the available commits with effort, data was presented to the software expert in order to help with detecting false-positives. For ASA Issues, three different types of High Impact Issues were detected via Coverity SA in the analyzed commits. Two out of these issues were agreed to be "real", while the third was proven to be false-positive. The detected Non-Allowed Dependencies were also validated to be positive, but the team members were unsure of their importance. The discovered Code Duplication was validated by the authors of this study, by comparing the outlined duplication blocks with the referred starting lines in other files as well as in the file itself. This comparison showed that the detected duplication blocks did indeed exist. The effect of the initial validation was that the source code to derive #_ISSUES was updated to not count the false-positive issue type, which in turn caused less files to be registered as over-threshold for that measure.

4.6.2 Final Validation

During the interview with the software expert, described in section 3.3.3.2, each TD type was first described in detail while summarized TD results of all files in the current application version were shown (see Table 4). The software expert was then asked to prioritize the types. Based on the type descriptions the expert pointed out that the files that contained a combination of high McCabe, high Halstead Error and many dependencies were files that had critical responsibilities and therefore are very complex. The expert also claimed that refactoring these very complex files was not worth it (based on the files with all three TD types visible in the summary) as they are not changed often enough. The expert also thought that Non-Allowed Dependencies was interesting, but was skeptical to them being refactored due to the potentially massive rework required. To put it simply, the principal was hypothesized to be more expensive than the interest, which would be consistent with the weak correlation results. On the other hand, duplication was not considered to be important at all for refactoring.

The interview then continued by asking about recent challenges and which parts of the code they originated from. This was then compared to the high TD files shown in the summary. The expert concluded that components that had been pointed during human identification of TD were still the most challenging components, and the comparison results were similar as well. It was still the case that files with large amounts of several TD types were commonly from components indicated as complex.

Table 4: Visualization of TD results per file

DUP_LOC	%_DUP_LOC	%_MC	HR_SUM	#_ISSUES	#_DEP	#_MV	#_HIGH_TD_TYPES
179	3.33%	50.10%	8.73776	3	43	2	6
114	2.03%	68.90%	13.24837	0	22	1	4
49	3.87%	80.80%	4.779	1	27	0	3
73	16.44%	56.60%	1.22255	0	7	0	3
204	52.71%	96.97%	2.30561	0	12	0	3
29	1.81%	22.26%	0	2	41	3	3
84	2.41%	60.38%	7.81148	0	30	1	3
24	6.61%	60.00%	1.71337	0	15	1	3
0	0.00%	34.75%	4.19038	0	14	1	2
0	0.00%	94.04%	1.30544	0	13	0	2
0	0.00%	69.67%	2.38057	0	17	0	2
0	0.00%	0.00%	1.23451	0	62	0	2
0	0.00%	61.29%	0	0	15	1	2
0	0.00%	33.17%	1.50526	0	25	2	2
0	0.00%	100.00%	2.07229	0	9	0	2
16	12.90%	0.00%	0	0	10	1	2
...	...	...	...	...	...	...	...
0	0.00%	68.52%	0	0	6	0	1
0	0.00%	0.00%	0	0	9	1	1
0	0.00%	23.84%	0	0	14	2	1

After discussing challenges, the interview then moved on to showing detailed tool results as well as selected examples from the production code where high levels of specific TD types had been found. These examples were chosen based on the strategies specified in section 3.2.4.2. This was used to validate if the tool results were expected and if they appeared to not be false-positive. The expert was also asked to reprioritize the TD types after seeing the detailed results. One Coverity issue was considered unexpected, as the expert could not explain its existence. For the other source code examples, it was considered clear why they were indicated to contain the specific TD type. Due to the unexpected issue, the priority of Coverity issues was increased in the expert's opinion. CppCheck issues on the other hand were considered to only consist of false-positives, thus being irrelevant. High dependency count was also considered irrelevant, with the motivation that the files with many dependencies were required by design to have them. Specifically those files were responsible for startup configuration of the application. The expert's opinion on McCabe and Halstead Error was practically unchanged. For McCabe specifically, the expert also claimed that in the shown example it was better to have everything in one sequence rather than separating into smaller methods.

The software expert also claimed that based on the chosen example, refactoring Code Duplication may be difficult in C specifically due to its lack of templates (a C++ construct that is used to create generic files and functions). He also claimed that duplication refactoring would be simple in C++ through the usage of such templates. The duplication results were also the the most difficult TD type to show, due to lack of good visualization. For example, investigating a block that is duplicated twice in the same file requires the file to be opened twice, so that both block starts can be viewed simultaneously. Modularity Violations were also shown to contain less false-positives than expected, and the expert seemed to judge them to be more important after seeing the example, as it proved that cyclic dependencies existed on the component level.

When asked which of these examples the expert would consider refactoring, the specific Coverity issue was considered the only candidate, while Modularity Violations were in need of further investigation. The expert had no particular opinions regarding the correlation results, as seen in Table 3, as none of them showed an existing strong correlation between TD and maintenance effort. The expert was also aware that these results did not prove lack of correlation as the sample size was small.

Shortly after, the build-integratability of the TD measurement & visualization process was validated by the tool expert, through showing and explaining Fig. 4. This expert concluded, based on the explanations, that mainly Code Duplication was not integratable enough. This was because of the manual “sorting data in excel”-step.

5. Discussion

In this chapter, the three main contributions of this study, the measurement system (answers **RQ1)**, the tool mapping (answers **RQ2)** and the process to measure TD impact on maintenance effort (answers **RQ3)**, and their results are discussed together with the processes that were used to create them. The resulting discussion is used to motivate why these three contributions answer their respective research questions and why they can be considered as contributions both from an academic perspective and a practical one. The validity of the results and how they were produced are also discussed and suggestions for changes, improvements and future work are highlighted. Finally, the limitations and ethical ramifications of this study are clarified.

5.1 Measurement System Discussion

One of the main contributions of this study is its measurement system for discovering files in need of refactoring, which aggregates several TD types and visualizes the result in order to answer **RQ1**. This in turn shows team members where in the system maintainability can be improved to avoid additional maintenance effort in the future. The understandable visualization allows experts to assess if the identified TD is in need of refactoring or if it can be postponed. The measurement system is a contribution to both practice and academia, as it can be applied on at least the design level (see Fig. 3) to other systems, either by team members or by researchers. The measures used in the system could even be changed to other source code level TD types (or different base/derived measures related to the currently used types) with other thresholds, while still using the overall indicator for refactoring priority based on TD levels found through source code analysis. Changing, adding or even removing measures is not an issue, due to the modular design of the measurement system. In the following three subsections we discuss how well the system fulfills best practice requirements, how correctly the activities to create it were conducted and how reliable its results are.

5.1.1 Measurement System Requirements

Staron et al. [20] have specified requirements for measurement system creation frameworks, and through several of these requirements (requirements 1,3,5,6 in [20]) it also becomes apparent how they believe the measurement system itself should be. Their first requirement of realizing ISO standard 15939:2007 [15] is certainly fulfilled, as evident by Fig. 3. As we incorporate two tools that were already available on-site (Coverity SA and Coverity AA) we have also achieved their third requirement, collecting data from existing tools in the company, to a degree. Requirement six, measurement system modularity, has also been fulfilled, as seen in Fig. 4, since the model is separated in different stages which represent base measures, derived measures and indicators. Specifically, from the tools we receive the base measures and through the ad-hoc parsers the derived measures and the indicator. The indicator is later visualized as an Excel sheet with color coded prioritization.

