UNIVERSITY OF TARTU
Institute of Computer Science
Curriculum

Octanty Mulianingtyas

A Microservice-based Platform for Software Technical Debt Analysis

Master's Thesis (30 ECTS)

Supervisors: Professor Ulrich Norbisrath, PhD, University of Tartu.
Professor Bruno Rossi, PhD, Masaryk University.

Tartu 2021

A Microservice-based Platform for Software Technical Debt Analysis

Abstract:

Technical debt (TD) is defined as developing a new feature by a poor code and design to fulfil a punctual release date that may provide shor-term advantage, but negatively impact the future. Reducing TD is crucial to improve the software quality and decrease the financial issue.

However, TD is cumbersome to calculate since every TD tool provided uses numerous metrics and models. This study aims to provide analysis about technical debt alternative measurements by developing a microservice-based platform on a set of large open-source projects.

The result of the analysis is expected can be used as a reference for the researchers to arrange a proper formula to calculate TD.

The microservice platform was selected since it has several benefits compared to monolithic applications. The benefits comprised simple deployment, fewer environment dependencies, and maintainability.

We implemented three TD identification methods in the microservice platform: Maintainability Index (MI), SIG Maintainability, and Sqale.

Through 124 directory java projects run in the microservice platform, the statistical analysis results show no correlation among the three TD identification methods.

The increasing value of the metrics of TotalOp, TotalOpr, CC, Duplication, UnitSize, and NumberParameters, WMC will decrease the ranking of result in each TD Identification method, and thus decrease the overall TD.

The scalability of the microservice is seen by the performance of the microservice by the total time for a query executed in the API. We used the load testing in the postman to see the total time. The API used for testing employed the aforementioned All TD Identification method. From the experiment, the time for a query executed for calculating 5 directories is 1m 40.38 s and it will continuously improve until 34m 20.70 s for 124 directories. We also tested the microservice platform for running in seven iterations. However, the number of iteration is not correlated with the time for a query executed. The time for a query executed is variated in every iteration and not increasing continuously as the increment of the number of iterations. Based on these facts, the microservice performance is only influenced by the number of directories upload instead of the number of iterations for executing.

Keywords:

Microservices; Maintainability Index; SIG Maintainability; Sqale; Technical Debt; Technical Debt Identification Method;

CERCS: P170

Mikroteenustepõhine platvorm tarkvara tehnilise võla analüüsiks

Lühikokkuvõte:

Tehniliseks võlaks nimetatakse situatsiooni kus tarkvarale uue funktsionaalsuse arendamise käigus on tähtaja nimel toodud ohvriks koodi kvaliteet ja disain. See võib anda lühiajalise eelise, kuid on pikas perspektiivis kahjulik. Tehnilise võla kahandamine on tarkvara kvaliteedi parandamiseks ja finantsprobleemide vähendamiseks ülioluline.

Samas on tehnilise võla arvutamise protsess kohmakas, sest erinevad tehnilise võla arvutamise meetodid kasutavad väga erinevaid meetrikaid ja mudeleid. Käesoleva magistritöö eesmärgiks on analüüsida alternatiivset meetodit tehnilise võla arvutamiseks, kasutades mikroteenustel põhinevat platvormi suuretmahuliste vabavaraprojektide kogumil.

Analüüsi oodatavav tulemus on viitematerjal mida teadurid saaksid kasutada tehnilise võla arvutamise etalonvalemi loomiseks.

Töö raames valiti mikroteenustepõhine platvorm sest sellel on monoliitsete rakenduse eesmitmeid eeliseid – lihtne juurutamine, väiksem arv keskkonnasõltuvusi, ja hooldatavus. Mikroteenuse platvormil kasutati kolme erinevat tehnilise võla leidmise meedotit: Hooldatavuse Indeks, SIG Hooldatavus, ja Sqale.

Olles mikroteenuste platvormi abil analüüsinud 124 java projekti, võib statistilise analüüsi tulemuste põhjal öelda et valitud kolme tehnilise võla arvutamise meetodil puudub vastastikune sõltuvus.

Kõigi kolme tehnilise võla leidmise meetodi puhul viib parameetrite TotalOp, TotalOpr, CC, Duplication, UnitSize, and NumberParameters, WMC kõrge väärtus madala hindestamiseni, ning seega kõrge (halva) tehnilise võlani.

Lahenduse skaleeruvust saab määrata läbi mikroteenuste jõudluse, mis selgub API päringutele kulunud ajast. Koormustestiks kasutati postmani. Testimiseks kasutatud API kasutas eelmainitud kolme erinevat tehnilise võla leidmise meedotit. Eksperimendi käigus kulus päringul 5 kausta analüüsimiseks 1m 40.38s, ning aeg paraneb järjekindlalt, kulutades 124 kausta analüüsimiseks 34m 20.70s. Platvormi testiti seitsmes interatsioonis ning iga kord saadi ajaliselt erinev tulemus. Päringute kiirus oli igas iteratsioonis erinev ning üldist kiiruse suurenemise tendentsi ei täheldatud. Tuginedes nendele andmetele, siis mikroteenuste jõudlust ei mõjuta mitte iteratsioonide, vaid analüüsitavate kaustade arv.

Võtmesõnad: Mikroteenuste; Hooldatavuse Indeks; SIG Hooldatavus; Sqale; Tehniliseks võlaks; Tehnilise võla leidmise meedotit.

CERCS: P170

Table of Contents

1 Introduction

1.1 Unit of Study

This thesis aims to provide the results of analysis about technical debt alternative measurements by a microservice-based platform on a set of large open-source projects. Technical debt (TD) represents making coding or design constructs that have advantages in the short term but result in adverse impacts in a long time (Avgeriou, P *et al.*, 2016). TD will enhance the cost of development and maintenance when it is left untracked (Nord, R, L *et al.*, 2012). The measurable technical debt can reduce the financial problem. Besides, TD is a method to measure the attempt required to improve software quality (Khomyakov, I *et al.*, 2019).

1.2 Motivation

Decreasing the amount of TD in the software is essential since it produces some issues during software developments. The research result from 107 practitioners from the Brazilian software industry indicates the low quality, delivery delay, low maintainability, rework, and financial are the impactful effects of TD. The impact is not only directing to the project but also disrupting the project team. Low quality refers to any condition that diminishes the quality of an artefact which encompasses errors and the unfixed known defects. Low maintainability includes problems throughout software maintenance activities (including expanded effort to resolve bugs and limited system evolution). Delivery delay refers to the deadlines agreed with the customer is not reached (Rios, N *et al.*, 2018).

Numerous platforms have been built to estimate technical debt utilizing static code analysis. However, the current calculation of TD is cumbersome because every platform utilizes various quality models, metrics, and technical debt remediation models. Each type of technical debt is correlated to many methods for quantitative classification and uses very different measurement units, and entails complicated setups (Brown, N *et al.*, 2010).

Furthermore, plenty of the tools built cannot be related to the identification of technical debt. The tools created a report of code smells instead of measuring a TD index (TDI) in circumstances of budgets or effort (Avgeriou, P *et al.*, 2020). The various tools lead to a complex condition for developers in deciding on the proper TD tools to be used in their tasks.

Therefore, providing the analysis of the alternative debt measurement from the usage of the microservice-based platform can support the researchers in deciding the proper technical debt formula to measure technical debt.

As a result, the technical debt formula can be utilized in building a TD tool in supporting the developer to refactor the code to be better quality and enhance the company's profit.

We selected to build the application in the microservice platform since compared to monolithic applications, microservices have benefits. First, regarding the deployment, microservices are simpler than monolithic applications since the scope is smaller and has less enviromental dependencies. The dependencies include OS versions, library versions, the amount of RAM, etc. Second, troubleshooting is easier to be conducted in the microservice application since the flow of data processed is obvious, therefore when the output is getting error, we can trace the descent of the data in the system (Hazelcast). Third, maintainability, this means the modification in a service of the microservice platform will not result impact the entire application (Viggiato Markos *et al.*, 2018).

I assume that monolithic application would not be appropriate for further research because it would make hard to implement the new TD identification method with the number of evaluated directory project. Besides, if the metric result from the calculation in the new method is as expected, it would be difficult to track the cause of the error.

1.3. Research Question

The research questions are represented below.

RQ1: How is the correlation of the metric result among the TD identification methods?

Purpose: The metric result from each TD identification methods comprised average maintainability rate, average maintainability ranking, and sqale rating. The correlation among TD methods is figured out in order to see whether the measurement result from one method can predict the measurement result from another method.

Approach: We used Kendall τ method correlation and Scatter plot to display the correlation between every two metric results of the technical debt identification method (MI & SIG Maintainability, Sqale & Software Improvement Group (SIG) Maintainability, and MI & Sqale).

RQ2: What is the correlation between different code metrics and results from the TD identification methods implemented?

Purpose: Each TD identification method has code metrics as parameters to be calculated. From this research question, we can see which code metrics has the greatest influence in the result of TD identification method.

Approach: We used Kendall correlation to see the correlation between each value of code metric used in each technical debt identification method and the value of metric result in each technical debt identification method.

RQ3: How is the microservice approach scalable for implementation?

Purpose: From this research question, we see the scalability of microservice in implementing the calculation of technical debt for every technical debt identification method.

Approach: We analyze the microservice platform's scalability by analyzing the platform's performance. The platform's performance is counted from the time for a query executed during load testing by Postman. Besides, we analyze how the implementation of microservice platform make easier to add more TD idenfication methods.

This thesis is structured as follows. Chapter 2 presents the background of TD definition from several perspectives, some TD identification methods, and microservices description and its benefits. Chapter 3 provides the research design and design steps. In Chapter 4, we describe the implementation of a microservice-based platform. In Chapter 5, we explain the analysis result from the usage’s platform. Chapter 6 concludes the thesis.

2. Background

2.1. Definition of Technical Debt

Warn Cunningham initially introduced technical debt as a metaphor to describe poor code, and the purpose of design to develop new features to gain short-term outcomes, creating interest payment in the future (Cunningham, W, 1992). Furthermore, the technical debt is researched via the several perspectives: enterprise perspective, stakeholder perspective, and broadening perspective, which will be elaborated on below.

2.2. Technical Debt from Enterprise Perspective

The authors reveal that there are two other sources of debt likely causing the TD apart from product architects. The first type is *Induced Debt*, which comes from stakeholders in the project or across the portfolio—the issue stems from the demanding requirement to fulfil a precise release date. Furthermore, the TD usually is resulted from the communication divergence between the technical and non-technical stakeholders.

The second type is *Unintentional Debt*, created by the conditions such as procurement, new adjustment requirements, or modifications in the market ecosystem. In the perspective of technical architects who authors interviewed, the Unintentional Debt typically was represented much more dubious than the Intentional Debt (Klinger, T. *et al.*, 2011).

2.3. Technical Debt from Stakeholder Perspective

Technical debt refers to any chasm in the technology infrastructure or its application which burdens the required level of quality. “The required level” diverges in each environment based on application’s supported process. The functionality of an application that is not necessary is not classified as technical debt. For example, access controls, this functionality is only required by an application that has sensitive data. The case that any application has no access controls would not determine as technical debt because that functionality is not requisite. Nevertheless, when an application has access controls that are not required but need a cost to manage, it might meet as debt. This condition will arise if the management reduces the productivity or the team is necessary to handle them.

When the principal quantity of debt linked to an essential investment in the technical environment, it will be counted as technical debt. The gaps in the supporting process are not to be

counted as technical debt. However, it will be categorized as debt only for technology prescribed to implement specific procedures (Theodoropoulos, T, Hofberg, M, and Kern, D, 2011).

2.4. Technical Debt from Broadening Perspective

TD represents the cost that should be paid for fixing and maintaining issue in a system as an impact of wrong decisions in the architecture, code, requirements, and other elements. Refactoring is one of the solutions that is applied by the development team for paying back debt (Avgeriou, P *et al.*, 2016).

McConnel classifies TD into two types. The first kind of technical debt is *Incurred Unintentionally*. This TD is caused by destructive code. In other circumstances, this debt can occur when the team flounders in developing a debt-laden platform and accidentally makes more debt.

The second type is *Incurred Intentionally*. This type usually appeared when an organization decides to enhance the present condition without considering further the effects which would yield in the future—for instance, writing a code that does not follow the coding standards and planning to clean the code later—another example, conducting unit test after the release of the software (McConnell, S, C, 2013).

Another classification comes from Krutchen. He defines the taxonomy as a possible activity of an organization related to technical debt. The technical debt landscape represents this taxonomy in Fig. 1.

