Master Thesis Project

On the Notions and Predictability of Technical Debt

2023/05/31

Author: Varun Dalal
Supervisor: Sebastian Hönel, Lic.
Examiner: Mauro Caporuscio, Ph.D.
Reader: Hemant Ghayvat, Ph.D.
Semester: VT 2023
Course Code: 5dv50e
Subject: Computer Science

Abstract

Technical debt (TD) is a by-product of short-term optimisation that results in long-term disadvantages. Because every system gets more complicated while it is evolving, technical debt can emerge naturally. The impact of technical debt is great on the financial cost for development, management, and deployment, it also has an impact on the time needed to maintain the project. As technical debt affect all parts of a development cycle for any project, it is believed that it is a major aspect of measuring the long-term quality of a software project. It is still not clear what aspects of a project impact and build upon the existing measure of technical debt. Hence in this experiment, the ultimate task is to try and estimate the generalisation error in predicting technical debt using software metrics, and adaptive learning methodology. As software metrics are considered to be absolute regardless of how they are estimated.

The software metrics were compiled from an established data set; Qualitas.class Corpus, and the notions of technical debt were collected from three different Static analysis tools; SonarQube, Codiga, and CodeClimate.

The adaptive learning methodology uses multiple parameters and multiple machine learning models, to remove any form of bias regarding the estimation.

The outcome suggests that it is not feasible to predict technical debt for small-sized projects using software-level metrics for now, but for bigger projects, it can be a good idea to have a rough estimation in terms of the number of hours needed to maintain the project.

Keywords: Technical debt error estimation; software metrics; Static Analysis tools; generalised error of predictions for machine learning algorithms

Preface

This master's thesis was a difficult undertaking as the goal was clear but the method was ever-changing. After trying several methods, it became clear that achieving an acceptable accuracy was not going to be possible hence the goal was slightly modified a bit to obtain some sort of an answer to the question.

I want to thank a few people who made it possible for me to undertake this challenge. First, I would like to thank Sebastian Hönel, my supervisor for his constant help and insight, which helped me achieve the results I have. We had many meetings regarding any doubts I had, ranging from a small doubts about the data to big doubts like discussing results or even methods used. He had an extremely long chain of conversations over e-mail as well, it was over 100 e-mails exchanged. Originally I was going to find 100 Python projects from GitHub and use them to create my data set but then he helped me by informing me about the existing data set, so the problem of ground truth is also solved while reducing some work for creating the data set. For certain situations, he broke down the steps into smaller steps for me to understand properly and not make any major mistakes. These are just some of the things, hence to say he helped me whenever I needed, would be an understatement.

Then I would like to thank Tibo Bruneel, he is an excellent friend. He enabled me to work more efficiently and even helped me by guiding me in the right direction when I was drifting away from my goal. Most importantly he helped me understand the importance of enjoying life while working, making my work-life balance a good one rather than a stressful one. I would also like to thank my partner for her unwavering support during my time of struggle as I was not able to pay attention to other things while writing the thesis, she helped me take care of myself.

Finally, I would like to thank my parents who made it possible for me to leave India and pursue my goals in Sweden, without their financial support, moral support, and emotional support, achieving this would have been impossible.

Without the support of all these people, it would have been impossible for me to get to the stage where I am right now.

Contents

List of Figures

1 Introduction

Modern software systems are complex and large. Hence software quality analysis is a very important aspect of the development phase, it involves more than just avoiding bugs. It also provides us with the chance to achieve an improved standard of code. There are several ways to achieve better software excellence, ranging from multiple quality levels to evaluating several quality attributes at every stage of development.
There are many quality attributes, on which the quality of a system is judged and quality contributors, that influence the attributes [1]:

Quality Attributes :	**Quality Contributors :**
Functional suitability	Software quality metrics
Performance efficiency	Software quality model
Compatibility	Software quality scorecard
Usability	Quality management
Reliability	Quality gates
Security	
Maintainability	
Portability	

Tools for evaluating the complexity and organisation of the code, detecting whether it complies with standards, confirming coverage, etc. are only a few of the many tools available for giving quantitative and qualitative indicators for software quality.

Static code analysis tools are one of the available options. They are popular and inexpensive means of automated code quality analysis [2]. They estimate different aggregated quality metrics, to attain one of the most important metrics, required to estimate time and financial costs, which is vital to accomplish the goal of better code quality. This metric is called technical debt.

Technical debt has significant impacts on financial costs for development, management, and deployment. It also has a major impact on the time needed to maintain the project and bring up the quality of the project itself. Technical debt is considered a fairly new research area. As there are different types of debts and various indicators present, the root cause is still slightly fuzzy [3].

In this section, you will find the following: sub-section.1.1 Brief background on technical debt, sub-section.1.2 Brief background on the selected tools, sub-section.1.3 Motivation for this study, sub-section.1.4 Problem statement and Research questions, sub-section.1.5 Thesis report, sub-section.1.6 Contribution of this study, sub-section.1.7 Target groups, and sub-section.1.8 Report structure.

1.1 Background

Ward Cunningham coined the term "technical debt" (TD) in 1992, describing it as those technical chores you decide not to undertake right now but which, although not performed, have the potential of triggering future issues [4].

Technical debt (TD) is a by-product of short-term optimisation that results in long-term disadvantages [5]. Businesses that prioritise speed above quality create technical debt that must be repaid later to avoid penalties. While development continues, team leaders frequently postpone features and functionality, take shortcuts, or accept less-than-ideal performance. This practice is known as technical debt. That happens due to the ethos of "build now, correct later." It is usually measured in terms of time.

According to Fowler[2], the causes for technical debt can be classified into four different categories: Reckless, Prudent, Deliberate, and Inadvertent (see Fig. 1.1). Fowler believes that they usually form a combination of two.

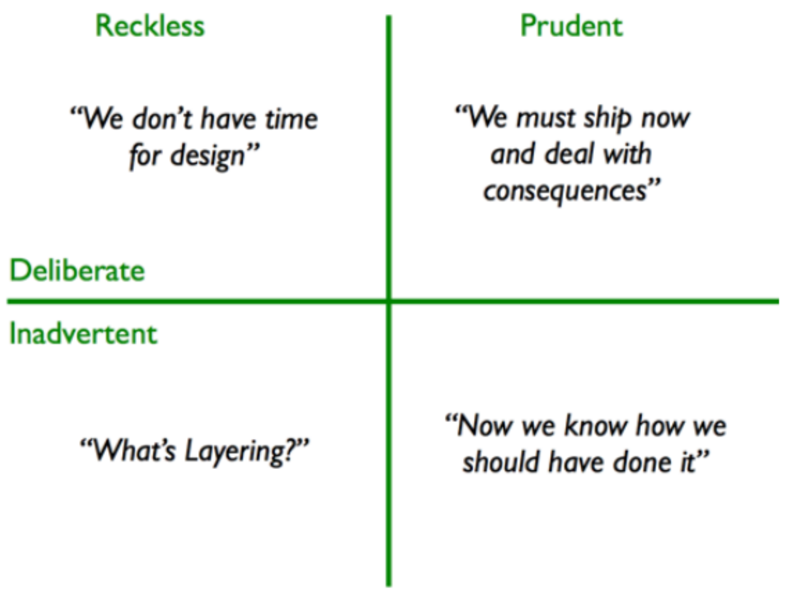

Reckless
Prudent
"We don't have time for design"
"We must ship now and deal with consequences"
Deliberate
Inadvertent
"What's Layering?"
"Now we know how we should have done it"

Figure 1.1: Technical Debt Quadrants.

Regardless of the mentality, as every project gets more complicated while the project is still being developed, technical debt rises naturally. There are mainly seven project-level metrics that can affect the technical debt score [6], which are as per many hypotheses by different researchers:

1. Bugs: defects in the project.
2. Code quality:
   (a) Cyclomatic complexity: is a measure for the complexity of a program.
   (b) Class coupling: number of inherited classes by one class.
   (c) Lines of code: quantitative metric to define the size of the program by adding up the number of lines.
   (d) Depth of inheritance: it is the length between the last child node and the root node in a hierarchy of classes.
3. Cycle time: time between the start of development and final release of product
4. Code churn: the size of code re-written for any reason.
5. Code coverage: is the measure of what size of the code base is tested.
6. Code ownership: is the measure of how the code base is divided between n number of personals.

[2]The "TechnicalDebtQuadrant": https://www.martinfowler.com/bliki/TechnicalDebtQuadrant.html.

Different tools estimate the same metrics in their own way [1]. It's not clear whether these differences affect the actual findings, in terms of aggregated metrics mentioned above, and even how they impact the final technical debt scores. As seen in Fig. 1.2, as of 2022 there were many static code analysis tools available in the market.

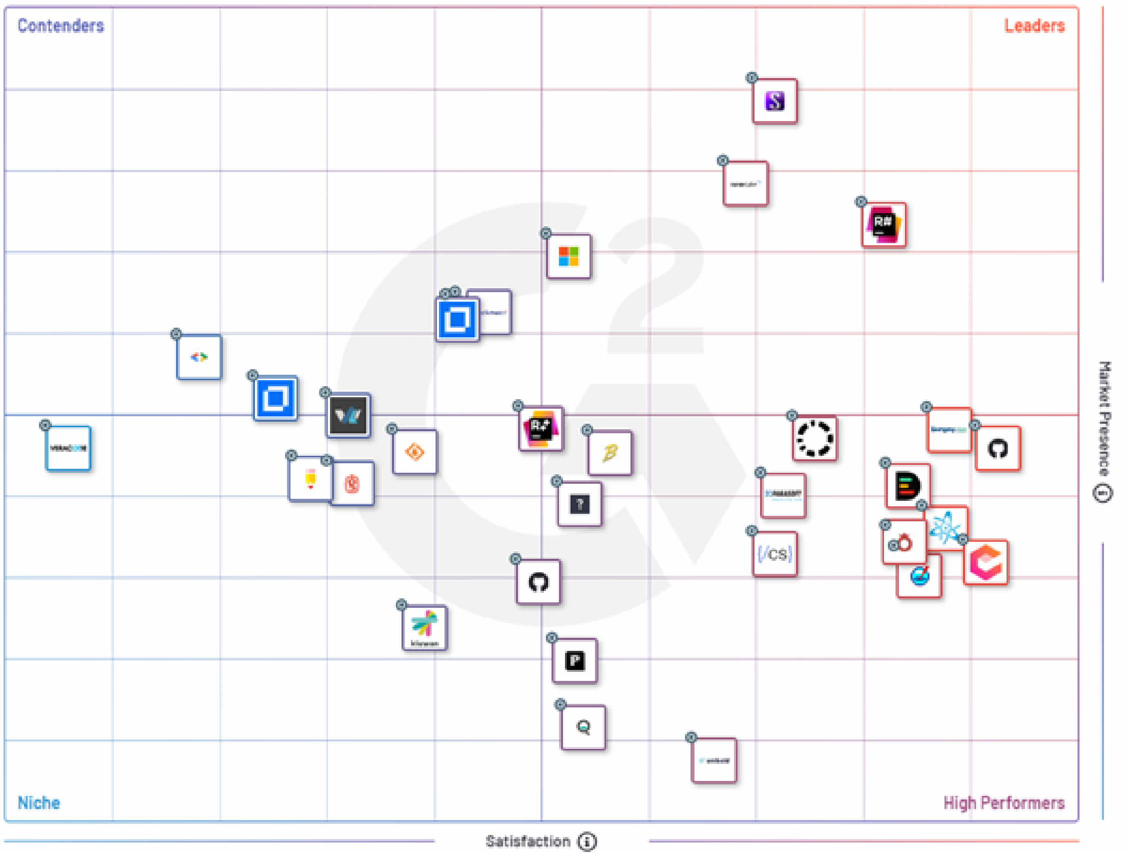


Figure 1.2: Available Static Analysis Tools[3]

Static analysis tools are debugging technique-enabled tools that examine the source code dynamically without the need for the program to be executed. This helps to ensure that the code base is compliant, secure, and safe by giving engineers an awareness of them. The procedure carried out by a static analysis tool, known as static code analysis, is the investigation of a software project against that set (or several sets) of coding rules.

1.2 Selected Tools

From the diverse tools seen in Fig. 1.2, the goal was to select some significantly distinct and some similar tools. As not all of them have a free version available, tool providers were contacted for a student access, for as many tools as possible. The other deciding factor for selecting the static analysis tools was the supported languages, as the list of projects were pre-selected beforehand. The selection criteria for tools was that they support both javascript and Java. In the end, three open-source and free tools were decided upon. As non of the paid front runners in the field responded to the request for student access to their respective tools.

[3]G2: https://www.g2.com/categories/staticcodeanalysis.

The first tool selected is SonarQube[4], released in 2006-2007. It is considered the state-of-the-art static analysis tool by many. SonarQube is an open-source Static Analysis tool, developed by SonarSource. It can be integrated with the development cycle for a project and can provide continuous analysis. SonarQube supports Java (including Android), C sharp, C, C++, JavaScript, TypeScript, Python, Go, Swift, COBOL, Apex, PHP, Kotlin, Ruby, Scala, HTML, CSS, ABAP, Flex, Objective-C, PL/I, PL/SQL, RPG, T-SQL, VB.NET, VB6, and XML. In the image Fig. 1.2, it can be seen that SonarQube is a high-performance tool and is considered to be one of the leaders in the field.

The second tool found is CodeClimate[5]. It is a fairly new tool. Released in 2022. It is also an open-source tool. In terms of the metrics returned, it is not as extensive as SonarQube and cannot be fully integrated into the development environment, but it can be accessed via GitHub. CodeClimate supports Ruby, Python, PHP, JavaScript, Java, TypeScript, GoLang, Swift, Scala, Kotlin, and C sharp. CodeClimate is not present in Fig. 1.2, as it is a fairly new tool, but the early impression is considerably good, and is also said to be fairly accurate in terms of measuring technical debt.

The last tool decided upon is Codiga[6]. Codiga has also been released recently, around 2020. Since its release it has quickly climbed the market. It is considered a top 10 tool in the field. It is a real-time static analysis tool, which can be either integrated into the development cycle using IDEs such as VS code, VS studio or JET beans or it can included using platforms such as GitHub, GitLab or BitBucket. Codiga supports many languages like C, shell, Dart, Scala, YML, JavaScript, Python, PHP, C++, Go, Apex, Java, Ruby, Kotlin and TypeScript. As seen in Fig. 1.2, Codiga is said to belong to the high-performance segment but is not considered a leader.

The tools selected are not very similar but not drastically different as well, it captures the current market well.

[4]SonarQube: https://www.sonarsource.com/products/sonarqube/.
[5]CodeClimate: https://codeclimate.com/.
[6]Codiga: https://www.codiga.io/.

1.3 Motivation

There has been drastic progress in the field of technical debt. The foundation studies such as the study of technical debt. Technical debt has existed from the beginning of software development, even though it has only lately become a formal notion. It has merely developed in form. Therefore what information can be understood from the prior versions of technological debt to enable us to get ready for its upcoming iterations? It can help identify several software system evaluations in recent software history. In this study, the author has looked at what technical debt has meant throughout the history of software systems and what it can signify going forward [7].

There are a few systematic literature reviews on the prioritisation of technical debt [8], Alfayez, Alwehaibi, Winn, *et al.* established 24 different approaches to maintaining reasonable technical debt. The examination of the listed methodologies showed a dearth of methods that take resource, cost, and value constraints into consideration as well as a dearth of industry review. Also, this Study identifies potential gaps in the existing technical debt prioritising research that future studies may investigate. There is another aspect people used systematic literature review on, for finding strategies, processes, tools and factors that impact technical debt [9]. When examining technical debt prioritisation, Code Debt and Architectural Debt have been the most commonly examined types of debt, while Test Debt and Requirement Debt are types of technical debt for which there is little evidence.

The majority of studies show that research on technical debt prioritisation is in its infancy and that there is no agreement on the key variables to include or how to assess them. As a result, the available study cannot be considered definitive.

There have been studies where different tools have been compared and analysed, only the most popularly used static analysis tools were selected for the study [10]. But, here only the results, that is the notion of technical debt. At first the analysis of the results between various tools and a thorough investigation of their precision that tool providers, practitioners, and academics may use to map the internal strengths of the tools and think about potential upgrades for the projects.

After looking at some other theoretical studies, there arises an interesting question, what low-level internal software metrics are responsible for the aggregated high-level metric, that is technical debt? This comes into the limelight as the aggregation of other metrics is not similar between all tools. None of the articles, research or studies tries to understand the lower-level root cause. It is believed that it will help better understand the cause of the increase in technical debt and help in general with writing code more efficiently, with the least amount of accumulating technical debt.