In regards to their fifth requirement, updating the measurement system without updating the source code, it is partially achieved in the measurement system. While the analysis models connected to the refactoring urgency indicator would need to be changed in the code, certain sub-indicators can effectively be ignored by passing an empty file representing that TD type to the source code that builds the indicator and rates files (see Fig. 4). For adding measures with their own sub-indicators, new code can be written that modifies the output file of the original, to add the new measurements on top of the existing measurements. The initial steps in Fig. 4 that create the base and derived measures could potentially also be connected to another indicator based on all or part of those measures, through new code.

5.1.2 Measurement System Activities

Staron et al. [22] have also outlined the activities that should take place during the creation of measurement systems. Several of these activities were also conducted in this study, while others were either overlooked or performed either incorrectly or partially Regarding information needs elicitation, the product guard and software expert had a general idea of TD related measures that they thought had potential to be interesting. As the overall information need was to discover if the existing TD was causing noticeable interest, previous TD literature was also studied to find the TD measures that appeared to cause high amounts of interest. Thus, our elicitation was not entirely stakeholder based, but we believe this extra step was needed to determine how TD and its interest could feasibly be measured. The purpose of knowing the cost of this interest was to know if it needed to be dealt with, which is why this information need was interpreted as which files have the highest refactoring priority (from a TD perspective). Performing the refactoring is analogous to paying the principal of the TD. The fact that the indicator is displayed on a per-file basis was based more on the assumption that refactoring one high TD file is preferable to refactoring several medium TD files. The stakeholders found the proposed indicator acceptable, but we did not involve them in its design to the degree they should have been according to Staron et al. [22].

For the analysis models the stakeholders were not involved beyond validation, as it was decided to use thresholds defined as “case independent” in previous literature where available, such as Heitlager et al.’s duplication threshold [13]. This is in direct violation of Staron et al.’s recommendations [22], but the stakeholders were not familiar enough with the TD types to have any case-specific thresholds in mind at the start of the study. Halstead Error, Modularity Violations and ASA Issues also did not have thresholds in previous literature, which led to arbitrary, but intuitive, thresholds such as zero-tolerance for defects, violations and issues. Also, only methods with a whole Halstead Error or more were considered as too complex, rather than taking the file’s entire Halstead Error score in account. Both the thresholds from previous research and the self-selected thresholds are discussed in section 5.2. The derived measures and their measurement functions, besides SUM_HR, were also defined based on previous literature while the base measures were largely identified from the beginning in the form of the TD types stakeholders were interested in.

Regarding the entity and its associated attributes, commits were considered by the stakeholders to be an ideal representation of the work that was performed. The commits also had the added benefit of allowing comparisons between files with high TD and files that were modified in difficult commits, through effort specified in commit messages. The measurement system's specification and architecture have also been detailed in Fig. 3 and Fig. 4 respectively. Fig. 4 also details the necessary information sources in the form of files. The necessary measurement instruments have been developed through a combination of existing tools and ad-hoc parsers, which in turn realize the measures and indicator specified in Fig. 3. However, ultimately the measurement system has not been fully integrated within the case context due to the weak effort correlation results and validation results on certain measures, as discussed in section 5.4.4.

5.1.3 Measurement System Result Reliability

Staron & Meding [23] also provide guidelines for assessing the quality of a measurement system's results. Regarding the quality of this study's results, the conclusion is that it is mixed. For example, the accessibility of the information is not optimal as the system has not been fully integrated on-site (several automatable parts are still performed manually rather than by scripting). As a result, it takes around 30-40 minutes to perform all the steps in Fig. 4. The timeliness of the information was accurate however, as it was obtained from either the system state before the commit being examined (in order find the possible impact TD could have had on the commit) or on the latest version of the system (to provide up-to-date examples of where the TD levels are high).

The results for the current system status were shown during the validation interview, where they were confirmed by the software expert in a sense that they believed that the TD types did in fact exist in their shown examples. However, %_MC, SUM_HR, #_DEP were considered less believable as their examples were considered to not need refactoring. This was also true for several ASA Issue categories due to false-positives. The volume of the data was based on the parts of the source code that the software expert was interested in seeing TD results for, and only files that have over-threshold levels of at least one TD type are included. This also contributed to the conciseness of the results. The design choice to visualize sub-indicators on the same row as the refactoring urgency indicator also contributes to conciseness.

The base and derived measures are also visible on their own inside separate pages in the same Excel file. While the same file filters have been applied to these pages, the format of this data is different from the indicator page, affecting the overall consistency. This data is however still required, in order to determine where in the high TD files the TD was found. Based on the validation results, all Excel pages have an adequate level of understandability, although DUP_LOC and %_DUP_LOC was more difficult to display compared to the other measures (especially when a block is present multiple times in the same file).

A quality that the measurement system results lack within the case context is completeness. As the TD types that are measured do not have a strong correlation to commit difficulty, the model may be missing a measure that does have this strong correlation. The measures provide correct results, but evidently in this case they were not the correct measures for finding causes of significant extra effort. The information was still highly relevant for this study however, as it was required for answering **RQ1**, and for providing the TD that was analyzed for correlation to effort as part of **RQ3**.

5.2 Measures Discussion

The base and derived measures that make up the measurement system and were gathered through tools found through answering **RQ2** have had different levels of success in regards to stakeholder acceptance and usefulness. However, the measures that were less successful in this context may be highly suitable in others. In any case, the measures used in this study are based on a combination of recommended TD types from the existing literature, and thus provide an excellent starting point for investigating TD for both team members from other contexts as well as for researchers in future studies. In the following six subsections, the process of gathering and evaluating each measure is discussed as well as their validation results. The results for where these measures were used in correlations to answer **RQ3** are discussed as a whole in section 5.4.4, as none of the measures are strongly correlated.

5.2.1 McCabe Cyclomatic Complexity: %_MC

The threshold used for %_MC (complexity of complex methods divided by the file complexity sum) was actually from another file-wise McCabe measure defined by Heitlager et al. [13]. However, this other measure was tested on several commits together with %_MC, and the resulting percentages were always very close. The other measure is based around LoC in complex methods divided by total LoC in a file. There are several reasons why this file-wise McCabe measure is worse than %_MC. First of all it was not possible to measure pure LoC (no comments, no blanks) for methods, which causes complex methods with many comments/blanks to seem worse than they are. In essence, calculating based on LoC ratio causes lines that have zero McCabe to be counted as complex. The other measure also requires separate thresholds depending on complexity level (such as 5% for source code with 100 or more complexity [13]) as methods with the same amount of LoC can have differences in complexity. On the other hand, %_MC already incorporates separate complexity values in its calculation. This means that it is theoretically easier to achieve 50% complexity with %_MC (although again, for the sampled commits the results were still close).

Regarding validation of this measure, the software expert and tool expert have both voiced hesitation on the usefulness of refactoring Cyclomatic Complexity in general. Specifically the software expert considered it to be better for readability to keep the related source code to one method in the shown production code example, rather than jumping between several smaller methods. The tool expert also voiced concerns on the mathematical validity of converting a method level measure to a file-wise one. As mentioned previously %_MC was however accepted by several stakeholders from the same company in the study it originates from [6] (named Effective_M%). Thus there are conflicting opinions. In the end, %_MC should still be more mathematically sound than Heitlager et al.'s [13] measure, for the previously mentioned reasons.

