

Value/Cost Analysis of Modularity Improvements

Citation for published version (APA):

Rawshan, L. (2022). *Value/Cost Analysis of Modularity Improvements*. Technische Universiteit Eindhoven.

Document status and date:

Published: 04/10/2022

Document Version:

Publisher's PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:

- A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
- The final author version and the galley proof are versions of the publication after peer review.
- The final published version features the final layout of the paper including the volume, issue and page numbers.

[Link to publication](#)

General rights

Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the "Taverne" license above, please follow below link for the End User Agreement:

www.tue.nl/taverne

Take down policy

If you believe that this document breaches copyright please contact us at:

openaccess@tue.nl

providing details and we will investigate your claim.



EngD THESIS REPORT

Value/Cost Analysis of Modularity Improvements

Lamisha Rawshan

September 2022

Department of Mathematics & Computer Science

EngD SOFTWARE TECHNOLOGY

Value/Cost Analysis of Modularity Improvements

Lamisha Rawshan

October 2022

Eindhoven University of Technology
Stan Ackermans Institute – Software Technology

EngD Report: 2022/067

Confidentiality Status:
Public

Partners

The ASML logo consists of the letters 'ASML' in a bold, blue, sans-serif font.

ASML Netherlands B.V.

The TU/e logo features the letters 'TU/e' in a large, red, sans-serif font, followed by the text 'EINDHOVEN UNIVERSITY OF TECHNOLOGY' in a smaller, red, sans-serif font stacked to the right.

Eindhoven University of Technology

Steering Group

Tom Verhoeff
Remko van der Vossen
William van Houtum
Yanja Dajsuren

Date

October 2022

Composition of the Thesis Evaluation Committee:

Chair: Prof.dr. M.G.J. van den Brand

Members: Dr.ir. Tom Verhoeff

Ir. Remko van der Vossen

Ing. William van Houtum

Dr.ir. Martijn van der Horst

Dr. Kees Huizing

The design that is described in this report has been carried out in accordance with the rules of the TU/e Code of Scientific Conduct.

Contact Address	Eindhoven University of Technology Department of Mathematics and Computer Science MF 5.072, P.O. Box 513, NL-5600 MB, Eindhoven, The Netherlands +31 402743908
Partnership	This project was supported by Eindhoven University of Technology and ASML
Published by	Eindhoven University of Technology
EngD-report	2022/067
Preferred reference	L. Rawshan, <u>Value/Cost Analysis of Modularity Improvements</u> . Eindhoven University of Technology, EngD Report, 2022/67, October 2022
Abstract	ASML's lithography machines are used in the process of manufacturing integrated circuits. These machines are driven by complex software components. The TWINSCAN software architects want to make the software more modular to improve the maintainability and extendibility of source code. A good refactoring plan is needed to improve the modularity. But existing tools are not sufficient to provide enough decisions to begin refactoring. The project's main goal is to develop a tool called MoVACA that will produce useful output for the TWINSCAN software architects and developers for making refactoring decisions to improve modularity. This report describes the motivation and process of developing MoVACA.
Keywords	EngD, TU Eindhoven, ASML, Modular Design, Software Dependencies, Technical Debt, Data-Visualization
Disclaimer Endorsement	Reference herein to any specific commercial products, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the Eindhoven University of Technology and ASML. The views and opinions of authors expressed herein do not necessarily state or reflect those of the Eindhoven University of Technology and ASML and shall not be used for advertising or product endorsement purposes.
Disclaimer Liability	While every effort will be made to ensure that the information contained within this report is accurate and up to date, Eindhoven University of Technology makes no warranty, representation or undertaking whether expressed or implied, nor does it assume any legal liability, whether direct or indirect, or responsibility for the accuracy, completeness, or usefulness of any information.
Trademarks	Product and company names mentioned herein may be trademarks and/or service marks of their respective owners. We use these names without any particular endorsement or with the intent to infringe the copyright of the respective owners.
Copyright	Copyright © 2022. Eindhoven University of Technology. All rights reserved. No part of the material protected by this copyright notice may be reproduced, modified, or redistributed in any form or by any means, electronic or mechanical, including photocopying, recording, or by any information storage or retrieval system, without the prior written permission of the Eindhoven University of Technology and ASML.

Foreword

Ever since the start of ASML in 1984, we have developed embedded software, software to be used in real-time, running highly automated equipment, our lithography machines. The embedded software supports the lithography machine in delivering speed and quality in producing chips with ever-increasing performance.

Software modularity has serious attention within ASML. Together with the increasing product portfolio and increasing demand in new features we need to keep our software maintainable. Software modularity is wanted for maintainable software, also we prefer commonality, share software modules over different product configurations. To get the features implemented many new hires start each month and more outsourcing is done, software modularity also plays an important role here.

With the assignment, we wanted to get more insights in which order to do software modularity improvements, also taking the effort on maintaining the modularity technical debt into account.

Due to the COVID situation, Lamisha has to do her assignment partly from home, in which it was harder to get the connection with ASML. As a project manager I joined the weekly project meetings in which she was well-prepared, so we could efficiently do the meetings remotely. It was good to see how she approached modularity, starting from literature, selecting metrics, and visualizing them in a tool.

The assignment gave ASML some good insights how to approach and depict modularity.

I am glad that Lamisha has decided to start at ASML's Metrology from the beginning of 2023.

Thanks, Lamisha!

Veldhoven, Sept 21 2022
William van Houtum
Software Platform Architect
TWINSCAN - Modularity at ASML

Preface

This report serves as the documentation for the development process and product of the project carried out by me as the partial fulfillment of the requirements for the Engineering Doctorate (EngD) in the Software Technology program. EngD is a two-year doctorate-level program provided by the Eindhoven University of Technology under the banner of 4TU.School for Technological Design, Stan Ackermans Institute.

This project took place in the Software Platform Architecture group of ASML Netherlands B.V. The goal of this project is to develop a tool that analyzes the source code of the TWINSCAN software and generates tabular and graphical reports for four metrics that relate to the modularity of source code and how this modularity evolves over time.

This report is primarily intended for readers with a technical background interested in modularity and software architecture. Nevertheless, this report may also interest the audience with a non-technical background. Readers who are interested in the non-technical parts of the report should read the first four chapters. Those who are interested in technical details should read Chapters 5 to 7. The project management methodology applied to this project is presented in Chapter 8. Chapter 10 presents a retrospective of the project, reflecting upon the entire project experience. It also revisited the design opportunities identified in the early phases of the project.

Lamisha Rawshan
September 2022

Acknowledgements

The last ten months at ASML Netherlands B.V. have been simply incredible. This project would not have been possible without the support of several people. To begin with, I would like to thank William van Houtum for giving me this opportunity. I want to thank him for his continuous support throughout the project.

I want to thank Remko van der Vossen, my supervisor at ASML, for his continuous support, knowledge, guidance, and expertise that I needed during the project, without which I could not have achieved the results and deliverables. Apart from my supervisors from ASML, I would like to thank Bram Schoenmakers for his assistance as needed.

I want to thank my university supervisor, Tom Verhoeff, for his constructive feedback and encouragement throughout the project. His critical thinking and ideas encouraged me to explore the project from a different view. I sincerely appreciate his exhaustive review of this report. Without his continuous assistance, this project would not have been a success.

I want to express my gratitude to Yanja Dajsuren for the opportunity to be part of the EngD program. I want to thank Yanja, Desiree van Oorschot, and all of the coaches for their encouragement and support over the last two years, which has allowed me to grow professionally and personally. In addition, I would like to thank my colleagues, the Software Technology Trainees of 2020, for our two years of teamwork, support, and friendship.

Lastly, I would like to thank my parents, whose love and guidance are with me in whatever I pursue. Most importantly, I am grateful to my husband for his continuous support, patience, and inspiration. I would also like to thank my daughter for providing me with a source of strength, happiness, and drive to succeed.

Lamisha Rawshan
September 2022

Executive Summary

ASML is the global leader in providing photolithography systems for the semi-conductors' industry to produce integrated circuits. The TWINSCAN software controls the lithography machine. The software has a large code base with millions of lines written in several programming languages. As the software grows, the architects encounter many unwanted dependencies, reducing the code's easy maintainability and extendibility. Hence, the architects think the current codebase's modularity needs to be improved. Technical debt needs to be identified to improve modularity. Ward Cunningham [1] first used the term technical debt in 1992. It describes how code quality is sacrificed to satisfy short-term goals. The result of the technical debt is code that is more expensive to maintain than usual.

This project aims to identify the technical debt at the architectural level. In addition, there should be a rank to prioritize the debt based on maintenance costs. Therefore, tool support is needed to visualize and prioritize the debt of the TWINSCAN software to help the architects.

To address the goal of the project, we researched the existing modularity tool and selected four metrics (hotspot, change coupling, complexity trends, and hotspot rank) from the book "Software Design X-ray" [2] based on the data available. Applying these four metrics helps to show the technical debt of the system using metadata of the source code (such as change frequencies, lines of code, and release data) from the version control system. We developed a tool, MoVACA, to present these four metrics in graphical and tabular format for user-specified scopes. We verified and validated the design and implementation of the tool against the requirements. To enable the extendibility of the tool, we designed the tool in a way that a new TWINSCAN-specific metric can be introduced.

Finally, we delivered the project successfully. The tool provided significant output to identify and prioritize the technical debt of the TWINSCAN software. We discovered some future opportunities during the development of the tool. Using the issue tracker system data will add more meaningful results to the tool and can be served as future work.

Table of Contents

Foreword.....	i
Preface.....	iii
Acknowledgements	v
Executive Summary	vii
Table of Contents	ix
List of Figures.....	xi
List of Tables	xii
1. Introduction	1
1.1 ASML.....	1
1.2 TWINSCAN.....	1
1.3 Project Context.....	1
1.4 Report Organization.....	2
2. Domain Analysis	3
2.1 TWINSCAN Software Organization	3
2.2 Dependencies in TWINSCAN	4
2.3 Modularity in TWINSCAN.....	4
2.4 Modularity Assessment Tool.....	4
3. Problem Analysis	6
3.1 Project Scope and Goal.....	6
3.2 Modularity Cost Analysis	6
3.2.1. Hotspot.....	7
3.2.2. Complexity Trends	7
3.2.3. Change Coupling	8
3.2.4. Hotspot Rank	9
4. Requirements Elicitation.....	10
4.1 Introduction.....	10
4.2 Use Case.....	10
4.3 MoSCoW.....	10
4.3.1. Project Requirements.....	11
4.3.2. Product Requirement	12
5. System Architecture and Design	15
5.1 High-Level Architecture	15
5.2 The 4+1 View Model of Architecture	16
5.3 Logical View.....	16

5.4	<i>Process View</i>	19
5.5	<i>Development View</i>	20
5.6	<i>Deployment View</i>	21
6.	Implementation	23
6.1	<i>Overview</i>	23
6.2	<i>Technology Choice</i>	23
6.3	<i>Modularity Cost Analysis Tool</i>	24
7.	Verification & Validation	27
7.1	<i>Validation</i>	27
7.2	<i>Verification</i>	27
7.2.1.	<i>Unit test</i>	27
7.2.2.	<i>Code coverage</i>	28
7.2.3.	<i>Coding convention</i>	29
8.	Project Management	31
8.1	<i>Work-Breakdown Structure (WBS)</i>	31
8.2	<i>Project Planning and Scheduling</i>	31
8.3	<i>Project Risk Analysis</i>	32
8.4	<i>Communication</i>	33
8.4.1.	<i>Weekly update meetings</i>	34
8.4.2.	<i>Project steering group meetings</i>	34
8.4.3.	<i>On-demand meetings</i>	34
9.	Conclusions	35
9.1	<i>Results</i>	35
9.2	<i>Recommendations and Future Work</i>	35
9.2.1.	<i>Future Work</i>	35
9.2.2.	<i>Recommendation</i>	36
10.	Project Retrospective	37
10.1	<i>Introduction</i>	37
	Glossary	39
	Bibliography	41
A.	Stakeholder Analysis	43
A.1	<i>Stakeholder Identification</i>	43
A.2	<i>Stakeholder Interests</i>	43
B.	TWINSKAN Software FCs Dependencies	45
	About the Author	46

List of Figures

Figure 1 TWINSCAN software organization	3
Figure 2 Formula of calculating modularity score	5
Figure 3 Hotspot diagram	7
Figure 4 Compelxity trends	8
Figure 5 Change coupling diagram.....	8
Figure 6 Use case diagram of MoVACA.....	10
Figure 7 System context diagram.....	15
Figure 8 Class diagram metric calculator	17
Figure 9 Class diagram metric calculator	17
Figure 10 Class diagram data formatter	18
Figure 11 Class diagram for change coupling	18
Figure 12 Sequence diagram for admin user	19
Figure 13 Sequence diagram for TWINSCAN developers/architects	20
Figure 14 System component diagram.....	21
Figure 15 System deployment diagram	22
Figure 16 Modularity cost analysis tool main page	24
Figure 17 Packages of MoVACA	25
Figure 18 Code snippet of coupling generator class	25
Figure 19 Algorithm for counting coupling	26
Figure 20 Log file	26
Figure 21 Test cases for the Data Formatter component	28
Figure 22 Code coverage result of MoVACA	28
Figure 23 UML class diagram for extending a new metric	30
Figure 24 Work-breakdown structure	31
Figure 25 Project timeline.....	32
Figure 26 Stakeholder analysis matrix.....	43
Figure 27 Essential FC dependencies of production.....	45

List of Tables

Table 1 TWINSCAN modularity KPI	5
Table 2 Project requirements	11
Table 3 Product requirements	12
Table 4 Existing sample test cases for MoVACA	27
Table 5 Project risks.....	32
Table 6 Main stakeholders' interest and involvement	44
Table 7 Additional stakeholders' interest and involvement.....	44

1. Introduction

This chapter provides a general introduction to the project by describing the context and domain in which it was carried out. The chapter concludes with an overview of the structure of this project.

1.1 ASML

In 1984, Philips and Advanced Semiconductor Materials International (ASMI) founded ASM Lithography (ASML) [3]. The vision of ASML is a world where affordable microelectronics improve the quality of life. ASML strives to achieve this vision by designing, developing, integrating, marketing, and servicing advanced systems for the semiconductor industry.