1.4 Problem Statement

Static Analysis tools are utilised by a user to establish high-level aggregated metrics for a software project. As there is no alignment or universal rules established for the calculation of such metrics, the notion of the scale of such metrics differs for each tool. The common denominator for all tools is the application/function/class level metrics, also known as low-level software metrics.

As low-level software metrics are not used as indicators or even displayed on the dashboard but are only used to find the aggregated metrics in the background. Even from the open-source tools, it is hard to estimate how the low-level internal software metrics impact the aggregated metrics as they are not even mentioned, and in general how the aggregated metric technical debt is calculated. Hence the questions that rise with different tools are:

- How different static code analysis tools give us the technical debt result for a project.
- How does the technical debt score compare between the tools?

This is important to understand how the tools achieve the technical debt score and the accuracy of the suggestions based on the pre-set rules.

My hypothesis is that there is a regression machine learning model, which is a technique for figuring out how independent features or predictors relate to a dependent prediction. Based on this, when low-level software metrics values are be used as predictors, they produce similar findings despite the variance in the different aggregated metrics values coming from different static code analysis tools.

These problems lead one to think about the following questions:

RQ1	What is the notion of technical debt across the selected static analysis tools?
RQ2	Which of the low-level software metrics are the most important predictors for technical debt and other aggregated quality goals?
RQ3	Which metric(s) correlate most significantly with technical debt?
RQ4	How confidently can a regression model trained on low-level software metrics for predicting technical debt achieve acceptable performance?

1.5 This Thesis Report

The projects and software metrics were collected from an already well-established and popular data set known as Qualitas.class Corpus data set. It contains real-world data for well know projects written in Java language.

To obtain answers to the research questions, multiple approaches were taken. First, Statistical analysis was conducted on the notions of technical debt to estimate if they are similar or not. **Pearsons sample correlation, Kendall correlation test, Spearman Correlation test, two-sample Kolmogorov–Smirnov test and Student's T-test** were conducted on the data collected from all three static analysis tools; **SonarQube, Code-Climate, Codiga**.

Afterwards for finding feature importance, multiple conceptually different machine learning models and methods were used such as **Linear Regression, RidgeCV, Random**

Forest, XGBoost and KNN models using **model coefficients, decision trees, permutation testing** methods.

and finally for predicting technical debt **Linear Regression, Random Forest, XGBoost and MLPRegressor** were used. The results were evaluated using MAE and RMSE metrics. The final conclusions were achieved using **Chebyshev's inequality** to obtain the Continuous Confidence Intervals.

1.6 Contributions

If the hypothesis is confirmed, it can contribute to the theoretical estimation of technical debt, as the calculation of the software metrics is definite, compared to the aggregated metrics estimated by different tools. This can be further used to either align the estimation of aggregated metrics or in general help align the predictability of technical debt. A common understanding can be found regardless of the domain of the project. Language still plays an important role, as the average value of each metric differs drastically for different languages. This can end the need to aggregate metrics for estimating technical debt. It can give a standardised way of maintaining source code quality no matter which tool is used as the user will be informed about which low-level software metric affects technical debt estimation the most. The most ideal outcome is where the predicted technical debt is more stable and consistent across all tools.

1.7 Target groups

The main target groups for this thesis are researchers, who are studying the impact and cause of technical debt, and developers, as this will help them archive a better project quality with better maintainability and companies/enterprises as low maintenance cost means less capital and time needs to be invested in the future for that project. There are three main target groups for this thesis.

1.8 Report Structure

This report's remaining sections are organised as follows: The research contributing to understanding the problems better is presented in Section 2. The technical background required to implement the experiment is presented in Section 3.

The methods of the experiments are described in Section 4. The actual implementation of the methods is described in Section 5. The results are presented and discussed in Section 66 In Section 7 the expected results and results achieved are discussed. Finally, in Section 8, conclusion to our findings and recommendations for further research.

2 Research for related studies

2.1 Research Protocols

To gain more knowledge about the problem and its causes of the problem, research was carried out. The research was divided into three sections.

The GQM method was followed for removing any kind of bias towards finding articles.
Goal: Finding how Software metrics affect the aggregated metric calculated by static code analysis tools
Question: Which Software metrics have an impact on aggregated quality goals?
Metrics: Quality goals and Estimated time for maintaining a project

2.1.1 Search Strategy

The planned to search across four search engines for papers and literature, they are IEEE Explore, ACM Digital, ScienceDirect and Google Scholar. These sources will be used to find papers and articles automatically using the following search strings:
Title: Static Analysis tools* OR Software metrics* OR Software Quality goals* OR Software Quality criteria* OR Technical debt*
Abstract: Software metrics* OR Low-level software metrics* OR Quality goals* OR Quality criteria*

A forward-backwards search for articles that are really interesting and relevant to the research questions, was carried out for the most relevant titles or referenced papers. Then after finding relevant papers and articles, there will be manual filtering to make sure that the paper is relevant as these are very open fields of research.

2.1.2 Selection Strategy

The selection of an article was based on the following conditions:

- Written in the English Language.
- Published between 1985 and the current period (February 2023). Such a huge range was set as software metrics were introduced in 1985 and technical debt was introduced in 1992.
- Papers that describe tools/software metrics in depth
- Papers that aim to give methods to resolve technical debt
- Papers that suggest how technical debt arises
- Papers that suggest the impact of metrics on the quality of the project

Articles were rejected if:

- Duplicate publications exist (The latest version was considered)
- Publications where the full paper cannot be located
- Paper does not have validated conclusion or results
- Database used is not mentioned or is not reliable or is for a very specific case

At first, only the abstract and conclusions were read, to make sure that the article is related to the topic, after compiling a list of articles, all of them were read in detail to extract the information required to answer questions that arose while researching about the field.

- What is the state of the art in the field of static code analysis tools?
- How are the aggregated metrics defined, Identified and Calculated?
- How are Software metrics defined, Identified and Calculated?
- How Software metric contribute to aggregated metric and Quality models?

2.2 Software metrics and Quality models

The first section is regarding, the software metrics, aggregated metrics and quality models for a software project.

A list of included articles can be found in Appendix A, which is a list of articles and studies that were found for this research section.
As mentioned in [11], software metrics are calculated in order to assess specific qualities of the created program. The most crucial criterion for software metrics is to offer data to enable quantitative management decision-making throughout the software life cycle, which has not been answered by the majority of them. Effective decision-making support requires help for risk estimation and mitigation. Managers that want to utilise measurement to analyse and reduce risk will find limited help from standard metrics techniques, which are frequently reliant on regression-based models for cost estimates and defect prediction. The future of software metrics is in the development of management decision-support tools that combine various facets of software development and testing and allow managers to make a variety of predictions, assessments, and trade-offs throughout the software life cycle using relatively simple existing metrics.

According to [12] the work of aggregating software measurements is difficult, and it becomes even more complicated when weights are taken into account to show the relative relevance of software metrics. As most of these weights are computed manually, the resulting models of subjective quality are challenging to understand. But it is clearly possible to utilise software metrics to gauge software quality as early as the requirements phase. Metrics can indicate possible problem areas at every stage of the development life cycle that could cause issues or errors. Identifying these regions early in the development life cycle lowers the cost and avoids potential rippling effects from the changes later on [13].

Much has been discovered and suggested since the early days of software quality, which were a few decades ago when the software's level of quality was considered to have reached its maximum potential as soon as it ran without crashing. First of all, software projects grew in complexity and interconnectedness with an expanding range of software layers. The same technique is then applied to the development of each layer, resulting in a complex and rich combination. Furthermore, software quality evaluation has expanded as well, including fresh ideas and methods and grouping them into standardised values. The main purpose of a quality model [14] is to give aggregated quality indicators, which are computed using quality measurements. This indicates that these indicators convert high-quality excellent practices into dependable and accepted computations.

All the articles helped me understand the role of software metrics, both at the application level and function/class level. As the metrics are used to estimate the quality of the project and aggregation of these function/class level metrics is not done in a definite manner, while quality models also differ for domains. The requirements of the architect also impact the results and evaluation process. It is essential to track these software metrics and maintain them in order to keep the quality of the project between good and excellent.

2.3 Technical Debt

The second section involves the questions needed for a deeper understanding of technical debt. The studies collected for these questions can be found in Appendix B.
Although technical debt is thought to be harmful to the main contributions of project management, it doesn't seem like academic literature fully understands it. The difficulty of identifying and managing technical debt is made more difficult by the lack of a precise definition and model, which prevents the realisation of technical debt's value as both a technical as well as conceptual medium of communication.

There has been an increase in the [15] development of a theoretical framework that offers a comprehensive view of technical debt, including a ranking of the phenomenon's measurements, attributes, precedents, and outcomes in addition to a taxonomy that categorises and includes its various manifestations, is a significant result of this research.

When looking at different perceptions of technical debt there are two different studies, first from the perspective of developers, and second from the perspective of enterprises.

The goal of the first study [16] was to determine how software engineers prioritise code technical debt in actual software projects.

To gather information from a broad range of open-source software projects, the authors conducted a poll. They used open coding, axial coding, and selective coding Straussian Grounded Theory methodologies to examine the data.

Participants were eager to pay off their technical debt when they made that decision. But, when they decided to forgo payment, it was frequent since the debt was created on purpose as a result of a project choice. As well, participants tended to select similar threshold values for their selections when utilising similar standards. Furthermore, the authors found that the techniques used to detect code technical debt need to be tailored to the project context for each software system.

In the second study, the authors mainly discuss the notion of reducing technical debt for the benefit of an enterprise. The decisions taken by project managers and developers to forego short-term gain in favour of a longer-term expense have been the subject of much of the technical debt study. Such a methodology might be unsuitable for enterprise software development, though. The authors [17] investigate the idea that corporate technical debt should be seen as a tool akin to financial leverage, enabling the company to take on debt to pursue possibilities that it otherwise couldn't afford. The authors put this notion to the test by speaking with a group of seasoned architects to learn how decisions about accumulating technical debt are made inside an organisation and how much leverage is gained.

The authors discovered that non-technical stakeholders, rather than technical architects, frequently decide to accrue technical debt for a project, either by initiating the accrual of new technical debt or by discovering previously hidden technical debt.

When faced with an enterprise-scale situation, with some preliminary findings and suggestions for corporations to effectively manage technical debt, as soon as possible.

The other important finding in this section of research was the financial cost of technical debt. The article [18] uses the findings of research on technical debt across 745 business applications with 365 million lines of code, conducted on data gathered from 160 organisations in 10 industry groups, which are summarised in this study. These programs were subjected to a static analysis, which assesses quality within and across application layers that might be written in several languages.

The study involves comparing the application to a database of more than 1200 guidelines for excellent architectural and coding standards. They provide a formula for computing technical debt that has movable parameters. Findings for technical debt for the full sample, as well as for various programming languages and quality criteria, are reported.

This study gives a greater understanding in terms of reasons, for resolving the issues causing an increase in technical debt.

2.4 Static Analysis Tools

The third and last section of this research involves a study of the available information regarding Static Analysis tools.
Static analysis is the technique used to obtain semantic information about a program at compilation time. As per an early study [19], there are two basic frameworks for undertaking static analysis: data flow analysis and abstract interpretation. The framework is not significant to this research, that's because the author shows that the two fundamental static-analysis tasks are tougher than previously accepted, independent of the framework utilised.

Where we also came across another study which helped us with the use of Static Analysis tools as a developer, [20] The authors have created a portrait as to how static analysis tools are utilised in the industry using a developer poll. When the viewpoint of the developer considered, it was discovered that they primarily employ analysis tools in their free time to fix warnings. Time limitations have a significant impact on the developers' objectives, process, and tool interactions.

To prioritise alerts, evaluate if they are true or false positives, and choose how to address them, developers develop internal knowledge and methodologies.

Based on this information, the researchers came up with ten recommendations using analysis tools, such as incorporating user-specific knowledge into the warning recommendation system, enabling users to encode useful heuristics in the analysis, and putting in place systems to deter harmful behaviour and promote beneficial behaviour, like collaboration and customisation, which are areas that are typically ignored in current static analysis research.

After further diving into the background functionality of the tools. It can be concluded that tools for static code analysis are rapidly being applied and used by software engineers in a variety of roles. Both existing and new tools are being developed.

Every day, tools get better, giving their users a higher-quality service [21]by giving them trust in the results of static analysis, allowing them to avoid some code flaws, and improving their software solution. With the ultimate goal of producing virtual assistants that should assist developers in the process of building high-quality software solutions, further development of tools for static code analysis entails their incorporation in development environments in the software development process.

The code review life cycle is not optimised to be the most efficient and Static Code Analysis tools can be easily integrated in this cycle. This will help the developers immensely. According to recent research in the field, they all jointly believe that the life cycle of code review is suggested in Fig. 2.3[22].

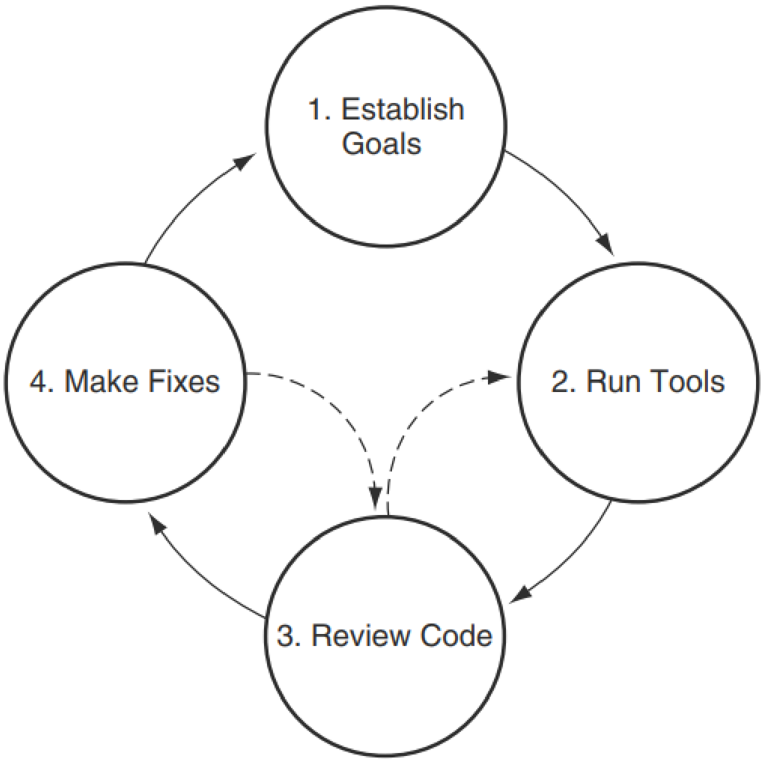


Figure 2.3: Life cycle of code review as of [22].

3 Background

3.1 Software metrics

The idea of excellence is varied, intricate, and ambiguous. Quality is subjective and relies on one's objectives and point of view. For various aims, the concept of quality may vary greatly. This is where software metrics come into use. Software metrics play an important role to measure every attribute and project success in the software development context.

Software metrics are measurements of quantifiable or countable software attributes. Measuring software performance, organizing tasks, gauging productivity, and many more purposes make software metrics crucial. The generalised classification of metrics can be seen in Fig. 3.4[23]

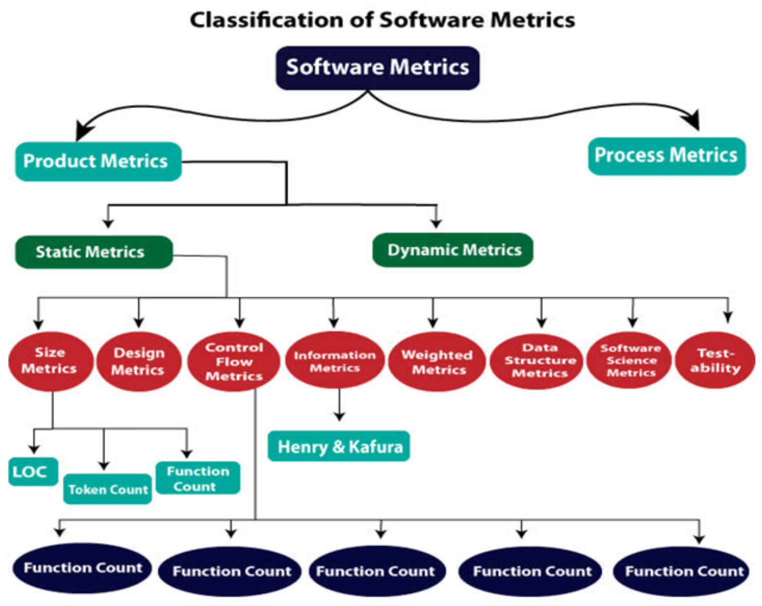


Figure 3.4: Classification of software metrics.