The elements are distinguished into two types. Firstly, visible components consisted of adding new features and fixing the defects—secondly, the invisible parts are visible for software developers. As the figure presented, they are taking care of evolution issues on the left, while on the right, they are taking care of internal and external quality issues. There is a limitation debt to the invisible elements in the rectangular box (Kruchten, P, Nord, R, and Ozkaya, I, 2012).

2.5. Identification of Technical Debt

Halepmollası describes the factors contributing to Technical debt by considering the phases of SDLC (Software Development Life Cycle) into the following categories:

• In the coding stage, code-level TD might be determined by utilizing Object-Oriented metrics, code metrics (e.g. Cyclomatic complexity), and other static source code analysis methods, e.g. code smells.

• In the design stage, TD would be identified by means of categorization methods and automated detection of changeability-related architectural parts.

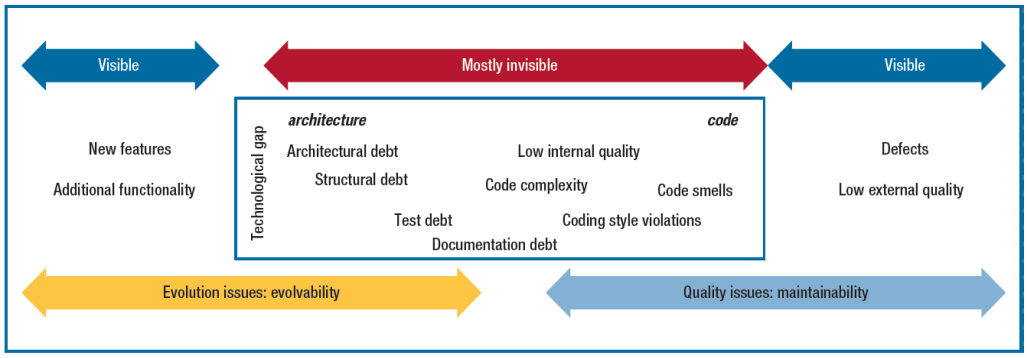


Fig. 1. Technical Debt Landscape (Source: https://resources.sei.cmu.edu/).

• In the test stage, types of measurement test will be used to specify the TD, such as test coverage and test effectiveness.

• In the code review stage, a set of process metrics is used to assess the review activities. The metrics such as the number of reviewers involved, review coverage with respect to the code/change modified, review discussions (Halepmollası, R., 2020).

Liu et al. elaborate the identification of technical debt into more detail by utilizing deep learning frameworks. There are five categories that have been discovered in non-deep learning projects consisted of design debt, defect debt, documentation debt, requirement debt and test debt.

1. *Design debt* is the debt that represents sub-optimal design, e.g., lack of abstraction, disorganized code, long methods, bad implementation, workarounds, or short-lived solutions on the consumption of other internal functions. Poor implementation code is another reason that induces the technical debt is, which implies an agreement code quality and specific non-functional or functional requirements. This technical debt is usually produced because the developers are not familiar with the code that they are implementing. Design debt can also be determined by statically evaluating source code or investigating code fulfilment to standards (Yuepo, G and Carolyn S, 2011).

2. *Defect debt* directs to code that performs in unexpected steps, and developers postpone fixing it because of subsidiary reasons (e.g., extra code complexity of repair or time insistence). The defect debt is often aroused through extensively adopted probability and statistics-based algorithm. The defect debt also contributed to the delay caused by the demand for cooperation with development teams to repair the defects. This debt can cause inconstent data dependencies in applications. Furthermore, the accumulation of the defects leads the framework developers to develop their future implementation to depend on the tricky code, which reduces the quality of frameworks. Another view states that defect debt can be determined by correlating test results to change reports. The defects items show the defects that are discovered but not repaired (Liu, J *et al.*, 2020).

3. *Documentation debt* corresponds to lacking, insufficient or incomplete documentation that describes the detail section of the program. The main factor that brings on documentation debt is that the expectation of several framework developers to other developers or experts to present an obvious description of the code (Liu, J *et al.*, 2020). The documentation debt can be detected by comparing modification reports to histories of documentation version. If the modifications are created without modification to documentation, the appropriated documentation that is not revised is documentation debt (Yuepo, G and Carolyn S, 2011).

4. *Requirement debt* means deficiency of the method, class or program at a specific time, which could refer to the initially prepared fulfilment of the task surpass the development timeline. It can also indicate events when new requirements are incurred during the development of existing requirements but cannot be implemented due to time insistence or other obstacles. Requirement debt can induce the application developers to realize the detailed functions by themselves. Requirement debt can induce the application developers to not be conscious of the methods, and the implementation of the methods by the developers produces unexpected results. To pay back the requirement debt, application developers have to develop these functions by themselves.

5. *Test debt* refers to the demand for enhancement to overcome incompleteness in the test sequence. Adequate testing is suitable for the project. However, the dead tests can produce the static analysis tools to be incapable of figuring out discarded source code. The remains

of discarded source code can result in huge damage (Liu, J *et al.*, 2020). Another way to identify testing debt through comparing test procedures to test results. The tests planned but not executed are identified as testing debt items (Yuepo, G and Carolyn S, 2011).

2.6. Self-Admitted TD

Self-Admitted TD (SATD) refers to a situation when developers know their commit code is either is incomplete, requires rework, produces errors, or is a temporary workaround. These quick fixes are often referred to as TD (Potdar, A and Shihab, E, 2014). Another study states that SATD is source code comments indicating that the corresponding source code is (temporarily) inadequate, e.g., because the implementation is poorly implemented, buggy, or smelly (Fucci, G et al., 2021). This statement, also similarly said by Wehaibi et al. that SATD is TD that developers themselves report through source code comments.

Potdar and Shihab performed an exploratory study to understand self-admitted TD better. They used source code comments to detect self-admitted TD and examined their analysis on four large open-source projects consisted of Eclipse, Chromium OS, ArgoUML and Apache HTTPd. They contribute to identify comment patterns that indicate self-admitted TD, on determining why self-admitted TD is introduced and how much of self-admitted technical debt is actually removed after their introduction. Their discovery presents 62 different comment patterns that indicate self-admitted technical debt from 101,762 code comments that they examined. The other finding is developers with higher experience tend to introduce more self-admitted TD and that time pressure as well as complexity do not correspond with self-admitted technical debt. Finally, they found that even after multiple releases, only between 26.3 -63.5% of the self-admitted technical debt is removed after its introduction (Potdar, A and Shihab, E, 2014).

2.7. Self-Admitted and Software Quality

Wehaibi, Shihab, Guerrouj examined the relationship between SATD and software quality in five open-source projects. Specifically, they analyzed whether (i) files with SATD have more defects compared to files without SATD, (ii) whether SATD modifications arouse future defects, and (iii) whether SATD-related modifications tend to be more cumbersome. They gauged the adversity of a modification in terms of the amount of churn, the number of files, the number of modified modules in a change, as well as, entropy of a change. They determined SATD at two levels: (i) file-level and (ii) change level and follow the methodology implemented by

Potdar and Shihab. The methodology used 62 patterns that indicate the appearance of SATD. In their study, they identified comments that indicate as SATD if they include any of the 62 patterns that signify SATD.

They referred to their SATD files to identify the SATD changes. They reviewed the modification and analyzed all the files that were affected by that adjustment. If one or more of the files by the adjustment is (are) SATD file(s).

They used regular expressions in changelogs from the Git version control system to identify whether a change resolves a defect. After determined the SATD files and SATD changes, they analyzed the bugs in each. The procedure to do this is following the techniques implemented in past research to identify the number of defects in a file and to identify defect-inducing changes. Their findings indicate that although TD shall have adverse effects, its effects are not associated with bugs, otherwise forming the system more cumbersome to modify in the future.

Fucci, G studied SATR in terms of what type of information is being transferred and how it is processed. They conducted an in-depth of the content provided in SATD comments and the expressed sentiment in order to gain a better understanding of developers' habits in SATD syntax and the expressed sentiment.

They manually investigated 1038 instances from an existing dataset and established a taxonomy consisted of 41 categories with nine top-level ones: functional issues, out-of-date SATD comments, bad implementation choices, waiting, deployment problems, partially/not implemented functionality, testing problems, documentation problems, and misalignment-related problems.

The findings of the study represent that (i) the SATD content is throughout life-cycle dimensions determined in prior work (ii) comments linked to functional issues, or on-hold SATD are commonly more adverse than poor implementation decisions or partly constructed functionality, and (iii) different from the observations in the prior work, only an outnumber of SATD comments influences external references (Fucci, G *et al.*, 2021).

2.8. SQALE Method

SQALE method is defined as Software Quality Assessment based on Life Cycle Expectations. The method was developed in France by Inspearit (formerly called DNV ITGS) to assess and maintain the source code quality of project. SQALE method is open source and can be applied to several language programming. There are numeous tools utilizing this method for measurement (SonarQube, SQuORE, CodeQ). The method represents 4 key concepts:

2.9.1. SQALE Quality Model

The SQALE Quality Model is applied for adjusting and coordinating the non-functional requirements that connect to code quality. It is arranged in three hierarchical levels. The first level is structured of characteristics, the second of sub-characteristics. The third level is structured of requirements that refer to the source code's internal attributes. These requirements usually based on software context and language. An approach related to the typical lifecycle of a source code file is used to determine the primary quality attributes of our model.

2.9.2. SQALE Analysis Model

Estimating a software source code in SQALE method is similar to measure the range between its current condition and its quality objective. This range enables normalizing all internal measures on a regular scale. Fig. 2 summarizes the approach of the SQALE quality model. The generic SQALE model is derived according to the implementation technologies (design and source code languages) and the tailoring needs of the project.

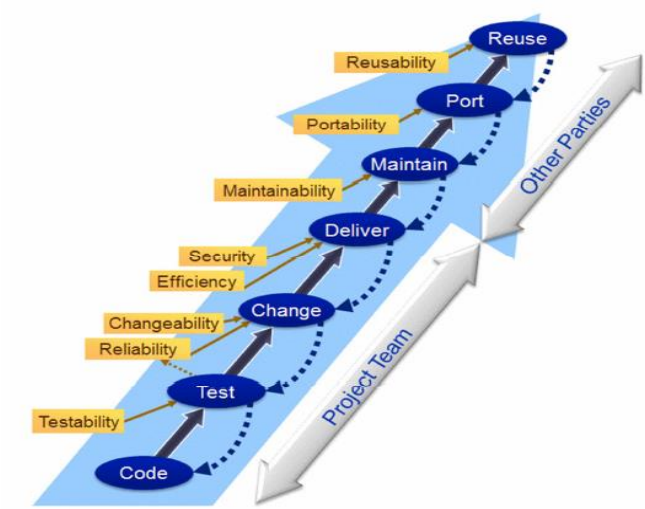


Fig. 2. An approach related to the typical lifecycle of a source code file

Source: (Letouzey, J, 2012)

To estimate the source code, the SQALE analysis model utilizes a remediation index, related to each part of the software source code (for instance, a file, a module or a class). The index means the remediation effort which would be required to fix the non compliances identified in the component, against the model requirements (Letouzey, J, 2012). The model requirements illustrated in Fig. 3.