ASML is the world leader in Lithography machines, and these are complex machines used for the most critical step in semiconductor manufacturing (such as memory chips and processors). Some of the major customers include Intel, TSMC, and Samsung. ASML is headquartered in Veldhoven, The Netherlands, and is spread across various locations over the world. The company manufactures three types of lithography systems [4].

- Ultra-Violet (UV)
- Deep Ultra-Violet (DUV)
- Extreme Ultra-Violet (EUV)

Software is a key component of ASML's lithographic machines. Therefore, the lithography machines combine high-tech hardware and advanced software. The software contributes to innovations and improvements in the process of chip production.

1.2 TWINSCAN

The lithography systems are sophisticated technologies used to manufacture integrated circuits and microchips. Lithography refers to a series of procedures in producing integrated circuits in which one of the circuit layouts is scanned (printed) onto a microchip. The lithography method is enabled by ASML TWINSCAN machines, which expose light to the wafer and create structures.

The TWINSCAN machines must operate extremely precisely to ensure the chips function correctly. As a result, ASML machines rely primarily on complicated software split into functional clusters, each of which has one or more components. Each functional cluster's components are developed and maintained by a separate team of software developers.

1.3 Project Context

The project focuses on TWINSCAN software, running on the Lithography Scanners. TWINSCAN software has grown rapidly over the past years. This complex software consists of several tens of millions of lines of code [5]. Updates, bug fixes, and new functionalities are continuously developed and tested before being released to the customer.

TWINSCAN software architects know there are a lot of unwanted dependencies between the modules. As the software has grown, the software architects have realized that they are introducing technical debt to fulfill deadlines. Technical debt is a term used to describe work in a software system that makes tradeoffs in the long term to meet urgent short-term needs [6].

First, technical debt needs to be addressed to remove it. The software architects have already taken the initiative to identify technical debt that will help TWINSCAN software developers, software architects, and managers to know the status of the code. They are migrating from the legacy to the modular software archive [7]. To help better refactoring, ASML is using different kinds of tooling. One of the tools is known as the Modularity Index Tooling (MDI) [8], which provides the modularity score and dependencies between the modules.

To improve modularizing the large code base, the architects need to estimate the budget as the refactoring costs are concrete and immediate. In contrast, the benefits of refactoring are long-term [9]. Therefore, it is difficult for the architects to make correct decisions regarding refactoring needs that cost time and effort. To realize such a large plan, software developers and architects require a tool to make informed decisions on which modularity improvements to prioritize.

The project aims to develop a tool that I will refer to as MoVACA to measure the technical debt to uncover limitations at the architecture level (e.g., violations of the modularity principle) and rank the identified debt based on their impact and the amount of work needed to fix them.

1.4 Report Organization

This report aims to provide a thorough technical overview of this ten-month project. Chapter 1 is a generic introduction to the project, the company, and the problem. Chapter 2 focuses on domain analysis, while Chapter 3 provides a more in-depth problem analysis. Chapter 4 contains the result of the requirement analysis.

Then, Chapter 5 describes the design architecture for modeling the tool. Chapter 6 discusses the implementation of the tool. The process of validation and verification of the new design is discussed in Chapter 7.

Finally, Chapter 8 presents the project management process, including project planning and risk management. Chapter 9 contains the conclusions and future recommendations. Chapter 10 concludes the report with a project retrospective. Appendix A discusses the stakeholders' interests and goals.

2.Domain Analysis

Before going into the problem analysis, it is essential to know where the problem is located. This chapter aims to provide an overview of the project domain. This chapter describes the organization (Section 2.1) and the modularity of the TWINSCAN software (Section 2.3). Finally, it represents the existing modularity assessment technique (Section 2.4) related to this project.

2.1 TWINSCAN Software Organization

To manage the rising complexity of the machines' hardware and software, ASML uses a divide-and-conquer technique. A system is divided into smaller subsystems utilizing this technique, which are small enough to be designed, tested, and delivered individually by a group of engineers before being assembled.

- **System Functions (SFs):** The TWINSCAN machine's functionality is decomposed into several SFs.
- **Functional Clusters (FCs):** Each SF has one or more FCs, either logical system functions or support facilities for system functions.
- **Building Blocks (BBs):** An FC is divided into one or more BBs, known as the unit of change management.
- **Software Components (CCs):** Each BB is also subdivided into ASML CCs, which are atomic software design units assigned to development teams. A CC can correspond with a specific part of the machine or play a general-purpose role (e.g., Error Logger). A component's source code and configuration parameters are unique to it.

Figure 1 depicts the machine's software breakdown. This architectural decomposition makes the testing and integrating software that runs on the computer easier.

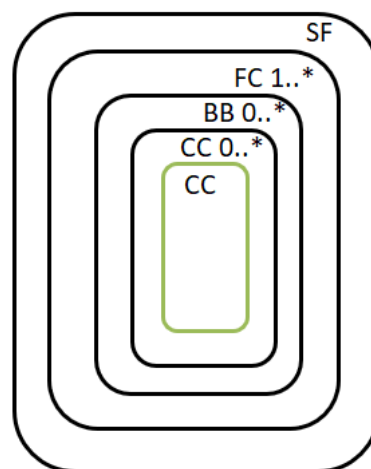


Figure 1 TWINSCAN software organization

2.2 *Dependencies in TWINSCAN*

Within the ASML software organization, interfaces exposed by one FC also become interfaces exposed to other FCs. That causes interdependencies between the FCs. There are two types of dependencies for the software.

- **Desired dependencies:** These are the software's essential dependencies. Appendix B contains the essential FC dependencies of the software.
- **Undesired dependencies:** These includes dependencies that are in the source code but not in the desired architecture. These dependencies are mainly introduced to fulfill the short-term goal.

2.3 *Modularity in TWINSCAN*

Modularity, reusability, analyzability, modifiability, and testability are all aspects of maintainability in ISO25010:2011 software quality standard [10]. Modularity is the degree to which a system or computer program is made of separate components, so changing one has minimal impact on others [11]. Modifying one module can have a cascading effect, causing other modules in the system to stop working correctly. ASML priorities for modular software are listed below:

1. Reuse of software across products and releases
2. Independent development of modules' lifecycle
3. IP Protection
4. Reduce regression risk for customers
5. Increase in innovation speed of tools

TWINSCAN software is developed in mainly [8] C, C++, and Python. TWINSCAN software contains a complex structure that has many undesired dependencies. The main motivation for modularity in TWINSCAN software is to decompose the software into modules or assemblies with clear and stable interfaces and minimum dependencies to do independent development and verification

An assembly is a collection of building blocks with its release cycle and branches. Modularity in TWINSCAN software is improved by introducing Distributed Assembly Integration (DAI). DAI introduces centralized to distributed assembly archives that allow only essential coupling with other assemblies. TWINSCAN engineers can reuse modules with an independent life cycle across releases.

2.4 *Modularity Assessment Tool*

Improving the software's modularity will provide independently evolving modules, efficient reuse, and maintainability. Researchers have used many ways to assess the modularity of software systems over the years [12]. They used the modules' degree of coherence and connectivity to determine modularity.

TWINSCAN software engineers developed MDI to assess the independence of current modules in 2017. The goal of assessing the modularity of the software is to know how well the containers are modularized.

TWINSCAN software uses its own modularity Key Performance Indicators (KPI) [13] to assess the modularity of its software. The KPIs are listed in Table 1. The locality of change KPI has more impact than the other KPIs. For each KPI in the table, a minimum and a maximum value are (manually) established. Each result is normalized to a value from 0 (worst score) to 10 (best score).

Table 1 TWINSCAN modularity KPI

Property	Metric	Weight
Interface Quality	Change frequency provided interfaces	15%
	Change frequency required interfaces	15%
Coupling	provided + required symbols	15%
	Deviation from referential module Dependencies	15%
Testability	Configuration space (prov. + req. parameters)	10%
Shareability	missing symbols in other releases	10%
Locality of Change	single module streams	20%

The modularity assessment tool generates a modularity score for each module by analyzing the release data. The modularity scores of the components are calculated using the formula in Figure 2. The modularity scores do not say what causes a low score. Therefore, more investigation is needed to know from where the refactoring can be started.

$$\text{Modularity Index of module } A = (\sum_{m \in \text{Metrics}} m(\text{Value}) * \text{weight}(m)) * C(A_{\text{size}})$$

Figure 2 Formula of calculating modularity score

3. Problem Analysis

The purpose of this chapter is to provide general information and an introduction to the problem this project explores. The project context is briefly discussed in the introduction. Section 3.1 addresses the scope of the project and the goals that need to be addressed, and Section 3.2 explains the metrics to be addressed for the project.

3.1 *Project Scope and Goal*

TWINSKAN software architects desire to modularize their software. TWINSKAN software platform architects proposed a modular reference architecture, discussed in Appendix A.

TWINSKAN software engineers monitor the health of the program code using TIOBE TICS [14] and other static code analysis tools (such as- PyLint). Among them, the Modularity Analysis Tool (Section 2.3) is used to manage the dependencies between the modules. At the component level, the tool indicates both desired and undesirable dependencies between modules, but to start refactoring, the architects need to address the dependencies inside and outside the module. Moreover, the existing tool does not prioritize undesirable dependencies. It is therefore challenging to determine which dependencies require more attention.

In TWINSKAN software, the undesired dependencies result in a steep increase of complexity, inefficient development, and slowing down tools running on it [15]. The code in TWINSKAN requires additional maintenance expenses. Technical debt refers to code that costs more to maintain than it should. The company does this by paying interest [16]. The TWINSKAN software architects are concerned with avoiding future technical debt.

To begin refactoring, the architects need a clearer picture of the system. The software architects believe that the health of the modularity should be measured and improved by a well-defined refactoring plan. Therefore, the software architects agreed that extra tool support is required for the software architects and developers.

The main company motivation for this project is to have a system that gives a complete picture of the code with maintenance problems in the system and gives fast feedback to the software developers and architects of the deliveries on the architectural level. Although there are commercially available tools on the market that perform the generic static code analysis, none of them can detect code with high maintenance effort.

Moreover, during the project scope meetings with the company stakeholders, we found that the architects want to

- Measure the technical debt to uncover limitations at a user-specified level
- Generate a rank to prioritize modularity improvement
- Visualize technical debt using the diagram

However, the answer to the above points cannot be provided directly. Therefore, we decided to compute more concrete metrics discussed in Section 3.2.

3.2 *Modularity Cost Analysis*

Refactoring complex code is a risky and costly process. It must be ensured that the development time is utilized effectively. System change history over the dimension of time is needed to start improving a large amount of code. However, the time dimension is not present in the code itself. The version control

system allows us to access that system evolution data. We chose behavioral code analysis for this project to identify and prioritize technical debt using time dimension data from the version control system. To achieve our project goal, we chose the following four metrics from the book [2].

3.2.1. Hotspot

A hotspot is a complex piece of code that developers frequently work with [17]. Hotspot analysis is the suggested first step in exploring a codebase because it identifies the code where most of the development time is spent. Hotspots are determined by combining the followings:

- Estimating the frequency of changes to each file as a substitute for the interest rate
- Counting lines of code to determine the complexity of the code

Figure 3 shows an example of the hotspot diagram. The hotspot of a system is represented by a map where each filled circle represents a file. The circle's size and color vary with the number of lines of code and the change frequencies, respectively.

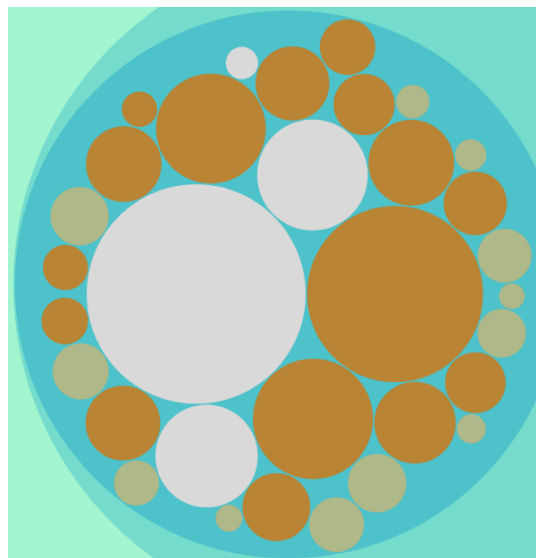


Figure 3 Hotspot diagram

3.2.2. Complexity Trends

We must evaluate the hotspots after identifying them. These hotspots are analyzed by the complexity trend [18]. The complexity trend defines how difficult it is for a human to understand the code. The trend is determined by retrieving each previous version of a hotspot and calculating the amount of code in each previous revision. The complexity (or lines of code for our analysis) and comments are used to calculate the complexity trends.

The complexity trends of a system are represented by a line graph where the two lines of the graph represent the number of changed lines and the comments for each version. An example of a complexity diagram is shown in Figure 4. The diagram shows the change of line and comments of version for three consecutive years.

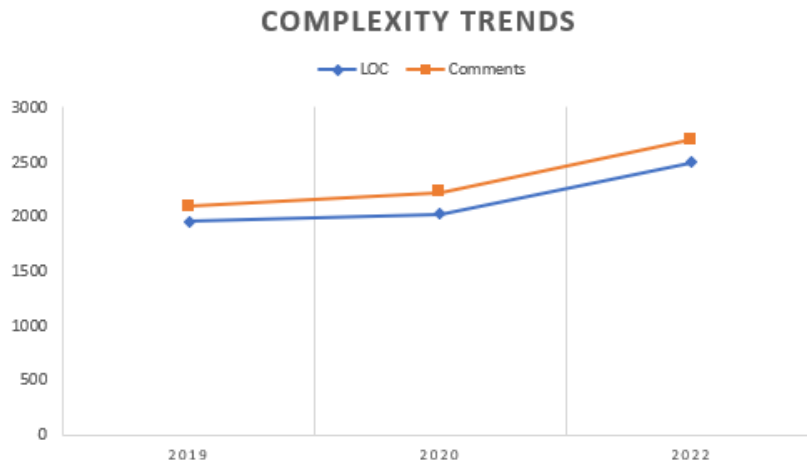


Figure 4 Complexity trends

3.2.3. Change Coupling

Change coupling refers to files changing together over time [19]. By examining the change patterns, change coupling can reveal hidden dependencies between groups of files. It provides an effective method for iteratively enhancing our system design based on feedback from how we work with the code. The change coupling is determined by analyzing release data to identify files that have changed within the same commit.

An example of the change coupling diagram is shown in Figure 5. The change coupling is represented by a chord diagram where each chord represents a change relation between two files. The chord size varies with the number of files that change together.