There are mainly three types of metrics:
Internal Metrics: Internal metrics are used to measure characteristics that software developers believe to be more important. LOC, for instance, is a measurement.
External Metrics: External metrics are those that are intended to gauge characteristics that are seen to be more crucial for the user, such as portability, dependability, usefulness, usability, etc.
Hybrid Metrics: Product, process, and resource metrics are all included in hybrid metrics. Cost per Function Point Metric, for instance.

3.2 Aggregated metrics

The project manager monitors the project's progress using project metrics. This allows the user to compare the effort, cost, and time as the project develops. This indicates that the measurements are employed to reduce risks, time commitments, and development costs further down the road.

Additionally, the project's standards can be raised. The number of mistakes, necessary time, and funds required to update or maintain a project decrease as the base standard of the project rises.

Hence, Software quality metrics or Aggregated metrics can be further divided into three categories:

- Product quality metrics
- In-process quality metrics
- Maintenance quality metrics

Road to Maintenance quality metric

Metric Type	Attribute	Software Metric
Process Measurement	Speed	Lead Time
		Cycle Time
		Sprint Burndown
Product Measurement	Size	Kilo Lines of Code
		Number of Attributes
		Number of Methods
	Complexity	Cyclomatic Complexity
		Depth of Inheritance Tree
		Lack of Cohesion of Methods
	Quality	Number of defects
		Code Duplication
Resource Measurement	Software	Reliability
		Usability
		Size

The attributes are considered as the aggregated quality metrics. Each tool calculates the attribute in a different manner but the basic metrics stay the same, ideally.

3.3 Statistical Tests

3.3.1 Pearson Correlation test

The Pearson sample correlation coefficient[7] is a descriptive statistic, which means it enumerates a data set's features. It explains specifically the magnitude and direction of the linear relationship between two quantitative variables. Despite the fact that how the link is interpreted varies among disciplines.

$$r_{xy} = \frac{n\sum x_i y_i - \sum x_i \sum y_i}{\sqrt{n\sum x_i^2 - (\sum x_i)^2}\sqrt{n\sum y_i^2 - (\sum y_i)^2} \tag{1}$$

[7]Pearson correlation coefficient: https://en.wikipedia.org/wiki/Pearson_correlation_coefficient#For_a_sample.

- r = Pearson's correlation coefficient
- xi, yi = individual sample points indexed with i
- n = sample size

Where r=0 denotes no association and ranges between -1 and +1. A precise linear relationship is implied by correlations of -1 or +1. Positive correlations suggest that while x grows, y grows as well. Negative correlations suggest that when x rises, y falls.

In addition to this, the 2-tailed p-value is also calculated. The p-value, in general, denotes the likelihood that an uncorrelated system will generate datasets with Pearson correlations at least as extreme as those calculated from these datasets.

$$t = \frac{r\sqrt{n-2}}{\sqrt{1-r^2}} \tag{2}$$

The p-value is 2 × P(T > t), where T has n - 2 degrees of freedom and follows a t distribution. Ideally at a significance level(p-value) of < 0.05, this would disprove the null hypothesis.

3.3.2 Kendall Correlation test

Kendall's rank-order correlation[8] is often referred to as Tau's Kendall coefficient. Based on ranked data, Tau's coefficient evaluates statistical association.

A non-parametric technique for assessing the relationship between ranked data columns is Kendall's correlation. When two columns are correlated, Kendall's coefficient returns 1 to represent the highest correlation and 0 to display the lowest correlation.

$$t = \frac{n_c - n_d}{n_0} \tag{3}$$

- t = Kendall's rank-order correlation coefficient
- n = number of pairs
- no = n(n-1)/2
- nc = Number of concordant pairs
- nd = Number of discordant pairs

$$Z_A = \frac{3(n_c - n_d)}{\sqrt{n(n-1)(2n+5)/2}} \tag{4}$$

One calculates ZA and determines the cumulative probability of a standard normal distribution to determine if two variables are statistically dependent.

[8]Kendall's rank-order correlation: https://en.wikipedia.org/wiki/Kendall_rank_correlation_coefficient.

3.3.3 Spearman Correlation test

The monotonicity of the relationship between two datasets is measured by the nonparametric Spearman[9] correlation. The Spearman correlation does not presuppose that the datasets are regularly distributed, in contrast to the Pearson correlation. This correlation coefficient, like others, ranges from -1 to +1, with 0 denoting no correlation. A precise monotonic relationship is implied by correlations of -1 or +1. Positive correlations suggest that while x grows, y grows as well. Negative correlations suggest that when the x value rises, the y value falls.

$$r_s = \rho R(X), R(Y) \frac{cov(R(X), R(Y))}{\sigma R(X) \sigma R(Y)} \tag{5}$$

- rs = Spearman's Correlation correlation coefficient
- ρ = stands for the standard Pearson correlation coefficient but uses the rank variables.
- R = dentones rank of the varaible
- X, Y = individual raw scores
- cov = covariance
- sigma = standard deviations

To estimate the statistical significance, it uses the same method as the Pearson correlation coefficient, as mentioned in 2.

3.3.4 Kolmogorov–Smirnov test

The two-sample Kolmogorov–Smirnov statistic[10] measures the difference between two samples' empirical cumulative distribution functions [24]. The samples were taken from the same distribution, which is the null hypothesis under which the null distribution of this statistic is produced.

$$D_{n,m} = sup_x |F_{1,n}(x) - F_{2,m}(x)| \tag{6}$$

- D_{nm} = The Kolmogorov-Smirnov 2 sample test coefficient
- sup = supremum function
- n = sample one size
- m = sample two size
- F1, F2 = empirical distribution functions of the first and the second sample respectively
- x = observation value

[9]Spearman Correlation test: https://en.wikipedia.org/wiki/Spearman%27s_rank_correlation_coefficient.
[10]Kolmogorov-Smirnov test: https://en.wikipedia.org/wiki/Kolmogorov%E2%80%93Smirnov_test.

The null hypothesis is rejected if:

$$D_{n,m} > c(\alpha)\sqrt{\frac{n+m}{n.m}} \tag{7}$$

- $c(\alpha)$ = similarity significance

3.3.5 Student's T-test

A two-sample location Student's T-test[11] is a test of the null hypothesis using two samples where the means of the two populations are equal. This test is appropriate only when the variances for the two populations are believed to be comparable.

This test is only applied when it is reasonable to believe that the variance of the two distributions is the same. The following formula can be used to compute the t statistic to determine whether the means are different:

$$t = \frac{\overline{X_1} - \overline{X_2}}{s_p\sqrt{\frac{1}{n_1} + \frac{1}{n_2}}} \tag{8}$$

Where:

$$s_p = \sqrt{\frac{(n_1 - 1)s_{x_1}^2 + (n_2 - 1)s_{x_2}^2}{n_1 + n_2 - 2}} \tag{9}$$

- t = The Kolmogorov-Smirnov 2 sample test coefficient
- X_1,X_2 = sample 1 mean, sample 2 mean
- s_p = pooled standard deviation of the two samples
- n_1,n_2 = sample 1 size, sample 2 size
- s = sample standard deviation

The estimated p-value is rejected in favour of the alternative hypothesis if it is less than the selected level of statistical significance. It is computed utilising the value of s_p if the variance is believed to be similar.

3.4 Models used for Feature Importance

Indicating the relative importance of each feature while producing a forecast, feature importance refers to a set of strategies for assigning scores to input features to a predictive model. For problems involving the prediction of a numerical value (regression problems), feature significance scores can be computed.

The scores are helpful and can be applied to a variety of predictive modelling issues, including:

- A better understanding of the data.
- Reducing the number of input features.

[11] Student's T-test: https://en.wikipedia.org/wiki/Student%27s_t-test.

Scores for feature relevance can shed light on the dataset. The features that may be most or least relevant to the objective might be highlighted by the relative scores, and vice versa. A subject matter expert may interpret this and use it as the foundation for collecting more or alternative data.

There are three different advanced methods to get feature importance:

- From model coefficients.
- From decision trees.
- From permutation testing.

From model coefficients: In a model where the predicted value is a weighted average of the values provided as inputs, linear machine learning techniques fit the data.

Examples include logistic and linear regression, as well as regularisation-adding extensions like ridge regression and the elastic net.

To create a forecast, each of these algorithms identifies an array of coefficients to add to the weighted total. One can utilise these coefficients directly as a basic kind of feature importance score.

From decision trees: The reduction in the criterion used to choose split points, such as Gini or entropy, is used to calculate important scores for ensembles of decision trees, such as the random forest and stochastic gradient boosting algorithms.

From permutation testing: A method for determining relative importance scores that is unrelated to the model being employed is called permutation feature importance.

First, a model that does not support native feature importance scores is fitted to the dataset. The model is subsequently applied to a dataset to create predictions, despite the dataset's values for a feature (column) being mixed up. For every feature in the dataset, this is repeated. The average importance score for every feature of the input is the result that is calculated.

This method, which can be applied to regression, calls for the selection of a performance metric as the foundation of the importance score, such as the mean squared error in the case of regression.

3.4.1 Linear Regression

In linear regression[12], linear predictor functions are used to model relationships, with the model's unknown parameters being estimated from the data. These models are referred to as linear models.

The conditional mean of the response is typically considered to be an affine function of the values of the explanatory variables (or predictors); the conditional median or another quantile is occasionally employed. In common with all other types of regression analysis, linear regression concentrates upon the conditional probabilistic distribution of the outcome provided the values of the predictors rather than the joint probability distribution of all these variables, which is the purview of multivariate analysis. A graphical depiction of the fundamentals of a Linear Regression can be seen in Fig.3.5[13]

[12]Linear Regression: https://en.wikipedia.org/wiki/Linear_regression.

[13]Fig.3.5: https://www.spiceworks.com/tech/artificial-intelligence/articles/what-is-linear-regression/.

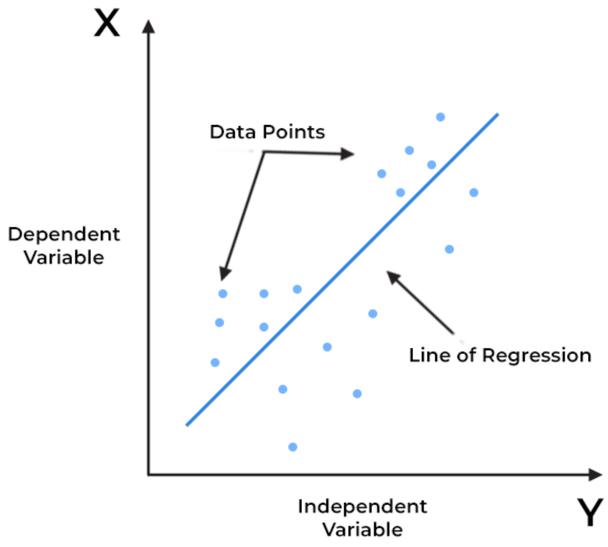


Figure 3.5: Linear Regression

3.4.2 RidgeCV

Ridge regression[14] is where the penalty applied to coefficient magnitude is equal to the square of the coefficients, this results in sparse models. RidgeCV implements ridge regression by default and includes built-in cross-validation for the alpha value. Alpha values denote the penalty for amount of shrinkage. The only difference is that it always uses Leave-One-Out cross-validation. K-fold cross validation is also known as leave-one-out cross validation, where K is equal to N, the total number of data points in the set. An example of RidgeCV results can be seen in Fig.3.6[15]

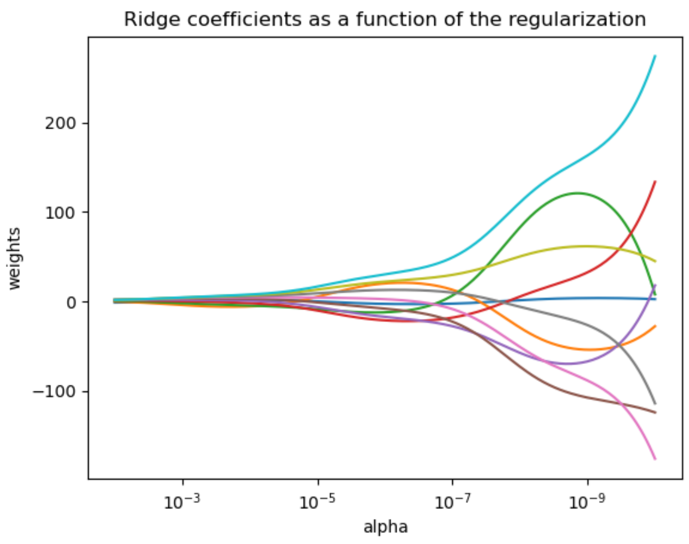


Figure 3.6: Ridge Regression with cross-validation

[14]RidgeCV: https://towardsdatascience.com/the-power-of-ridge-regression-4281852a64d6.
[15]Fig.3.6: https://machinelearninghd.com/ridgecv-regression-python/.

3.4.3 Random Forest

With the aid of several decision trees and a method known as Bootstrap and Aggregation, also referred to as bagging, Random Forest is an ensemble methodology capable of handling both regression and classification tasks. This method's fundamental principle is to integrate several decision trees to get the final result rather than depending solely on one decision tree. Multiple decision trees serve as the fundamental learning models in Random Forest.[16]

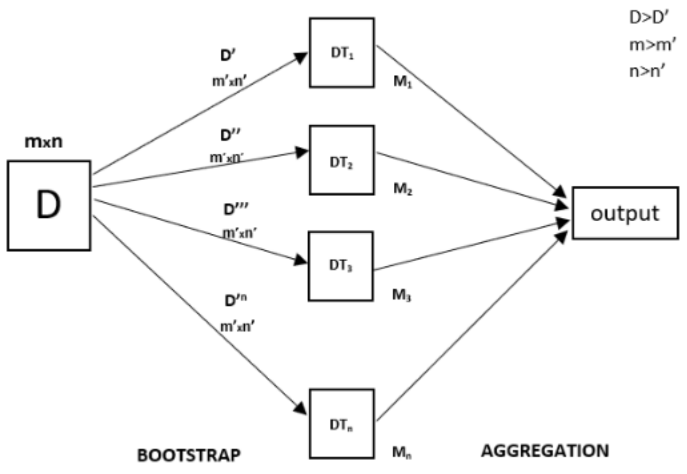


Figure 3.7: Random Forest

Where each decision tree follows a similar method as seen in Fig.3.8[17]

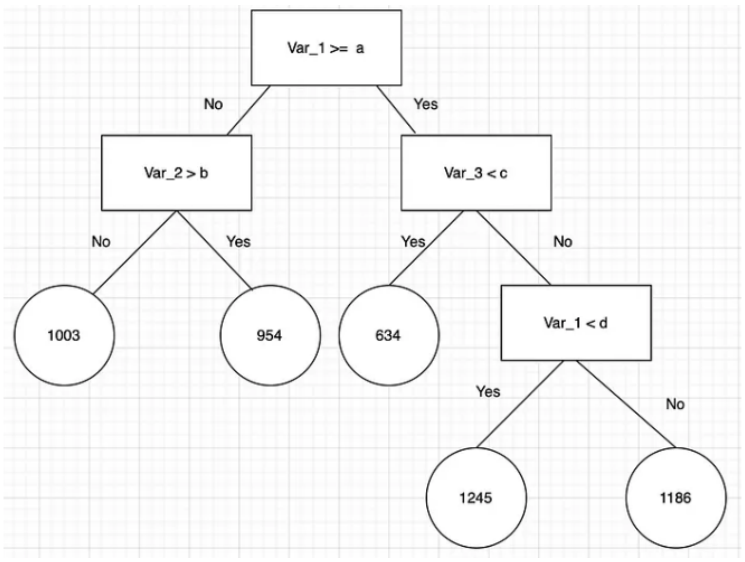


Figure 3.8: Decision Tree

[16]Random Forest: https://www.geeksforgeeks.org/random-forest-regression-in-python/.
[17]Fig.3.8: https://towardsdatascience.com/random-forest-regression-5f605132d19d.

3.4.4 XGBoost

Extreme Gradient Boosting (XGBoost)[18] is a distributed, scalable gradient-boosted decision tree (GBDT) machine learning framework. The top machine learning package for regression, classification, and ranking issues, it offers parallel tree boosting.

comprehending the machine learning ideas and techniques that supervised machine learning, decision trees, ensemble learning, and gradient boosting are built upon is essential to comprehending XGBoost.

In supervised machine learning, a model is trained using algorithms to discover trends in a dataset of features and labels, and the model is then used to forecast the labels on each feature of a new dataset.

The concept of "boosting" or strengthening one poor model by merging it with a number of additional weak models in order to get a cumulatively strong model. As a variant of boosting, gradient boosting formalises the process of cumulatively creating weak models as a gradient descent method over an objective function.

3.4.5 KNN

The k-nearest neighbors algorithm or KNN for short, is a supervised learning regressor that employs proximity to produce predictions about how a particular data point will be predicted. KNN method is a member of the "lazy learning" model family because as it memorises the training data set rather than learning an indicator function using the training data.