5.2.2 Halstead Error: SUM_HR

The threshold used for Halstead Error and its file-level representation was exploratory, as Halstead Error was not used in any of the reviewed previous literature. However the software expert agreed that the reasoning to only count methods that reached a full error felt intuitive, and did not explicitly state that the example would be made worse through refactoring (unlike with the %_MC example).

In our opinion, the example with high Halstead Error but low Cyclomatic Complexity appeared less readable than the example with reversed conditions. This was due to how the constructs that cause Cyclomatic Complexity ("if" and "else" statements, "switch" statements, loops etc.) are often quite self-contained, so the high Cyclomatic Complexity example could easily be read in stages.

5.2.3 Code Duplication: DUP_LOC & %_DUP_LOC

The threshold for %_DUP_LOC was taken from Heitlager et al. [13], while the threshold for DUP_LOC was an arbitrary figure based on what we considered to be an excessively large duplication block. As part of the %_DUP_LOC calculation it is compared to the amount of pure LoC, which means that the percentage can become inflated if the duplicated blocks contained comments. If the comparison had been to pure LoC plus comment lines instead, the percentages would risk being too low, as the duplication block may not have had any comments, even if the file itself has plenty. In the end, this did not have a significant effect, as most files with %_DUP_LOC ≥ 10% but not DUP_LOC ≥ 100 had a percentage that was several units above 10%.

Regarding the validation of these measures, the software expert agreed that excessive duplication should be avoided, and that the tool was able to find real duplication. Through the chosen example we were also able to gain new insights from the software expert in how refactoring of duplication in C specifically may in some cases be more difficult than expected (due to the lack of templates from C++). The measurement function for this derived measure is in need of modifications to take such situations in mind and ignore them, possibly through keyword recognition. This was however not possible during the timeframe of the study.

5.2.4 ASA Issues: #_ISSUES

The threshold for this category was based on an assumption of zero-tolerance for High Priority Issues. This assumption was strengthened by the existing commit procedure within the case context, which stops commits if issues from another (excluded from the tool mapping), proprietary tool are found. As ASA Issues are already actively refactored on-site, the correlation for this measure was never expected to be high. Additionally, a lot of the discovered issues did end up being confirmed as false-positive by the software expert. There was however one interesting issue, and the existing policies regarding committing proves that this is a measure that is already prioritized. It appears that the issue type is more important than the total number, even within the same priority level. However, the frequency of false-positives may very well be due to the real issues being fixed before any commit is made. Thus, the validation and correlation results for this measure are perhaps the most case specific results.

5.2.5 Dependency Count: #_DEP

The threshold that was used to indicate too many dependencies is taken from Rapu et al. [10]. As it turns out, only four files out of over a thousand actually reached this threshold, and the software expert considered that these files are not refactorable due to their complex responsibilities. From an improvement perspective this measure was thus not very useful, but it did manage to only point out complex files, meaning it could be very useful from an informative perspective when dealing with unfamiliar applications (to determine the “core” files).

5.2.6 Non-Allowed Dependencies: #_MV

The threshold for this category was based on an assumption that Non-Allowed Dependencies could be considered as a type of High Priority Issue. As previously mentioned it was only possible to measure for one application, but it was easy to verify by hand that the tool results were not false-positive (through examination of dependencies directly in a file). The software expert considered the diagram that was inputted into the tool to be relatively up-to-date, and found the results interesting as they showed circular dependencies on a component level. This was considered as a possible target for refactoring. Our opinion is that the tool used for this measure also serves an informative purpose, as it is easy to read how everything in the application is connected in the dependency treeview before even adding the manually specified diagram.

5.3 Tool Evaluation Discussion

Another of the main contributions of this study is the identification of C/C++ compatible TD measurement tools (see Table 1) and the evaluations of several of these tools that were used to answer **RQ2** (see sections 4.3.2.1-4.3.2.6). Although these tools allegedly support other programming languages as well, that aspect has not been verified during this study and as such the tool evaluation results are the most context-specific contribution. Still, any system made in C/C++ should be analyzable by the selected tools, whether it be by team members themselves or as part of a future research project. This remedies the gap identified in existing literature regarding how to apply existing TD theories in a C/C++ context. The identified gaps regarding open source tool support for certain TD types also highlights possible future projects to develop such alternatives.

The set of tools that were found and considered build-integratable were tested on-site at Ericsson to assess their actual compatibility towards the case context's work process. Completing the tool mapping with information from this on-site testing addresses **RQ2**. There may definitely exist tools that were overlooked, but for the purposes of this study, the tools found through this method, with some additional ad-hoc parsers, were enough to implement the planned measurement system (see Fig. 3). Currently however, two TD categories require proprietary tools, which puts a limit on where the measurement system can be used. Another limitation of the measurement system is that it focuses on providing a quick overview of TD levels in a file. The indicator will point out files with large amounts of TD, but the tool results for each measure still need to be examined if a decision to refactor is made (in order to see exactly where in the file the TD is located). This led to a visualization design containing separate pages for each TD type in addition to the main indicator page. Based on results with the tools used for Non-Allowed Dependencies (Coverity AA and Structure101) it was also realized that those tools can detect a form of documentation debt as well, as the found Non-Allowed Dependencies can show where the documented architecture differs from the real architecture.

The selected tools were considered as sufficiently integratable due to potential for automation, but reaching this automation still requires work. For calculating (%)DUP_LOC, %_MC, SUM_HR, #_ISSUES and #_DEP, this is just a matter of creating scripts that link together commandline arguments to run the tools and the ad-hoc parsers. For measuring #_MV however there is an upfront cost in that a diagram must be designed at least once. A separate tool from the Coverity suite can then be used to analyze source code for Non-Allowed Dependencies.

The ad-hoc derived measure parsers were mainly necessary to reformat tool outputs to make them compatible with the ad-hoc indicator parser. Doing this formatting through code eliminates several manual steps. As this code was developed as a throwaway prototype [24], it did not take too much time away from the study to create it (approximately two days).

However, if an open source version of the measurement system was to be created then these ad-hoc parsers should be improved (which is why they are not publicly available).

As seen in Table 1, open source tools for McCabe and Code Duplication are readily available. On the other hand, the tested open source ASA Issues tool is inferior to the proprietary alternative, and tools for Halstead Error and Non-Allowed Dependencies are proprietary-exclusive. As the case context requirement was to support both programming languages, the open source tool Dependometer (Non-Allowed Dependencies for C++) has been ignored. To solve the "lack of open source solutions"-issue, there are several steps to take. While Non-Allowed Dependencies may be easier to implement as a case specific solution, a generic Halstead calculation tool should not only be feasible but also fairly simple, as the creators of CMT++ describe in detail how Halstead metrics are calculated for C/C++ code [19].

For Code Duplication, the visualization of results from PMD also has room for improvement, especially in regards to its Eclipse plugin. Specifically, the plugin would benefit immensely both from highlighting of duplication blocks, and context menus for navigating from a block to the other occurrences of the same block. The current solution for calculating the duplication in a specific file is also dependent on manually sorting and removing unnecessary data from the results in Excel, due to how PMD mixes the block's start and size values with a copy of the actual block, resulting in output files with several thousand lines of data. This step of the measurement system could be sped up by removing the block copies programmatically instead of manually deleting their column in Excel.