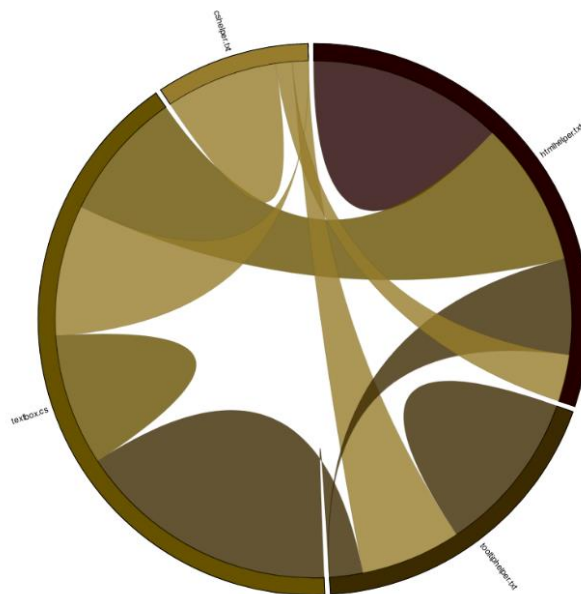


Figure 5 Change coupling diagram

3.2.4. Hotspot Rank

The hotspot rank indicates which files are quickly moving up the hotspot ranking [20]. To determine the hotspot rank, we need to conduct two calculations to find rising hotspots. A hotspot analysis based on the current state of the code and another hotspot analysis based on the previous state of the code is computed. The analysis will be illustrated using a tabular format where the first column contains the file name, followed by the rank of the two different time frames. It reveals the files that are rapidly changing over time.

In this chapter, the metrics that are needed to fulfill the stakeholders' needs are described. The concrete requirements to show the metrics are presented in Chapter 5.

4. Requirements Elicitation

4.1 Introduction

The goal of this chapter is to describe the requirements that have been identified for this project. The requirements are categorized into two major parts: project requirements and product requirements. The requirements were defined by analyzing the problem and discussing it with stakeholders. This chapter contains a primary use case and the requirements.

4.2 Use Case

A use case describes potential interactions between a system and its users. MoVACA has two primary categories of actors. TWINSCAN software architects and developers. In the context of this project, an architect or designer of the TWINSCAN software will use the tool to observe the metrics described in Section 3.2 at different levels. They will make decisions on refactoring to see the metrics. The version control actor is responsible for providing historical data for the analysis.

Figure 6 shows the potential actions that a user can perform. MoVACA has four primary use cases: the hotspot, the complexity trend, the change coupling diagram, and the rank of refactoring cost.

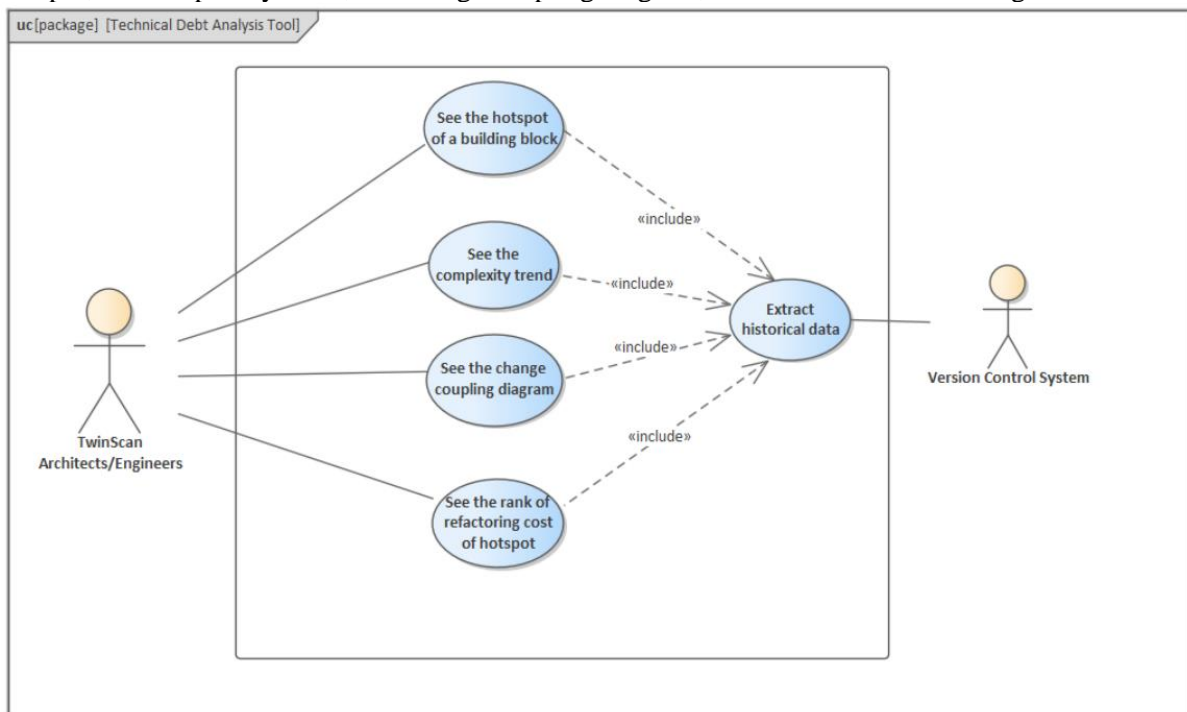


Figure 6 Use case diagram of MoVACA

4.3 MoSCoW

In this project, the MoSCoW [21] method is used to prioritize the elicited set of requirements. The word MoSCoW is an abbreviation of four terms, each of which defines different priorities. They are:

- Must have (M): The requirements under this category must be included in the final delivery.
- Should have (S): The requirements under this category are suggested to include in the project.

- Could have (C): The requirements under this category could be satisfied depending on the project's timeline
- Won't have(W): The requirements under this category will not be addressed in the project scope.

4.3.1. Project Requirements

This subsection lists the set of project requirements. The project requirements were developed by analyzing the domain, investigating the modularity tool, and studying existing literature. Table 2 illustrates the list of requirements obtained from the interests and priorities of stakeholders. Each project requirement is described in detail, including motivation and a verification method.

Table 2 Project requirements

ID	Description, Motivation, and Verification	Priority
Req-1	<p>Description The backend of the system shall be implemented in the Python (Python -3.7) programming language</p> <p>Motivation The backend is targeted to be implemented using OOP languages such as C++, Java, and Python. Python is the target for this project as the existing modularity tool is built using Python and is a standard at ASML</p> <p>Verification This requirement will be tested by a source code inspection and running the code using Python</p>	Must
Req-2	<p>Description The Python implementation of the tool shall follow the Python coding standard of ASML.</p> <p>Motivation The implementation shall follow the ASML coding standard to be consistent with the existing code</p> <p>Verification This requirement will be verified by using the TICS tool that checks the compliance of the code with the coding standard</p>	Must
Req-3	<p>Description The front end of the system shall be a web interface (supports Microsoft Edge, Google Chrome, Mozilla Firefox browser) to present the tool's features.</p> <p>Motivation The user does not have to install any additional software to use the tool.</p> <p>Verification This requirement will be verified by running the code.</p>	Must
Req-4	<p>Description The web interface of the implementation shall use JavaScript (Version-ES2015) to visualize generated diagrams.</p> <p>Motivation JavaScript is needed to add interactivity and data visualization to the web page.</p> <p>Verification</p>	Must

	This requirement will be tested by code inspection and running the source code.	
Req-5	<p>Description The implementation of the tool shall use Pytest automatically by Continuous Integration (CI) to write and execute test code.</p> <p>Motivation The developed tool will be tested to fulfill the requirements.</p> <p>Verification This requirement can be verified by source code inspection.</p>	Must
Req-6	<p>Description All the deliverables (source code, test code, and documentation) shall be under version control in a Git repository.</p> <p>Motivation This requirement is needed to capture the delivery history and keep the change.</p> <p>Verification This requirement can verify by reviewing all the deliverables of the Git by the date.</p>	Must
Req-7	<p>Description The system should follow an agile development process with two weeks sprint that starts with story selection with the customer.</p> <p>Motivation The deliverables can be accessible to the stakeholder in the development phase so the customer can give more feedback.</p> <p>Verification The stakeholders will review all the deliverables iteratively.</p>	Must

4.3.2. Product Requirement

This subsection describes the product requirements of the tool. The product requirements were assessed using ISO standard 25010 [22], a quality model. This model identifies which features should be considered when evaluating the system's quality attributes. Table 3 presents these requirements with priority, aspect, and motivation.

Table 3 Product requirements

ID	Priority	Attribute	Description and Motivation
PR-1	Must	Functional	<p>Description The tool shall be able to visualize architectural hotspots [18] in user-specified level-file levels, component levels, and building block levels for a user-defined timeframe</p> <p>Motivation The user can quickly view the whole codebase where the complicated code exists.</p>
PR-2	Must	Functional	Description

			<p>The tool shall visualize the complexity trends [18] of a selected hotspot (components or files) using a line graph for a user-specified timeframe.</p> <p>Motivation This requirement is needed to see the complexity of a hotspot over time.</p>
PR-3	Must	Functional	<p>Description The tool shall detect change coupling [18] between files by analyzing revision history.</p> <p>Motivation This requirement will uncover the hidden relationship between files by analyzing the changing pattern.</p>
PR-4	Must	Functional	<p>Description The tool shall rank the detected hotspot.</p> <p>Motivation This requirement is needed to determine which part of the building block can be refactored fast.</p>
PR-5	Must	Functional	<p>Description The tool shall separate files of a building block into four major categories - makefiles, scope files, machine-generated files, implementation files.</p> <p>Motivation This requirement gives the opportunities to filter out less essential files.</p>
PR-6	Could	Functional	<p>Description The tool shall extract metadata from ClearCase based on file extension.</p> <p>Motivation This requirement allows filtering files based on needs.</p>
PR-7	Should	Functional	<p>Description The tool shall save generated diagrams and results (file format is to be determined).</p> <p>Motivation This requirement is needed to use and compare the result in the future.</p>
PR-8	Must	Compatibility	<p>Description The tool must be able to extract and process historical data from ClearCase.</p> <p>Motivation The historical data from ClearCase is needed to measure architectural hotspots and rank the refactoring cost.</p>
PR-9	Could	Compatibility	<p>Description The tool shall be able to extract data from Git</p>

			<p>Motivation To be prepared for future development environment (GreenHouse Github) and development teams working in Break-out-archive (BOA)</p>
PR-10	Must	Usability	<p>Description The generated diagrams shall contain the timeframe, file scope, and type of analysis.</p> <p>Motivation The diagram needs to be self-descriptive for the general user</p>
PR-11	Could	Maintainability	<p>Description The tool's implementation shall be extensible to add a new visualization and metric.</p> <p>Motivation A new visualization or metric can be introduced soon.</p>
PR-12	Must	Reliability	<p>Description The tool's implementation shall write log messages to a file when an exception occurs.</p> <p>Motivation This requirement is needed to analyze and understand the exception.</p>
PR-13	N/a	Security	The security requirement is not considered for this project

5. System Architecture and Design

After the requirements elicitation process, the next step was to formulate the system architecture as a solution direction toward the project objectives. In this chapter, we discuss the system architecture, the rationale behind the design decisions, and alternative design ideas. The system architecture and design are constructed based on the system requirements of Chapter 4.

Section 5.1 includes the high-level architecture of the system. Section 5.2 contains the 4+1 view model of architecture (Kruchten, 1995), which is used in this chapter to describe the design and architecture of the system. We explain how we covered the use cases (i.e., the +1 view) in Section 4.2 and the other four views in Sections 5.3 to 5.6.

5.1 High-Level Architecture

In this section, we describe the high-level architecture of the system, which clarifies the context before going to the detailed architecture. The system context diagram in Figure 7 is used to show the system's high-level architecture. The Version Control System and Release Portal are the two external interfaces the tool access.

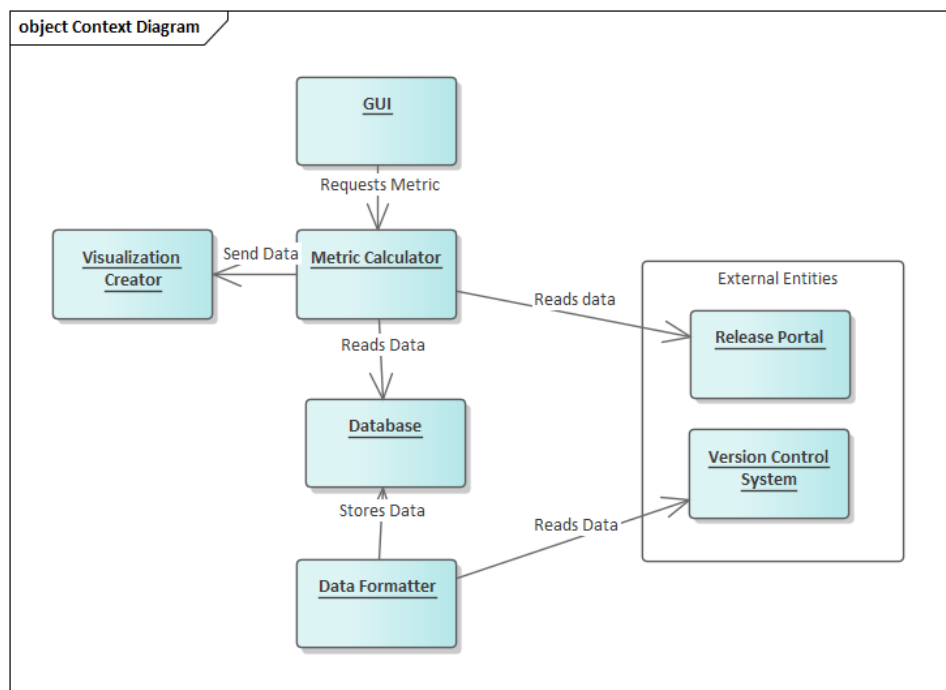


Figure 7 System context diagram

Database: Holds data retrieved from the version control system. The database is needed to get good performance.

Data Formatter: Reads and formats data from the version control system. Later, the data is stored in the database.

Metric calculator: Retrieves data from the database and release portal. Later, it calculates the four metrics described in Section 3.2.

Diagram Generator: Visualizes the diagrams using values from the metric calculator.