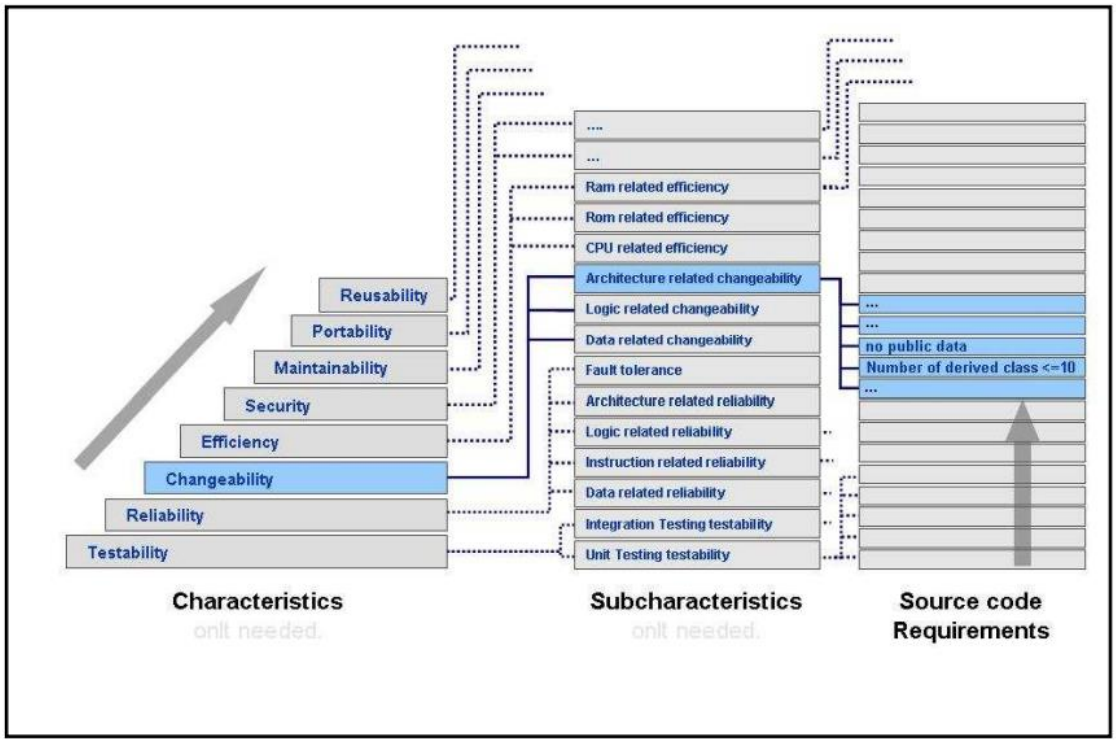


Fig. 3. Model requirements in Sqale Model

Source: (Letouzey, J, 2012).

To calculate TD in Sqale, we compute Remediation Costs (RC) that are represented by the following formula:

$$RC = \frac{\sum ruleeffortToFix(violationsrule)}{8\left[\frac{hr}{day}\right]} \qquad (1.1)$$

Source: (Strečanský, P, Chren, S, and Rossi, B, 2020)

2.10. Maintainability Index

2.10.1. Formation of the MI

The Maintainability Index (MI) is defined as a source-code estimation for maintainability. Around 50 regression models were established to find out the efficient and simple models that could be implemented to a wide range of software systems. A four-metric polynomial based on average Halstead's volume metric (V), average cyclomatic complexity V(g), average lines

of code (LOC) and an average number of comment lines (PerCM) was used in the regression model. The formula of MI exists in two variants that are represented below.

$$MI = 171 - 5.2 \times ln(aveV) - 0.23 \times aveV(g) - 16.2 \times (ln(aveLOC)) \quad (1.2)$$

or

$$MI = 171 - 5.2 \times ln(aveV) - 0.23 \times aveV(g) - 16.2 \times ln(aveLOC) + 50 \times sin(sqrt(2.46 \times perCM)) \quad (1.3)$$

In which,

aveV = average Halstead Volume V per module
aveV(g) = average extended cyclomatic complexity per module
aveLOC = average count of line of code per module
perCM = average percent of lines of comments per module

The components are measured at the module level and then averaged. The word 'module' utilized here intends to the modest unit of functionality. Regarding the programming language, this means a procedure, function, method, subroutine or section.

Formula (1.3), which consists of percentage comments, should only be implemented if the comments are valid, instead of, e.g. pieces of program code that have been commented out. Otherwise, formula (1.2) should be implemented. A greater MI value implies better maintainability.

2.10.2. MI Components

MI subsists of four components that are explained further below.

a. Lines of code

This metric indicates the program or module size. Lines of Code (LOC) still is a favoured metric, not in the least because of its directness.

b. Halstead Volume

Halstead Volume is one metric of the family of metrics called Software Science, initiated by Maurice Halstead. All Halstead metrics are regarding four scalar figures produced directly from a program's source code:

n1 = the number of distinct operators

n2 = the number of distinct operands

N1 = the total number of operators

N2 = the total number of operands

From these figures, Halstead first gauged vocabulary (n) and length (N):

$n = n1 + n2$ *(1.4)*

$N = N1 + N2$ *(1.5)*

Finally, program volume (V) is measured as:

$V = N * 2 \log(n)$ *(1.6)*

(Oppendijk, 2018).

Halstead Metrics is a measurement to calculate a program module from the source code. The measurement is determined by identifying the size of complexity quantitative from operator and operand in the system module (M.A.M.Najm, N, 2014).

2.11. Software Improvement Group Maintainability Method

Software Improvement Group (SIG) Maintainability method is developed as a layered model for measuring and rating the technical quality of a software system in terms of the quality characteristics of ISO/IEC 9126.

In the first layer, analysis of source code is conducted to gain measurement data about the software system. The analysis relates to several metrics such as McCabe complexity numbers, parameter counts, dependency counts, LOC, and duplicated LOC. These metrics are gathered on the basic building blocks level that involves lines, units (e.g. methods or functions) and modules (e.g. files or classes). Subsequently, these building blocks metrics are mapped onto ratings for properties at the level of the entire software products comprises volume, duplication, and unit complexity.

In the SIG model, an ordinal scale between 0.5 and 5.5 is used on all levels, with a rounding convention to integral numbers of stars between 1 and 5 (Bijlsma, D. *et al.*, 2012). The mapping functions for volume and duplication are straightforward translations of LOC-based metrics to ratings. The remaining mapping functions make use of *Risk Profiles* as an intermediate device.

Each profile is a partition of the volume of a system (measured typically by LOC) into four risk categories: low, moderate, high, and very high risk. Source code property ratings are mapped to ratings for ISO/IEC 9126 sub-characteristics of maintainability following dependencies summarized in a matrix that is described in Fig. 4.

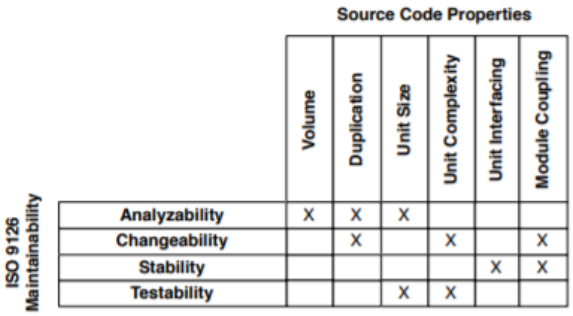


Fig. 4. SIG's quality model for software maintainability

Source: (Nugroho, A, Visser, J, and Kuipers, T, 2011)

In this matrix, an x is located whenever a property is assumed as a prominent influence on a certain sub-characteristic—the explanation of each property of SIG Maintainability as follows.

a. Volume

The evaluation of the volume property is measured on the basis of the number of lines of the source code.

b. Duplication

A line of code is acknowledged redundant if it is part of a code fragment (larger than six lines of code) that is duplicated literally (modulo white-space) in at least one other location in the source code.

c. Unit Size

The size of the units of a software product is defined by total the number of lines of code within each unit.

d. Unit Complexity

The complexity of each unit is defined in terms of the McCabe cyclomatic complexity number. This number performs the number of non-cyclic paths through the control-flow graph of the

unit and can be gauged by counting the number of decision points that are appeared in the source code.

e. Unit Interfacing

The size of the interface of a unit can be counted as the number of parameters (also known as formal arguments) that are determined in the signature or declaration of a unit.

f. Module Coupling

The coupling of the modules of a software product can be quantified as the number of incoming dependencies, such as invocations, per mõdule (‘SIG/TÜViT Evaluation Criteria Trusted Product Maintainability: Guidance for producers Version 11.0’, 2019).

The sub-characteristic rating is gained by averaging the ratings of the properties where an x is placed in the sub characteristics line in the matrix. Finally, the overall maintainability rating is accumulated by averaging all sub-characteristic ratings.

2.12. CAST Application Intelligence Platform

The CAST Application Intelligence Platform (Cast 2011) is a commercial product of an automated system to measure the software quality and the performance of labour. This system utilizes a quality model based upon a number of characteristics that is similar to, but not explicitly based upon, ISO 9126 (Hegeman, J, H., 2011).

Bill Curtis et al. propose the estimation of TD Principal as a calculation of three variables — the number of should-fix violations in an application, the time to fix each violation, and the cost of labour. They gauge each of these variables to develop an equation for calculating TD-principal. The hours to resolve each violation could be feasible from historical effort data. The cost to resolve violations can be appointed to the average load rate for developers charged to the application development. The formula for measuring TD-principal is represented as follows.

TD-principal = ((S high-severity violations) × (percentage to be fixed) × (average hours needed to fix) × (US$ per hour)) + ((S medium-severity violations) × (percentage to be fixed) × (average hours needed to fix) × ($ per hour)) + ((S low-severity violations) × (percentage to be fixed) × (average hours needed to fix) × ($ per hour)).

Table 1. Parameter values for three estimates of TD-principal

Variable	Parameter values		
	Estimate 1	Estimate 2	Estimate 3
Violations that must be fixed			
High-severity violations	50%	100%	100%
Medium-severity violations	25%	50%	
Low-severity violations	10%		
Hours to fix			
High-severity violations	1 hour	2.5 hours	10%—1 hour 20%—2 hours 40%—4 hours 15%—6 hours 10%—8 hours 5%—16 hours
Medium-severity violations	1 hour	1 hour	
Low-severity violations	1 hour		
US$ per hour			
All violations	75	75	75

Source: (Curtis, B, Sappidi, J, and Szynkarski, A, 2012).

They utilized three contexts for these parameters (see Table 1) to examine their consequences on TD-principal estimates. They found that the average costs for many IT organizations, especially where they have a mix of on and offshore operations, ranks at US$70 to $80 per hour. Therefore, they used the same hourly standard in all three estimates. In each estimation, they ranged the percentage of violations at each harshness rank that would be prioritized for remediation, focusing on fewer harshness rank in estimates 2 and 3. The sample and the data reported here are gained from the Appmarq benchmarking repository maintained by CAST Software. They analyzed 700 applications submitted by 158 organizations utilizing CAST's Application Intelligence Platform (AIP) for this examination. They reviewed the whole application, implementing more than 1,200 rules to detect good architectural and coding practices violations. They gained these rules from software engineering articles, repositories, online discussion groups of structural quality problems, and complaints from customer experience regarding defect logs and application architects.

The AIP contains parsers for the 28 languages displayed in Fig. 5, and a universal analyzer provides a limited parse of languages that do not own devoted parsers. The AIP implements several rules to recognize violations of its architectural and coding rules (the examples of the rules implemented in the Detected violations column in Fig. 5).

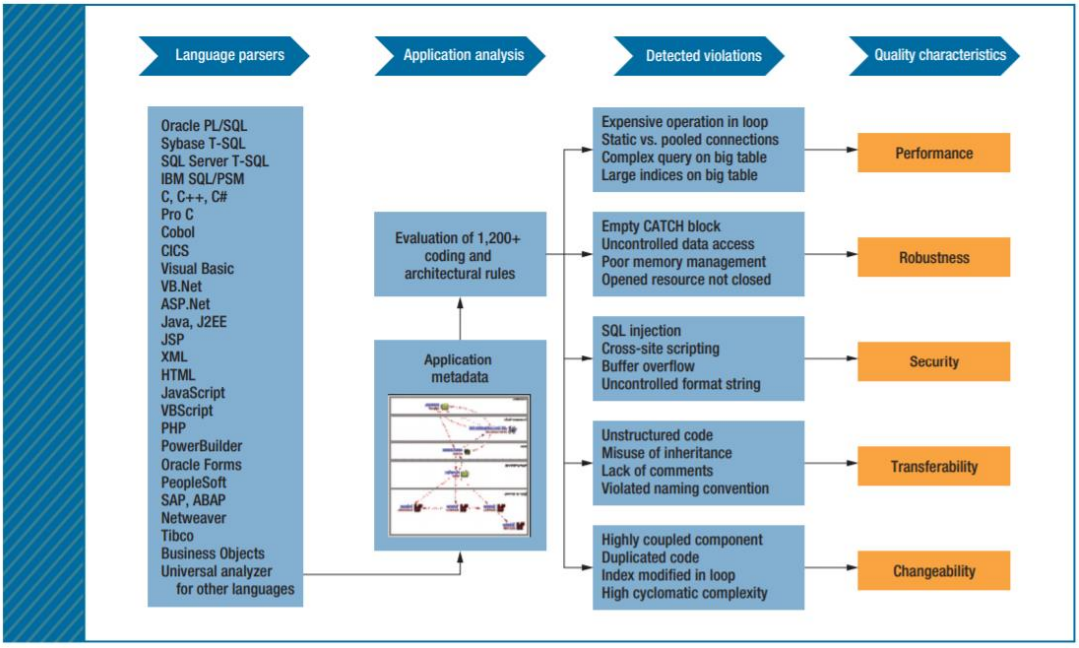


Fig. 5. Approach for estimating Rework Fraction from the Software Monitor.