The 'feature similarity' component of the KNN method is used to forecast the values of any additional data points. In other words, the value given to the new point depends on how much it resembles the points in the training set.

The distance between the query point and the other data points must be determined in order to determine which data points are closest to a specific query point. Most frequently, euclidean distance is employed.

3.5 PCA

Principal component analysis, or PCA[19], is a method for reducing the number of dimensions in large data sets by condensing a large collection of variables into a smaller set that retains the majority of the large set's information.

Accuracy naturally suffers as a data set's variables are reduced, but the answer to dimensionality reduction is to sacrifice a bit of accuracy for simplification. Because machine learning algorithms can analyse data points considerably more quickly and easily with smaller data sets because there are fewer irrelevant variables to process.

[18]XGBoost: https://www.nvidia.com/en-us/glossary/data-science/xgboost/.

[19]PCA: https://builtin.com/data-science/step-step-explanation-principal-component-analysis.

3.6 Models used for error generalisation

The only new model added to the evaluation process in this part of the experiment was MLPRegressor model.

MLPRegressor : A potent machine learning algorithm for regression tasks is Sklearn's MLPRegressor[20]. It offers a high level of accuracy and has the ability to work with complicated, non-linear datasets. In order to increase prediction accuracy, the MLPRegressor artificial neural network model employs backpropagation to modify the weights between neurons.

MLPRegressor can handle data with a lot of features, which is one of the key advantages it has over other regression algorithms. With datasets comprising tens of thousands of features, ordinary regression models would struggle, but MLPRegressor can quickly identify trends and make precise predictions.

3.7 Methods for estimating the error

3.7.1 Distributions

PDF : The probability density function (PDF)[21], in contrast to a continuous random variable, is a statistical expression which describes a probability distribution (the probability of a certain occurrence). A discrete random variable differs from a continuous random variable in that the variable's precise value may be determined. A typical example of a PDF that creates the well-known bell curve shape is the normal distribution.

CDF : The chances that a random variable will have values less than or equal to x are expressed by a cumulative distribution function (CDF)[22]. Because it adds the total probability up to that point, it is a cumulative function. It always produces a value between 0 and 1. Both discrete and continuous variables have cumulative distribution functions. Although discrete functions represent the probabilities of every discrete value below or equal to each value, continuous functions use integrals for determining solutions.

Probabilities for random variables are provided via cumulative distribution functions and probability density functions (PDFs). However, CDFs provide the chances for $\leq x$, whereas PDFs compute probability densities for x.

3.7.2 Metrics

It is particularly advantageous to have a single number that needs interpretation, such as a score, for evaluating a model's performance in machine learning, whether during training, cross-validation, or monitoring after deployment. Hence the following metrics were used to evaluate the experiment:

[20]MLPRegressor: https://vitalflux.com/sklearn-neural-network-regression-example-mlpregressor/.
[21]PDF: https://en.wikipedia.org/wiki/Probability_density_function.
[22]CDF: https://en.wikipedia.org/wiki/Cumulative_distribution_function.

MAE : A measure of mistakes between paired observations reflecting the same phenomena in statistics is called mean absolute error (MAE)[23]. The mean or average absolute variance between X and Y is known as MAE. Additionally, any error influences MAE in a proportional manner to its absolute value.

$$MAE = \frac{\sum_{i=1}^{n} |y_i - x_i|}{n} \tag{10}$$

RMSE : The quadratic mean of the differences between predicted values and observed values, or the square root of these differences, is what is represented by the RMSE[24]. When computations are made outside of the data sample that was used for estimation, the deviations are referred to as errors (or prediction errors) instead of residuals.

$$RMSE = \sqrt{\frac{\sum_{t=1}^{T} (\hat{y}_t - y_t)^2}{T}} \tag{11}$$

Since it is important for evaluating the experiment and have more interpretability, MAE was used in place of MSE because we don't want to train a model that focuses on reducing large outlier errors.

The RMSE is used to combine the sizes of prediction mistakes for different data points into a single indicator of predictive power. Since RMSE is scale-dependent, it should only be used to compare forecasting errors of various models for a single dataset and not between datasets. Therefore, even though researchers recommended using MAE over RMSE for the experiment, both were taken into consideration.

3.7.3 Continuous Confidence Intervals

The mean of your estimate plus and minus the range of that estimate constitutes a confidence interval. Within a certain level of confidence, this is the range of values you anticipate your estimate to fall within if you repeat your test.

Chebyshev's inequality : According to Chebyshev's inequality, no more than a certain number of values can be more than a certain distance from the mean for a wide class of probability distributions [25]. When combined with other theorems, Chebyshev's inequality is a very helpful theorem and the foundation of confidence intervals. The probabilistic equation is as follows:

$$Pr(|X - \mu| \geq k\sigma) \leq \frac{1}{k^2} \tag{12}$$

- X = random variable
- μ = expected value of X
- k = k > 0, the number of standard deviations, only values > 1 are useful
- σ = standard deviation of X

[23]MAE: https://en.wikipedia.org/wiki/Mean_absolute_error.
[24]RMSE: https://en.wikipedia.org/wiki/Rootmeansquare_deviation.

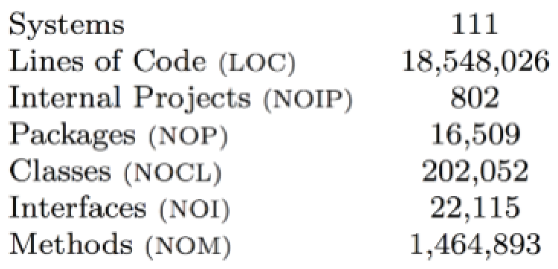
4 Method

First a statistical test was performed to determine the correlations between each notion of technical debt in order to examine the viability of forecasting distinct notions of technical debt using software metrics. Ultimately, a methodical evaluation test was conducted to determine the reliability of various machine-learning models for deriving technical debt estimates from software metrics.

One can argue that choosing only Java projects introduces a certain amount of bias and prevents the results from being definitive and generalizable for multiple languages, but as it contains multiple domains of projects, it removes some bias regarding the domains. The dataset is already well-established and is utilised for numerous other research projects, which is the justification for choosing this particular list of projects. As a result, the values can be regarded as accurate and free of nominal errors.

4.1 Data Collection and Preparation

In order to conduct empirical research on code artefacts, an assembled set of software systems has been accumulated, known as the Qualitas Corpus [26], [27]. The main objective is to offer a resource that aids with software study replication. The present edition of the corpus includes many open-source Java software platforms. The University of Auckland's Ewan Tempero is presently responsible for maintaining the Qualitas.class Corpus. The summary of the data set can be seen in Fig. 4.9 [26] and the available metrics for each dataset can be seen in Table. 4.1 [26]. The quantiled QCC data and all notions of technical debt can be found herehttps://zenodo.org/record/7981663



Systems	111
Lines of Code (LOC)	18,548,026
Internal Projects (NOIP)	802
Packages (NOP)	16,509
Classes (NOCL)	202,052
Interfaces (NOI)	22,115
Methods (NOM)	1,464,893

Figure 4.9: Summary of Qualitas.class Corpus

List of available metrics

Basic Metrics:	**Complexity Metrics:**
• Number of Lines of Code (LOC) • Number of Packages (NOP) • Number of Classes (NOCL) • Number of Interfaces (NOI) • Number of Methods (NOM) • Number of Attributes (NOA) • Number of Overridden Methods (NORM) • Number of Parameters (PAR) • Number of Static Methods (NSM) • Number of Static Attributes (NSA)	• Method Lines of Code (MLOC) • Specialisation Index (SIX) • McCabe Cyclomatic Complexity (VG) • Nested Block Depth (NBD) • Normalised Distance (RMD)
CK Metrics:	**Coupling Metrics:**
• Weighted Methods per Class (WMC) • Depth of Inheritance Tree (DIT) • Number of Children (NOC) • Lack of Cohesion in Methods (LCOM HS)	• Afferent Coupling (CA) • Efferent Coupling (CE) • Instability (I) • Abstractness (A)

While Qualitas.class corpus [26] provides the compiled versions of the projects, they are not compressed in a zip format hence each project needs to be downloaded individually. After collecting all the projects into one location, three different static analysis tools, ranging from the most well-known and well-established in the business to relatively new yet validated tools, as described in Section.1.2, were used to acquire various conceptions of technical debt. Testing every project and obtaining data from every tool was not practical.

Regarding scanning, SonarQube allows projects to be submitted locally, by simply uploading them on a local server, while Codiga and CodeClimate demanded that the projects be uploaded to a Git repository. SonarQube and Codiga had REST APIs for data retrieval, while CodeClimate needed manual contributions. In terms of other limitations regarding data collection, Codiga and CodeClimate both have a limit of 100 projects.

4.2 Scientific approach

Answering concerns about applied machine learning (ML) requires systematic experimentation. Repeated trials are necessary to establish the expected result and its range of variation because the findings of one test are probabilistic and subject to variance.

In systematic experiments, we have total control over the surroundings and run numerous tests to learn from the outcomes. To ascertain their impacts on the dependent variable, the independent variables are changed one at a time while the dependent variables are held constant. The software metrics, randomness, usage of the domain as a quantitative value, PCA components, model architectures, model hyperparameters, and experiment objectives are the covariates in this study. The estimated prediction performance on unobserved data is the dependent variable.

4.2.1 Experiment objectives

The first objective was to quantitatively distinguish the similarities or differences in all notions of technical debt.

The second objective was to find which software metrics are the most important predictors for estimating technical debt and how they correlate with each notion of technical debt.

The final and main objective of this experiment was to determine how confident we are in predicting the notion of technical debt for any project without using the project metrics estimated by each tool. As mentioned before these project metrics vary as the method for calculation is either not disclosed or very different compared to other tools.

Therefore, software metrics acquired from Qualitas.class corpus [26] were used. These values are said to be the ground truth and there is no variance even if the same metrics were to be calculated by an individual again. All available metrics, for example, LCOM the number of available measurements are not fixed for each project as it depends on the size of the project, hence to get a better understanding of these values, each metric was divided into a range of quantiles plus standard deviation to obtain fixed-length feature vectors.

Using the quantiles of software metrics and each notion of technical debt, a dataset was created hence in total there are three datasets made and used for the experiment.

4.2.2 Statistical analysis of Technical Debt

To answer research question 1; What is the notion of technical debt across the selected static analysis tools? In total five different tests are carried out to understand the notions of technical debt better statistically. Where the quantitative results tell us how the notions are distributed, using which a quantitative judgment can be made of whether the notions are similar or at least fall on the same plane or not.

4.2.3 Model Selection for Feature Importance

To keep the results unbiased for research question 2; Which of the low-level software metrics are the most important predictors for technical debt and other aggregated quality goals? A total of five models were selected which estimate feature importance using all most known methods for regression as mentioned in Section.3.4.

The resulting quantitative values were absolute as mainly to see the impact of each software metric regardless if it is a negative or positive impact.

4.2.4 Correlation Metrics of Software metrics vs Technical debt

To obtain an answer for research question 3; Which metric(s) correlate most significantly with technical debt? A simple correlation metric was created between the quantiles of software metrics and all notions of technical debt. The result was also quantitative to interpret. This was important to understand the impact of software metrics on each notion of technical debt, as one metric could be really important as it is highly correlated for predictions for one notion, but not important at all for all the other notions.

4.2.5 Adaptive training for estimating generalisation error

To obtain a probabilistic answer to research question 4; How confidently can a regression model trained on low-level software metrics for predicting technical debt achieve acceptable performance? To achieve the distributions of all notions of technical debts PDF and CDF plots were made. After which four possible combinations were run with 1000 seeds each; with domain with PCA, with domain without PCA, without domain with PCA and without domain without PCA.

As four models are used for predicting, mostly for removing any sort of biases and two metrics were calculated; RMSE and MAE, hence it results in 4000 RMSE and MAE values for each combination. The lowest MAE values were considered for each notion of technical debt achieved from the three different tools. Using the methods described in research by Hönel, Ericsson, Löwe, *et al.* [28], mostly using CDF transformation, which mainly uses the CDF values and the metric values and then normalises them, this way we get scores and can compare all combinations. This was done for

After this using Chebyshev's inequality the Champion model's MAE was compared for the predicted values. This gives us the confidence level by which we can predict technical debt and also the possible lowest and highest difference in the prediction.

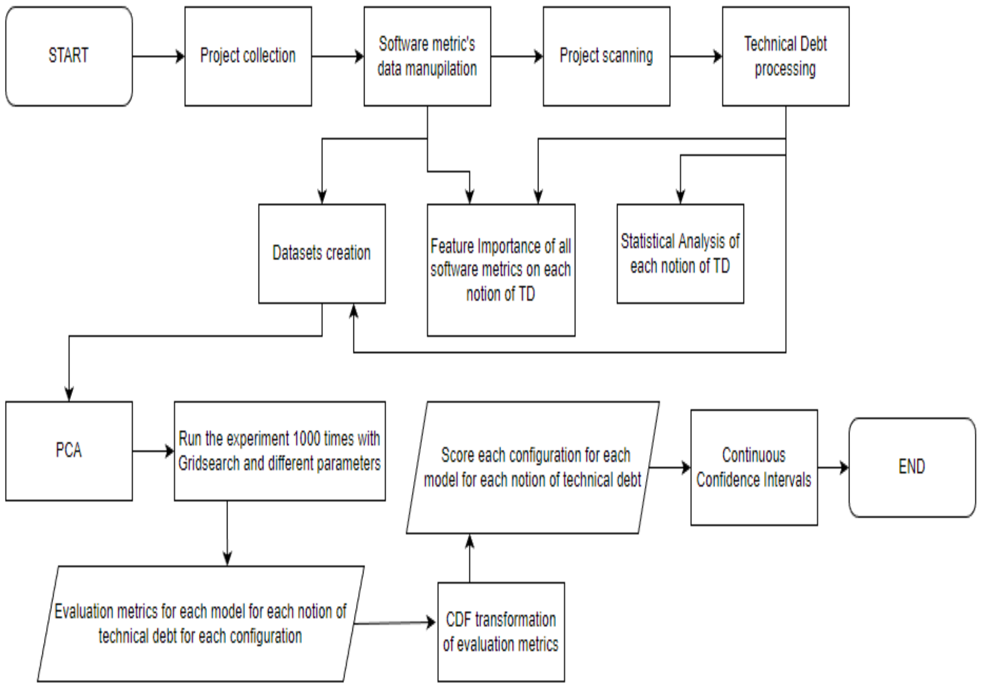

START
Project collection
Software metric's data manupilation
Project scanning
Technical Debt processing
Datasets creation
Feature Importance of all software metrics on each notion of TD
Statistical Analysis of each notion of TD
PCA
Run the experiment 1000 times with Gridsearch and different parameters
Score each configuration for each model for each notion of technical debt
Continuous Confidence Intervals
END
Evaluation metrics for each model for each notion of technical debt for each configuration
CDF transformation of evaluation metrics

Figure 4.10: Method flowchart

4.3 Reliability and Validity

Machine learning relies heavily on randomness, which makes it difficult to reproduce results. But as all random seeds have been pre-set, reproducing the exact same result is not going to be possible, but similar results can be achieved. Even pre-defining the random states and random seeds for splits, the parameters chosen after the grid search can be slightly different and the champion model can differ to some extent. However, by repeating the experiments many times using bootstrapping, we can obtain estimates for the robustness, and the generalisation error, of the trained models.

Nevertheless, the data has been made public and the method is described in detail, hence if not the same but similar results can be achieved easily.

We have tried to use different models, and a grid search to make the models as valid as possible. The only other validity issue we believe arises is that the projects used are fairly old, although the latest versions were considered, but they were all Java projects, maybe for a different language the results could vary. While that is true, you could easily gather a similar set of metrics for many other languages. The point was to show that your approach works (to some degree).

5 Implementation

In this section, we describe the implementation of the methodology described in the last section.
The experiment was conducted in Jupyter Notebooks. In total, there are six different Notebooks, one for collecting data from tools and pre-processing the collected data, one for Data engineering where data is structured in the required format, then one for each research question.

5.1 Technical Requirements

The first two research questions experiments were conducted locally on an AMD Ryzen 9 4900 HS with 8 cores and an NVIDIA GeForce RTX 2060 with Max-Q Design. For the third research question, an Epyc 7742 64-core processor with 256GB Ram was used, and the computation of the entire grid took $\approx$12 hours. The experiment is reproducible as none of the data is randomised except for the splitting of data for training and testing, but in this case, it was done deterministically, and eventually, a grid of models was run as described in the Section.4.2.5. The **random state** was determined using the pre-defined iteration value, **test size** for repeated cross-validation was **25%** and **train size** was **75%** and **stratify**, which makes sure that both train and test data have at least some part of one class from the predictors was set to **Domain**.
3.11.1 version of Python was used with the following libraries:

- pandas [29]
- numpy [30]
- matplotlib [31]
- seaborn [32]
- scipy [33]
- sklearn [34]

5.2 Data Preparation

For each project, for example, ant-1.8 has multiple instances of metrics except certain metrics; NOP and TLOC. The other metrics have a number of instances that are not constant across all projects. Hence to obtain fixed-length feature vector values .01, .2, .4, .5, .6, .8, and .99 quantiles in addition to these standard deviation value was calculated, this creates X, predictor features.