5.2 *The 4+1 View Model of Architecture*

Kruchten [23] first proposed the 4+1 view model of architecture, which is a model for describing the architecture of software-intensive systems based on multiple views:

Logical View: The class diagram of the design is represented by the logical view

Process View: The process view captures the design's synchronization and concurrency features.

Development View: The software's static organization in its development environment is described by the development view.

Deployment View: The distributed nature of the software and its mapping to the hardware are described by the deployment view.

These four views can be used to structure the description of the architecture and the decisions made, and a fifth view can be created by selecting a few use cases or scenarios to serve as examples.

5.3 *Logical View*

This section describes the logical view of the MoVACA. It contains four modules. Figure 8 is the high-level class diagram for the system. The design uses strategy pattern to select the metric at runtime. The MetricCalculator class is responsible for passing the requests from view to different concrete classes. Every concrete class has a specific responsibility to deal with user requests. The HotspotRank, ComplexityTrends, and Hotspot are the concrete metrics that use the DataConnection class to create and return the database connection.

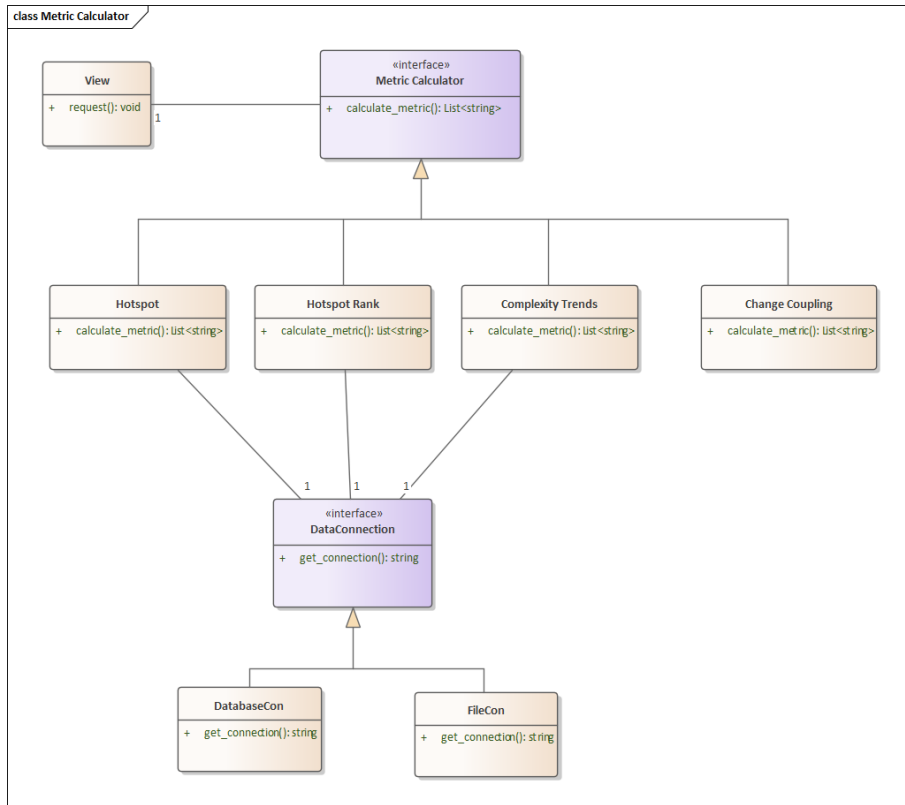


Figure 8 Class diagram metric calculator

The ChangeCoupling class uses the release data connection. Hence, the release data connection can be passed with the DataConnection abstract class, so the structure and behavior of all the concrete classes will be similar. DataConnection object can be given to the MetricCalculator abstract class. An updated class diagram is shown in Figure 9.

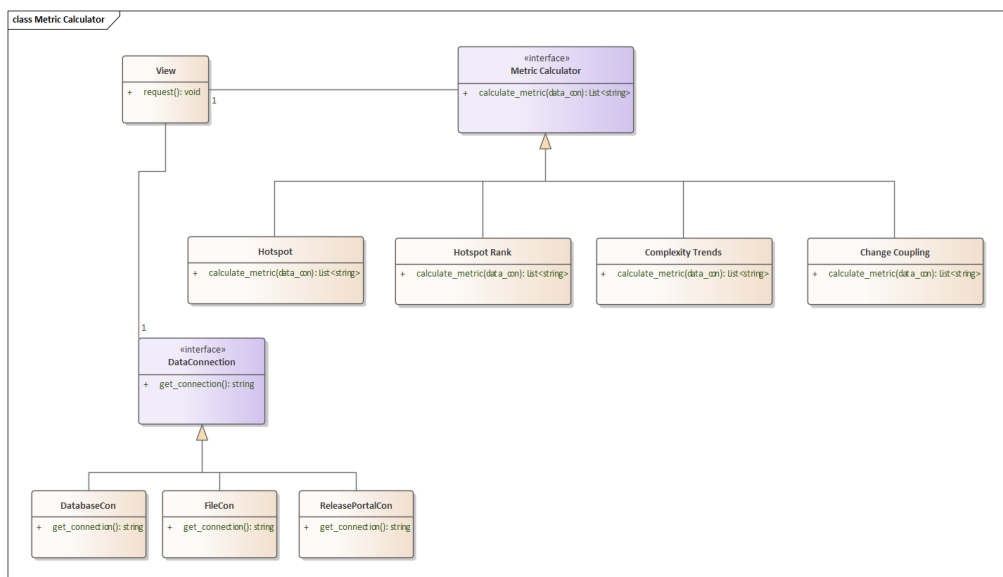


Figure 9 Class diagram metric calculator

The class diagram for the DataFormatter component is shown in Figure 10. The ChangeInfo class is responsible for calculating the version list of paths using the VersionList class. The LatestQbl class returns the latest recommended Qualified Baseline (QBL) from ClearCase (CC). ListOfContainer class returns the list of functional clusters, building blocks, and components from CC. Version-sChangedAmount class is responsible for calculating the number of inserted lines, deleted lines, lines of code, and single-line comments of versions using the abstract class VersionInfo. This approach is known as the strategy method design pattern. The ChangeInfo class is responsible for gathering all data from the ClearCase and inserting this data using QueryRunner. QueryRunner will use DatabaseCon-nection class to insert the data into the database.

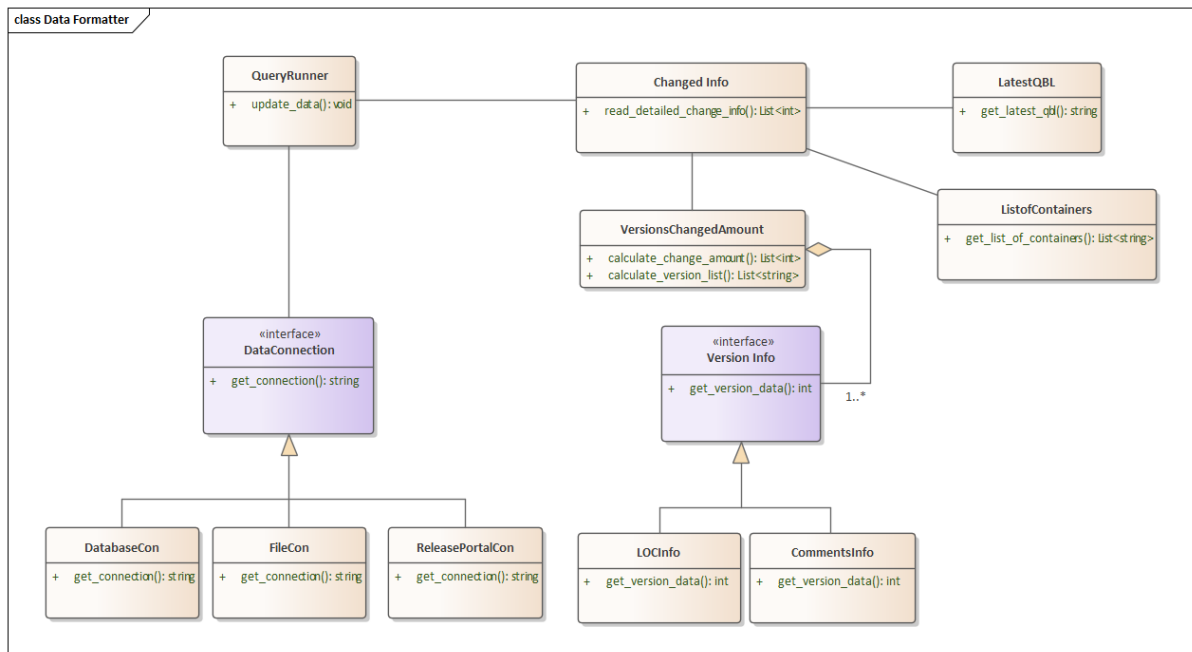


Figure 10 Class diagram data formatter

The class diagram for the change coupling is shown in Figure 11. In the class diagram, each of the classes has a specific responsibility to handle the client’s request. The view class deals with the request to see change coupling from the users. This request is dispatched to the CouplingFactory class and this class delegates the coupling generator task to the concrete classes responsible for calculating the coupling values dynamically using the Coupling generator abstract class. This design approach is commonly known as the factory design pattern.

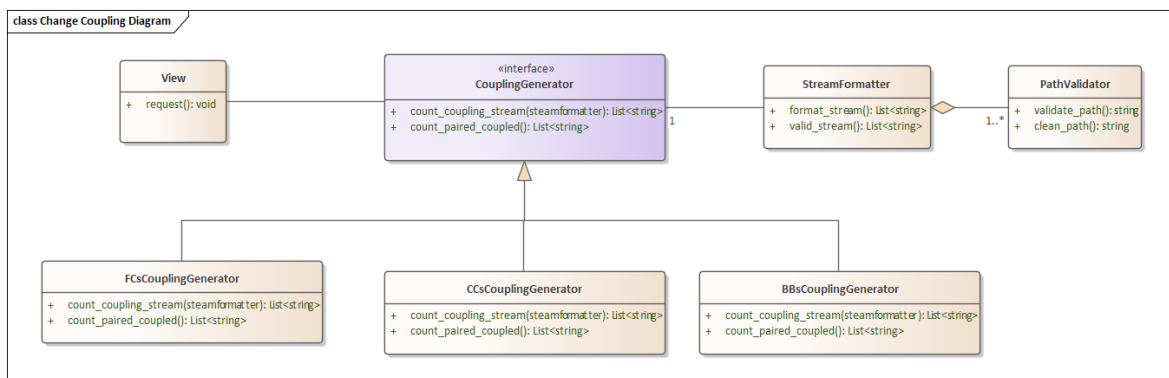


Figure 11 Class diagram for change coupling

5.4 Process View

The process view includes the UML sequence diagram of the change coupling. Sequence diagrams are used to illustrate how classes behave and interact to accomplish a specific use case functionality. It is often easier to understand the dynamic behavior of a given system's use case processes by the sequence diagram. The Figure 12 shows the sequence diagram for the admin user. Initialization would start from the admin. The sequence is broken into steps and listed below.

- The admin requested the DataFormatter module to update data.
- The DataFormatter checks whether the data from the database is the latest or not.
- If the database contains the latest updated data, dataformatter returns a response to the user. If it doesn't contain updated data, the dataformatter will go to ClearCase. It retrieves data and saves it to the database.

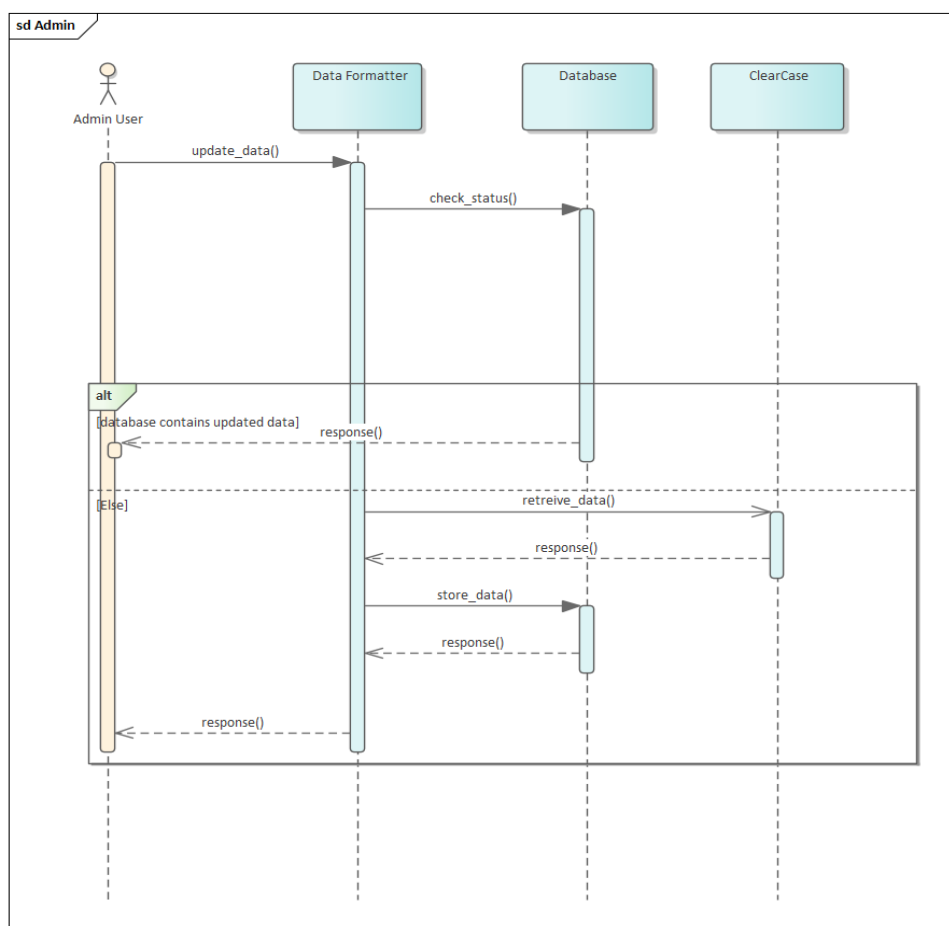


Figure 12 Sequence diagram for admin user

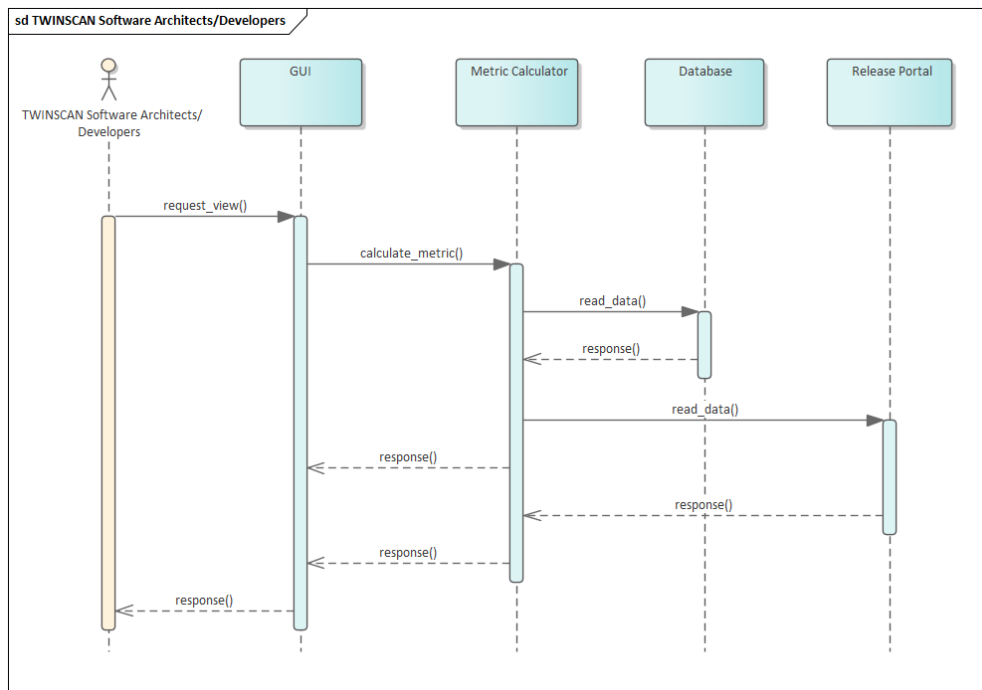


Figure 13 Sequence diagram for TWINSCAN developers/architects

The sequence diagram is for use TWINSCAN software architects or developers. The sequence is broken into several steps