AIP scoring starts by recognizing the amount of occasion for a rule to be triggered and the percentage of times that rule was violated. Each violation is weighted by its severity and aggregated to the application level. (Users can set harshness weights for each violation to fit local priorities, but the results they provide here are depending on the original weights.) The AIP presents a sequence of management reports and a portal to lead developers to locations of violations in the source code can be remediated. The management report accumulates violation scores into scales for the five quality characteristics shown in Fig. 5.

2.13. Microservices

2.13.1. Definition of Microservices

Microservices is a software architecture style that is purposed to design and structure an application as sets of fine-grained and loosely coupled service. Every part of microservice comprised functionality of business logic of systems. Their effect of loose coupling allows their software development to utilize variant programming language and data. The communication of microservices is handled directly by HTTP resource API or indirectly using message brokers (see Fig. 6).

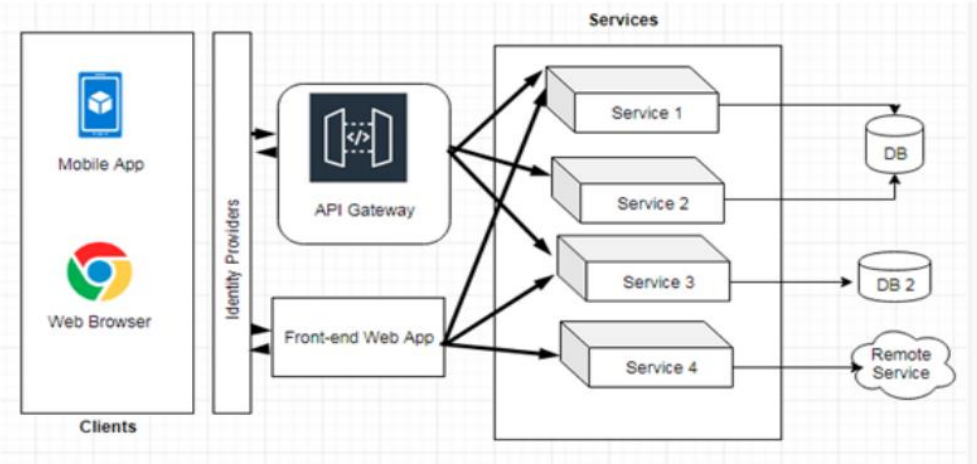


Fig. 6. Microservices Architecture (Source:(Irudayaraj, Prathap and P, Saravanan, 2019))

The primary aspect of microservices architectural style that is distinguished from monolithic and service-oriented architectures is the smaller size and scalability (Hannousse et al., 2020). The other advantages are microservices presumes a greater level of service-specific self-reliable related to the form of structure, technology, and functionality. Furthermore, they can also be deployed and operated without modification in other microservices (Rademacher et al., 2018).

2.13.2. Benefit of Microservices

There are several advantages that are represented by microservices. Firstly, continuous delivery. In which each service can be executed separately in its environment and domain. Secondly, deployment quickness; allows the system to be deployed at the maximum speed. Thirdly, the tools and IDE provided facilitate developers to build a more forceful software product. Furthermore, minimize cost; Compared to monolithic, microservices does not need each piece of code to be communicated with other parts. In which a modification in one area leverage other features. Next, fault tolerance. It enables them to adapt more handily rapidly against the changes that occurred in industries (Ali, 2020).

Moreover, microservices architecture aims to dissolve the application into loosely coupled service, put on actualizing single functional service and allow it to be individually maintained. The scalability which owned by microservices enables to enhance application workload and performance (Aksakalli et al., 2021).

3. Research Design

The research method proposed is a design science research methodology. This methodology was selected since the methodology of this research corresponds to the steps of the design research methodology, which consisted of programming, data collection and analysis, synthesis of the objectives and analysis results, development prototyping, and documentation (Peffers, K, Rothenberger, M, A, and Tuunanen, T, 2008). The design science research of this thesis comprised several steps that is described in Fig. 7.

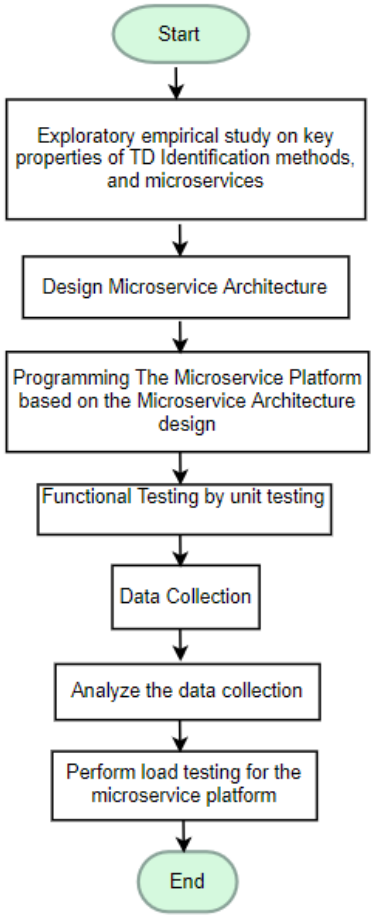


Fig. 7. The design research steps

In the first stage, we conducted an exploratory empirical study on the critical properties of the TD identification method to see which TD identification methods are appropriate to implement in the microservice platform. From this activity, we selected three TD identification methods includes Maintainability Index, Software Improvement Group (SIG) maintainability Group and Sqale.The reasons for selecting these methods are all methods can calculate the TD from the source code extracted, availability package for implementation all metrics for TD calculation in all TD identification methods.

The research study also focused on microservices to gain knowledge of the benefit of microservice than the other software implementation type. Besides, to design and implement a microservice platform for calculating TD.

The second stage refers to designing microservice architecture. The microservice is selected as a platform implementation for this thesis because this platform is assumed can run calculation TD for plenty of directory projects more efficiently. Furthermore, the platform would make the development easier if the new TD identification method wants to be added.

The third stage is implementing the microservice platform based on the microservice architecture design. The microservice consisted of five services, including the first service for uploading and extracting directory, the second service for running all TD identification methods, and there remaining services for implementing calculation of three TD identification methods. The detail of each service will be explained further in the implementation chapter.

After programming the microservice platform, we performed functional testing for all functions that handle the calculation process in all TD identification methods in the fourth stage. The functional testing aims to ensure that all functions for the calculation process work well and produce the right output.

The fifth stage is data collection. The data is collected from the database in the microservice platform. To obtain the data, we should first run the calculation for all TD identification methods in the microservice platform and run the function of extracting the data to .csv format.

The data which is collected will be utilized in the sixth stage. In the sixth stage, we run statistical code for the data collection by python before analyzing the data. The analysis result obtained in this stage can answer RQ1 and RQ2.

The last stage is performing load testing for the microservice platform. The load testing uses Postman with API for calculating all TD identification methods. The result from this stage will be analyzed and can answer RQ3.

3.1. Design and Development

The technical debt calculation method literature was used as the theoretical basis for the calculation methods in the microservice platform. Some metrics for the calculation method were implemented from the “Maven” library. Furthermore, the microservice literature was also used for designing a detailed technical architecture between some calculation methods, which was implemented in every service in the microservice platform. The database is needed to save the

list of directory that will be measured by the calculation method and to save the result of the measurement.

3.2. Implementation

The microservice platform was built in Java and proposed to measure applications that are developed in Java. The microservice contains the following services: The first service is the directory service that will be responsible for uploading and extracting a zip file and save the list of directories in the zip file to the database. Then, the list of directories will be tested in the measurement service.

The second service namely measurement service will run the calculation method from three services consisted of MIService, SIGService, and SqaleService that perform technical debt method calculation services respectively. The result of the calculation from every technical debt measurement service that is saved in the database can be extracted to the .csv document in order to be analysed. By implementing this microservice, we can answer RQ3.

3.3. Evaluation

The microservice platform was used to calculate technical debt in a set large open source project. The open-source project which was used is a set of projects from the faculty of Informatics, Masaryk University. The result from the calculation process will be continued to the analysis process as the evaluation. The analysis process is implemented by comparing the usage of the microservice platform. The usage consists of the calculation result from every technical debt measurement service. By the analysis process, we can answer RQ1 and RQ2.

3.4. Communication

The theories of technical debt calculation which was implemented for this project were provided in the Open Access Archive – HAL (Letouzey, J and Coq, T, 2010) , QUATIC 2007 (Heitlager, I, Kuipers, T., and Visser, J, 2007), and IOSR-JCE (M.A.M.Najm, N, 2014). Meanwhile, the comparison between technical debt measurement methods was reported in Hindawi Scientific Programming (Strečanský, P, Chren, S, and Rossi, B, 2020). Then, the library of metrics that were used in the microservice platform referenced from the metrics that were explained in Computer Standards & Interfaces (Thomas, P, Escalona, M, J, and Mejías, M, 2013).

3.5. Contribution

The contribution of this study is developing a microservice-based platform that implements three TD identification methods consisted of Maintainability Index, SIG Maintainability, and Sqale. Then, based on the usage platform, we provide the analysis result, which that be used

later as a reference for the researchers in deciding the proper technical debt formula to measure technical debt.

3.6 Validation of The Results

The validation of the result is the implementation of unit testing for every code metric to ensure the code that is implemented can produce a valid result. The unit testing implemented for the code metrics, which is not derived from the maven-library and the calculation for every code metric. The unit testing implementation for the code metric based on the implementation of unit testing of the origin code. Meanwhile, the unit testing implementation for the calculation of code metric depends on the documentation or paper of its technical debt identification method.

3.7 Interpret the result

This research will be interpreted by quantitative analysis. Compared to previous research (Strečanský, P, Chren, S, and Rossi, B, 2020), this research experimented on a larger scale, which on a set of large open-source projects and there is a microservice built which can run the technical debt calculations simultaneously. The analysis process is also getting more explanatory insight by looking at the correlation between different code metrics and result from the technical debt identification methods implemented and the comparison of technical debt identification methods itself.

4. Implementation

In this study, we selected 124 projects from projects in Faculty informatics, Masaryk University. There are three technical debt identification methods implemented in three services respectively in the microservice platform that we developed. The technical debt identification methods consisted of Maintainability Index, Software Improvement Group Maintainability, and Squale. Each technical debt contains some different code metrics.

4.1. Sequence Diagram of the Microservice Platform

The microservice platform was developed in Java and consisted of five services. Firstly, one service named directory handle uploading and extracting the .zip directory file that will be processed. Secondly, one service called MeasurementService will run the three technical debt identification methods services. Lastly, three technical debt identification methods services named MaintainabilityIndexService that will process the directory by Maintainability Index Method, SIGMaintainabilityService that will process the directory by SIG Maintainability Index Method and SqaleService that will process the directory by Sqale method. The flow of the application is described by the Sequence Diagram in Fig 8.