There is also a feature gap in the tested LoC category tools. None of them can provide the non-blank, non-comment LoC for a single function rather than a whole file. It would have been interesting to use such a value in comparisons, as many of the TD types have base measures on function level as well.

5.4 Commit Difficulty Analysis & Correlation Discussion

The final main contribution of this study is the process for assessing if a certain TD type (or TD overall) is responsible for increased commit difficulty. This will show if the TD has had a tangible interest, whose level can be used for the prioritization mentioned in **RQ3**. To make this assessment the commits themselves were graded for difficulty, which allows both unusually difficult and unusually easy commits to be identified. We believe modified LoC divided by time spent on a commit is a more accurate representation of maintenance effort compared to previously used interest indicators such as change frequency and defect density, as one is actual effort and the other two are effort surrogates. A more accurate representation of interest in turn improves TD Management decision making, where the goal is to assess if the interest is larger than the principal estimated by experts. As per the goals of this study, we have provided one such accurate representation and a process for how to compare it to TD levels, which is adoptable by team members independently or as part of future research projects. The following four subsections discuss the reliability of how commit difficulty was calculated and how it was used in correlations, as well as possible improvements.

5.4.1 Measuring Commit TD

When measuring the commit TD levels that were used to correlate with commit difficulty as part of **RQ3**, it was chosen to exclude files with significantly less modified LoC compared to other files in the same commit. This restriction allowed for faster analyzing of the commits with few large changes followed by one to three modified LoC in multiple files. A potentially more accurate solution would have been to link a file change to the method that was modified rather than the entire file. This would have allowed for less deriving as many TD types are initially measured on method level. It is also more likely that TD in the specific method has had a negative effect on the change compared to the TD levels of the entire file. These results could then be converted back to a file-level indicator based on the amount of high TD methods the file has. However, as far as we are aware there is no easy way to link a file modification in a Git commit to the specific method that was changed. Also, there is a weakness in the method-based approach due to the fact that new methods are much more common than new files, and that the modifications can consist of new and modified methods in the same file. In the end, correlating commit difficulty to file-level rather than method-level TD measurements trades accuracy for simplicity.

5.4.2 Calculating Average Ratio

As previously stated, the average ratio (and all correlations based on it) was recalculated using the MAD-Median-rule [18], as the previous usage of 20% Trimmed and one Standard Deviation from the trimmed mean was too lenient. It was chosen to try a different method rather than changing the previous one to be based on two Standard Deviations, as no correlations existed even on the too lenient level. These methods were needed to find the threshold for transforming commit ratios into "1" or "0" for several of the correlation algorithms that were used to answer **RQ3**. In the end, no strong correlations were found between TD and commit difficulty anyway, but the mean from the MAD-Median-rule sample should be closer to the population mean compared to the trimmed one.

5.4.3 Correlating Modified LoC & Effort

Based on the correlation results, there is a strong relation between time spent and modified LoC, as the commits where a lot of source code was modified are often the commits that took large amounts of time to complete. In a way this is not surprising as 21 of 25 commits are within a normal distribution according to the MAD-Median-rule. The Pearson's result for this correlation should be valid, and we are also satisfied with how modified LoC and effort values were transformed to "1" and "0", as the resulting datasets pair high modified LoC with large effort, thus being consistent with the Pearson's result.

5.4.4 Correlating TD & Commit Difficulty

As seen in section 4.5, no significant correlations between specific over-threshold TD types and commit difficulty or overall over-threshold TD and commit difficulty were found. As it stands now, lack of correlation cannot be proven either, due to the sample size. It is likely that the low number of commits with effort is a major factor, but these results still warrant the question; are the right TD types being measured? This study has at least covered Source Code TD extensively, but perhaps the true causes of significant interest lie in a different TD area such as build-, test-, infrastructure- or versioning TD. The correlation results do indicate that the chosen TD types have an interest cost, as the Pearson's Correlation value is negative as expected. The value is just not significant enough based on Zazworka et al.'s [3] thresholds. When comparing to the value required for 95% significance level with 23 degrees of freedom [25], one finds that the threshold from the significance table [25] is actually lower than 0.6, but the TD correlation results of this study do not reach this more lenient threshold either. You can still prioritize the TD types by their interest as intended by **RQ3**, but their priority would be as low as their interest.

One thing that we in hindsight wish had been done differently would have been to challenge the notion that effort should only be specified on production code commits and not test code commits. The argument that these commit types are too different could have been circumvented by keeping two commit datasets, with separate ratios and correlation results. This could have also been used to analyze if the commit types truly are as different as was assumed, by comparing the two datasets.

For the correlation process we are slightly doubtful regarding the correctness of how the results are transformed to be used in Conditional Probability, Chance Agreement and Cohen's Kappa (while the Pearson's results should be without any issue). We believe that it makes sense to attempt to pair occurrence of TD with commits that were not unusually easy, but perhaps the "1" and "0" for specific over-threshold TD type occurrence should have been based on deviation from mean, rather than if the commit had at least one occurrence. The conditions for becoming "1" rather than "0" may be too lenient for the transformed specific TD type occurrence datasets. One issue however is that different TD types have more/less frequent occurrences, so that their occurrence datasets also have different means. This was the reasoning for choosing the "one or more occurrence"-approach. The differences in occurrence also affect the overall TD correlation results, as certain TD types may be more important. The threshold for becoming "1" in transformation could potentially be divided into a chain of thresholds for each TD type instead, where at least one threshold has to be fulfilled. As it stands now, commits with many occurrences of less important TD may have been treated as more important than they actually were.

5.5 Validity

In this section the overall validity of this study is discussed using the three types of validity outlined by Runeson & Höst [4].

5.5.1 Construct Validity

To ensure the construct validity of the tool mapping, the tools themselves were tested. In order to select tools that gather accurate measurements, data triangulation was performed by trying out several tools both to choose the most suited for the case context as well as to compare if they give (almost) identical measurements on the same measures. However due to the lack of measurement overlap Code Duplication could not be compared between tools. Also, the other tools with the DUP category were based on PMD, making result comparison pointless.

The construct validity of the process this study uses for TD and effort correlation should be similar to its original use by Zazworka et al. [3], where they claim that the thresholds were set higher than strictly necessary (based on their findings from previous literature) in order to reduce uncertainty. However, The TD interest surrogates they correlate to are different. They also applied this process and measured TD from system versions rather than commits, which potentially could mean their dataset values were much closer to each other (many files are present in every system version). These similarities should support their smaller sample size (13 system versions vs 25 commits). In any case, a process designed for correlating over system versions has been used to correlate over commits instead, which may be less suitable.

To ensure the construct validity of human validation of measurements/indicator, this study aimed to organize and visualize the measurement system results in an understandable way as per the recommendation of Kruchten et al. [8]. Based on the validation interview results, this was also successful. In regards to construct validity of the validation interviews themselves the guidelines by Martini et al. [2] were followed. This included summarizing major findings and presenting them to the interviewee, to ensure their answers had not been misunderstood.

5.5.2 Internal Validity

The internal validity of the method that was used to find extra effort is potentially affected by several outside factors, such as varying skill levels in the participating team members. Another factor is that the accuracy of the effort values different team members provide may not be perfect. However the team members already use a reporting system based on hours in their work, which makes the effort in commit method fit in fairly naturally. The fact that modified LoC and Effort was strongly correlated over the commits from seven different team members also implies that there is not that much variation in the skill levels (most commits were similar productivity-wise). The validity of each separate measure that is correlated to this effort is discussed in sections 5.2.1-5.2.6.