- The TWINSCAN software architect and developer will request the GUI of the modularity analysis
- The user sends this request to the metric calculator
- The metric calculator requests data from the database and release portal and gets the response
- The metric calculator returns the response to the client.

5.5 Development View

The development view depicts a system from a programmer's viewpoint and is focused on software management. It describes system components using a UML Component diagram. The Package diagram is one of the UML diagrams used to depict the development view.

A visual representation of the relationship between various components is shown in Figure 14. The Data formatter component retrieves data from ClearCase and uses the database to store the data. Hotspot, Hotspot Rank, and Complexity Diagram use the data from the database. Complexity Trends depends on the hotspot to get the file name to calculate the trends. Change coupling uses release data to generate the coupling values.

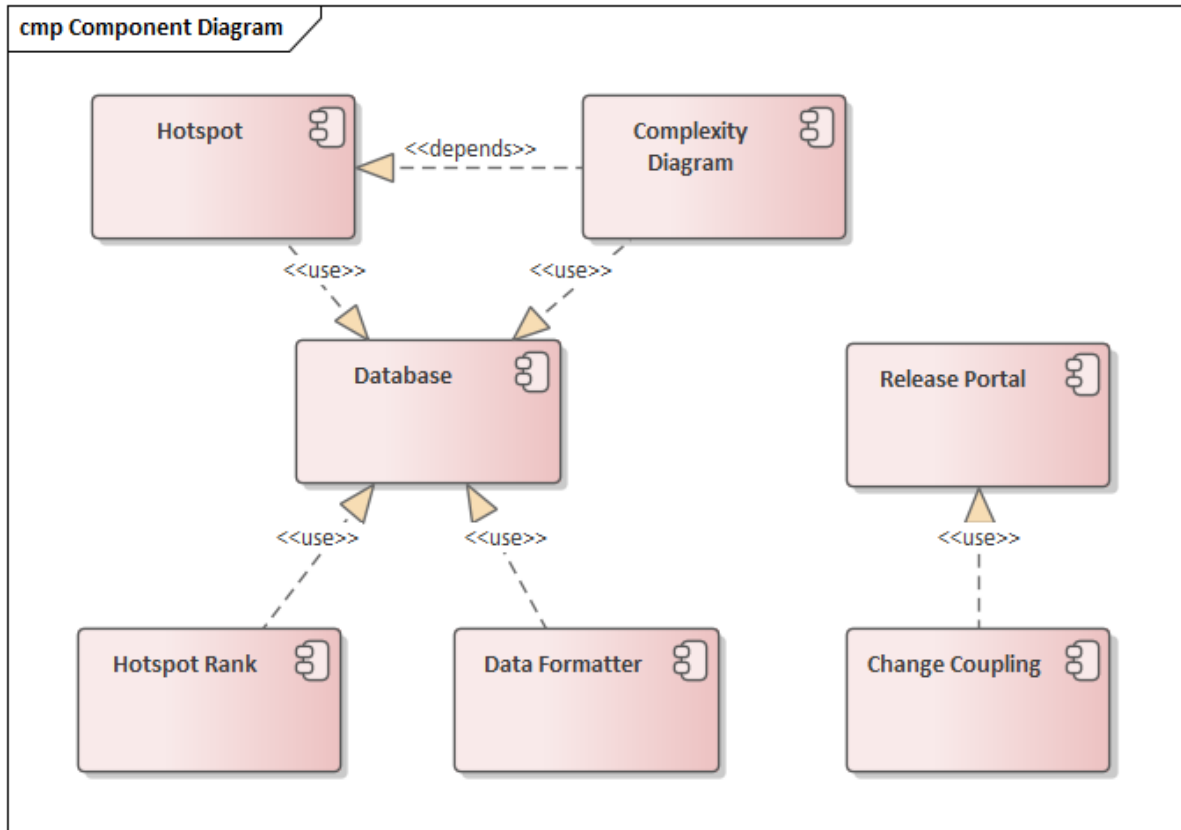


Figure 14 System component diagram

5.6 Deployment View

Figure 15 presents a visual overview of the deployment of various entities in the system. It shows us the different components of the system. The server is where the Python execution environment and the web interface reside within the ASML network. The server needs to start first by running the base python file. Then, the user can send HTTP request different metric calculator. The server interacts with two external entities—the Release Portal and ClearCase. Information regarding the file changes is retrieved from the ClearCase and release portal, which is then preprocessed and stored in the database. All data in the database can be visualized and embedded within the web interface presented to the user upon request.

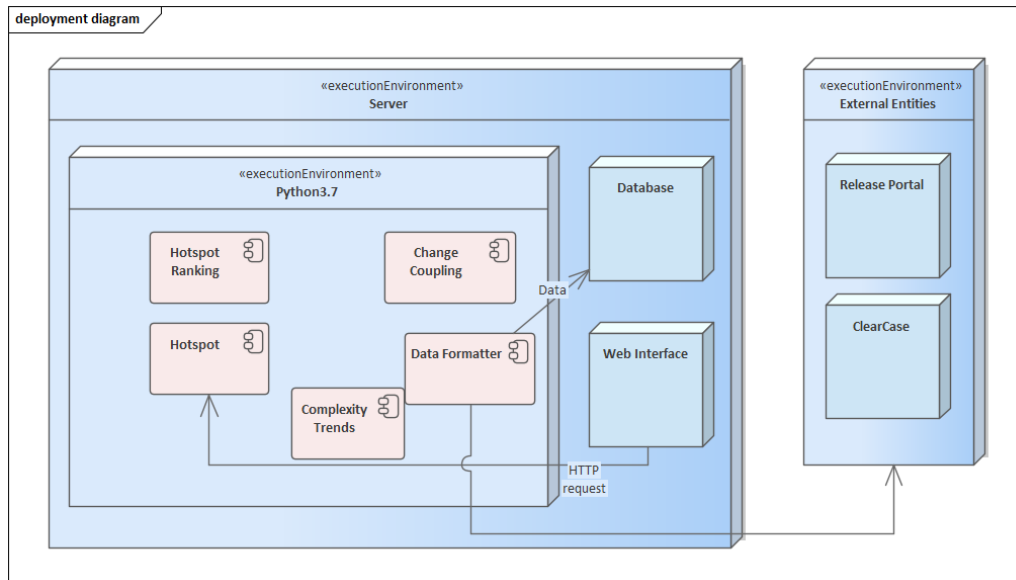


Figure 15 System deployment diagram

6. Implementation

We explain our system architecture and design in Chapter 6. This chapter describes the implementation of MoVACA following the design of the previous chapter.

In Section 6.1, we report the overview, including the development environment and structure of the solutions. Afterward, in Section 6.2, we explain the choices for selecting tools and technologies to develop this project. Finally, in Section 6.3, we describe the output of the modularity cost analysis tool.

6.1 Overview

We used Python 3.7 to develop the modularity cost analysis tool. The implementation was done in the ASML Linux Virtual Development Infrastructure (VDI) to access the ClearCase data. ASML Bitbucket was used as a version control system for tracking changes. We used PyCharm 2021.3 (Professional Edition) as our IDE.

6.2 Technology Choice

The project was developed according to the architecture explained in Chapter 5. This selection was made during the architecture phase. This section explains the choices for selecting tools and technologies to develop this project. In addition, high-level design decisions are listed at the end of the section.

Technology Choice 1: Use of Flask

Rationale:

- Flask is a micro-framework [24] and is generally easier to use to get started building applications.
- Flask provides full flexibility to change the modules of a project. Developers can add more functionality to Flask because of the expandable nature of the framework.
- Flask is easy to learn.

Discarded Alternatives:

Django: Django is harder to learn and use. Hence, the learning curve is steeper. It is feature packed. It does not provide flexibility for dynamically changing projects.

Technology Choice 2: Use of SQLite for storing data

Rationale:

- SQLite is a self-contained, serverless database [25]. It is also an embedded database because the database engine is integrated within the program.
- SQLite stores data in a single file, making it efficient to transfer
- It uses most of the standard SQL syntax, making it easy to use

Discarded Alternatives:

MySQL and Oracle: MySQL and Oracle require a server to run. These databases need to be compressed into a single file before transferring. Additionally, MySQL is not fully supported within ASML. It would add additional costs for maintenance, backup, and recovery of databases. Given these factors, SQLite is preferred over MySQL and Oracle.

Technology Choice 3: Use of D3 for graph visualization

Rationale:

- D3 provides the user with rich and easy interaction with the visualization in the web interface [26]. It provides several modules that help people to add interactivity, such as zooming, panning, and brushing.
- Rather than providing full components, D3 gives data-driven helper functions to create those components. Hence, the types of charts generated by D3 are not predefined.
- D3 is a JavaScript library that provides an easy bridge between the power of SVG (Scalable Vector Graphics) in the browser and the JavaScript applications.

Discarded Alternatives:

Chart.js: Chart.js generates a small number of charts. It uses html5 canvas tag, which is pixel based. By considering these factors, we prefer to use D3 to visualize the diagram.

Implementation Decisions:

After discussing with the stakeholders, the following high-level design decisions were made:

- Release data is not stored in the database: Release data will be accessed from the release portal instead of ClearCase to calculate the change coupling. The rationale behind this decision is that accessing the release stream directly from ClearCase needs more effort and time.
- Managing rename of files: When the file scope is changed in ClearCase, the file path also changes. MoVACA does not track the renaming of files path. It retrieves data from the latest recommended baseline.

6.3 Modularity Cost Analysis Tool

TWINSCAN software developers and architects will interact with MoVACA through a graphical user interface web browser.

Figure 16 shows the initial page of the tool.

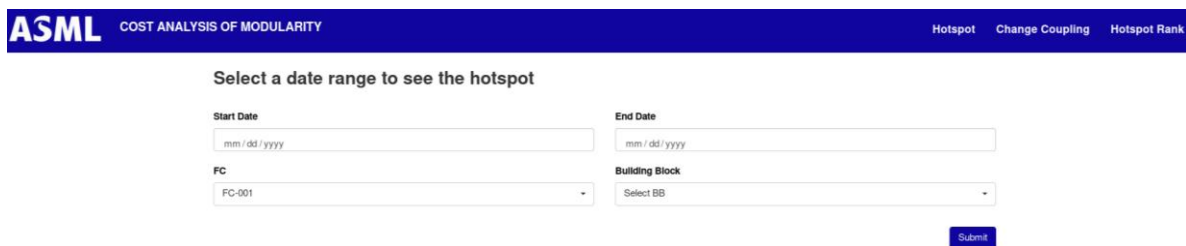


Figure 16 Modularity cost analysis tool main page

For testability purposes, we implemented our tool in four modules. The modules are

- Data Formatter
- Hotspot
- Hotspot ranking
- Change Coupling

Figure 17 shows the implemented modules of MoVACA. The implementation was done using Python 3.7 and SQLite 3 for the backend and HTML, CSS, and JavaScript for the front end. Flask Blueprint

[27] is used to create application components and supports common patterns within applications. The frontend files are stored in the templates folder.