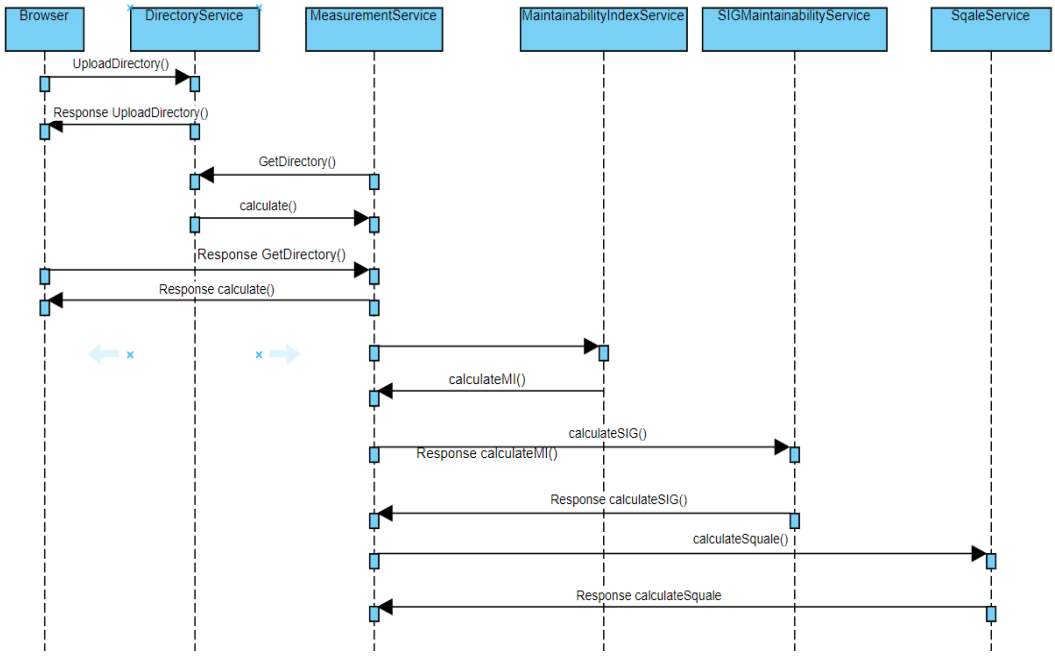


Fig. 8. Sequence Diagram of the Microservice Platform

4.2. Hexagonal Architecture of Microservice Platform

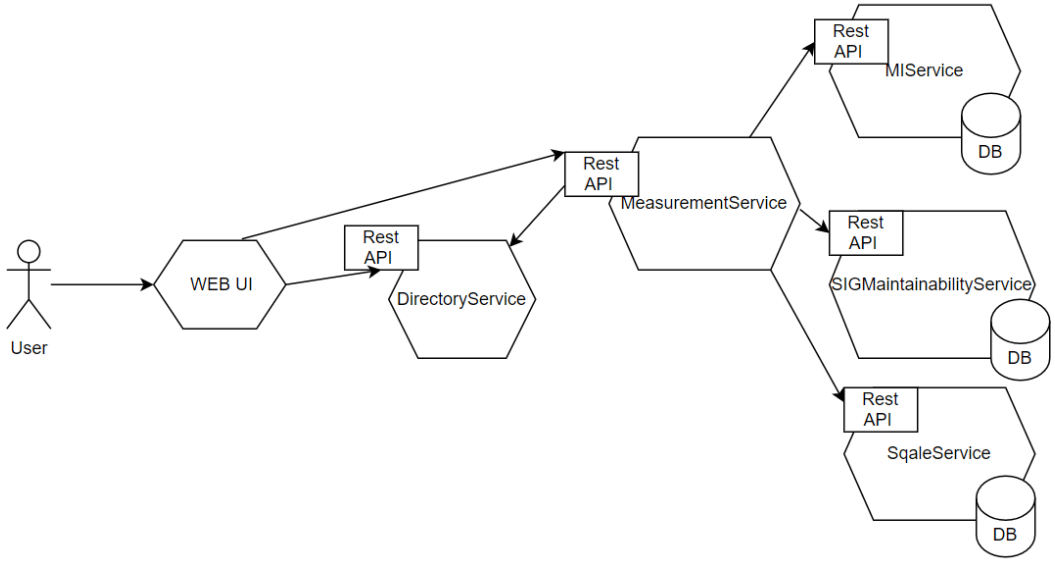


Fig. 9. Hexagonal Architecture of the Microservice Platform

As represented in Fig 9, the process begins from the user who accesses the WEB UI of the application, then when the user uploads the file of the directory project that will be analysed, WEB UI will run the upload function in the rest api in the directory service, after the file uploaded, the user will click the button calculate which will call the calculate function in the measurement service. Then the measurement service will call the rest api in the directory service to get the directory project that will be analysed and run all rest api in each technical debt identification method (MIService, SIGMaintainabilityService and SqualeService). Next, the result from each technical debt identification method will be restored in the repository of each technical identification method service.

The list requirements that were implemented in the microservice for DirectoryService and MeasurementService are represented below.

Table 2. The functions implemented in DirectoryService and MeasurementService

Requirements	Function Name	Service Name/Controller Name
Uploading Directory	upload()	DirectoryService
Extract Directory project that is uploaded	getListDirectory()	DirectoryService
Save list of directory projects to the database	saveToRepository()	DirectoryService
Run calculation by All TD identification method for all directory uploaded	calculate()	MeasurementService

4.3. Implementation for Each Technical Identification Method

To interpret the result of the measurement in each technical debt identification method, we use some acronyms that represent in Table 1 in the appendix part

We imported some packages for implementing some metrics in each technical identification methods.

- We imported package from org.eclipse.jdt.core.dom to implement code metric of TotalOp, TotalOpr, TotalUnqOp, TotalUnqOpr.
- We imported package from com.github.mricioaniche.ck. to implement code metric of For code metric of DIT, WMC, and CBO.
- We imported package from com.github.javaparser to implementd code metric of CC, NumberParameters, TotalCom, and UnitSize. This package is also used for check iteration and condition statement in the code
- We imported package from net.sourceforge.pmd.cpd to implement code metric of Duplication.
- We imported package from javancss.Javancss to implement code metric of LOC

4.3.1. Implementation for Sqale Method

We implemented the Sqale method in the Sqale service. In the implementation of Sqale Method, we use some rules in the code that refers to the Quality Model Example for the Java Language. The Sqale model for the microservice implementation is represented in the Table 3.

For all generic requirement, we measure it by the number of file, except for the duplication, because in the duplication metric, it can be measured in a directory instead of each file.

Table 3. Sqale Model for Java Language

Characteristic	SubCharacteristic	Generic Requirement Description	Abbreviation
Maintainability	Understandability	No use of "continue" statement within a loop	M1
Maintainability	Understandability	File comment ratio (COMR) > 35% in a file	M2
Maintainability	Readability	Variable name start with a lower case letter	M3
Maintainability	Readability	File size (LOC) < 1000 in a	M4

		file	
Efficiency	RAM related efficiency	Class depth of inheritance (DIT) < 8 in a file	E1
Changeability	Architecture related changeability	Class weighted complexity (WMC) < 100 in a file	C1
Changeability	Architecture related changeability	Class specification does not contains public data in a file	C2
Reliability	Fault Tolerance	Switch' statement have a 'default' condition	R1
Reliability	Logic related reliability	No assignment '=' within 'if' statement	R2
Reliability	Logic related reliability	No assignment '=' within 'while' statement	R3
Testability	Integration level testability	Coupling between objects (CBO) <7	T1
Testability	Unit Testing testability	No duplicate part over 100 token	T2
Testability	Unit Testing testability	Number of ind. test paths within a module (v(G)) <6	T3
Testability	Unit Testing testability	Number of parameters in a module call <6	T4

Source: (Letouzey, J, 2016)

To determine the remedian factor, there are some category that is shown in the Table 4.

Table 4. Remediation Factors associated with various types for Squale Model

NC Type Name	Description	Sample	Remediation Factor
Type1	Corrigible with an automated tool, no risk	Change in the indentation	0.01
Type2	Manual remediation, but no impact on compilation	Add some comments	0.1
Type3	Local impact, need only unit testing	Replace an instruction by another	0.25
Type4	Medium impact, need integration testing	Cut a big function in two	0.5
Type5	Large impact, need a complete validation	Change within the architecture	1

Refers from: (Letouzey, J, 2016)

Then, we mapped the characteristic of the Squale model into various types of Remediation Factors. The mapping is represented in Table 5.

Table 5. Mapping the characteristic of Squale Model into type of Remediation Factor

Abbreviation of Characte-ristic	NC Type Name
M1	Type1
M2	Type2
M3	Type3
M4	Type5
E1	Type4
C1	Type4
C2	Type2
R1	Type1
R2	Type3
R3	Type3
T1	Type3
T2	Type2
T3	Type4
T4	Type2

The RF is used to calculate the RC by multiplying with the amount of metric level that should be fixed. Then, we total the RC and calculate the index of each characteristic level by divide the total RC by the working hours. The total working hours are based on the total of KLOC for every Project.

For example, the assessed Project has 62 KLOC of code. SQALE has been configured so that a LOC costs 0.06 man-days to develop, which is the default value. The total development cost, therefore, is 62*0.06*1000=3720 days (Hegeman, J, H., 2011).

Then, we calculate all RC from each characteristic index and divide it with working hours to obtain the value of All Index. Finally, this value will be rated based on the rating in the Sqale Rating. The Sqale rating is described in Table 6.

Table 6. The Sqale Rating

Index	Rating
0-0.1	A
0.1-0.2	B
0.2-0.5	C
0.5-1.0	D
>1.0	E

Source: (Hegeman, J, H., 2011)

Fig. 9 explains the example of the Squale model implementation in a directory project.

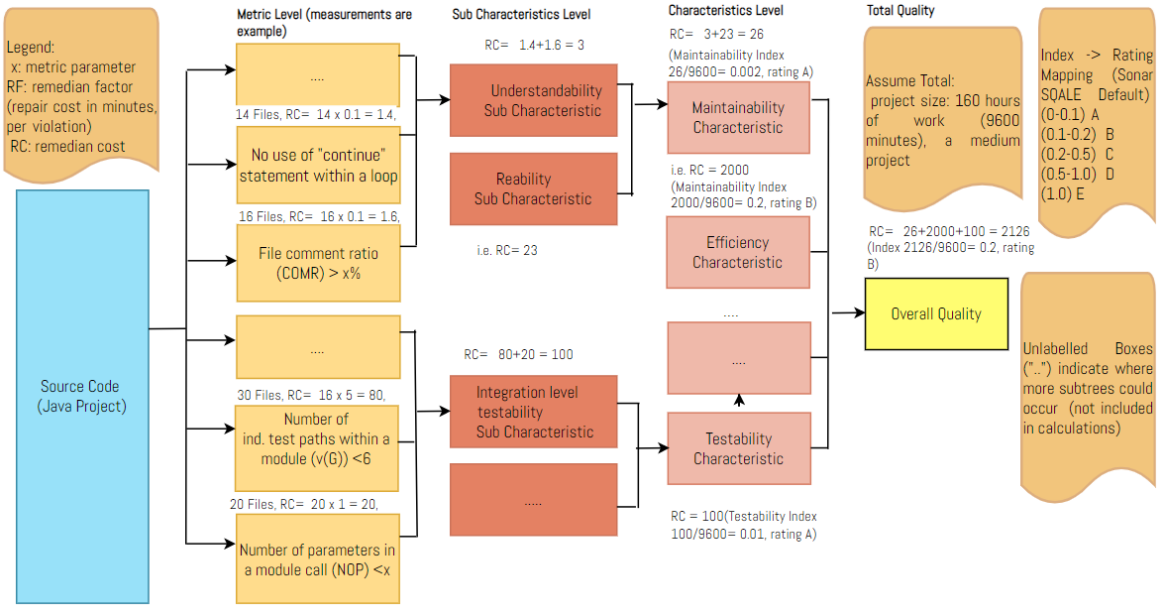


Fig. 9. Full Squale Example

Source: (Hegeman, J, H., 2011)

We implemented some functions in the SqaleService based on the description above that is described in the Table 7.

Table 7. The functions that are implemented in the SqaleService

Requirements	Function	Service name or controller name
Run calculation for directory project by Sqale model	calculate()	SqaleService
Get result total LOC for calculation in Sqale	getLOC(string files)	MetricResult
Get total Ratio Comment for calculation in Sqale	getRatioComment()	MetricResult
Get total DIT for calculation in Sqale	getDIT()	MetricResult
Get total WMC for calculation in Sqale	getWMC()	MetricResult
Get total CBO for calculation in Sqale	getCBO()	MetricResult
Get total CC for calculation in Sqale	getTotalCC()	MetricResult
Get total number parameters for calculation in Sqale	getTotalParameters()	MetricResult

Get remedian cost for maintainability in the directory	All functions in the controller	MaintainabilityCheck
Get remedian cost for efficiency in the directory	All functions in the controller	EfficiencyCheck
Get remedian cost for changeability in the directory	All functions in the controller	ChangeabilityCheck
Get remedian cost for reliability in the directory	All functions in the controller	ReabililityCheck
Get remedian cost for testability in the directory	All functions in the controller	TestabilityCheck
Get total remedian cost	getTotalRCAll()	RC
Get sqale rating and rating for each characteristic	getSqaleRating()	SqaleRating
Calculate working hours	calculateTime(int totalLoc)	WorkingHours
Get number duplication for calculation in Sqale	getNumDuplication(string directoryName)	MetricResults
Insert the Sqale calculation result to the database	insertToDatabase()	SqaleService
Extract data result of MI calculation from the database to the .csv format	export()	ExcelSqaleService

4.3.2. Implementation for SIG Maintainability Method

In the SIG Maintainability service, we implemented the technical debt identification method of SIG Maintainability. There are some metrics implemented consisted of LOC, Duplication, CC, Unit, Size and NumberParameters. The interpretation ranking of LOC is described in Table 8.