Regarding the correlation process itself, the triangulation gained from using several correlation algorithms should ensure result validity for the specific sample they are applied to. As mentioned previously however, Conditional Probability, Chance Agreement and Cohen's Kappa require that the values in the examined datasets are transformed into "1" and "0" for pairwise comparisons. Section 3.3.2.4 details how these transformations were determined. We believe that the decision criterias for these transformations makes sense, but it was still a decision that had to be made based on intuition rather than scientific background. Ultimately the decision on how to transform the values had a significant effect on the results of the related methods. The results of Pearson's Correlation are based on the untransformed data, and are thus not affected.

Extra effort could also hypothetically be a byproduct of paying back principal rather than dealing with interest. To examine this, the levels of system-wide TD were compared between the end of January and the end of April (the time-range for the commits with effort). The results in Table 5 show that only a minimal reduction in TD levels has occurred, and the team members themselves claim that none of the commits with effort were specifically TD refactoring tasks. This shows that the effort spent during this time-period did not include any major attempts at paying back principal. Specifically, the columns in Table 5 represent:

- **Point:** The point in time.
- **DUP__LOC Total:** The total amount of duplicate LoC
- **#_MC Total:** The number of functions with too high Cyclomatic Complexity (>15)
- **#_HR Total:** The number of functions with too high Halstead Error (≥1)
- **#_ISSUES Total:** The number of High Priority Issues.
- **#_DEP Total:** The number of files with too many dependencies (≥40).
- **#_MV Total:** The number of Non-Allowed Dependencies.
- **Files with Extensive TD:** The number of files with over-threshold levels of at least one TD measure.

Table 5. System-wide TD at two points in time

Point	DUP_LOC Total	#_MC Total	#_HR Total	#_ISSUES Total	#_DEP Total	#_MV Total	Files with Extensive TD
Start	1562	57	25	23	4	101	131
End	1519	54	24	18	4	101	122

5.5.3 External Validity

For external validity, the generalizability of the correlation and validation results is limited since the case only covers a single department of a large company. Especially the validity of the (lack of) TD correlation to effort may be insufficient due to the lack of commits that specify effort. For the overall generalizability of this study, two strategies described by Wieringa and Denava [26] have been considered. The Statistical Learning strategy could be considered slightly applicable, as the Mad-Median-rule based mean ratio of the 24 previous commits turned out to be an accurate predictor of the 25th commit's ratio (as it's ratio was less than one StDev away from the current Mad-Median-rule based mean). This would imply that the ratio can be used to for example judge if a future commit within the case context is more difficult than expected. As Statistical Learning is designed for discovering statistical patterns in large sample sizes [26], it is unlikely that this ratio will hold if it were to be applied to commits made by other teams from the case context. A strategy that fits better with this study is the Case-based Generalization, which consists of the following three steps [26]:

1. Observe case event.
2. Explain the event architecturally.
3. Generalize the theory to architecturally similar cases.

The observed event in this study is the impact of existing TD on commit difficulty, while the architecture is the commits with effort specified as well as the measurement tools that were used to measure if the levels of the chosen TD types are above the specified thresholds. In order to replicate this architecture in similar cases it can be generalized by replacing the commits (which of course would happen anyway), with/without also replacing the measurement tools. By changing tools this makes the study generalizable even to other programming languages, as long as the necessary measures can be measured. Additionally, the correlation process, tool recommendations and measurement system implementation should be of interest to companies in similar contexts and maintainers of C/C++ projects of any size. This includes the method for finding difficult commits, based on specifying effort as hours spent and comparing to modified LoC, which could easily be applied to other programming languages.

5.6 Limitations

The study is delimited to one case (due to the established contract with the company in question). The scope is limited to identifying and visualizing for easier validation of Source Code TD from historical/current data. The possibility of build integration was considered an important criterion during tool selection due to the fact that the product guard and software expert in the case context explicitly declared their requirement for lightweight tools that do not require big changes to their current process. Using lightweight tools was also considered to be a key factor for making sure tool setup/configuration would not take up too much of the time designated for this study. This led to preference being given to tools available on-site (whose functionality otherwise requires open source tools and ad-hoc parsers to mimic) as they were already integrated.

Our amount of measures are also to a degree limited compared to previous studies such as [3], as we can only measure what the tools we have found and the code we have written can measure. This limitation stems from the fact that C/C++ support is required. Specifically one of the four approaches from [3] is not covered at all (Grime), although it is also the approach most closely tied to object-oriented programming [16][17] (which is not relevant to the case context). For effort correlations we were also mainly limited to one of four applications, due to it being the one that was most actively worked on by the team members that specify effort. The on-site experts we validated measurements with also ended up being limited in number (the two expert stakeholders). Finally, there is a major limitation in using modified LoC per hour as an indicator of extra effort in that it is a manual process that is heavily affected both by people forgetting to write effort in some of their commits, as well as the number of people who participate to begin with. This limitation can cause the time needed to gather enough commits with effort to become extensive. The process of choosing “irregular” commits to study (unusually low or high modified LoC per hour) also falls apart if commits are too similar (dataset is free of outliers).

5.7 Ethical Ramifications

This study uses information from the team members such as effort, which was the software expert’s own suggestion, and the other team members also voluntarily chose to provide the data. However, the effort that is provided in the commits by the team members could potentially be used to grade their performance. Even though the data presented in this report cannot be connected specifically to any member as we avoided mentioning the names of the team members, the commits with effort are still available inside the company. As such the effort values can be accessed by other development teams and managers. However, the team members were already required to specify their total amount of hours worked, as part of the company procedure. Specifying effort in commits should not be much worse from an ethical perspective, although the mandatory time specifications are less accessible by others. There should be no cause for concern regarding comparison of commit effort to the existing work time reporting schedule. This is due to the fact that it is already known that the commits with effort do not cover the effort spent on commits of testing code, which there are plenty of.

We have carefully reviewed this report to avoid publishing any company sensitive information. It will also be further reviewed for this purpose within the company itself. To have full consent of the information during the interviews the participants were asked if recording the session was acceptable, and it was also explained what the results would be used for.

6. Conclusion

Technical Debt (TD) in the form of Source Code TD is considered to be a problem for many software projects; especially in projects using Agile methodology as the rapid software evolution and frequent deadlines causes developers to use more short-term solutions. To neglect TD instead of properly managing it will result in growth of TD interest, which will increase both the effort required to maintain software as well as to extend it with new features. However, to be able to properly manage TD it must be correctly identified and presented in an understandable way. This thesis addressed this by measuring and visualizing source code level TD measures in an industrial context as a joint indicator. This thesis also analyzed the correlation strength between the chosen TD measures and commit difficulty, in order to indicate which measures had the strongest impact on maintenance effort within the case context. Conducting the study in such a context has also allowed for the results to be validated by expert team members. The key findings of this study are:

- *A measurement system based on ISO standard 15939:2007 that indicates refactoring priority of files based on levels of several TD types:* A measurement system that combines tools, ad-hoc parsers and manual processes to identify files in need of refactoring has been designed and implemented. This in turn shows team members where in the system maintainability can be improved to avoid additional maintenance effort in the future. The measurement system's indicator and its visual representation provide answers for the aggregation and understandability parts of **RQ1** respectively. This visualization has also been confirmed as suitable by stakeholders on-site.