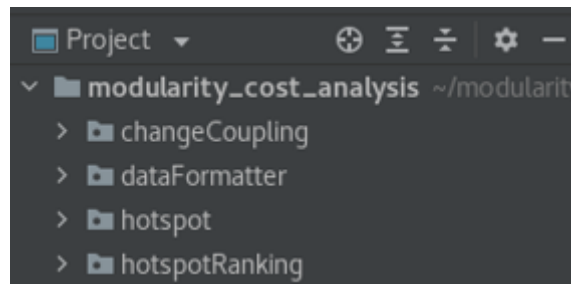


Figure 17 Packages of MoVACA

The strategy design pattern discussed in Figure 8 will pass user requests to the hotspot, change coupling, and hotspot rank metrics. The dataformatter module reads data from ClearCase and stores it in the SQLite database. The hotspot, hotspot ranking, and complexity diagram read data from the database for a user-specified time frame and sent JSON requests to the user.

```

from abc import ABC, abstractmethod

class ChangeCouplingGenerator(ABC):
    """
    This is the base class to generate coupling from the release data.
    """

    @abstractmethod
    def calculate_stream_coupling(self, stream_formatter):
        """
        Reads release data from the release portal and returns formatted streams
        to calculate couplings
        """
        pass

    @abstractmethod
    def count_coupled_pair(self, data):
        """
        Returns coupled filenames, coupling values, and stream list
        by analyzing release data
        """
        pass

```

Figure 18 Code snippet of coupling generator class

The change coupling module gets user requests with specific scopes and granularity. The change coupling's scope and granularity are implemented using the factory design pattern discussed in Figure 11. Figure 18 shows the code snippet of the abstract class for the factory design pattern. The class contains two functions. `calculate_stream_coupling` function takes the release stream from the release portal and returns a dictionary with commit ID as keys and file changed on that commit as values. The `count_coupled_pair` takes the dictionary as an input and returns the coupling between files. Figure 19 shows the algorithm for counting coupled pairs of files.

```
Function count_coupled_pair (D1)
Input D1: mapping with commit ID as keys and set of IDs of changed files as values
Output D2: mapping with sets of two file IDs as keys and frequencies as values
    D2=Counter(file_ID)
    for commit, files in D1.items():
        for file_pair in itertools.combinations(files, 2):
            if set(file_pair)<=files:
                D2[file_pair]+=1
        end for
    end for
end
```

Figure 19 Algorithm for counting coupling

A log file for the project is implemented. In Section 4.3, requirement PR-12 describes the need for logging. Logging is used to understand the frequency of exceptions and errors as well as to speed up the process of identifying them when they happen. Figure 20 is an example of the log lines captured during the development.

```
2022-08-23 13:00:02,411 WARNING werkzeug MainThread : * Debugger is active!
2022-08-23 13:00:02,425 INFO werkzeug MainThread : * Debugger PIN: 141-144-650
2022-08-23 13:03:56,518 INFO werkzeug MainThread : * Detected change in '/home/lrawshan/modularity_cost_...
2022-08-23 13:03:57,431 INFO Flask_sqlite MainThread : Info level log
2022-08-23 13:03:57,431 WARNING Flask_sqlite MainThread : Warning level log
2022-08-23 13:03:57,439 WARNING werkzeug MainThread : * Debugger is active!
2022-08-23 13:03:57,448 INFO werkzeug MainThread : * Debugger PIN: 141-144-650
2022-08-23 13:04:15,287 INFO Flask_sqlite MainThread : Info level log
2022-08-23 13:04:15,287 WARNING Flask_sqlite MainThread : Warning level log
2022-08-23 13:04:15,296 INFO werkzeug MainThread : * Running on http://127.0.0.9:5000 (Press CTRL+C to qu
2022-08-23 13:04:15,297 INFO werkzeug MainThread : * Restarting with stat
2022-08-23 13:04:16,001 INFO Flask_sqlite MainThread : Info level log
2022-08-23 13:04:16,001 WARNING Flask_sqlite MainThread : Warning level log
2022-08-23 13:04:16,009 WARNING werkzeug MainThread : * Debugger is active!
2022-08-23 13:04:16,017 INFO werkzeug MainThread : * Debugger PIN: 141-144-650
2022-08-23 14:01:19,048 INFO werkzeug MainThread : * Detected change in '/home/lrawshan/modularity_cost_...
```

Figure 20 Log file

7. Verification & Validation

This chapter explains the process of verification and validation used in this project, as well as the result of the process, to guarantee that MoVACA is working well. As a result, the verification and validation processes were employed to ensure that the tool implementation met the project's requirements.

7.1 Validation

Validation is primarily an external process that evaluates whether the developed system meets the stakeholder's needs. In the context of this project, the verification process was carried out by the EngD trainee and the project steering group. For verification, the following procedure was used in this project. This project followed an incremental tool development process. Multiple recurring meetings were held throughout the project to ensure that the process and created products were aligned with the requirements and the formulated requirements were correct. In addition, the system's requirements and the implementation of the tool were discussed during the monthly PSG meetings to ensure that the project was on track and the implemented system was built according to the agreed specifications.

7.2 Verification

Verification is usually an internal process of evaluating the correctness of the system. To verify the components discussed in Section 5.5, we performed the unit tests, checked the ASML coding standard using PyLint, and measured the code coverage of the system by the unit tests.

7.2.1. Unit test

A unit test is a piece of automated code that calls a unit of work in the system and then verifies a single test about that unit of behavior [28]. A unit can be a class's function or method. We developed test cases for every component. Some of these test cases for the system are listed in Table 4.

Table 4 Existing sample test cases for MoVACA

Test case number	Test case
Test_case_1	Test if a given path of a file is valid
Test_case_2	Test if the total number of functional clusters is correct
Test_case_3	Test if the 11nc for a building block is correct
Test_case_4	Test if the list of building blocks for a given functional cluster is valid
Test_case_5	Test if the LOC of a given file is correct
Test_case_6	Test if a given stream is valid
Test_case_7	Test if a given date is valid
Test_case_8	Test if a given path of a file is cleaned
Test_case_9	Test that the components from a file path is correct
Test_case_10	Test if two files are from the same building block

We used the unittest module to perform unit testing for the python code. Unittest is the built-in python module to write automated tests for applications [29]. Figure 21 contains the result of the unit test cases for the Data Formatter component.

```
test_get_bb_nc (__main__.TestDataFormatter)
test the nc from bb ... ok
test_get_comments (__main__.TestDataFormatter)
test the number of comments of a file ... ok
test_get_fcs_length (__main__.TestDataFormatter)
test the number of fc ... ok
test_get_fcs_list (__main__.TestDataFormatter)
test the list of fc ... ok
test_get_latestatqbl (__main__.TestDataFormatter)
test the latest recommended qbl ... ok
test_get_loc (__main__.TestDataFormatter)
test the number of lines of a file ... ok
test_get_nc11 (__main__.TestDataFormatter)
test the 11nc returned of a building block ... ok
test_get_nc11_list (__main__.TestDataFormatter)
test the 11nc returned of a building block ... ok
test_get_nc_bb_clearcase (__main__.TestDataFormatter)
test the nc from bb from clearcase ... ok
```

Figure 21 Test cases for the Data Formatter component

We used similar data [30] to test our JavaScript code. We used file connection to connect JSON files from the front end and tested the generated visualizations.

7.2.2. Code coverage

Code coverage is a measure to express how well a system's source code is tested. Coverage uses different methods (such as branch coverage, statement coverage, and decision coverage) to measure test efficiency. Programs with high test coverage indicate more source code has been executed during testing and are less likely to contain undetected software bugs than programs with low test coverage. For measuring code coverage, we use the Python library 'coverage.py.' It showed the percentage of the source code tested by the unit test cases. Due to limited time, the tests cover only the main features of the tool. The coverage report is shown in Figure 22. It shows that the system achieved 70% of code coverage.

Coverage report: 70%

Module ↓	statements	missing	excluded	branches	partial	coverage
changeCoupling\CCsCouplingGenerator.py	68	26	0	16	1	58%
changeCoupling\CouplingGenerator.py	6	2	0	0	0	67%
changeCoupling\FCsCouplingGenerator.py	104	41	0	26	1	57%
changeCoupling\PathFormatter.py	14	0	0	8	0	100%
changeCoupling\StreamFormatter.py	55	14	0	18	3	71%
changeCoupling__init__.py	0	0	0	0	0	100%
dataFormatter\Containers.py	40	18	0	4	0	50%
dataFormatter__init__.py	0	0	0	0	0	100%
hotspotRanking\DatabaseConnection.py	5	0	0	0	0	100%
hotspotRanking__init__.py	0	0	0	0	0	100%
hotspots\hotspot.py	145	17	0	4	1	87%

Figure 22 Code coverage result of MoVACA

7.2.3. Coding convention

We used PyLint to check the python coding style's consistency throughout the system. PyLint is a tool that evaluates Python code for errors, attempts to enforce coding standards, and detects undesirable code smells. PyLint's default code style is similar to PEP 0008. A Pylint-based scope check is performed on the Python code at compile time.

PyLint produces different levels of messages. ASML has its own PyLint rules defined. The PyLint messages from the community version are mapped to the ASML version of the PyLint messages. ASML uses the TICS framework to check the code quality. The PyLint messages are also mapped to the TICS level.

To check the coding standard of MoVACA, we used the ASML PyLint rules for the components of the tool. The PyLint score for hotspot, hotspot ranking, change coupling, and data formatter component were 7.10, 7.07, 7.07, and 7.86, respectively.

Extendibility quality

The validation process of the extendibility of MoVACA will be discussed in this section. This is performed using a qualitative method. When a new metric is introduced, the developers only need to implement the specific metric. The steps are listed below:

- First, the new metric needs to register to the blueprint using an app.py file.
- A folder with the name of the metric needs to be created inside the project.
- We used the strategy pattern to pass user requests for the four metrics. The class diagram is discussed in Figure 9. The diagram in
- Figure 23 explains the way to introduce a new metric to the pattern. The diagram assumes a new metric, MetricFive, like the hotspot diagram and hotspot ranking, and a new data connection DataConFour, like the DatabaseCon and the ReleasePortalCon. The diagram shows that the code extendibility to add a new feature is achieved by the tool.

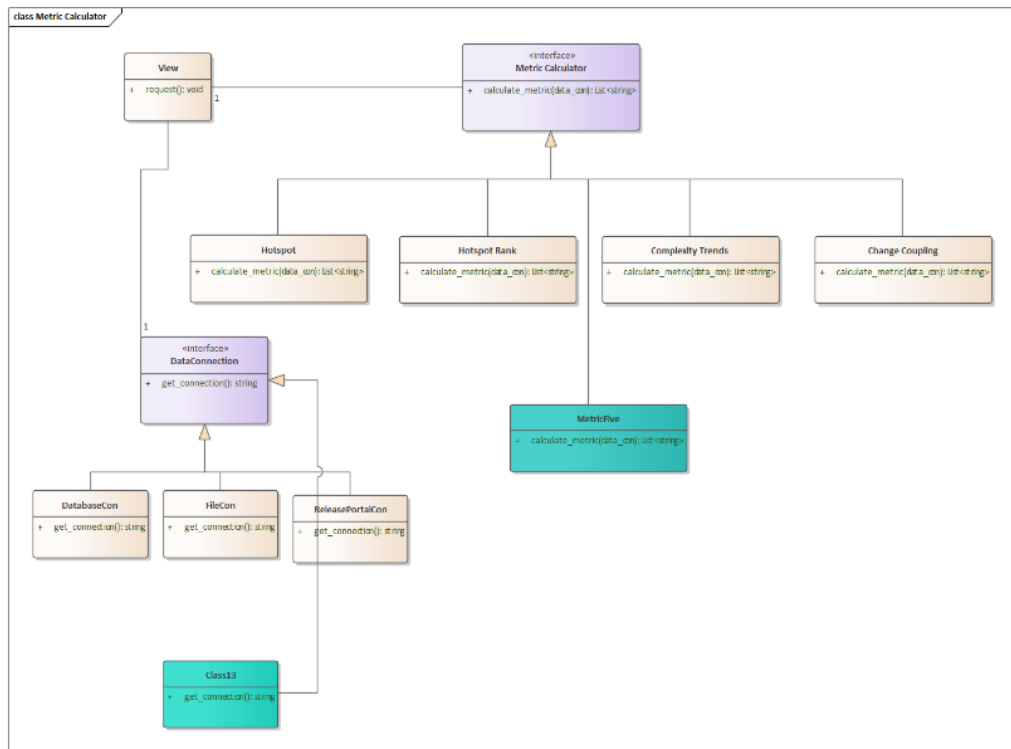


Figure 23 UML class diagram for extending a new metric

8. Project Management

One of the essential aspects of project success is project management. The primary purpose of project management is to set up the project's roadmap and strategies for dealing with challenges and risks. Therefore, fulfilling the project's goal will be more manageable within the restrictions.

The project management process followed in this project is explained in this chapter. Every EngD ST project is technically and organizationally challenging. The project uses an iterative approach. The iterative approach gave us the flexibility to demonstrate the prototype to the product owner, get valuable feedback, and be flexible. The rest of the chapter is structured as follows. Subsection 8.1 presents the work breakdown structure; Subsection 8.2 shows the project timeline; Subsection 0 describes the risks.

In this chapter, we report project management and planning of this project. First, we describe how we managed the project. Second, we show the risk table with an explanation.

8.1 Work-Breakdown Structure (WBS)

In this section, the Work-Breakdown Structure of the project is discussed. The project is divided into five major categories: planning and management, research, design & Implementation, validation, and project closure. Figure 24 shows the detailed activities conducted in each category.

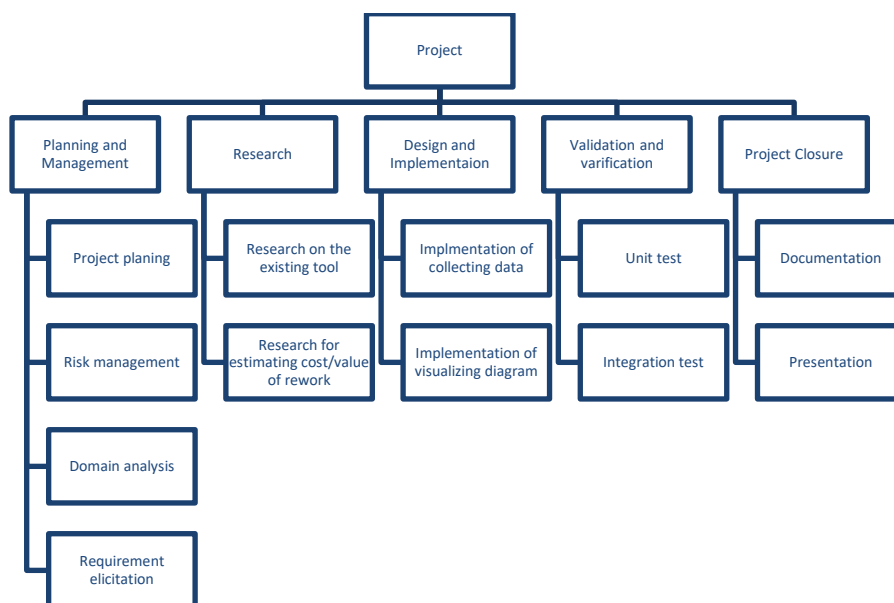


Figure 24 Work-breakdown structure

8.2 Project Planning and Scheduling

At the beginning of the project, there were some questions about the project requirements. We came up with the initial plan to start the project and after each PSG meeting, we refined it until it was more concrete.