After collecting and scanning all projects with individual tools, all notions of technical debt were converted into the number of hours needed to maintain the project. SonarQube gives sqale index (technical debt) in terms of minutes, hence the column is divided by 60, whereas Codiga measures technical debt in terms of money, considering 70 dollars per hour, hence the notion of debt from Codiga was divided by 70 to get hours, and CodeClimate estimates technical debt in terms of numbers of hours so no changes were required. All three notions make up the Y1, Y2, and Y3 respectively, they are considered as response features or the true output, of which the aim is to predict it.

5.3 Statistical Tests

Without any centralising or standardisation, all notions of technical debt were compiled in one data frame. Where multiple statistical tests were conducted against 2 notions of technical debt.
The following are the combinations of technical debt notions (as A vs B is the same as B vs A):

- SonarQube vs Codiga
- SonarQube vs CodeClimate
- Codiga vs CodeClimate

Each combination was tested for the following statistical tests, using the stats package from Scipy, mainly to determine if they are similar or not:

- Pearson correlation test
- Kendall correlation test
- Spearman correlation test
- Kolmogorov—Smirnov test [24]
- Student's T-test

5.4 Feature Importance

Before using the data for estimating feature importance, all the data was centred and scaled using Z-standardisation. Each notion of technical debt was standardised separately, so no similar data was available between training and testing. No splitting of data was done at this part of the experiment so the models can use all the features for all records and give a better result.

From all the features created, to find the most important features for training the model, five models were selected.

- Linear Regression Model
- RidgeCV model
- Random Forest model
- XGBoost model
- KNN model

Each Model was run with X and each notion of technical debt, that is Y1, Y2, and Y3. That means for Y1 there were 5 models, the results were averaged to find the most important features for Y1, after obtaining the averages for all the Ys, the result for further averaged to have a non-biased result.

5.5 Correlation Matrix

After determining the relation between the notions of technical debt. We check for what features most or least correlate to each notion of technical debt. For this seaborn heatmap was used.

5.6 Regression Models to estimate generalisation error

5.6.1 Domain

After a few trials, it was understood that if the models are given some context of where the metrics come from, that in this case was the domain, the models performed differently. Hence considering it was considered a parameter itself. When it was used it was encoded from categorical to numerical data.

5.6.2 PCA

As the number of features in quantile software metrics dataset is high, PCA was considered as a parameter as well. To make sure not all components were used, as soon as the explained variance reaches 95%, that component was considered.

5.6.3 Making the grid

In the beginning, a for loop was made for in the range of 0 to 1000, where the value is also the random seed value and random split value, to make train and test data sets. After fitting the train data set to the Standard scalar, both train and test data were transformed.

If Domain was set to True, then the domain was encoded, else after the domain was used to stratify the split, it was dropped.

Then if PCA was set to True, the train part of the quantile software metric dataset was used to fit on which test data was transformed. Each component's explained variance was calculated and cumulatively summed. For the first component, the sum is more than 0.95, that component was considered to transform the original train and test data.

Considering each combination, a grid search was conducted for all models:

- RandomForest
- XGBoost
- MLPRegressor
- LinearRegression

RandomForest parameter grid (4800 **permutations**):

- Max depth: [5, 10, 50, 80]
- Max features: [2, 3, 4, 5]
- Min samples Leaf: [2, 3, 4, 5, 6]
- Min sample split: [2, 3, 4, 6, 8]
- number of estimators: [1, 5, 10, 15]

XGB parameter grid (2400 **permutations**):

- loss: ['absolute_error', 'squared_error']

- learning rate: [0.01, 0.12, 0.13, 0.14]
- Max depth: [2, 4, 6, 8, 10]
- Min sample split: [0.5, 2, 3,4]
- number of estimators: [10, 25, 50, 80, 100]

MLP parameter grid (360 **permutations**):

- hidden layer sizes: [(64,16), (64,8), (32,16),(32,8),(16,4)]
- max_iter: [50, 100, 150]
- Solver: ['sgd', 'adam']
- alpha: [0.0001, 0.05]
- learning rate: ['constant', 'adaptive']

LinearRegression parameter grid (12 **permutations**):

- fit_intercept: [True, False]
- copy_X: [True, False]

Parameters:

- number of estimators: number of trees.
- Max depth: maximum depth of trees.
- Max features: number of features to look at when looking for the best split.
- Min samples Leaf: number of samples required to be considered as a leaf node.
- Min sample split: number of samples required to split internal node.
- loss: loss functions (evaluating function to test fit of model on dataset) to be used.
- learning rate: how quickly should the model adapt to solving the problem
- hidden layer sizes: number of neurons per hidden layer
- max_iter: number of max iterations before the model converges or max iteration is reached.
- Solver: how to optimise weights.
- alpha: strength of regularisation.
- fit_intercept: if intercept should be calculated for the model or not
- copy_X: if copy values of the features for new trees or overwrite them

After the training, the predicted values were reverse transformed, and then MAE and RMSE values were calculated. All of this data; seed, Domain boolean, PCA boolean, model, best model parameters, MAE and RMSE are stored in a data frame.

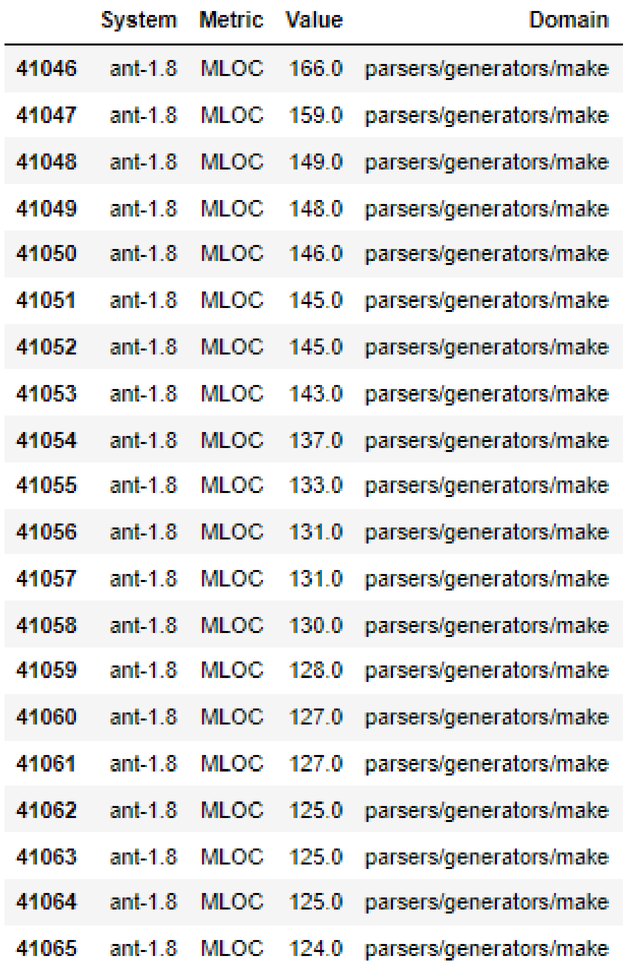
The champion models are selected on the basis of their MAE and RMSE values, for each model type and then the MAE is converted into scores using CDF normalisation as explained in [28].

Using the scores continuous confidence intervals are found, mainly using Chebyshev's inequality.

6 Evaluation

6.1 Data Preparation

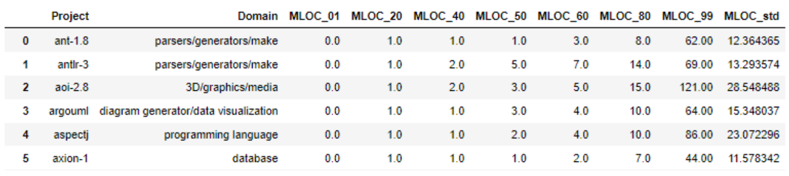
The structure of the data had to drastically be changed compared to the original data received from [26]. The original shape of the data $7,943,452$ rows $\times$ five columns which were converted to 109 rows $\times$ 172 columns.
For example the first project, "ant-1.8" has multiple instances of metric MLOC (only first 20 rows are shown in Fig. 6.11):



	System	Metric	Value	Domain
41046	ant-1.8	MLOC	166.0	parsers/generators/make
41047	ant-1.8	MLOC	159.0	parsers/generators/make
41048	ant-1.8	MLOC	149.0	parsers/generators/make
41049	ant-1.8	MLOC	148.0	parsers/generators/make
41050	ant-1.8	MLOC	146.0	parsers/generators/make
41051	ant-1.8	MLOC	145.0	parsers/generators/make
41052	ant-1.8	MLOC	145.0	parsers/generators/make
41053	ant-1.8	MLOC	143.0	parsers/generators/make
41054	ant-1.8	MLOC	137.0	parsers/generators/make
41055	ant-1.8	MLOC	133.0	parsers/generators/make
41056	ant-1.8	MLOC	131.0	parsers/generators/make
41057	ant-1.8	MLOC	131.0	parsers/generators/make
41058	ant-1.8	MLOC	130.0	parsers/generators/make
41059	ant-1.8	MLOC	128.0	parsers/generators/make
41060	ant-1.8	MLOC	127.0	parsers/generators/make
41061	ant-1.8	MLOC	127.0	parsers/generators/make
41062	ant-1.8	MLOC	125.0	parsers/generators/make
41063	ant-1.8	MLOC	125.0	parsers/generators/make
41064	ant-1.8	MLOC	125.0	parsers/generators/make
41065	ant-1.8	MLOC	124.0	parsers/generators/make

Figure 6.11: original data

The metric specific metric is converted to the following format as seen in Fig. 6.12, and this was done for all metrics for all projects.



	Project	Domain	MLOC_01	MLOC_20	MLOC_40	MLOC_50	MLOC_60	MLOC_80	MLOC_99	MLOC_std
0	ant-1.8	parsers/generators/make	0.0	1.0	1.0	1.0	3.0	8.0	62.00	12.364365
1	antlr-3	parsers/generators/make	0.0	1.0	2.0	5.0	7.0	14.0	69.00	13.293574
2	aoi-2.8	3D/graphics/media	0.0	1.0	2.0	3.0	5.0	15.0	121.00	28.548488
3	argouml	diagram generator/data visualization	0.0	1.0	1.0	3.0	4.0	10.0	64.00	15.348037
4	aspectj	programming language	0.0	1.0	1.0	2.0	4.0	10.0	86.00	23.072296
5	axion-1	database	0.0	1.0	1.0	1.0	2.0	7.0	44.00	11.578342

Figure 6.12: quantiled data

6.2 Statistical Analysis of technical debt

For research question 1: What is the notion of technical debt across the selected static analysis tools??
A Statistical approach was taken. First, to check for the distribution, Pearsons, Kindall and Spearman correlation tests were conducted between all 3 notions of technical debt.

The null hypothesis established is that the notions of technical debt for all the tools should be similar, as an identical list of projects has been tested with different tools.

6.2.1 Pearson sample correlation test

Pearsons Analysis		
Tools	**Pearson coefficient**	**p-value**
SonarQube vs Codiga	0.197	0.0398
SonarQube vs CodeClimate	0.107	0.270
CodeClimate vs Codiga	0.933	3.40e-49

When looking at Pearson's correlation coefficient, we can see that except CodeClimate vs Codiga, non of the other combinations of tools have a strong linear relationship with one another. Although the p-values suggest that the combination of tools other than SonarQube vs CodeClimate are statistically significant, suggesting they are not similar in notion, that is the distribution is not similar.

6.2.2 Kendall correlation test

Kendall Analysis		
Tools	**Kendall coefficient**	**p-value**
SonarQube vs Codiga	0.437	1.61e-11
SonarQube vs CodeClimate	0.409	2.87e-10
CodeClimate vs Codiga	0.750	6.12e-31

When looking at Kendall's correlation coefficient, we can see that there is a strong monotonic relation between CodeClimate and Codiga but the others are weak, while all of the combinations of tools reject the null hypothesis suggesting non of the notion of technical debt are similar.

6.2.3 Spearman correlation test

Spearman Analysis		
Tools	**Spearman coefficient**	**p-value**
SonarQube vs Codiga	0.561	2.11e-10
SonarQube vs CodeClimate	0.536	1.86e-09
CodeClimate vs Codiga	0.897	8.70e-40

Spearman's correlation coefficient for all tools suggests that the notion of technical debts has a degree of association, while the p-values reject the null hypothesis, similar to

Kendall.

So after looking at all the relations between the notions of technical debt for all tools and the distributions, the distance between the empirical distribution and the cumulative distribution is measured, to have more evidence for the comparison of the notion of technical debt across all three tools.

6.2.4 Kolmogorov–Smirnov test

KS2 Analysis		
Tools	**Ks2 score**	**Ks2 p-value**
SonarQube vs Codiga	0.440	7.00e-10
SonarQube vs CodeClimate	0.275	4.84e-4
CodeClimate vs Codiga	0.615	1.50e-19

As we know Kolmogorov–Smirnov test (KS2) test is used to compare the distance between two distributions of two different sets of data's cdfs. Looking at the KS2 scores, we can easily see that the set of distributions is far apart from each other. The ideal score should be below 0.1 to demonstrate similarity in cdfs. Hence the closest is SonarQube and CodeClimate but not significantly close.

6.2.5 Student's T-test

T-test Analysis		
Tools	**T-test score**	**T-test p-value**
SonarQube vs Codiga	4.33	2.32e-05
SonarQube vs CodeClimate	-1.87	0.063
CodeClimate vs Codiga	0.933	3.40e-49

Although most p-values for the t-test are less than 0.05, suggesting that they reject the null hypothesis, that is come from the same population of distribution the scores also tell us that the mean-variance for each comparison is far apart. As the values are closer to 0.1 plotting cdfs for each notion of technical debt further provided details about the distributions and their locations.

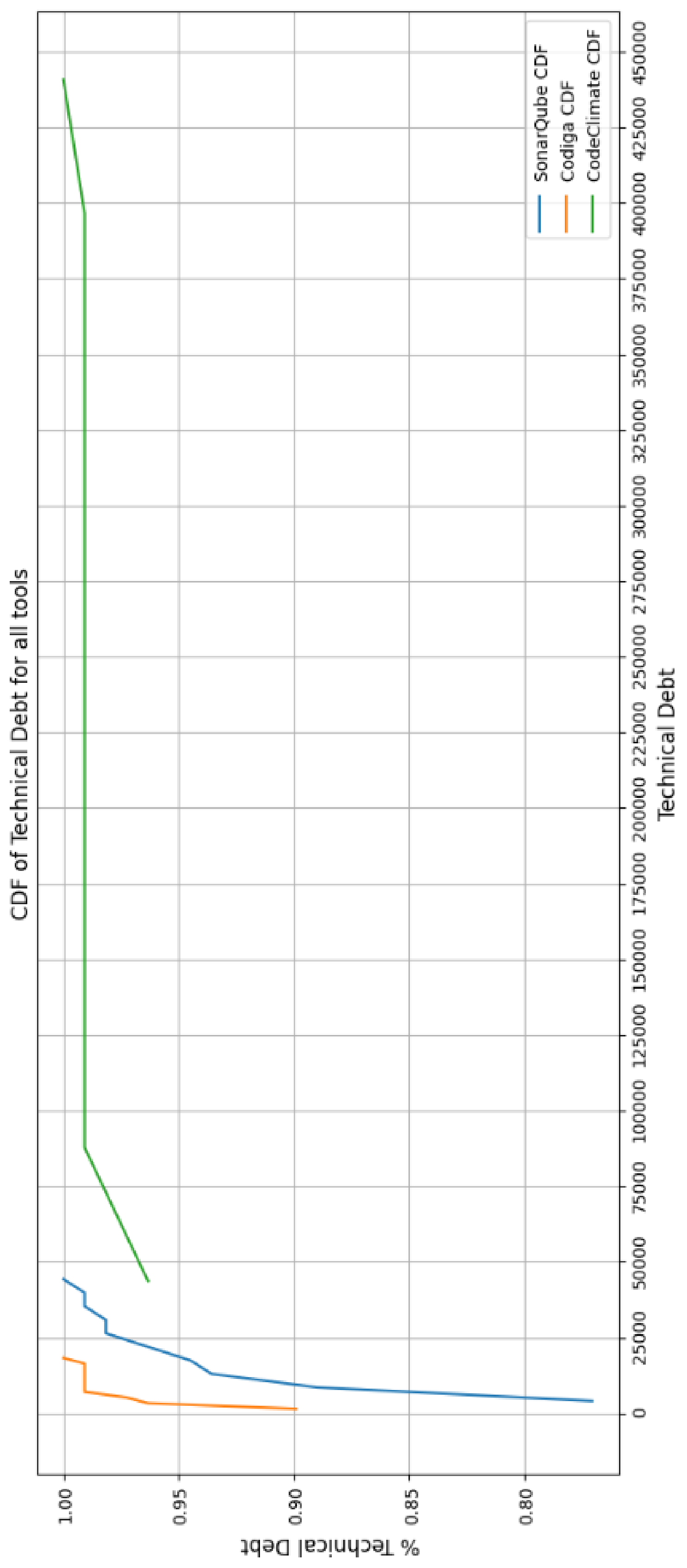


Figure 6.13: CDF plot for all notions of TD

6.3 Feature Importance

As mentioned in Section.5.4 five different models were implemented for each notion of technical debt. All these scores were further averaged down to have 1 set of feature importance scores for each notion of technical debt, which can be seen in Fig. 6.14.