The measurement system has a modular design, as it was expected to disable measures based on correlation results and stakeholder feedback. Due to the case specific nature however, the measures that were less correlated to effort or less useful to stakeholders in this context may be more important and useful in others. In any case, the measures used in this study provide a starting point for investigating TD in other contexts. Further research could also integrate additional TD type measures into the file-level indicator. While the implementation of the measurement system is limited to measuring TD in C/C++ systems, it should be applicable at the design level (see Fig. 3) to systems written in other languages.

- *A mapping of Source Code TD tools for C/C++, with accompanying recommendations:* Tools for measuring different forms of TD through C/C++ source code have been mapped based on the TD types they can measure, the output formats they provide and how integratable they are within the case context. Most of these tools have also been tested on-site to ensure their functionality, and a selection of recommended tools has been established. The tool results and how those results are presented provide multiple answers to **RQ2**. However, open source tool support for Halstead metrics and Non-Allowed Dependencies is lacking, which may prevent further research in certain contexts unless such tools are developed.

  The selected tools have also been confirmed by stakeholders to fulfill their integratability requirements, besides the tool for duplication. That specific tool is in need of additional coding to automatically derive measures from its results. Any system made in C/C++ should be analyzable by the selected tools, which remedies the gap identified in existing literature regarding how to apply existing TD theories in such a context. The identified gaps regarding open source tool support for certain TD types also highlights possible future projects to develop such alternatives.

- *A process for correlating specific TD types and overall TD to commit difficulty, allowing for TD type prioritization:* By rating the difficulty of commits through their ratio of modified LoC per hour, the TD levels of these commits can then be correlated to a more accurate representation of effort spent, thus answering **RQ3**. This will show which (if any) of the TD types are strongly correlated to increased commit difficulty. This process is also programming language independent, as all it requires is change size and the time taken to make said change.

  By applying triangulation of several correlation methods, it is ensured that the overall process only finds relations that are truly strong. As shown with this study's results, none of the chosen measures were strongly correlated, which is consistent with the stakeholders' opinions on the urgency of the problems that the measures indicate (based on examples in their own production code). However, they did agree that the examples were at least minor problems, which is consistent with the fact that the correlation results show a minor relation. Of course, these results may very well be skewed by the small sample size (25 commits over approximately three months). The correlation result between modified LoC and effort shows the commits are in fact not that different however, so the sample size may have been large enough to make case-specific inferences.

The prioritization process could possibly be refined further in future studies, by evaluating the TD levels of the previous state of modified functions rather than the previous state of modified files. It would also be valuable to conduct a comparison study that uses modified LoC per hour and effort surrogates from previous studies in order to establish which of these interest indicators is the most suitable.

In summary, the findings of this study will aid others to identify, quantify, present and prioritize Source Code TD in C/C++ applications. The design of the measurement system and the process for prioritizing TD types through correlation strength can also be applied to any programming language that TD can be measured for. By implementing the designed measurement system based contextual needs, files with high levels of multiple TD types will be identified which can then be taken into account during TD Management. This will in turn improve decision making regarding when and where to conduct refactoring.

References

[1] J. Grundèn and B. Lexell, 'Finding Architectural Debt in Historical Data', Master of Science Thesis Software Engineering, Chalmers University of Technology | University of Gothenburg, 2014.

[2] A. Martini, J. Bosch and M. Chaudron, 'Architecture Technical Debt: Understanding Causes and a Qualitative Model', in Software Engineering and Advanced Applications (SEAA), 2014 40th EUROMICRO Conference on, Verona, 2014, pp. 85-92.

[3] N. Zazworka, A. Vetro', C. Izurieta, S. Wong, Y. Cai, C. Seaman and F. Shull, 'Comparing four approaches for technical debt identification', Software Qual J, vol. 22, no. 3, pp. 403-426, 2013.

[4] P. Runeson and M. Höst, 'Guidelines for conducting and reporting case study research in software engineering', Empirical Software Engineering, vol. 14, no. 2, pp. 131-164, 2009.

[5] A. Nugroho, J. Visser and T. Kuipers, 'An Empirical Model of Technical Debt and Interest', in Proceedings of the 2nd Workshop on Managing Technical Debt (MTD '11), Waikiki, Honolulu, USA, 2011, pp. 1-8.

[6] V. Antinyan, M. Staron, W. Meding, P. Osterstrom, E. Wikstrom, J. Wranker, A. Henriksson and J. Hansson, 'Identifying Risky Areas of Software Code in Agile/Lean Software Development: An Industrial Experience Report', in Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE), 2014 Software Evolution Week - IEEE Conference on, Antwerp, Belgium, 2014, pp. 154-163.

[7] N. Zazworka, R. Spínola, A. Vetro', F. Shull and C. Seaman, 'A Case Study on Effectively Identifying Technical Debt', in Proceedings of the 17th International Conference on Evaluation and Assessment in Software Engineering (EASE'13), Porto de Galinhas, PE, Brazil, 2013, pp. 42-47.

[8] P. Kruchten, R. Nord and I. Ozkaya, 'Technical Debt: From Metaphor to Theory and Practice', IEEE Software., vol. 29, no. 6, pp. 18-21, 2012.

[9] Z. Li, P. Avgeriou and P. Liang, 'A systematic mapping study on technical debt and its management', *Journal of Systems and Software*, vol. 101, pp. 193-220, 2015.

[10] D. Rapu, S. Ducasse, T. Girba and R. Marinescu, 'Using history information to improve design flaws detection', in *Software Maintenance and Reengineering, 2004. CSMR 2004. Proceedings. Eighth European Conference on*, Tampere, Finland, 2004, pp. 223-232.

[11] D. Coleman, D. Ash, B. Lowther and P. Oman, 'Using metrics to evaluate software system maintainability', *Computer*, vol. 27, no. 8, pp. 44-49, 1994.

[12] M. Rieger, S. Ducasse and M. Lanza, 'Insights into system-wide code duplication', in *Reverse Engineering, 2004. Proceedings. 11th Working Conference on*, Washington, DC, USA, 2004, pp. 100-109.

[13] I. Heitlager, T. Kuipers and J. Visser, 'A Practical Model for Measuring Maintainability', in *Quality of Information and Communications Technology, 2007. QUATIC 2007. 6th International Conference on the*, Lisbon, Portugal, 2007, pp. 30-39.

[14] M. D'Ambros, A. Bachelli and M. Lanza, 'On the Impact of Design Flaws on Software Defects', in *Quality Software (QSIC), 2010 10th International Conference on*, Zhangjiajie, China, 2010, pp. 23-31.

[15] Iso.org, 'ISO/IEC 15939:2007 - Systems and software engineering – Measurement process', 2015. [Online]. Available: http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=44344. [Accessed: 22- Mar- 2015].

[16] C. Izurieta and J. Bieman, 'How Software Designs Decay: A Pilot Study of Pattern Evolution', in *Empirical Software Engineering and Measurement, 2007. ESEM 2007. First International Symposium on*, Madrid. Spain, 2007, pp. 449-451.

[17] C. Izurieta and J. Bieman, 'Testing Consequences of Grime Buildup in Object Oriented Design Patterns', in *Software Testing, Verification, and Validation, 2008 1st International Conference on*, Lillehammer, Norway, 2008, pp. 171-179.

[18] R. Wilcox, *Modern statistics for the social and behavioral sciences.* Boca Raton: Taylor & Francis, 2012.