A project timeline is created at the initial stage of the project, and it is updated to ensure that all activities are on track. This timeline helps evaluate project progress and see the influence of one activity on others. Microsoft Project is used to create the Gantt chart. Figure 25 shows a high-level overview of the activities involved and the estimated time allocated for each task. Project Initiation, Domain Analysis, Research, Implementation, Validation, and Documentation are the five essential tasks.

	Task Mode	Task Name	Duration	Start	Finish
1		1 Cost/Value Analysis of Modularity Improvements	210 days	Mon 1/3/22	Sat 10/22/22
2		1.1 Project Initiation	64 days	Mon 1/3/22	Thu 3/31/22
3		1.1.1 kick off	1 day	Mon 1/10/22	Mon 1/10/22
4		1.1.2 Environment Setup	10 days	Tue 1/4/22	Mon 1/17/22
5		1.1.3 Problem analysis	25 days	Wed 2/16/22	Tue 3/22/22
6		1.1.4 Project scope definition	36 days	Mon 1/17/22	Mon 3/7/22
7		1.2 Domain analysis	81 days	Mon 1/10/22	Sun 5/1/22
8		1.2.1 Review Previous Codebase	31 days	Tue 1/25/22	Tue 3/8/22
9		1.2.2 Initial Domain Analysis	30 days	Tue 1/18/22	Mon 2/28/22
10		1.2.3 Stakeholder analysis	40 days	Mon 1/3/22	Fri 2/25/22
11		1.2.4 Requiriement Analysis	65 days	Tue 2/1/22	Sun 5/1/22
12		1.3 Research	80 days	Mon 1/3/22	Fri 4/22/22
13		1.3.1 Dependency Analysis Research	43 days	Tue 1/11/22	Thu 3/10/22
14		1.3.2 Cost Analysis Research	74 days	Tue 1/11/22	Fri 4/22/22
15		1.3.2.1 Technology Research	22 days	Tue 3/1/22	Wed 3/30/22
16		1.4 Design and Implementation	87 days	Mon 4/4/22	Tue 8/2/22
17		1.4.1 Tool to Estimate Cost for Dependencies	87 days	Mon 4/4/22	Tue 8/2/22
18		1.5 Validaiton and Verification	140 days	Tue 2/15/22	Mon 8/29/22
19		1.5.1 Design review	15 days	Tue 2/15/22	Mon 3/7/22
20		1.5.2 Solution Test	20 days	Tue 8/2/22	Mon 8/29/22
21		1.6 Documentaion and Presentation	190 days	Tue 2/1/22	Sat 10/22/22
22		1.6.1 PDEng Thesis Report	181 days	Tue 2/1/22	Tue 10/11/22
23		1.6.1.1 First Draft Report	65 days	Tue 2/1/22	Sun 5/1/22
24		1.6.1.1.1 Create report outline	5 days	Tue 2/1/22	Sat 2/5/22
25		1.6.1.1.2 Regularly update relevant secions	55 days	Tue 2/8/22	Mon 4/25/22
26		1.6.1.2 Review PDEng Report by Supervisors	7 days	Tue 6/7/22	Wed 6/15/22
27		1.6.1.3 Final Report	81 days	Tue 6/21/22	Tue 10/11/22
28		1.6.2 Final presentation	16 days	Thu 9/15/22	Thu 10/6/22
29		1.6.3 Project Booklet	7 days	Fri 10/14/22	Sat 10/22/22
30		1.6.4 Graduation	1 day	Thu 10/20/22	Thu 10/20/22

Figure 25 Project timeline

8.3 Project Risk Analysis

The risk management process starts by identifying an initial set of risks at the beginning of the project. Each identified risk was assigned a severity and probability score. During the execution of the project, the initial set of risks was updated by adding newly identified risks and adjusting previously identified ones when they became obsolete. This section discusses the identified risks, their impact, and the mitigation plan. Table 5 Project risks present the list of the risks.

Table 5 Project risks

Risk Id	Description	Impact	Effect	Mitigation Plan
1	Requirement change or adding extra features and tools	High	Documentation and operation phase will have limited time.	<ul style="list-style-type: none"> Make the SMART requirements with identifiers and priorities Frequently review the requirements

2	Miscommunication with Stakeholders	High	Misunderstanding of the project goals that result in a delayed and inefficient development process	<ul style="list-style-type: none"> • Frequent communication with Stakeholders • Confirm important decisions with stakeholders
3	High priority requirements cannot be satisfied	High	Might result in an incomplete solution	<ul style="list-style-type: none"> • Raise the issue as early as possible, discuss with stakeholder for a possible workaround • Manage the stakeholders' expectations
4	Lack of domain knowledge	Medium	Change in deliverables, delay in the project completion	<ul style="list-style-type: none"> • Pre-study and taking Online courses • Trying to build a network of experts to ask for help
5	Selecting improper matrix/method that cannot satisfy the requirements	High	Slow down development	<ul style="list-style-type: none"> • Conducting comprehensive research and feasibility study • Keeping stakeholders in an early feedback loop with a regular demo on achievements or challenges
6	Main Stakeholders' illness (such as COVID)	Medium	Depending on the role, could lead to the project delay for a different period	<ul style="list-style-type: none"> • Try to find some alternatives for asking questions and help • Taking necessary measures stated by the government
7	Data need to predict cost is not available	High	Might result in an incomplete solution	<ul style="list-style-type: none"> • Try to make relation between data to generate new data • Keep stakeholders updated about data insufficiency
8	Limited data available to predict technical debts	High	Might depict partial result	<ul style="list-style-type: none"> • Keep stakeholders updated

8.4 Communication

Establishing a clear and regular communication channel is essential to avoid misunderstandings and promote early input to monitor and manage the project's progress and direction. Most of the meetings held during the project's execution fell into one of three categories: weekly update meetings, monthly update meetings, and other meetings called on-demand meetings.

8.4.1. Weekly update meetings

The weekly update meetings were set up at the start of the project every week. The EngD trainee conducted a separate meeting with ASML supervisors and the TU/e supervisor. These meetings aimed to demonstrate incremental and minor updates about the project's progress to the major stakeholders. These updates included the tasks completed in the previous week, any blocking issues, and plans for the upcoming week. Weekly meetings ensured that the project was on track and allowed to spot any misunderstanding as early as possible.

8.4.2. Project steering group meetings

Project Steering Group (PSG) meetings were held every month with ASML and TU/e supervisors. The primary purpose of these meetings was to encourage feedback from the stakeholders. Typically, these meetings started with a demonstration where the trainee explained the significant updates implemented since the previous PSG meeting. A high-level plan for the coming month was also discussed towards the end of these meetings.

8.4.3. On-demand meetings

In addition to the regular weekly and monthly meetings, several other meetings were held during the project. Although these meetings were not periodic and were mainly scheduled as needed, they played an essential role in communicating with key stakeholders and understanding the project context.

9. Conclusions

This chapter focuses on the project's findings, elaborating on the achieved results and adding value to the stakeholders. It also covers the recommendations for future work.

9.1 Results

This section describes the result achieved based on the requirements listed in Chapter 4. The project's main goal is to develop a tool that will visualize the architectural debt in tabular and graphical formats. To meet the project's purpose, we developed MoVACA for the TWINSCAN software developers and architects that will visualize the refactoring target of a system.

MoVACA requires detailed information about the time dimension and change history data of the TWINSCAN software to detect architectural debt. Therefore, we invested most of the development time interacting with ClearCase and the release portal to get the change history for the files in the system.

To visualize architectural debt of different scopes, we chose four metrics-hotspot, change coupling, complexity trends, and hotspot diagram based on the available data. MoVACA clearly shows how the source code changes for a specific period. It should help the architects and developers of TWINSCAN software to get the overall view to make a better design decision.

This report shows that the prototype of the tool meets all the must-have and could-have requirements except PR-5 and PR-9 from the requirements listed in Chapter 4. However, the should-have product requirements are left for future work. We tested the main use cases and verified the code quality using the ASML Python coding standard.

We divided the architecture of the tool into components. The tool is designed in a way that it can be extended to add a metric. All the source code, test cases, and documentation are stored in the Git repository to allow easy project delivery.

9.2 Recommendations and Future Work

During the development of the project, we identified future possibilities, improvements, and features that were not implemented due to the time limit. The ideas for the recommendations and future works are listed below.

9.2.1. Future Work

- The MoVACA tool calculates the complexity trends at file levels. But the complexity trends at the function level can reveal more insights into trends. Information related to the lines of code of a function and the commit number will reveal more about a trend. The function level change history can be tracked using cyclometric complexity.
- TWINSCAN uses the TIOBE TICS tool for accessing code quality. The developed tool can be integrated with the TICS tool to get more insights at the code and architecture levels.
- MoVACA doesn't track the renaming of scopes. It retrieves source code metadata from the latest recommended baseline. All the files are migrated to the destination scope when a scope is changed. Hence, we cannot get previous metadata of the source code. The renaming of scopes can be tracked to get the change history of files.

9.2.2. Recommendation

- Many research works [9], [2] suggest using social data to rank the refactoring of a system. The issue tracking system can obtain the development effort for each module. It will provide a complete picture of the module in terms of cost and effort. It is recommended for ASML to manage issue tracker data in a way that it can extract for future analysis.
- MoVACA extracts metadata of source code from ClearCase. However, ClearCase does not provide an easy way to extract source code metadata. Hence it is recommended that ASML adopt a better VCS policy to capture more detailed meta-data systematically to see how their software evolves.

10. Project Retrospective

This chapter concludes the report with a reflection on the project from the author's perspective.

10.1 Introduction

My ten-month journey was challenging, but it was also full of opportunities to learn from domain experts. With this project, I experienced a new domain and a number of challenges. These challenges provided numerous opportunities to improve my personal and professional skills.

The first challenge was to understand the domain and context of the problem. I spent the first three months on domain study and finding ASML-compliant computation platforms and technology through different trainings. Thus, I have gathered sufficient experience in the relevant technologies quickly.

Another challenge was to select the type of code analysis for the project. At the beginning of the project, it was not clear how to achieve the goal. To know the refactoring target, which data is needed, and whether it is possible to get the data. Hence, I went through several kinds of literature and a book to make a clear decision [2].

One of the most challenging parts was to define the requirements because I needed to examine a legacy source code of existing software to understand the improvement points. I implemented the hotspot metric to determine the available data and develop the requirements. In the beginning, the requirements were vague. I refined more concrete requirements with the suggestions and feedback from the supervisors and domain experts.

Furthermore, the project was heavily dependent on version control data. Most of the data related to the project reside in ClearCase. I didn't have prior knowledge about ClearCase. I had to dedicate a substantial amount of time finding a way of retrieving data from ClearCase.

Throughout the design and implementation phase, I applied design patterns to make an extensible and maintainable solution. Therefore, the design was changed iteratively then; accordingly, implementation also incrementally improved.

In this project, I have gained valuable experience in process management. The project was a personal project, so I was the project designer and the project manager. I have gained a lot of experience in organizational skills such as planning the project, creating the communication plan, managing risk, and taking initiatives.

However, there are a few things I could have done better. Documenting source code from the beginning of the implementation is one of the lessons that I learned. From the beginning of the implementation, I wasn't concerned about making my code comply with the PyLint coding standard and adding unit tests, even if it was related to managing the process of my project. Hence, it was more work for me to document the source code at the end.

I have gained a lot of experience managing projects and balancing the workload. Overall, I have had a challenging and beneficial experience with this project.

Glossary

EngD	Engineering Doctorate
MoVACA	Modularity Values Cost Analysis
TICS	Code quality measurements tool
UML	Unified Modeling Languages
LOC	Line of Code
GUI	Graphical User Interface
JSON	JavaScript Object Notation
KPI	Key Performance Indicator
TU/e	Eindhoven University of Technology
PSG	Project Steering Group

Bibliography

- [1] W. Cunningham, "The WyCash portfolio management system.," *ACM SIGPLAN OOPS Messenger*, 4(2), 29-30., 1992.
- [2] A. Tornhill, *Software Design X-Rays: Fix Technical Debt with Behavioral Code Analysis*. Pragmatic Bookshelf, 2018.
- [3] "ASML history," [Online]. Available: <https://www.asml.com/en/company/about-asml/history..> [Accessed 20 07 2022].
- [4] "ASML products," [Online]. Available: <https://www.asml.com/en/products..> [Accessed 20 07 2022].
- [5] "Tiobe TICS Viewer," [Online]. Available: <https://tics-scanner.asml.com/tiobeweb/TICS/>.
- [6] A. V. J. & K. T. Nugroho, An empirical model of technical debt and interest, ACM, 2011.
- [7] W. v. Houtum, "Software modularity Past-Future," [Online]. Available: https://asml.sharepoint.com/:p:/r/sites/SMART-program/_layouts/15/Doc.aspx?sourcedoc=%7B901846A2-431B-4A7D-97D9-0BC237257802%7D&file=Software%20modularity%20Past%20-%20Future.pptx&wdLOR=c71317051-6D0C-46A0-B181-5338B8AC5330&action=edit&mobileredirect=true.
- [8] B. Schoenmakers, "Modularity Index - ASML Technology Wiki," ASML, [Online]. Available: https://techwiki.asml.com/index.php/Modularity_Index.
- [9] R. C. Y. M. R. F. Q. X. L. H. S. .. & S. A. Kazman, A case study in locating the architectural roots of technical debt, IEEE/ACM 37th IEEE International Conference on Software Engineering (Vol. 2, pp. 179-188). IEEE., 2015.
- [10] "Systems and Software Engineering: Systems and Software Quality Requirements and Evaluation (SQuaRE): System and Software Quality Models. ISO.," in *International Organization for Standardization, & Technical Committee ISO/IEC JTC 1, Information technology. Subcommittee SC 7, Software and systems engineering*, 2011.
- [11] C. Y. C. K. B. & C. K. B. Baldwin, „Design rules: The power of modularity (Vol. 1). MIT press.,” 2000.
- [12] Y. e. a. DAJSUREN, "Simulink models are also software: Modularity assessment.," *In: Proceedings of the 9th international ACM Sigsoft conference on Quality of software architectures.*, 2013.
- [13] B. Schoenmakers, "PSA Modularity Index-D000466131-01," [Online]. Available: <https://tc-eu.asml.com:3000/#/com.siemens.splm.clientfx.tcui.xrt.showObject?uid=1IoxTIbsQS4FxA>.
- [14] "Tiobe TICS," [Online]. Available: <https://www.tiobe.com/tics/tics-framework/>. [Accessed 20 07 2022].
- [15] W. v. Houtum, "PSA SMART Modularity Roadmap-D001185115-01," [Online]. Available: <https://tc-eu.asml.com:3000/#/downloadFile?uid=3mwJw46pQS4FxA>. [Accessed 20 09 2022].
- [16] A. Alliance, "Introduction to the Technical Debt Concept," [Online]. Available: <https://www.agilealliance.org/introduction-to-the-technical-debt-concept>. [Accessed 16 09 2022].
- [17] A. Tornhill, "Software Design X-Rays: Fix Technical Debt with Behavioral Code Analysis. Pragmatic Bookshelf.," p. 19, 2018.
- [18] A. Tornhill, "Software Design X-Rays: Fix Technical Debt with Behavioral Code Analysis.," Pragmatic Bookshelf, 2018, p. 25.
- [19] A. Tornhill, "Software Design X-Rays: Fix Technical Debt with Behavioral Code Analysis. Pragmatic Bookshelf.," p. 35, 2018.