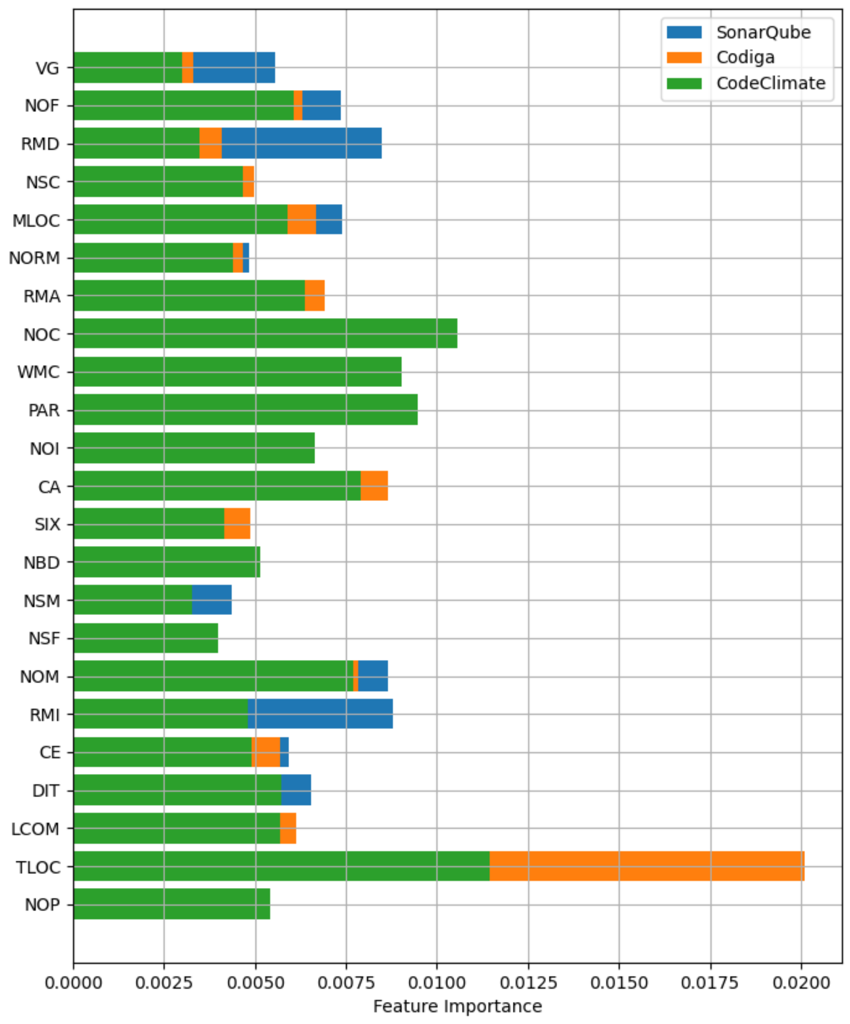


Figure 6.14: Average Feature Importance

To determine the most important features, an average of all three averages was taken into consideration. The results can be seen in Fig. 6.15. Absolute value was taken into consideration.

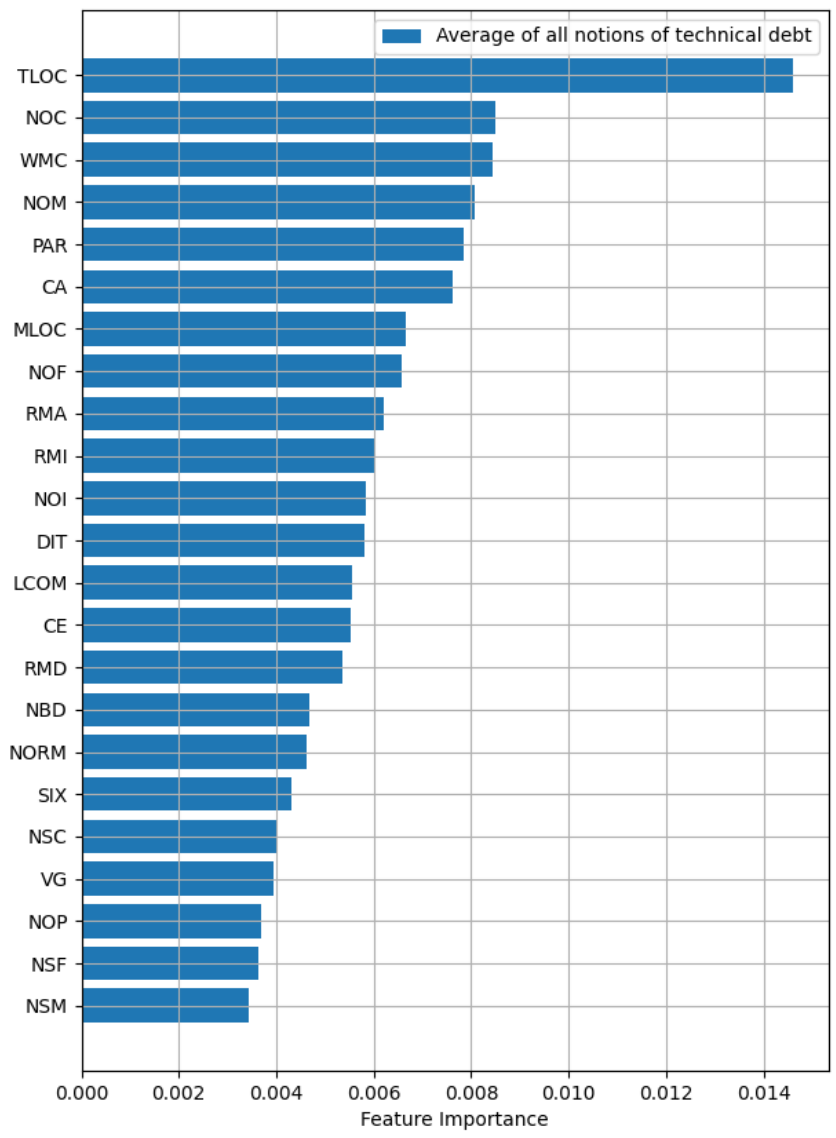


Figure 6.15: Average Feature Importance of all three notions of technical debt

6.4 Correlation of all features with all notions of technical debt

To understand which features affect each notion of technical debt the most, a correlation metric was made between the features.
As seen in Fig. 6.16 the white spaces in the figure itself are quantiles with 0.0 values or nan values, hence no correlation is possible, and the other features do not highly contribute to the technical debt.

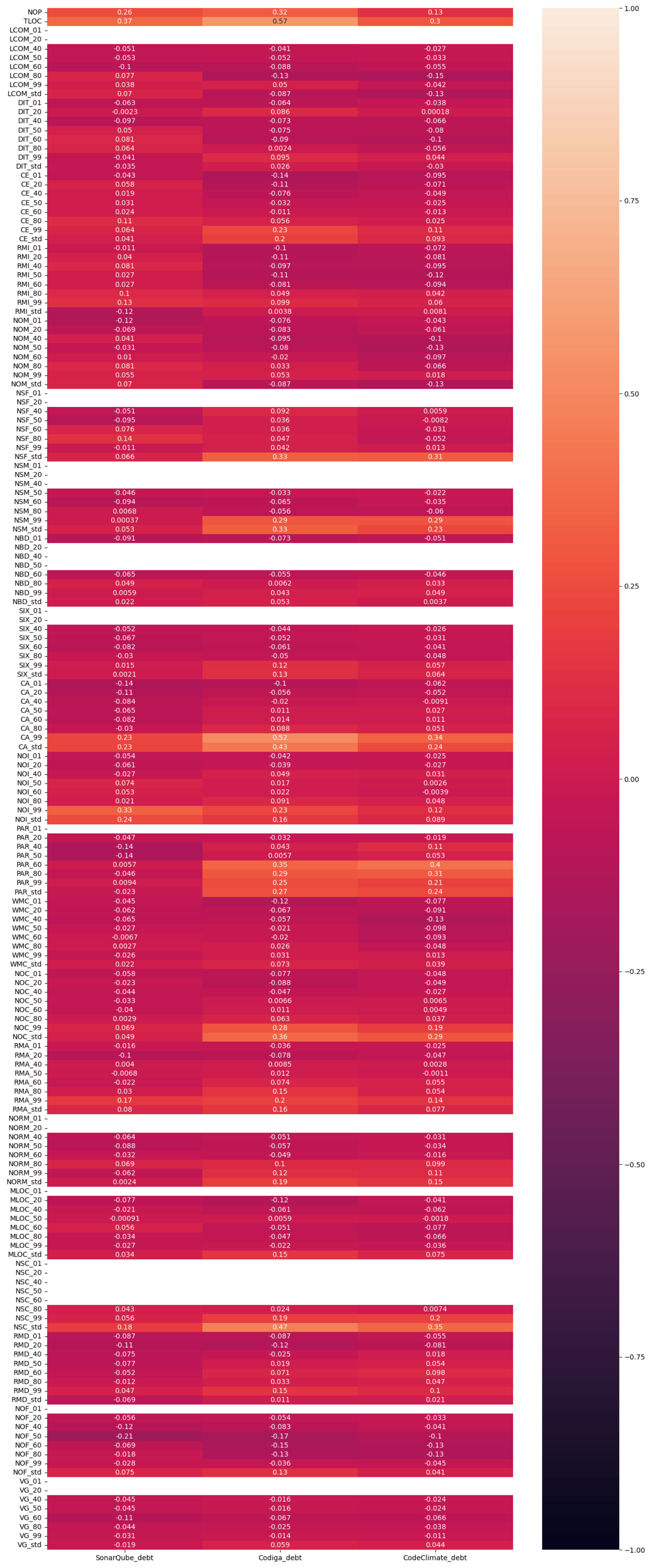


Figure 6.16: Correlation Matrix between all features and all notions of technical debt

6.5 Generalisation error

6.5.1 PCA

PCA was conducted on the features converted to quantiles + the standard deviation of each feature. With each notion of technical debt, the goal was to archive a 95% explained variance. On average each notion of technical debt required 37 components to achieve a cumulative explained variance of 95%.

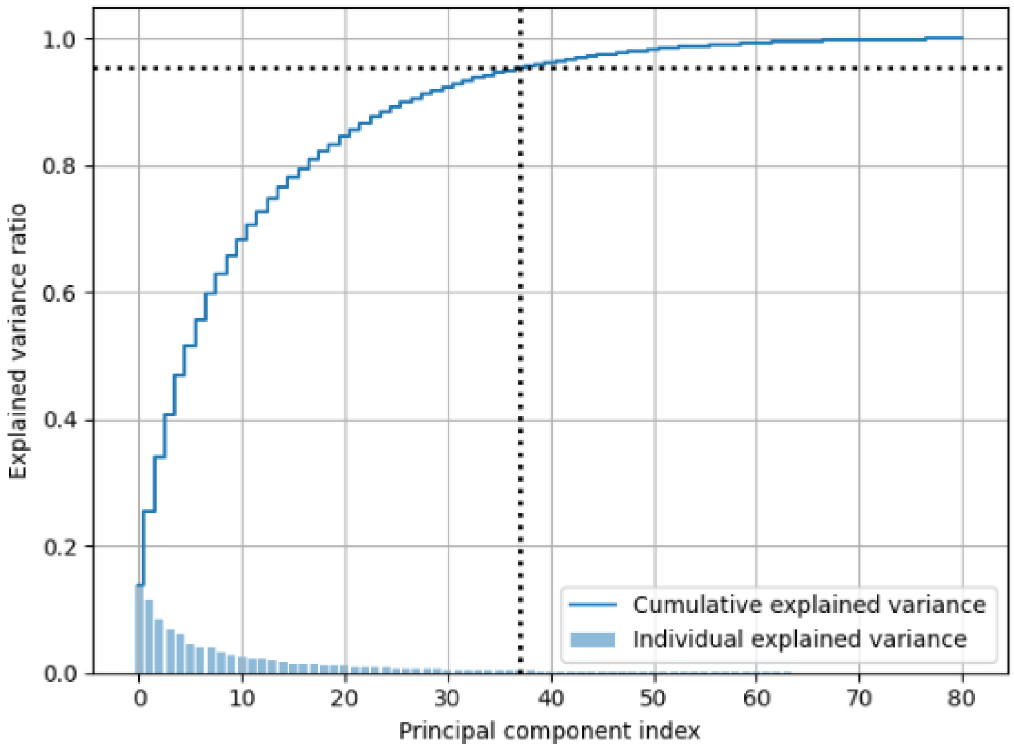


Figure 6.17: PCA

6.5.2 CDF and PDF for all notions of technical debt

We know that the technical debt notions of Codiga and SonarQube differ, it is much more common to have a large value for the latter, as seen in Fig.6.18. This makes them difficult to compare. However, we can transform the corpus' projects' TD into CDFs and normalize the predicted values. In other words, the CDF-transformation will give us results that we can compare.

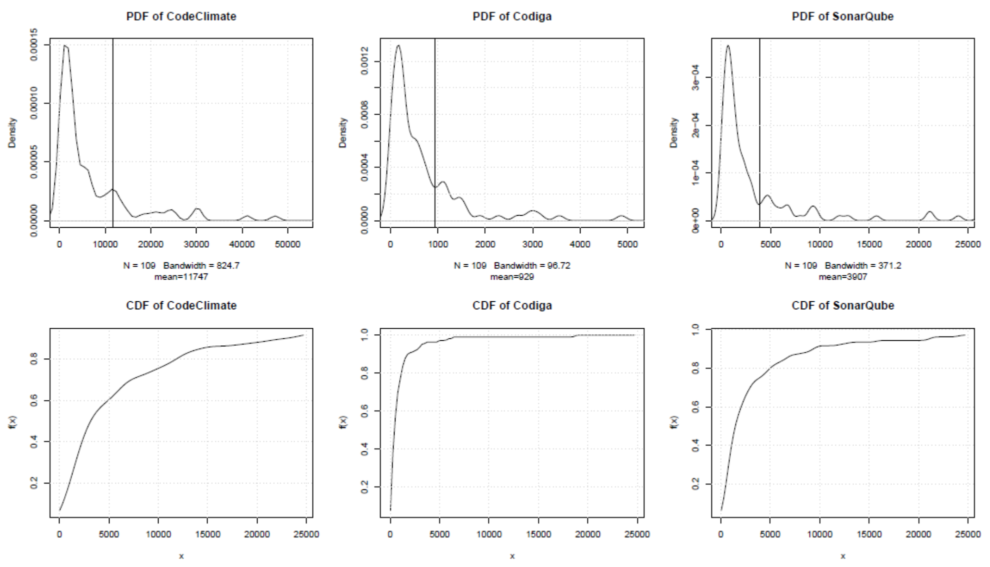


Figure 6.18: CDF and PDF plots for all notions of technical debt

We take the obtained MAEs and convert them into probabilities using the corresponding CDFs. We will get a good idea for each flavour of estimator whether it predicts low or high deviations. Using the CDFs also allows us then to compare these results, which would otherwise not have been possible.