[19] Verifysoft.com, 'Verifysoft → Halstead Metrics', 2015. [Online]. Available: http://www.verifysoft.com/en_halstead_metrics.html. [Accessed: 03- May-2015].

[20] M. Staron, W. Meding and C. Nilsson, 'A framework for developing measurement systems and its industrial evaluation', *Information and Software Technology*, vol. 51, no. 4, pp. 721-737, 2008.

[21] M. Staron and W. Meding, 'Using Models to Develop Measurement Systems: A Method and Its Industrial Use', in *IWSM '09 /Mensura '09 Proceedings of the International Conferences on Software Process and Product Measurement*, Amsterdam, 2009, pp. 212 - 226.

[22] M. Staron, W. Meding, G. Karlsson and C. Nilsson, 'Developing measurement systems: an industrial case study', *J. Softw. Maint. Evol.: Res. Pract.*, vol. 23, no. 2, pp. 89-107, 2011.

[23] M. Staron and W. Meding, 'Ensuring Reliability of Information Provided by Measurement Systems', in *IWSM '09 /Mensura '09 Proceedings of the International Conferences on Software Process and Product Measurement*, Amsterdam, 2009, pp. 1-16.

[24] I. Sommerville, *Software engineering*, 8th ed. Harlow, England: Addison-Wesley, 2007.

[25] Radford.edu, 2015. [Online]. Available: http://www.radford.edu/~jaspelme/statsbook/Chapter%20files/Table_of_Critical_Values_for_r.pdf[Accessed: 18- May- 2015].

[26] R. Wieringa and M. Daneva, 'Six strategies for generalizing software engineering theories', *Science of Computer Programming*, vol. 101, pp. 136-152, 2015.

Appendix A - Results for One StDev from 20% Trimmed

Table 6: Normal or one StDev below 20% Trimmed mean table

TD Type	Pearsons	Conditional Probability commit|TD & TD|commit	Chance Agreement	Cohen's Kappa	Correlation strength commit|TD (x/3)	Correlation strength TD|commit (x/3)
McCabe	-0.36	15.38% & 33.33%	49%	0.25	0/3	0/3
Halstead	-0.47	16.67% & 50%	39%	0.19	0/3	0/3
Issues	-0.48	22.22% & 33.33%	57%	0.35	0/3	0/3
Duplication	-0.29	21.05% & 66.67%	36%	0.23	0/3	1/3
High Dep Count	-0.37	16.67% & 33.33%	51%	0.28	0/3	0/3
Modularity Violations	-0.09	9.1% & 50%	30%	0.16	0/3	0/3
High Total TD	-0.39	0% & 0%	74%	0.40	0/3	0/3
Modified LOC / Effort	0.98	33.33% & 33.33%	79%	0.53	1/3	1/3