- [20] A. Tornhill, "Software Design X-Rays: Fix Technical Debt with Behavioral Code Analysis. Pragmatic Bookshelf.," p. 196, 2018.
- [21] Agile Business Consortium, 2022. [Online]. Available: https://www.agilebusiness.org/page/ProjectFramework_10_MoSCoWPrioritisation..
- [22] "ISO/IEC 25010," [Online]. Available: <https://iso25000.com/index.php/en/iso-25000-standards/iso-25010>. [Accessed 22 09 2022].
- [23] P. B. Kruchten, The 4+ 1 view model of architecture, IEEE software 12.6, 1995.
- [24] "Flask Web Development," [Online]. Available: <https://flask.palletsprojects.com/en/2.2.x/>. [Accessed 22 09 2022].
- [25] "SQLite," [Online]. Available: <https://www.sqlite.org/index.html>. [Accessed 22 09 2022].
- [26] "Data-Driven Documents," [Online]. Available: <https://d3js.org/>. [Accessed 22 09 2022].
- [27] "Modular Applications with Blueprints," [Online]. Available: <https://flask.palletsprojects.com/en/1.1.x/blueprints/>. [Accessed 19 09 2022].
- [28] P. Hamill, Unit test frameworks: tools for high-quality software development., O'Reilly Media, Inc., 2004.
- [29] "Unit testing framework," [Online]. Available: <https://docs.python.org/3/library/unittest.html>.
- [30] M. Bostock, "Zoomable Circle Packing," [Online]. Available: <https://observablehq.com/@d3/zoomable-circle-packing>.
- [31] S. De Mascia, "Project psychology: using psychological models and techniques to create a successful project. Routledge," 2016.
- [32] "System and software quality models. Standard, International Organization for Standardization," Systems and software engineering Systems and software Quality Requirements and Evaluation (SQuaRE), 2011.
- [33] R. G. D. H. S. & K. M. A. Benkoczi, "A design structure matrix approach for measuring co-change-modularity of software products," in *15th International Conference on Mining Software Repositories (MSR)*, 2018.
- [34] H. J. M. & K. J. Gall, "CVS release history data for detecting logical couplings," in *In Sixth International Workshop on Principles of Software Evolution.Proceedings.*, 2003.
- [35] R. L. O. I. S. R. S. & K. R. J. Nord, "Architectural dependency analysis to understand rework costs for safety-critical systems.," in *In Companion Proceedings of the 36th International Conference on Software Engineering*, 2014.
- [36] R. S. N. R. L. & O. I. Sangwan, "Architectural Dependency Analysis: Addressing the Elephant in the Room.," *Computer*, 54(3), pp. 73-78., 2021.

A. Stakeholder Analysis

Stakeholder analysis helps to understand the concerns and requirements of the main stakeholders in the project [31]. This chapter illustrates the detailed interests and involvements of the stakeholders in the project. The ASML and the Eindhoven University of Technology are the parties involved, and each of them has specific interests in the project. The following section identifies the stakeholders and presents each party in detail.

A.1 Stakeholder Identification

A stakeholder matrix identifies the project stakeholders and determines the activities required to match their interests with their goals. Power and interest are the two central variables of the matrix. This matrix organizes stakeholders into a 2x2 grid based on their power, interest, and level of involvement in the project. The stakeholder matrix for this project is shown in Figure 26.

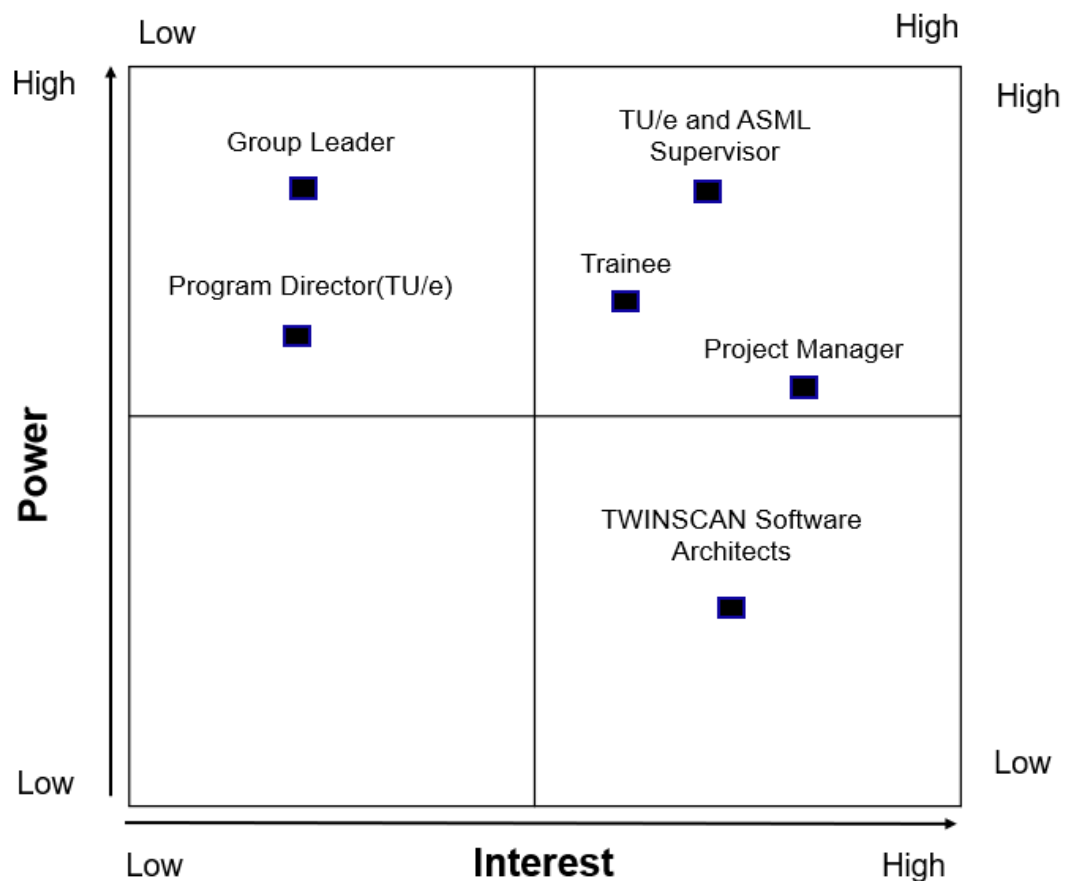


Figure 26 Stakeholder analysis matrix

A.2 Stakeholder Interests

This section describes the identified stakeholders' interests and involvement. Based on the expertise and interest, the involvement level of the stakeholders is different. ASML is the owner and initiator of the project. This project's outcome may provide additional value to the company. The stakeholder interest represents the source of knowledge, requirements, and expectations for this work.

Table 6 Main stakeholders' interest and involvement

Name	Role	Interest	Involvement
Tom Verhoeff	TU/e Supervisor	<ul style="list-style-type: none"> • Monitor the project progress and provide feedback. • Guide the trainee with technical and non-technical skills throughout the project. • Ensure quality of the graduation report. • Evaluate the final deliverables 	<ul style="list-style-type: none"> • Weekly meetings and monthly PSG meetings throughout the project duration
Remko Van der Vossen	ASML Supervisor	<ul style="list-style-type: none"> • Evaluate the progress and provide feedback. • Provide information regarding the needs and requirements of the project. • Ensure quality of the project result • Evaluate the final deliverables 	<ul style="list-style-type: none"> • Weekly meetings and monthly PSG meetings throughout the project duration
William Van Houtam	Project Manager	<ul style="list-style-type: none"> • Support the project by providing the necessary information on the ASML domain • Evaluate the progress and provide feedback • Evaluate the final deliverables 	<ul style="list-style-type: none"> • Weekly meetings and monthly PSG meetings throughout the project duration
Lamisha Rawshan	EngD Software Technology Trainee	<ul style="list-style-type: none"> • Successful and timely completion of the EngD graduation project • Design and develop a system that fulfills customer requirements • Gather experience in software design, architecture, communication, and project management 	<ul style="list-style-type: none"> • Throughout the project

The Engineering Doctorate (EngD) in Software Technology is conducted and assessed by the Eindhoven University of Technology as an educational program. Table 1 shows the interest and involvement of the main stakeholders of the project. The additional stakeholders' interests are listed in Table 2.

Table 7 Additional stakeholders' interest and involvement

Name	Role	Interest
Yanja Dajsuren	EngD ST Program director	<ul style="list-style-type: none"> • Quality of the project result • Successful graduation of the trainee
Henk Kant	Group Leader	<ul style="list-style-type: none"> • Ensure successful collaboration with TU/e • Provide necessary access required for the project
All architects of the TWINSCAN software	Software Architect Team	<ul style="list-style-type: none"> • Evaluate and use the project result

B. TWINSCAN Software FCs Dependencies

The essential dependencies between the FCs are divided based on two modes of operation. The modes are production and CPD. The essential FC dependencies for the production are shown in Figure 27. The essential FC dependencies for the CPD are shown in Figure 27.

Essential FC dependencies – Lot Production

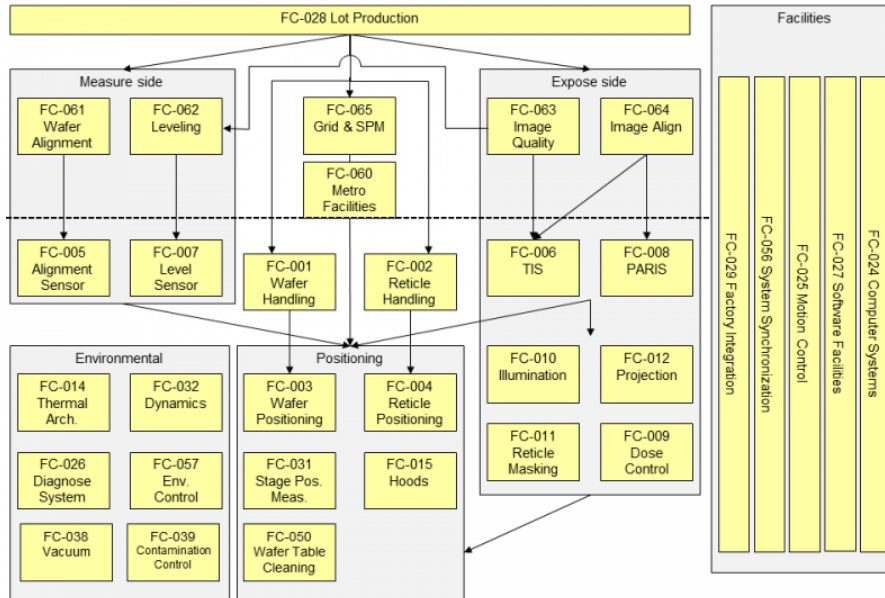
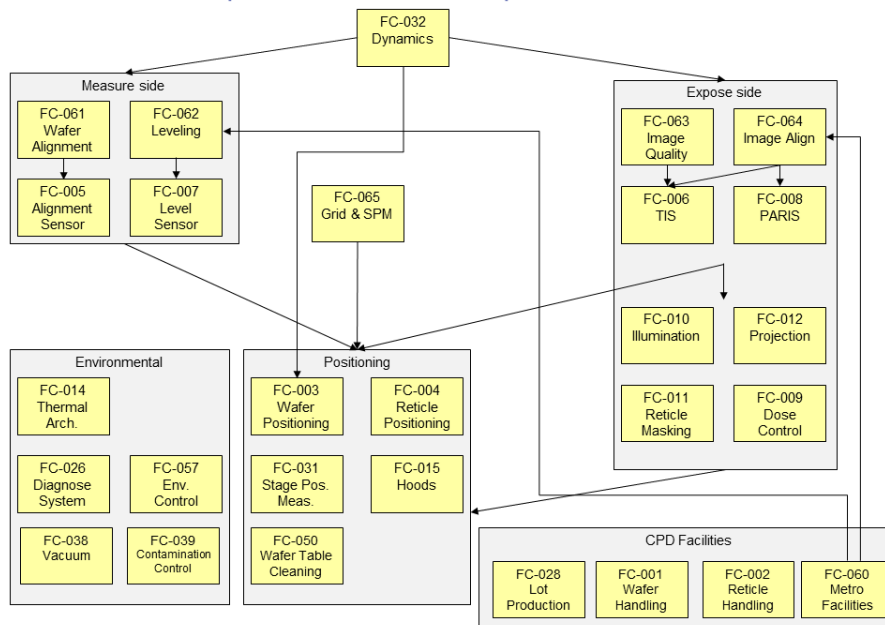


Figure 27 Essential FC dependencies of production

Essential FC dependencies – CPD operation



About the Author



Lamisha Rawshan received her Bachelor's and Master's degrees in Software Engineering from University of Dhaka, Bangladesh in 2014 and 2016, respectively. Her Master's thesis was titled "Time-Waved Monitoring and Emergent Self Adaption of Software Components in Open Source Cloud". In this project, she worked on cloud monitoring problem and proposed a self-adaptation solution for open source cloud. After graduation, she worked as a lecturer at Daffodil International University, Bangladesh for four years. She published research papers on cloud computing during this period. Her interests are in Software Architecture & System Design, Cloud Computing and Data Science.

PO Box 513
5600 MB Eindhoven
The Netherlands
tue.nl

PDEng SOFTWARE TECHNOLOGY

TU/e EINDHOVEN
UNIVERSITY OF
TECHNOLOGY