6.5.3 Grid of combinations

Grid results post CDF normalisation

Model	**WithDomain UsePCA**	**NoDomain UsePCA**	**WithDomain NoPCA**	**NoDomain NoPCA**
CodeClimate RF	0.7606750	0.7660496	0.7452672	0.7519722
CodeClimate XGB	0.7498625	0.7581327	**0.7086790**	0.7194362
CodeClimate MLP	0.9056478	0.9087426	0.9162333	0.9183945
CodeClimate LR	0.9328974	0.9311435	0.9530537	0.9524264
Codiga RF	0.6795504	0.6834638	0.6536617	0.6601129
Codiga XGB	0.6525374	0.6550226	**0.5765633**	0.5856857
Codiga MLP	0.8165407	0.8165647	0.8252067	0.8308815
Codiga LR	0.8253077	0.8213767	0.8698335	0.8683150
SonarQube RF	0.7396159	0.7412417	0.7301272	0.7341624
SonarQube XGB	0.7335765	0.7376694	**0.7178658**	0.7215078
SonarQube MLP	0.8185164	0.8204274	0.9079272	0.9083246
SonarQube LR	0.8076266	0.8096504	0.9079272	0.9083246

The results in Table 6.5.3 tell us the following: the lower the average value, the better the flavour. The best value for CodeClimate is XGB, including domain, not doing PCA (0.709). The same flavour is best for Codiga (0.577), as well as SonarQube (0.718).

That also means, Codiga > CodeClimate is almost the same as SonarQube in terms of what we can predict best.

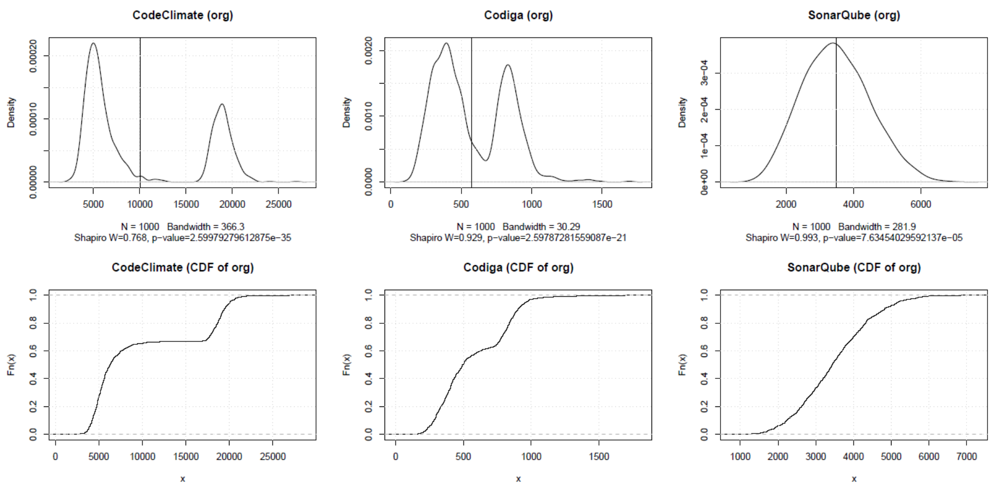


Figure 6.19: CDF and PDF plots for Distribution of MAE

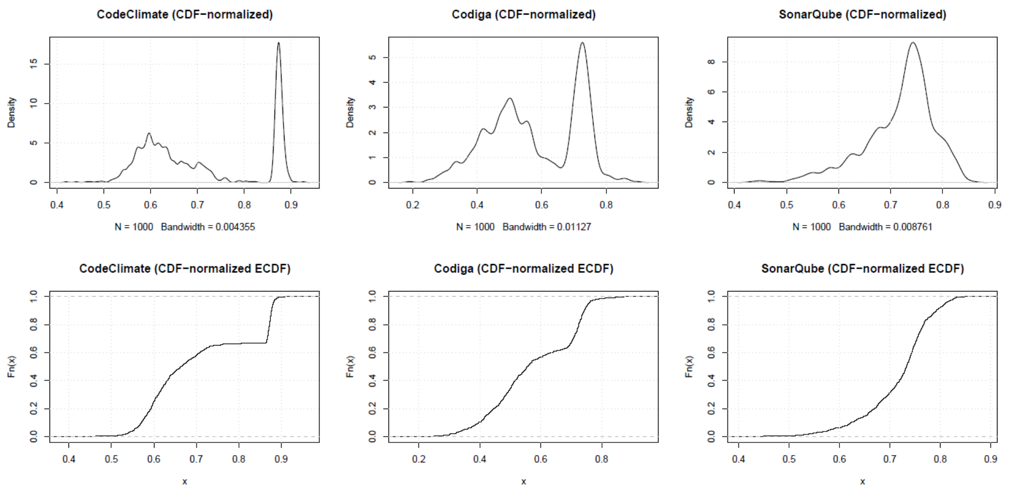


Figure 6.20: CDF and PDF plots for Distribution of MAE normalised

None of the residual MAEs are normally distributed, nor do they have a unimodal distribution (the residual MAEs for SonarQube are almost normal though). Therefore, we can only apply Chebyshev's inequality for estimating confidence intervals [35].

6.5.4 Confidence intervals

	Inequality		
Confidence	**Chebyshev Low**	**Mean**	**Chebyshev High**
5.00	321.7156	575.04	837.3734
15.00	297.7135	575.04	852.3756
25.00	279.8031	575.04	870.2860
35.00	257.9044	575.04	892.1847
45.00	230.2768	575.04	919.8123
50.00	213.4491	575.04	936.6400
55.00	193.8895	575.04	956.1996
65.00	142.8553	575.04	1007.2338
68.27	121.1312	575.04	1028.9578
75.00	63.6714	575.04	1086.4177
80.00	3.3120	575.04	1146.7771
85.00	0.0000	575.04	1235.2245
90.00	0.0000	575.04	1383.5965
95.00	0.0000	575.04	1718.5097
95.45	0.0000	575.04	1773.7217
97.50	0.0000	575.04	2192.1485
98.00	0.0000	575.04	2383.0217

Only for very low confidence, we expect our champion model to deviate rather slightly from the expected mean MAE. For example, our confidence is only 5% that the predictions for the type of champion model on Codiga's TD notion deviates between 312 and 837 around a mean of 575.

In other words, even the champion model is "usually" on average off by 575. However, it may be off less down to 312 or more up to 837. We can "guarantee" this with confidence of 5%.

7 Discussion

7.1 RQ1

We can conclude that the distribution of technical debt belongs to the same family, that is it is similar across all three tools but the tools themselves have different notions as the distributions are located at different locations on the x-axis, hence the notion of technical debt for each tool is not the same.

7.2 RQ2

We can graphically see which features are most important in terms of predicting technical debt, as absolute values were considered the impact of these metrics can be positive or negative but in terms of estimating the predictions, they play a huge role.

7.3 RQ3

The correlation metric does not give any important information as there are no metrics that correlate highly or do not correlate at all, hence it was considered that it is not a good idea to use only a sample of features as the models would not learn properly if the software metrics are not significantly correlated.

7.4 RQ4

To predict technical debt for each notion, only 4 different models were initially taken into consideration. However, after extensive hyperparameter tuning, we concluded that the variance between the predicted and actual values was too great for technical debt to be predicted with high accuracy. As a result, we switched to looking for generalised estimation error, which would reveal more about each notion of technical debt and its prediction.

It was previously stated that the Domain is of great importance because applications and their metrics are significantly different from each other if you take their Domain into account [28].

We had NOT considered the domain as a feature beforehand, because we decided only to use the numerical metrics. That the domain is important was only discovered while conducting the experiments for RQ4.

8 Conclusions and Future Work

8.1 Conclusion

If we recall that a typical mean value for Codiga's notion of technical debt is 929. In other words, running Codiga on a random project will on average yield a technical debt of 929. If we're off by more than $1,300$, it could mean that we predict roughly that a project has no technical debt when it actually has technical debt, or that we more than twice the technical debt. These are just examples of how "off" we could be with those 90%.

Since low confidence has no real practical usage. So, if we look at a more reasonable example, say, 90%, the deviation from the expectation (mean) must be more extreme. So with 90% confidence, the deviation from the actual technical debt, for Codiga, is $\leq 1,384$. But here is where it also gets interesting.

Suppose you are happy with lower confidence, just to get "ballpark" kind of figure. So if, for example, 50% confidence is enough, this predictor might become useful. Let's say you are dealing with projects that usually have very high technical debt, then being off by about $1,000$ is perhaps acceptable for getting a first quick impression. Remember that this here is a trade-off; evaluating technical debt is perhaps really expensive in real life, as you need experts who have to do this qualitatively, evaluating lots of unstructured data [36]. The model quantitatively has an increased variance in inequality and prediction, as the confidence level increases.

Although the data set used consists of very well-known software, the result can be generalised to conclude that the current model is a stepping stone towards predicting technical debt using internal software metrics, hence if we can predict more confidently, this method can be used to align the estimation of technical debt regardless the language, domain or size of the software.

Practical usage of this method would benefit companies mainly as it would help them develop more efficiently and reduce costs for the future, which ideally will help them make more profits and create a better market reputation.

8.2 Future Work

While it is true that the data is fairly old and only in one language, one could gather a similar set of metrics for many other languages, this usually takes the longest time in terms of executing the experiment. The point is to show that our approach works to some degree. Future work could be to attempt this with other languages, i.e., to replicate your work while altering some of the parameters

The other thing that could be considered is using more data itself, as the data was so little the models could have not trained properly. While the list of models could also be reconsidered. but as we can see the results do gives us hope that it is possible to estimate technical debt using software metrics rather than using aggregated metrics.

The main possibility of this method is that it can provoke other researchers to use this method to develop a better model and collect more data in this field.

References

[1] M. Nachtigall, L. N. Q. Do, and E. Bodden, “Explaining static analysis - A perspective”, in *34th IEEE/ACM International Conference on Automated Software Engineering Workshops, ASE Workshops 2019, San Diego, CA, USA, November 11-15, 2019*, IEEE, 2019, pp. 29–32. DOI: 10.1109/ASEW.2019.00023.

[2] D. Stefanovic, D. Nikolic, D. Dakic, I. Spasojevic, and S. Ristic, “Static code analysis tools: A systematic literature review”, pp. 0565–0573, 2020. DOI: 10.2507/31st.daaam.proceedings.078.

[3] N. S. R. Alves, L. F. Ribeiro, V. Caires, T. S. Mendes, and R. O. Spínola, “Towards an ontology of terms on technical debt”, in *Sixth International Workshop on Managing Technical Debt, MTD@ICSME 2014, Victoria, BC, Canada, September 30, 2014*, IEEE Computer Society, 2014, pp. 1–7. DOI: 10.1109/MTD.2014.9.

[4] W. Cunningham, “The WyCash portfolio management system”, in *Addendum to the Proceedings on Object-Oriented Programming Systems, Languages, and Applications, OOPSLA 1992 Addendum, Vancouver, British Columbia, Canada, October 18-22, 1992*, J. L. Archibald and M. C. Wilkes, Eds., vol. 4, New York, NY, USA: Association for Computing Machinery, 1992, pp. 29–30. DOI: 10.1145/157709.157715.

[5] D. Falessi and A. Voegele. “Validating and prioritizing quality rules for managing technical debt: An industrial case study”. (2015), [Online]. Available: https://doi.org/10.1109/MTD.2015.7332623.

[6] A. Nugroho, J. Visser, and T. Kuipers, “An empirical model of technical debt and interest”, in *Proceedings of the 2nd Workshop on Managing Technical Debt, MTD 2011, Waikiki, Honolulu, HI, USA, May 23, 2011*, I. Ozkaya, P. Kruchten, R. L. Nord, and N. Brown, Eds., ACM, 2011, pp. 1–8, ISBN: 9781450305860. DOI: 10.1145/1985362.1985364.

[7] E. Woods, “The past, present and future of technical debt: Learning from the past to prepare for the future”, in *Proceedings of the 2018 International Conference on Technical Debt*, ser. TechDebt ’18, Gothenburg, Sweden: Association for Computing Machinery, 2018, p. 61, ISBN: 9781450357135. DOI: 10.1145/3194164.3194181.

[8] R. Alfayez, W. Alwehaibi, R. Winn, E. Venson, and B. Boehm, “A systematic literature review of technical debt prioritization”, in *Proceedings of the 3rd International Conference on Technical Debt*, ser. TechDebt ’20, Seoul, Republic of Korea: Association for Computing Machinery, 2020, pp. 1–10, ISBN: 9781450379601. DOI: 10.1145/3387906.3388630.

[9] V. Lenarduzzi, T. Besker, D. Taibi, A. Martini, and F. A. Fontana, “A systematic literature review on technical debt prioritization: Strategies, processes, factors, and tools”, *Journal of Systems and Software*, vol. 171, p. 110 827, 2021. DOI: 10.1016/j.jss.2020.110827.

[10] V. Lenarduzzi, F. Pecorelli, N. Saarimaki, S. Lujan, and F. Palomba, “A critical comparison on six static analysis tools: Detection, agreement, and precision”, *Journal of Systems and Software*, vol. 198, p. 111 575, 2023, ISSN: 0164-1212. DOI: https://doi.org/10.1016/j.jss.2022.111575.

[11] H. F. Li and W. K. Cheung, “An empirical study of software metrics”, *IEEE Transactions on Software Engineering*, vol. 13, no. 6, pp. 697–708, Jun. 1987, ISSN: 0098-5589. DOI: `10.1109/TSE.1987.233475`.

[12] M. Ulan, W. Löwe, M. Ericsson, and A. Wingkvist, “Towards meaningful software metrics aggregation”, in *Proceedings of the 18th Belgium-Netherlands Software Evolution Workshop, Brussels, Belgium, November 28th to 29th, 2019*, D. D. Nucci and C. D. Roover, Eds., ser. CEUR Workshop Proceedings, vol. 2605, CEUR-WS.org, 2019.

[13] L. Rosenberg, T. Hammer, and J. Shaw, “Software metrics and reliability”, in *9th international symposium on software reliability engineering*, 1998.

[14] Y. A. Alsultanny and A. M. Wohaishi, “Requirements of software quality assurance model”, in *2009 Second International Conference on Environmental and Computer Science, ICECS 2009, Dubai, UAE, 28-30 December 2009*, IEEE Computer Society, 2009, pp. 19–23. DOI: `10.1109/ICECS.2009.43`.

[15] E. Tom, A. Aurum, and R. T. Vidgen, “An exploration of technical debt”, *Journal of Systems and Software*, vol. 86, no. 6, pp. 1498–1516, 2013. DOI: `10.1016/j.jss.2012.12.052`.

[16] D. Pina, C. Seaman, and A. Goldman, “Technical debt prioritization: A developer’s perspective”, in *Proceedings of the International Conference on Technical Debt*, ser. TechDebt ’22, Pittsburgh, Pennsylvania: Association for Computing Machinery, 2022, pp. 46–55, ISBN: 9781450393041. DOI: `10.1145/3524843.3528096`.

[17] T. Klinger, P. Tarr, P. Wagstrom, and C. Williams, “An enterprise perspective on technical debt”, in *Proceedings of the 2nd Workshop on Managing Technical Debt*, ser. MTD ’11, Waikiki, Honolulu, HI, USA: Association for Computing Machinery, 2011, pp. 35–38, ISBN: 9781450305860. DOI: `10.1145/1985362.1985371`.

[18] B. Curtis, J. Sappidi, and A. Szynkarski, “Estimating the size, cost, and types of technical debt”, in *Proceedings of the Third International Workshop on Managing Technical Debt, MTD 2012, Zurich, Switzerland, June 5, 2012*, P. Kruchten, R. L. Nord, I. Ozkaya, and J. Visser, Eds., IEEE/ACM, 2012, pp. 49–53. DOI: `10.1109/MTD.2012.6226000`.

[19] W. Landi, “Undecidability of static analysis”, *LOPLAS*, vol. 1, no. 4, pp. 323–337, 1992. DOI: `10.1145/161494.161501`.

[20] L. N. Q. Do, J. R. Wright, and K. Ali, “Why do software developers use static analysis tools? A user-centered study of developer needs and motivations”, *IEEE Trans. Software Eng.*, vol. 48, no. 3, pp. 835–847, 2022. DOI: `10.1109/TSE.2020.3004525`.

[21] D. Nikolić, D. Stefanović, D. Dakić, S. Sladojević, and S. Ristić, “Analysis of the tools for static code analysis”, in *2021 20th International Symposium INFOTEH-JAHORINA (INFOTEH)*, 2021, pp. 1–6. DOI: `10.1109/INFOTEH51037.2021.9400688`.

[22] B. Chess and J. West, *Secure Programming with Static Analysis: Getting Software Security Right with Static Analysis* (Addison-Wesley Software Security). Addison Wesley, 2007, 624 pp., ISBN: 0321424778.

[23] S. Jaiswal, *Software engineering: Software metrics - javatpoint.*

[24] M. A. Stephens, "Edf statistics for goodness of fit and some comparisons", *Journal of the American Statistical Association*, vol. 69, no. 347, pp. 730–737, 1974. DOI: 10.1080/01621459.1974.10480196.

[25] P. Tchébychef, "Des Valeurs Moyennes", *Journal de Mathématiques Pures et Appliquées*, 2nd ser., vol. 12, pp. 177–184, 1867, Traduction du Russe par M. N. de Khanikof, ISSN: 0021-7874.