Table 8. Interpretation ranking of LOC

LOC Range	Ranking
0<LOC<=66000	5
66000<LOC<=246000	4
246000<LOC<=665000	3
665000<LOC<=1310000	2
LOC>1310000	1

Source: (Heitlager, I, Kuipers, T., and Visser, J, 2007)

The duplication metric should be calculated in the percentage before interpreted in ranking. The percentage duplication is calculated as follows.

*Percentage Duplication = Total Duplication / Total LOC * 100.* *(1.4)*

Meanwhile, the interpretation ranking of the percentage duplication range is described below. The code can be said as duplication which has minimum 100 token (SIG, 2019). Then, to rank the duplication, first we make the total duplication into percentage by dividing it with total loc and multiply by 100. The interpretation ranking of the percentage duplication is shown in Table 9.

Table 9. Interpretation ranking of the percentage duplication

Percentage Duplication Range	Ranking
0< Percentage Duplication <=3	5
3< Percentage Duplication <=5	4
5< Percentage Duplication <=10	3
10< Percentage Duplication <=20	2
Percentage Duplication > 20	1

Source: (Heitlager, I, Kuipers, T., and Visser, J, 2007)

For CC, UnitSize and UnitInterfacing, the ranking is not interpreted by number but by some category consisted of moderate, high, and very high. The interpretation ranking of CC, Unit-Size, and UnitInterfacing are shown in Table 10, Table 11, and Table 12.

Table 10. Interpretation ranking of CC

CC Range	Ranking Percentage
11<=CC<=20	Moderate
21<= CC<=50	High
CC>50	Very High

Source: (Heitlager, I, Kuipers, T., and Visser, J, 2007)

Table 11. Interpretation ranking of UnitSize

UnitSize Range	Ranking Percentage
16<=UnitSize<=30	Moderate
31<=UnitSize<=60	High
UnitSize>60	Very High

Source: (SIG, 2019)

Table 12. Interpretation ranking of UnitInterfacing

UnitInterfacing Range	Ranking Percentage
3<=param<=4	Moderate
5<=param<=6	High
param>6	Very High

Source: (SIG, 2019)

Table 13. Interpretation ranking of Coupling

Coupling Range	Ranking Percentage
11<=coupling<=20	Moderate
21<=coupling<=50	High
coupling>50	Very High

Source: (SIG, 2019)

Then, the interpretation ranking of CC, UnitSize, and UnitInterfacing will be converted to the ranking in the range from 1 to 5. In which the lowest ranking is 1, and the highest ranking is 5. Before converting the interpretation ranking, we should calculate the interpretation ranking into a percentage. The way how to measure the percentage is described below.

*Percentage Moderate = TotalLOC for moderate ranking /TotalLOC *100; (1.5)*

*Percentage High = TotalLOC for high ranking /TotalLOC *100; (1.6)*

*Percentage Very High = TotalLOC for very high ranking /TotalLOC *100; (1.7)*

Furthermore, the percentage will be ranked. The interpretation ranking of the percentage is described as follows.

Table 14. Interpretation ranking of Moderate, High, and Very High

	maximum relative volume		
rating	moderate	high	very high
⋆ ⋆ ⋆ ⋆ ⋆	25%	0%	0%
⋆ ⋆ ⋆ ⋆	30%	5%	0%
⋆ ⋆ ⋆	40%	10%	0%
⋆ ⋆	50%	15%	5%
⋆	-	-	-

Source: (Nugroho, A, Visser, J, and Kuipers, T, 2011)

We implemented some functions in the SIGService based on the SIG Maintainability method calculation concept that is illustrated in the Table 15.

Table 15. The functions implemented in SIGService

Requirements	Function	Service name or controller name
Run calculation for directory project by SIG Maintainability method	calculate()	SIGService
Get result total LOC for calculation in SIG Maintainability	getLOC(string files)	MetricMeasure
Get total CBO for calculation in Sqale	getCbo()	CKMetric
Get total CC and total line of CC in each file for calculation in SIG Maintainability	All functions in the controller	CCVisitor
Get total number parameters and total line of total number parameters in each file for calculation in SIG Maintainability	All functions in the controller	VoidVisitorAdapters
Get number duplication for calculation in SIG Maintainability	getNumDuplication(string directoryName)	MetricResults
Get LOC ranking	getRankLOC()	MetricRank
Get CC ranking	All functions in the controller	CCRanking
Get Duplication ranking	getRankDuplication()	DuplicationRanking
Get Coupling ranking	All functions in the controller	CouplingRanking
Get Unit Interface ranking	All functions in the controller	UIRanking
Get Maintainability ranking	All functions in the controller	MaintainabilityRanking
Insert the SIG Maintainability calculation result to the database	insertToDB()	SIGService
Extract data result from SIG Maintanability calculation from the database to the .csv format	export()	ExcelSIGService

4.3.3. Implementation for Maintainability Index

We implemented Maintainability Index in MI service. There are 9 Metrics that were implemented in this service, consisted of TotalOp, TotalOpr, TotalUnqOpr, TotalCom, AvgCom, TotalLOC, AvgLOC, TotalCC, AvgCC that will be utilized in the Halstead Metric calculation

and Maintainability index computation. The value of the computation will be ranked adjusted the cutoff points. The cutoff points specifically have been identified as follows:

Table 16. MI cutoff values

Maintainability	Formula (1.2)	Formula (1.3)
High	MI >= 50	MI >= 85
Moderate	-	65 <= MI < 85
Low	MI < 50	MI < 65

Source: (Oppedijk, F, R, 2008).

To implement the calculation process of technical debt, we create some function in the micro-service that is illustrated in the Table 17.

Table 17. The function implemented in MI service based on the requirements

Requirements	Function	Service name or controller name
Run calculation for directory project by Maintainability Index	calculate()	MIService
Get number of total operator, total operand, total unique operaator, total unique operand.	All functions in the controller	AstVisitorCheck
Get result total LOC for calculation in MI	getLOC()	MetricCode
Get result total CC for calculation in MI	getCC()	MetricCode
Get result total comment for calculation in MI	getTotalComment(String file)	MetricCode
Calculate Halstead Metric	All functions in the controller	HalsteadMetric
Calculate The Average Maintainability Index	getAvgMaintainabilityIndex()	MaintainabilityIndex
Calculate The Average Maintainability Rate	getAvgMaintainabilityRate()	MaintainabilityIndex
Insert the MI calculation result to the database	setDirResult()	MIService
Extract data result of MI calculation from the database to the .csv format	export()	ExcelFileService

4.4. API Implementation

We Implemented some APIs to support the microservice. Table. 18 provides The detail of API.

Table 18. List of API in the microservice platform

Service	Method	URL API	Detail
DirectoryService	POST	/directories/upload	For uploading directory project
DirectoryService	POST	/directories/saveTo-Repository	For saving the name of folder directory Project to the database
MeasurementSer-vice	POST	/measurements/cal-culate	For running calculation in all technical debt identification methods
MaintainabilityIn-dexService	POST	/mi/calculate	For running calculation techni-cal debt in the method of Maintainability Index
MaintainabilityIn-dexService	GET	/mi/downloadExcel-File	For downloading result in .xlsx format from technical debt cal-culation in method of Maintaina-bility Index
SIGService	POST	/sig/calculate	For running calculation techni-cal debt in the method of SIG Maintainability
SIGService	GET	/sig/downloadExcel-File	For downloading result in .xlsx format from technical debt cal-culation in method of SIG Maintainability
SqaleService	POST	/sqale/calculate	For running calculation techni-cal debt in the method of Sqale
SqaleService	GET	/sqale/downloadEx-celFile	For downloading result in .xlsx format from technical debt cal-culation in SIG Maintainability method

4.4. Functional Testing

We performed unit testing in some functionalities to ensure the microservice platform produce the right output. The function that was tested is presented below. We conducted the unit testing for the functions that are not utilized code from imported packages. All results of the unit testing are valid.

Table 19. The unit testing result

Function	Service name or controller name	Service Name	Result
All functions in the controller	MaintainabilityCheck	SqaleService	Valid
All functions in the controller	EfficiencyCheck	SqaleService	Valid
All functions in the controller	ChangeabilityCheck	SqaleService	Valid
All functions in the controller	ReabililityCheck	SqaleService	Valid
All functions in the controller	TestabilityCheck	SqaleService	Valid
getTotalRCAll()	RC	SqaleService	Valid
getSqaleRating()	SqaleRating	SqaleService	Valid
calculateTime(int totalLoc)	WorkingHours	SqaleService	Valid
getRankLOC()	MetricRank	SIGService	Valid
All functions in the controller	CCRanking	SIGService	Valid
getRankDuplication()	DuplicationRanking	SIGService	Valid

All functions in the controller	CouplingRanking	SIGService	Valid
All functions in the controller	UIRanking	SIGService	Valid
All functions in the controller	MaintainabilityRanking	SIGService	Valid
All functions in the controller	HalsteadMetric	MIService	Valid
getAvgMaintainabilityIndex()	MaintainabilityIndex	MIService	Valid
getAvgMaintainabilityRate()	MaintainabilityIndex	MIService	Valid

4.5. Related Works

Some literature has been proposed for comparing technical debt calculation and technical debt tools. Mostly, the literature focuses on measuring software quality (Strečanský, P, Chren, S, and Rossi, B, 2020) compared three main techniques related to TD Identification that consisted of Maintainability Index (MI), SIG TD models, and SQALE. The three methods experimented on a set of 20 Python projects. An increasing trend of TD was reported from all methods for the projects. However, different patterns appeared in the final evolution of the measurement time series. SIG TD and MI show a growing trend compared to SQALE, which has more periods of steady TD. MI is the method that shows more repayments of TD widely compared to the other methods. SIG TD and MI have similarity in the form TD expands, while MI and SQALE are the slighter similar.

Oppedijk compared two maintainability models based on measuring internal product attributes are discussed consisted of Maintainability Index (MI) and SIG Maintainability Model (SMM) (Oppedijk, F, R, 2008). The experiment is conducted in 73 software projects written in C, C++ and Java, which from 52 closed-source systems and 21 open-source systems. MI and SMM maintainability models perform a moderate, significant degree of positive correlation when both models are implemented side-by-side on similar software projects written in the programming language C, as well as when the models are applied on the OO systems in this research.

The SMM-MI correlations for the C systems are identical to those for the Java systems, while the OO systems researched indeed have a higher average MI than the procedural systems, the leverage of access routines on the height of the average MI of OO systems is limited. The research has reported that access routines account for one-quarter to one-third of the higher MI that OO systems have, compared with procedural systems. The SMM components of analysability and changeability, as well as the SMM components of changeability and testability, could have too high a multicollinearity. A similar result also happened for all pairs of the MI components average Halstead Volume, average extended cyclomatic complexity and average unit size (Oppedijk, F, R, 2008).

Griffith I. et al. compared the Quamoco quality model as a plugin to the SonarQube and SQALE. A comparison is reported from 18 open-source and two commercial projects. The quality was ranked in the security, reliability and maintainability categories. As a result, as a point of view of Reliability and Maintainability, the assessments are very different. When Quamoco ranks Reliability much higher than SQALE, the Maintainability result in the opposite value. In the part of Reliability, SQALE is significantly less forgiving because it only focuses on the number of vulnerabilities found, whereas Quamoco takes into consideration other factors aggregated at a higher level of abstraction. In the Maintainability case, SQALE draws a technical debt index to a letter rank, whereas Quamoco aggregates many metrics into quality aspects (Griffith, I, Izurieta, C, and Huvaere, C., 2017).

5. Analysis Result

The usage platform was then extracted from the database to the .csv format to be analyzed. Before conducting data analysis, data is examined using statistical code in python.

We calculated the source code from the projects by the technical debt identification services in the microservice platform. The usage platform was extracted from the database to the .csv format to be analyzed. Before conducting data analysis, data is examined using statistical code in python.

RQ1: How is the correlation of the metric result among the TD identification methods?

The hypotheses regarding this research question are described below.

Hypothesis 1.

MI & SIG Maintainability, MI & Sqale, and SIG Maintainability & Sqale have a strong correlation.

Hypothesis 2.

MI & SIG Maintainability, MI & Sqale, and SIG Maintainability & Sqale have a positive correlation.

To analyze the result from TD identification methods on a set large of open source projects, we used non-statistical parametric, named Kendall 'tau' (τ) method correlation. Motivation to use this method because the data is not normally distributed and has a monotonic function relationship which is one that either never increases or never decreases as its independent variable increases. Then, comparing to other correlation methods for a monotonic function named Spearman correlation, Kendall Correlation is more robust and efficient (Croux, C and Dehon, C, 2008). In addition, some studies report Kendall's τ as the preferred measure when the sample size is large, and there is impulsive noise in the data. The results are based on unbiased estimations of Spearman's ρ and Kendall's τ correlation coefficients (Mamun, M, A, A, Berger, C, and Hansson, J, 2019).