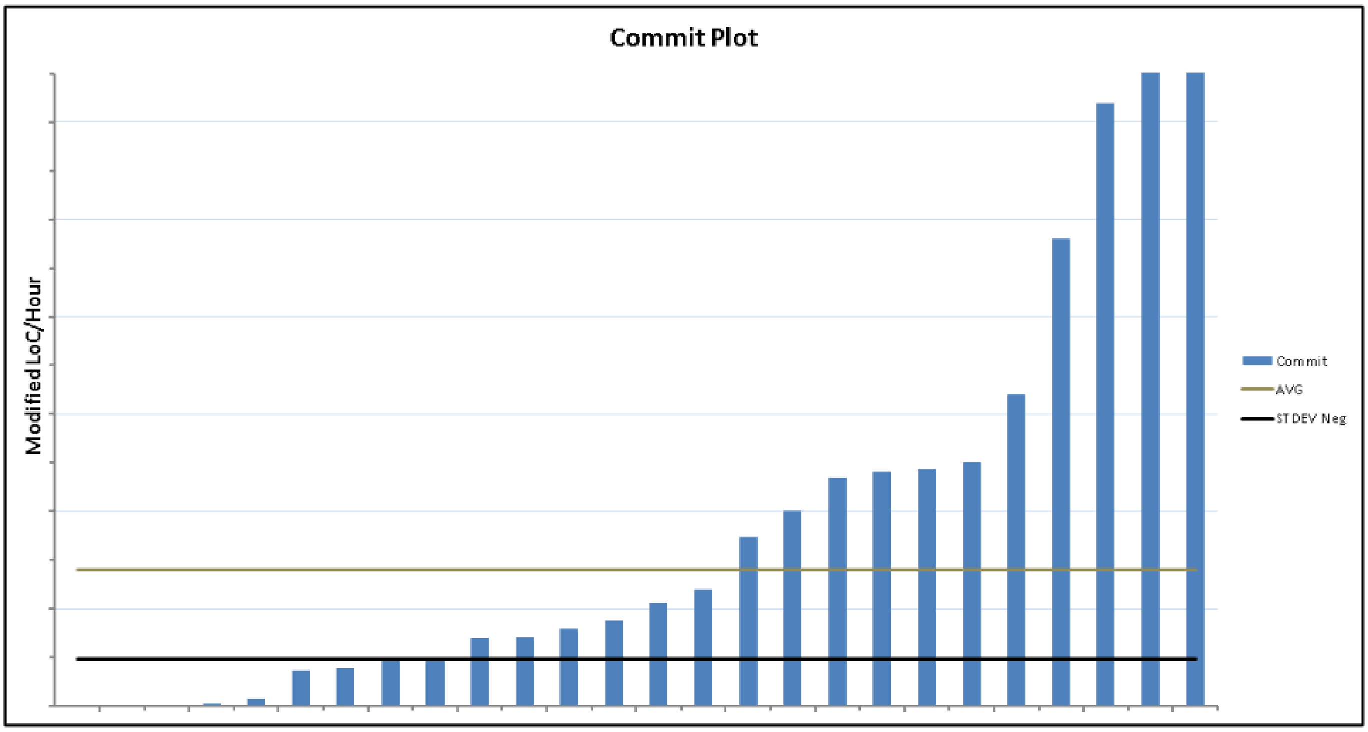

Figure 7: Modified LoC per hour ratio for each commit 20% Trimmed

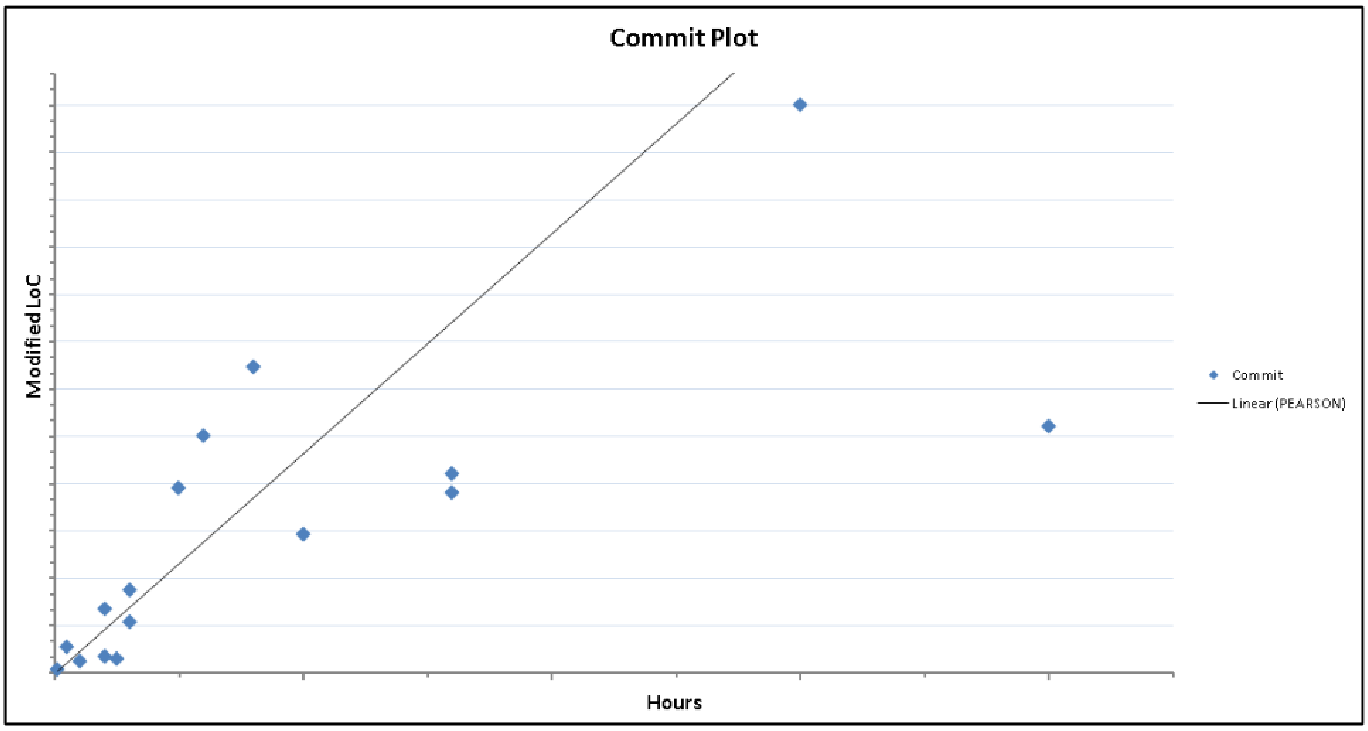

Figure 8: Chart showing distribution of commit modified lines/commit effort pairs 20% Trimmed

Appendix B - TD Questionnaire with Definitions List

Technical Debt Definition

Technical debt (TD) is a metaphor reflecting technical compromises that yield short-term benefits with a risk of causing long-term side-effects. Neglecting TD on actively developed software can cause inefficiency and expansion difficulties in the system, and such overhead cost is referred to as the interest of TD. If TD is not properly managed, the growth of TD over time will result in the growth of the interest, which will increase the required effort to maintain the software (which includes extending it with new features). However, repayment of TD also requires effort (referred to as the principal of TD). Therefore, it is important to decide when to pay the principal during TD management.

Out of the TD types (see glossary), our study focuses on Code TD, while also touching upon Architecture TD through modularity violations as well as Defect TD by evaluating issue categories not covered by Lint. If we manage to find correlations between commits that needed extra effort and certain subcategories of these TD areas, then that would also be an indicator of Infrastructure TD (lack of support).

Regarding TD management activities (see glossary), our study focuses mainly on TD Identification (which is required for TD Monitoring, Communication & Documentation). By correlating TD measures to effort, we will also be able to perform TD Prioritization.

As a complementary source of data for TD identification, we would very much like you to answer the following questions. Your answers may lead to further discussion, and they may also point out TD in the application that our current tools do not cover.

Questions

1. If you were forced to address TD in the application to make it more maintainable for the future (i.e. excluding new features or bug fixes), which part of the code would you spend your time on? Why?

2. Which part of the application, that you think contains TD, will not be changed in the foreseeable future?

3. What kind of “code compromise” would you personally like to keep track of, or find out about earlier?

4. What do you consider to be extra time consuming when working on/with the application?

5. What do you consider to be “poor” code that can still be found within the application? Where can it be found?

Questionnaire Glossary

Technical Debt Types

TD can be split into ten areas based on the origin of the compromise [9]:

Requirements TD
Refers to differences between an existing requirements specification and the actual product, under domain assumptions and constraints.

Documentation TD
Refers to insufficient, incomplete, or outdated documentation.

Test TD
Refers to lack of tests/test coverage.

Build TD
Refers to flaws in a software system, in its build system, or in its build process that make the build overly complex and difficult.

Versioning TD
Refers to problems in source code versioning, such as unnecessary code forks.

Infrastructure TD
Refers to sub-optimal configurations of development-related processes, technologies and supporting tools.

Architectural TD
Refers to architecture decisions that make compromises in certain internal quality aspects, such as maintainability.

Design TD
Refers to technical shortcuts taken during detailed design.

Code TD
Refers to code that violates general or context specific coding practices/coding rules.

Defect TD
Refers to defects, bugs, or failures found in software systems.

Not TD
Things that are not TD include logical correctness as well as runtime properties such as performance.

Technical Debt Management

Technical debt management (TDM) is divided into activities centered on either dealing with existing TD or preventing potential future TD [9]. These activities include:

TD Identification

Deals with detecting TD caused by intentional or unintentional technical decisions in a software system through specific techniques, such as static code analysis.

TD Measurement

Deals with quantifying the benefit and cost of known TD through estimation techniques, or estimates the level of the overall TD in a system.

TD Prioritization

Deals with ranking identified TD according to certain predefined rules to support deciding which TD items should be repaid first and which TD items can be tolerated until later.

TD Prevention

Deals with preventing potential TD from being incurred.

TD Monitoring

Deals with tracking the changes of the cost and benefit of unresolved TD over time.

TD Repayment

Deals with resolving TD through techniques such as reengineering and refactoring.

TD Communication

Deals with making identified TD visible to stakeholders so that it can be discussed and further managed.

TD Documentation

Deals with representing TD in a uniform manner, addressing the concerns of particular stakeholders.

Appendix C - TD & Correlation Validation Questions

Asked after explaining TD type concepts

- Do you have any questions regarding the TD types we have explained?
- Which of the TD types would you consider important/not important just based on our definitions?

Tying summarized results to recent challenges

- Can you describe any time-consuming problems encountered by you or anyone else from the team during the last two weeks?
- Do any of our summary results for the time-consuming file(s), highlight the function(s) that caused extra work?

After showing detailed tool results and examples in the production code

- Were any of these results unexpected to you (concerning the code areas they point out)?
- How would you reprioritize the TD types after seeing the tool results and our selected examples?
- Which (if any) of our selected examples would you consider refactoring first?

Asked after showing correlation results

- What do you think of relative importance between TD types according to the correlation results?

Assessing tool integratability after detailing result summarizing process

- Based on our explanations of the tools and our result summarizing process, how well do they fulfill your integratability requirements?

Appendix D - Tool Licence List

- **GPLv2** = http://www.gnu.org/licenses/gpl-2.0.html
- **GPLv3** = http://www.gnu.org/copyleft/gpl.html
- **LGPLv3** = http://www.sonarqube.org/downloads/license/
- **ALv2** = http://www.apache.org/licenses/LICENSE-2.0
- **BSD-4** = https://spdx.org/licenses/BSD-4-Clause
- **BSD-3** = http://docs.oclint.org/en/dev/devel/license.html
- **VPL** = http://source.valtech.com/display/dpm/License
- **EPL** = https://www.eclipse.org/legal/epl-v10.html
- **USC** = http://csse.usc.edu/csse/affiliate/private/license.txt
- **MIT** = http://opensource.org/licenses/MIT