[26] R. Terra, L. F. Miranda, M. T. Valente, and R. da Silva Bigonha, "Qualitas.class corpus: A compiled version of the qualitas corpus", 5, vol. 38, 2013, pp. 1–4. DOI: 10.1145/2507288.2507314.

[27] E. D. Tempero, C. Anslow, J. Dietrich, *et al.*, "The qualitas corpus: A curated collection of java code for empirical studies", in *17th Asia Pacific Software Engineering Conference, APSEC 2010, Sydney, Australia, November 30 - December 3, 2010*, J. Han and T. D. Thu, Eds., IEEE Computer Society, 2010, pp. 336–345. DOI: 10.1109/APSEC.2010.46.

[28] S. Hönel, M. Ericsson, W. Löwe, and A. Wingkvist, "Contextual operationalization of metrics as scores: Is my metric value good?", in *22nd IEEE International Conference on Software Quality, Reliability and Security, QRS 2022, Guangzhou, China, December 5-9, 2022*, IEEE, 2022, pp. 333–343. DOI: 10.1109/QRS57517.2022.00042.

[29] T. pandas development team, *Pandas-dev/pandas: Pandas*, version 1.5.3, Feb. 2020. DOI: 10.5281/zenodo.3509134.

[30] C. R. Harris, K. J. Millman, S. J. van der Walt, *et al.*, "Array programming with NumPy", *Nature*, vol. 585, no. 7825, pp. 357–362, Sep. 2020. DOI: 10.1038/s41586-020-2649-2.

[31] J. D. Hunter, "Matplotlib: A 2d graphics environment", *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007. DOI: 10.1109/MCSE.2007.55.

[32] M. L. Waskom, "Seaborn: Statistical data visualization", *Journal of Open Source Software*, vol. 6, no. 60, p. 3021, 2021. DOI: 10.21105/joss.03021.

[33] P. Virtanen, R. Gommers, T. E. Oliphant, *et al.*, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python", *Nature Methods*, vol. 17, pp. 261–272, 2020. DOI: 10.1038/s41592-019-0686-2.

[34] F. Pedregosa, G. Varoquaux, A. Gramfort, *et al.*, "Scikit-learn: Machine learning in Python", *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[35] P. Tchébychef, "Des valeurs moyennes (traduction du russe, n. de khanikof.", fre, *Journal de Mathématiques Pures et Appliquées*, pp. 177–184, 1867.

[36] P. G. F. Matsubara, B. F. Gadelha, I. Steinmacher, and T. U. Conte, "Sextamt: A systematic map to navigate the wide seas of factors affecting expert judgment software estimates", *Journal of Systems and Software*, vol. 185, p. 111 148, 2022, ISSN: 0164-1212. DOI: https://doi.org/10.1016/j.jss.2021.111148.

[37] L. Schrettner, L. J. Fülöp, Á. Beszédes, Á. Kiss, and T. Gyimóthy, "Software quality model and framework with applications in industrial context", in *16th European Conference on Software Maintenance and Reengineering, CSMR 2012, Szeged, Hungary, March 27-30, 2012*, T. Mens, A. Cleve, and R. Ferenc, Eds., IEEE Computer Society, 2012, pp. 453–456. DOI: 10.1109/CSMR.2012.57.

[38] F. Huang, Y. Wang, Y. Wang, and P. Zong, "What software quality characteristics most concern safety-critical domains?", in *2018 IEEE International Conference on Software Quality, Reliability and Security Companion, QRS Companion 2018, Lisbon, Portugal, July 16-20, 2018*, IEEE, 2018, pp. 635–636. DOI: 10.1109/QRS-C.2018.00111.

[39] N. E. Fenton and M. Neil, "Software metrics: Successes, failures and new directions", *Journal of Systems and Software*, vol. 47, no. 2-3, pp. 149–157, 1999. DOI: 10.1016/S0164-1212(99)00035-7.

[40] K. Lochmann and A. Goeb, "A unifying model for software quality", in *Proceedings of the 8th international workshop on Software quality, WoSQ@ESEC/FSE 2011, Szeged, Hungary, September 4, 2011*, S. Wagner, S. Chulani, and B. Wong, Eds., ACM, 2011, pp. 3–10. DOI: 10.1145/2024587.2024591.

[41] N. Nagappan, T. Ball, and A. Zeller, "Mining metrics to predict component failures", in *28th International Conference on Software Engineering (ICSE 2006), Shanghai, China, May 20-28, 2006*, L. J. Osterweil, H. D. Rombach, and M. L. Soffa, Eds., ACM, 2006, pp. 452–461. DOI: 10.1145/1134285.1134349.

[42] N. F. Schneidewind, "Methodology for validating software metrics", *IEEE Trans. Software Eng.*, vol. 18, no. 5, pp. 410–422, 1992. DOI: 10.1109/32.135774.

[43] H. F. Li and W. K. Cheung, "An empirical study of software metrics", *IEEE Trans. Software Eng.*, vol. 13, no. 6, pp. 697–708, 1987. DOI: 10.1109/TSE.1987.233475.

[44] N. E. Fenton and M. Neil, "Software metrics: Roadmap", in *22nd International Conference on on Software Engineering, Future of Software Engineering Track, ICSE 2000, Limerick Ireland, June 4-11, 2000*, A. Finkelstein, Ed., ACM, 2000, pp. 357–370. DOI: 10.1145/336512.336588.

[45] S. A. Mengel, "Software metrics: Views from education, research, and training", in *12th Conference on Software Engineering Education and Training, 22-24 March, 1999, New Orleans, Louisiana, USA*, IEEE Computer Society, 1999, pp. 126–128. DOI: 10.1109/CSEE.1999.755191.

[46] T. Honglei, S. Wei, and Z. Yanan, "The research on software metrics and software complexity metrics", in *2009 International Forum on Computer Science-Technology and Applications*, vol. 1, 2009, pp. 131–136. DOI: 10.1109/IFCSTA.2009.39.

[47] E. A. Alikhashashneh, R. R. Raje, and J. H. Hill, "Using software engineering metrics to evaluate the quality of static code analysis tools", in *1st International Conference on Data Intelligence and Security, ICDIS 2018, South Padre Island, TX, USA, April 8-10, 2018*, IEEE, 2018, pp. 65–72. DOI: 10.1109/ICDIS.2018.00017.

[48] L. Lavazza, S. Morasca, and D. Tosi, "Technical debt as an external software attribute", in *Proceedings of the 2018 International Conference on Technical Debt*, ser. TechDebt '18, Gothenburg, Sweden: Association for Computing Machinery, 2018, pp. 21–30, ISBN: 9781450357135. DOI: 10.1145/3194164.3194168.

[49] C. B. Seaman, Y. Guo, C. Izurieta, *et al.*, "Using technical debt data in decision making: Potential decision approaches", in *Proceedings of the Third International Workshop on Managing Technical Debt, MTD 2012, Zurich, Switzerland, June 5, 2012*, P. Kruchten, R. L. Nord, I. Ozkaya, and J. Visser, Eds., IEEE/ACM, 2012, pp. 45–48. DOI: 10.1109/MTD.2012.6225999.

[50] I. Khomyakov, Z. Makhmutov, R. Mirgalimova, and A. Sillitti, "Automated measurement of technical debt: A systematic literature review", in *Proceedings of the 21st International Conference on Enterprise Information Systems, ICEIS 2019, Heraklion, Crete, Greece, May 3-5, 2019, Volume 2*, J. Filipe, M. Smialek, A. Brodsky, and S. Hammoudi, Eds., SciTePress, 2019, pp. 95–106. DOI: 10.5220/0007675900950106.

[51] D. Saraiva, J. G. Neto, U. Kulesza, G. Freitas, R. Rebouças, and R. Coelho, "Technical debt tools: A systematic mapping study", in *Proceedings of the 23rd International Conference on Enterprise Information Systems, ICEIS 2021, Online Streaming, April 26-28, 2021, Volume 2*, J. Filipe, M. Smialek, A. Brodsky, and S. Hammoudi, Eds., SCITEPRESS, 2021, pp. 88–98. DOI: 10.5220/0010459100880098.

[52] S. Bougouffa, Q. H. Dong, S. Diehm, F. Gemein, and B. Vogel-Heuser, "Technical debt indication in plc code for automated production systems: Introducing a domain specific static code analysis tool", *IFAC-PapersOnLine*, vol. 51, no. 10, pp. 70–75, 2018, 3rd IFAC Conference on Embedded Systems, Computational Intelligence and Telematics in Control CESCIT 2018, ISSN: 2405-8963. DOI: https://doi.org/10.1016/j.ifacol.2018.06.239.

[53] B. Curtis, J. Sappidi, and A. Szynkarski, "Estimating the size, cost, and types of technical debt", in *Proceedings of the Third International Workshop on Managing Technical Debt, MTD 2012, Zurich, Switzerland, June 5, 2012*, P. Kruchten, R. L. Nord, I. Ozkaya, and J. Visser, Eds., IEEE/ACM, 2012, pp. 49–53. DOI: 10.1109/MTD.2012.6226000.

[54] J. Letouzey, "The SQALE method for evaluating technical debt", in *Proceedings of the Third International Workshop on Managing Technical Debt, MTD 2012, Zurich, Switzerland, June 5, 2012*, P. Kruchten, R. L. Nord, I. Ozkaya, and J. Visser, Eds., IEEE/ACM, 2012, pp. 31–36. DOI: 10.1109/MTD.2012.6225997.

[55] P. Emanuelsson and U. Nilsson, "A comparative study of industrial static analysis tools", *Electronic Notes in Theoretical Computer Science*, vol. 217, pp. 5–21, 2008, Proceedings of the 3rd International Workshop on Systems Software Verification (SSV 2008), ISSN: 1571-0661. DOI: https://doi.org/10.1016/j.entcs.2008.06.039.

[56] P. Thomson, "Static analysis: An introduction: The fundamental challenge of software engineering is one of complexity.", *Queue*, vol. 19, no. 4, pp. 29–41, Sep. 2021, ISSN: 1542-7730. DOI: 10.1145/3487019.3487021.

[57] D. Singh, V. R. Sekar, K. T. Stolee, and B. Johnson, "Evaluating how static analysis tools can reduce code review effort", in *2017 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, 2017, pp. 101–105. DOI: 10.1109/VLHCC.2017.8103456.

[58] F. Wedyan, D. Alrmuny, and J. M. Bieman, “The effectiveness of automated static analysis tools for fault detection and refactoring prediction”, in *Proceedings of the 2009 International Conference on Software Testing Verification and Validation*, ser. ICST ’09, USA: IEEE Computer Society, 2009, pp. 141–150, ISBN: 9780769536019. DOI: 10.1109/ICST.2009.21.

[59] D. Nikolić, D. Stefanović, D. Dakić, S. Sladojević, and S. Ristić, “Analysis of the tools for static code analysis”, in *2021 20th International Symposium INFOTEH-JAHORINA (INFOTEH)*, 2021, pp. 1–6. DOI: 10.1109/INFOTEH51037.2021.9400688.

[60] T. Muske and P. Bokil, “On implementational variations in static analysis tools”, in *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, 2015, pp. 512–515. DOI: 10.1109/SANER.2015.7081867.

[61] F. A. Fontana, R. Roveda, and M. Zanoni, “Technical debt indexes provided by tools: A preliminary discussion”, in *2016 IEEE 8th International Workshop on Managing Technical Debt (MTD)*, 2016, pp. 28–31. DOI: 10.1109/MTD.2016.11.

A Related studies: Software metrics and quality models

no.	Title	Author(s)	Year Published
[37]	Software Quality Model and Framework with Applications in Industrial Context	L. Schrettner, L. J. Fülöp, Á. Beszédes, *et al.*	2012
[38]	What Software Quality Characteristics Most Concern Safety-Critical Domains?	F. Huang, Y. Wang, Y. Wang, *et al.*	2018
[39]	Software metrics: successes, failures and new directions	N. E. Fenton and M. Neil	1999
[40]	A unifying model for software quality	K. Lochmann and A. Goeb	2011
[41]	Mining metrics to predict component failures	N. Nagappan, T. Ball, and A. Zeller	2006
[13]	Software metrics and reliability	L. Rosenberg, T. Hammer, and J. Shaw	1998
[42]	Methodology For Validating Software Metrics	N. F. Schneidewind	1992
[43]	An Empirical Study of Software Metrics	H. F. Li and W. K. Cheung	1987
[14]	Requirements of Software Quality Assurance Model	Y. A. Alsultanny and A. M. Wohaishi	2009
[44]	Software metrics: roadmap	N. E. Fenton and M. Neil	2000
[12]	Towards Meaningful Software Metrics Aggregation	M. Ulan, W. Löwe, M. Ericsson, *et al.*	2019
[45]	Software Metrics: Views from Education, Research, and Training	S. A. Mengel	1999
[46]	The Research on Software Metrics and Software Complexity Metrics	T. Honglei, S. Wei, and Z. Yanan	2009
[47]	Using Software Engineering Metrics to Evaluate the Quality of Static Code Analysis Tools	E. A. Alikhashashneh, R. R. Raje, and J. H. Hill	2018

B Related studies: Technical Debt

no.	Title	Author(s)	Year Published
[15]	An exploration of technical debt	E. Tom, A. Aurum, and R. T. Vidgen	2013
[48]	Technical Debt as an External Software Attribute	L. Lavazza, S. Morasca, and D. Tosi	2018
[3]	Towards an Ontology of Terms on Technical Debt	N. S. R. Alves, L. F. Ribeiro, V. Caires, *et al.*	2014
[49]	Using technical debt data in decision making: potential decision approaches	C. B. Seaman, Y. Guo, C. Izurieta, *et al.*	2012
[41]	Mining metrics to predict component failures	N. Nagappan, T. Ball, and A. Zeller	2006
[7]	The Past, Present and Future of Technical Debt: Learning from the Past to Prepare for the Future	E. Woods	2018
[8]	A Systematic Literature Review of Technical Debt Prioritization	R. Alfayez, W. Alwehaibi, R. Winn, *et al.*	2020
[16]	Technical Debt Prioritization: A Developer's Perspective	D. Pina, C. Seaman, and A. Goldman	2022
[17]	An Enterprise Perspective on Technical Debt	T. Klinger, P. Tarr, P. Wagstrom, *et al.*	2011
[44]	Software metrics: roadmap	N. E. Fenton and M. Neil	2000
[50]	Automated Measurement of Technical Debt: A Systematic Literature Review	I. Khomyakov, Z. Makhmutov, R. Mirgalimova, *et al.*	2019
[51]	Technical Debt Tools: A Systematic Mapping Study	D. Saraiva, J. G. Neto, U. Kulesza, *et al.*	2021
[9]	A systematic literature review on Technical Debt prioritization: Strategies, processes, factors, and tools	V. Lenarduzzi, T. Besker, D. Taibi, *et al.*	2021
[52]	Technical Debt indication in PLC Code for automated Production Systems: Introducing a Domain Specific Static Code Analysis Tool	S. Bougouffa, Q. H. Dong, S. Diehm, *et al.*	2018
[53]	Estimating the size, cost, and types of technical debt	B. Curtis, J. Sappidi, and A. Szynkarski	2012
[54]	The SQALE method for evaluating technical debt	J. Letouzey	2012

C Related studies: Static Analysis Tools

no.	Title	Author(s)	Year Published
[1]	Explaining Static Analysis - A Perspective	M. Nachtigall, L. N. Q. Do, and E. Bodden	2019
[19]	Undecidability of Static Analysis	W. Landi	1992
[55]	A Comparative Study of Industrial Static Analysis Tools	P. Emanuelsson and U. Nilsson	2008
[56]	Static Analysis: An Introduction: The Fundamental Challenge of Software Engineering is One of Complexity.	P. Thomson	2021
[20]	Why Do Software Developers Use Static Analysis Tools? A User-Centered Study of Developer Needs and Motivations	L. N. Q. Do, J. R. Wright, and K. Ali	2022
[57]	Evaluating how static analysis tools can reduce code review effort	D. Singh, V. R. Sekar, K. T. Stolee, *et al.*	2017
[58]	The Effectiveness of Automated Static Analysis Tools for Fault Detection and Refactoring Prediction	F. Wedyan, D. Alrmuny, and J. M. Bieman	2009
[59]	Analysis of the Tools for Static Code Analysis	D. Nikolić, D. Stefanović, D. Dakić, *et al.*	2021
[60]	On implementational variations in static analysis tools	T. Muske and P. Bokil	2015
[10]	A critical comparison on six static analysis tools: Detection, agreement, and precision	V. Lenarduzzi, F. Pecorelli, N. Saarimaki, *et al.*	2023
[61]	Technical Debt Indexes Provided by Tools: A Preliminary Discussion	F. A. Fontana, R. Roveda, and M. Zanoni	2016