Kendall's τ has three different types that are τ -A, τ -B, and τ -C. Both τ-A and τ -B is suitable for square-shaped data, meaning data with the same variables on both rows and columns. τ-C is used for rectangular-shaped data tables with different sized rows and columns. An essential difference between τ-A and τ-B is that τ-B can perform tied ranks although having the exact characteristics of τ -A. Since our result data has tied ranks, we selected τ-B for measurement. Correlation coefficients of τ-B are divided into a set of parameters regarding their level of strength and significance level. The collection of parameters is represented in Table 20.

Table 20. Set of parameter τ-B

Statistical significance	Negative τ_b value	Positive τ_b value	Label
$\alpha = 0.05$	$-1.0 \leqslant \tau_b \leqslant -0.9$	$0.9 \leqslant \tau_b \leqslant 1.0$	Very strong τ_b
	$-0.9 < \tau_b \leqslant -0.7$	$0.7 \leqslant \tau_b < 0.9$	Strong τ_b
	$-0.7 < \tau_b \leqslant -0.4$	$0.4 \leqslant \tau_b < 0.7$	Moderate τ_b
	$-0.4 < \tau_b \leqslant -0.0$	$0 \leqslant \tau_b < 0.4$	Weak τ_b

Source: (Mamun, M, A, A, Berger, C, and Hansson, J, 2019)

We compared the correlation from the metric result in each TD identification method. The metric result includes of AvgMaintainability for MI, AvgMaintainabilityRanking for SIG Maintainability, and AllIndex for Sqale.

The result represented in Fig. 10 shows SIG Maintainability & Sqale, MI & SIG Maintainability, Sqale & MI have a weak correlation, which stands at -0.26, 0.22, and 0.21. Therefore, SIG Maintainability & Sqale is the highest correlation compared to other relationship between two technical debt identification methods.

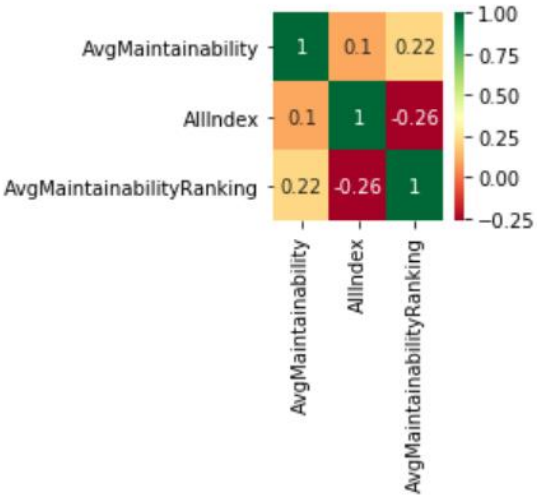


Fig. 10. Result of the τ-B method for each final result among the different metric results of TD identification methods.

We also see the correlation between the two technical debt identification methods through the scatter plots. Fig. 11 shows no correlation between the metric results in MI & SIG Maintainability, while Fig.12 describes that there is also no correlation between metric results in Sqale and SIG Maintainability. Lastly, MI and Sqale also have no correlation, which is shown in Fig. 13.

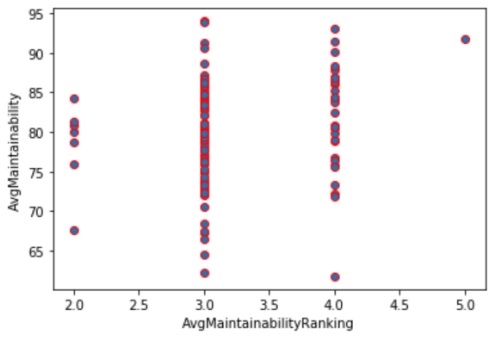


Fig. 11. Scatter plot comparing AvgMaintainability and AvgMaintainabilityRanking

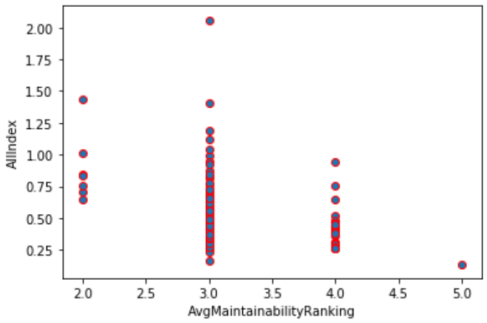


Fig. 12. Scatter plot comparing AllIndex and AvgMaintainabilityRanking

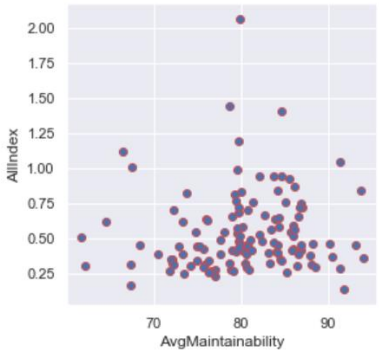


Fig. 13. Scatter plot comparing AllIndex and AvgMaintainability

RQ2: What is the correlation between different code metrics and the metric results from the technical debt identification methods implemented?

To see the correlation between different code metrics and the metric results from each TD identification method implemented, we also use non-statistical parametric Kendall τ method correlation. The motivation to use this method is also the same as RQ1. Meanwhile, we use a boxplot for representation to see the correlation and distribution value between different code metrics and each result in every technical debt identification method. There are some hypotheses regarding the research question.

Hypothesis 1

The increasing value of the metrics of TotalOp, TotalOpr, TotalUnqOp, TotalUnqOpr, TotalCom, TotalLOC, AvgLOC, AvgCC, TotalCC, UnitSize, NumberParameters, Duplication, CBO, DIT, and WMC will decrease the rate of AvgMaintainabilityRate.

Hypothesis 2.

The increasing value of the metrics of TotalOp, TotalOpr, TotalUnqOp, TotalUnqOpr, Total-Com,

TotalLOC, AvgLOC, AvgCC, TotalCC, UnitSize, NumberParameters, Duplication, CBO, DIT, and WMC will decrease the ranking AvgMaintainabilityRanking.

Hypothesis 3.

The increasing value of the metrics of TotalOp, TotalOpr, TotalUnqOp, TotalUnqOpr, Total-Com, TotalLOC, AvgLOC, AvgCC, TotalCC, UnitSize, NumberParameters, Duplication, CBO, DIT, and WMC will decrease the rating of SqaleRating.

1. The Correlation result from Maintainability Index (MI)

The metric result from the MI is AvgMaintainability. Based on the result in Fig. 14, we can see that RatioComment, AvgLOC and AvgCC have a moderate correlation with the AvgMaintai-nabilityRate, which stands at 0.45 0.5, and 0.35. Then, the remaining metrics rank in a weak correlation with AvgMaintainability.

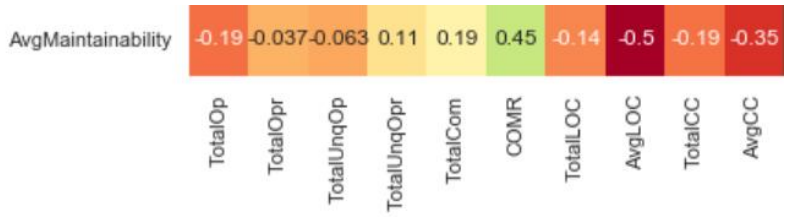


Fig. 14. Result of τ-B method for MI

As the boxplot is represented in Fig. 15, the code metric of TotalOp, TotalOpr, TotalLOC, AvgLOC, TotalCC, AvgCC ranks at a lower rate AvgMaintainabilityRate with a higher value, then the rate increases gradually as the amount of the metrics declines. As a result, we can conclude that the decline value of these code metrics will affect the increase rate of Avg-MaintainabilityRate. Conversely, both rate of AvgMaintainabilityRate in TotalCom and COMR increase as the value of these code metrics rises. Therefore, the increment value of these code metrics increases the rate of AvgMaintainabilityRate.

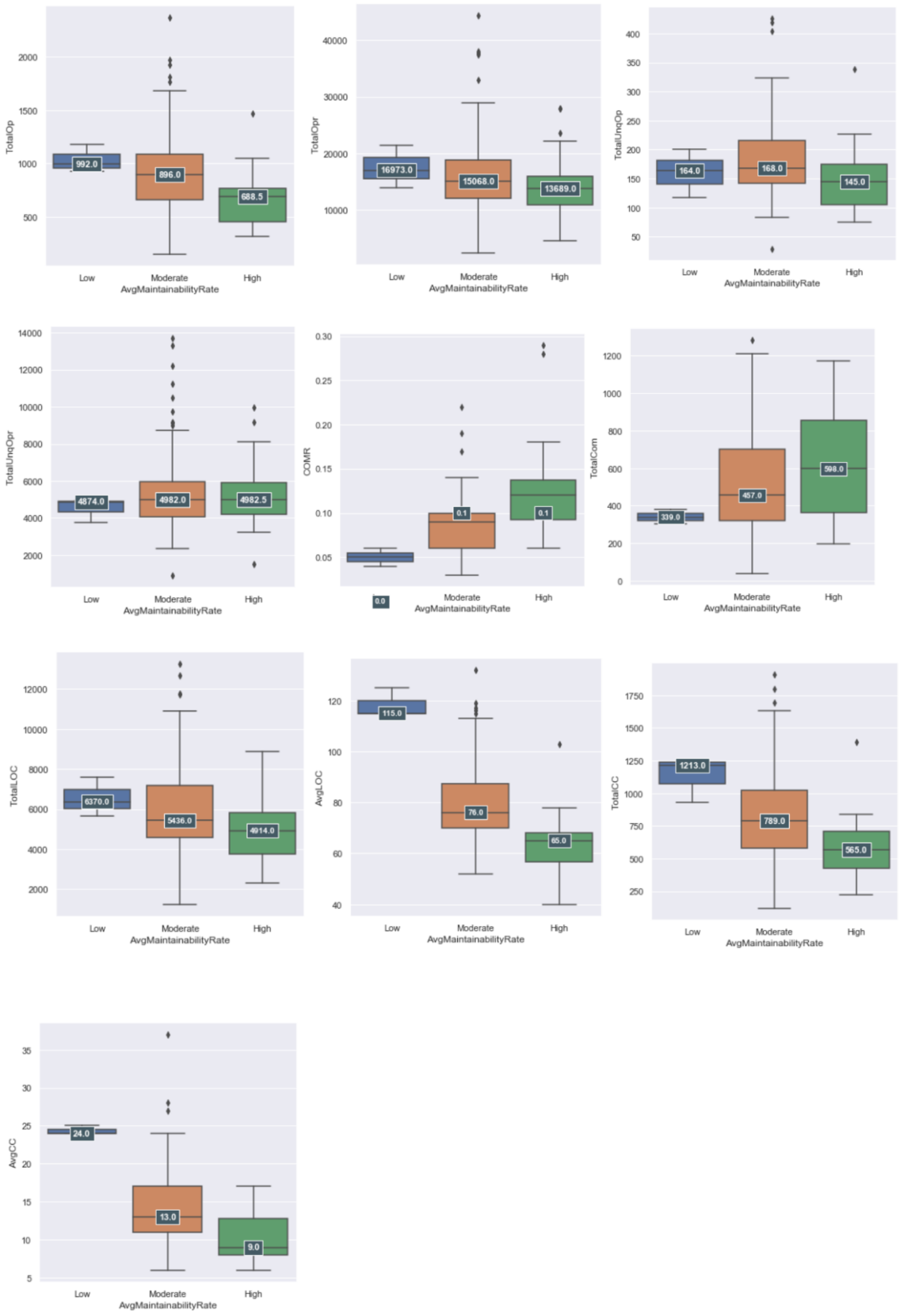


Fig. 15. Boxplot for the relationship between AvgMaintainabilityRate and each code metric in MI

2. The Correlation result from SIG Maintainability Service

Fig. 16 presents that CC, UnitSize and Duplication has a moderate correlation with Avg-MaintainabilityRanking, which stands at 0.39, 0.35, and 0.37 . Meanwhile, the remaining metrics rank at a weak correlation.

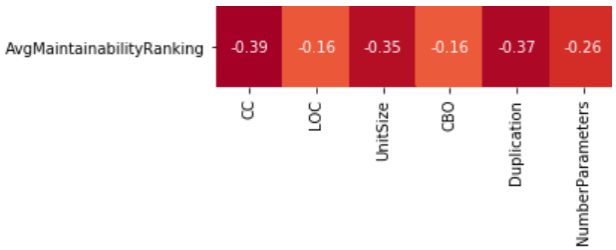
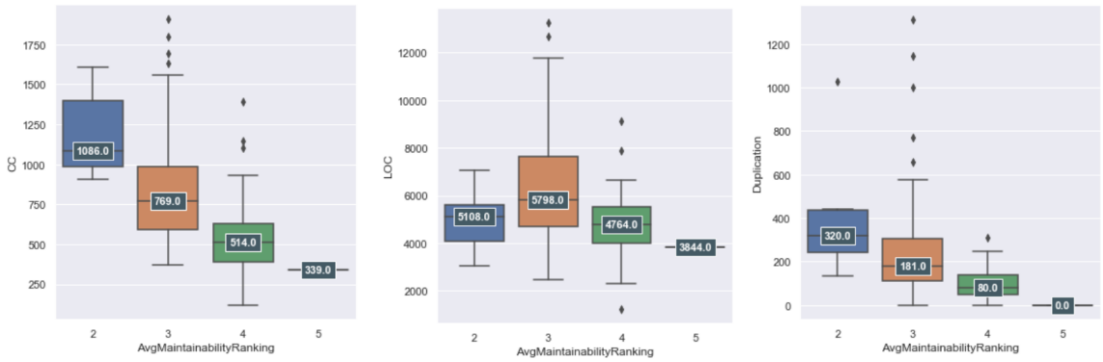


Fig. 16. Result of τ-B method for SIG Maintainability

AvgMaintainabilityRanking consisted of five rankings. In which, the lowest ranking is the lowest maintainability. However, based on the platform's usage, all projects only rank in the four orders, including 2,3,4 and 5.

As shown in Fig. 17, the lowest ranking (2) of AvgMaintainabilityRanking has the highest point in the code metric CC, Duplication, UnitSize, and NumberParameters compared to other rankings. We can state that the decrease figure of these code metrics influences the increment value of AvgMaintainabilityRanking.



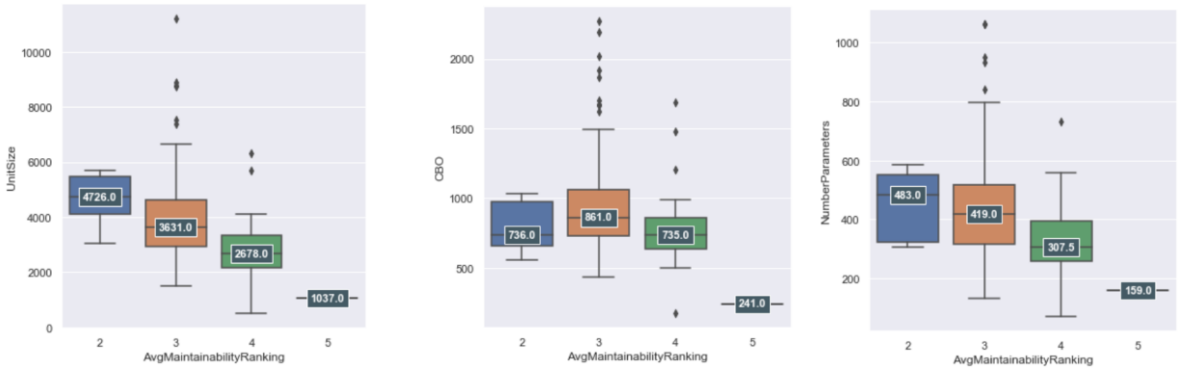


Fig. 17. Boxplot for the relationship between AvgMaintainabilityRanking and each code Metric in SIG Maintainability.

3. The Correlation result from Sqale Service

The metric result from the Sqale service is AllIndex. AllIndex is calculated from total remediation cost divided by total working hours. Fig. 18 describes the correlation between Allindex and all code metrics utilized in Sqale model.

As we can see in Fig. 18, All code metrics have a weak correlation with All Index, except Duplication, which has a moderate correlation with AllIndex, which stands at 0.63.

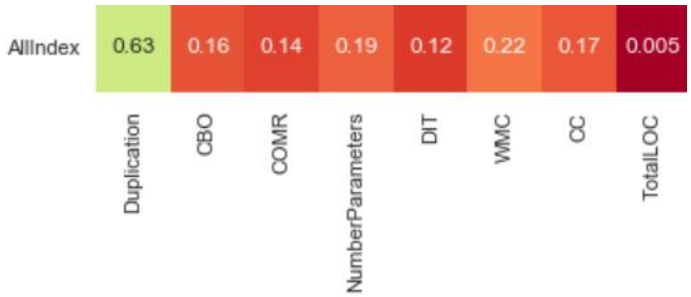


Fig. 18. Result of τ-B method for Sqale

As Fig. 19 shown, all value in all code metrics except COMR touch the lowest point in the lowest rating of SqaleRating. Then, all value increase gradually as rating of SqaleRating decreases. We can conclude that the increment value of code metric CC, NumberParameters, CBO, WMC, Duplication, and DIT will decrease the rating of SqaleRating.

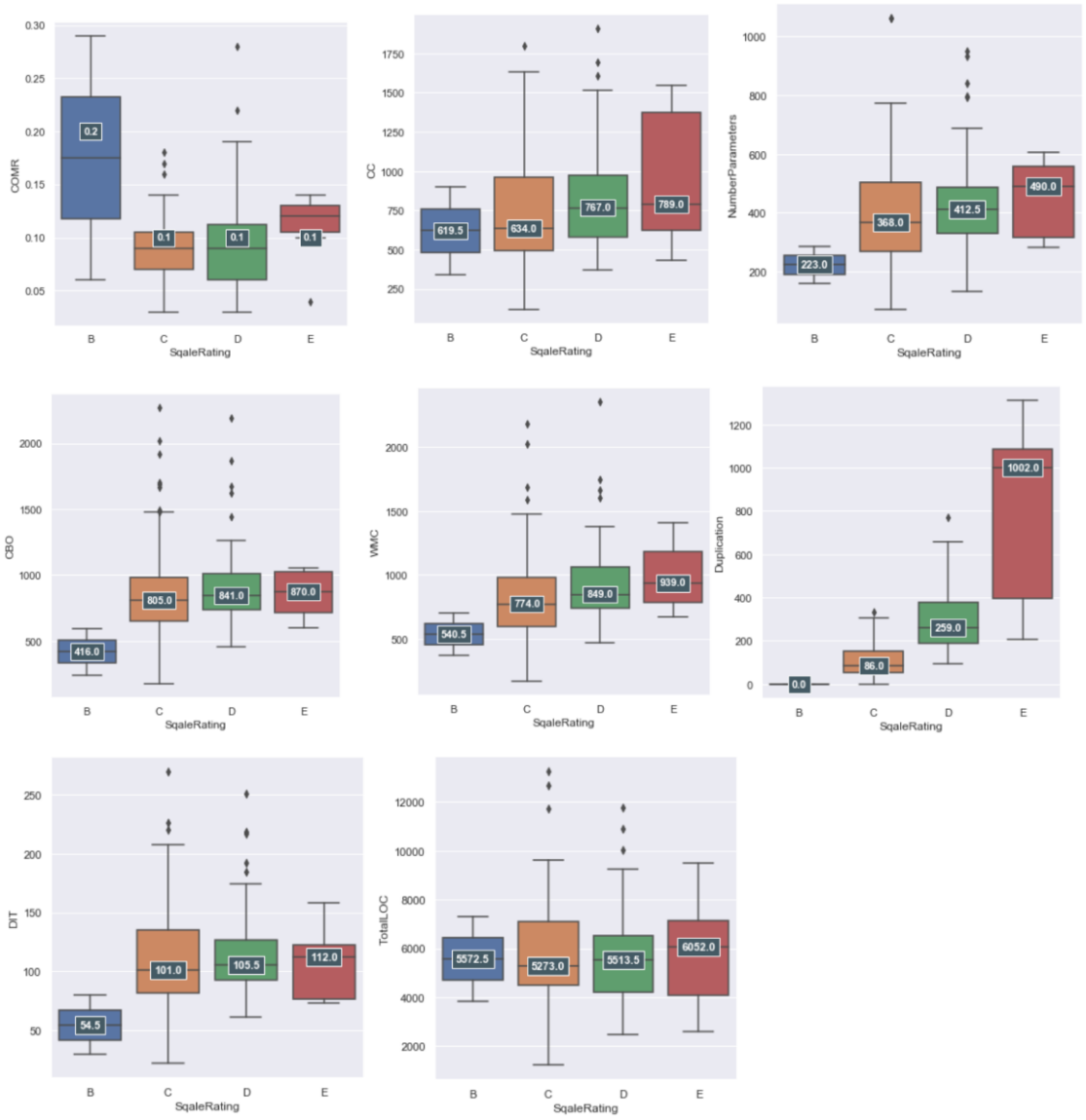


Fig. 19. Boxplot for the relationship between SqaleRating and each code Metric in Sqale.

4. The correlation result in all technical identification method

Fig. 20 describes the correlation result between all results in each technical identification method and all code metric. As you can see, COMR, AvgLOC, AvgCC have a moderate correlation with the AvgMaintainability, which has figures 0.45, 0.5, and 0.35. However, this metric has a weak correlation with the AverageMaintainabilityRanking and AllIndex. In AverageMaintanaibilityRanking, the correlation for these metrics is 0.095, 0.15, and 0.31. Meanwhile, the correlation for these metrics in AllIndex is 0.14, 0.085, and 0.07. Another fact, while UnitSize has a moderate correlation with AvgMaintainabilityRanking, which stands at 0.35, this metric is only in a weak correlation with AvgMaintainability and All index which ranks at 0.14 and 0.16. The interesting thing is although AvgMaintainability owns TotalOp, this metric is

only in a weak correlation with AvgMaintainability as well as in Allindex. However, this metric in a moderate correlation with AvgMaintainabilityRanking, which stands at 0.39. CC is a code metric utilized in each technical debt identification method that only has a moderate correlation with AvgMaintainabilityRanking, which has a value at 0.39. Lastly, although the duplication metric has a moderate correlation with AllIndex and AvgMaintainabilityRanking, which ranks at 0.63 and 0.37, this code metric has a weak correlation with AvgMaintainability, which stands at 0.049.

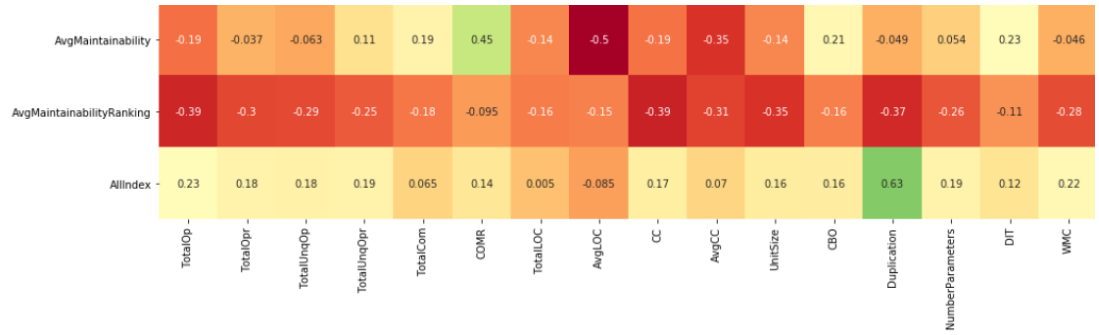


Fig 20. Result of τ-B method for All Technical Identification Method

Furthermore, each boxplot represents below will describe the relationship between each code metric utilized by each technical debt identification method with the final value result of each technical debt identification method. If the code metrics used among the technical debt identification method is the same, it will be eliminated.

Fig. 21 represents the relationship of each code metric in the SIG Maintainability with the AvgMaintainabilityRate. Meanwhile, in the boxplot result from Fig. 21, the code metric of Duplication, UnitSize, and NumberParameters rank in the highest point when the AvgMaintainabilityRate is low. Then, it will decrease gradually as an incline of AvgMaintainabilityRate. Conversely, the number of CBO touches in the lowest point when the AvgMaintainabilityRate is low, increasing progressively as an increment of AvgMaintainabililityRate. We can conclude that the decrease of code metric of Duplication, UnitSize, and NumberParameters will increase the rate of AvgMaintainabilityRate.

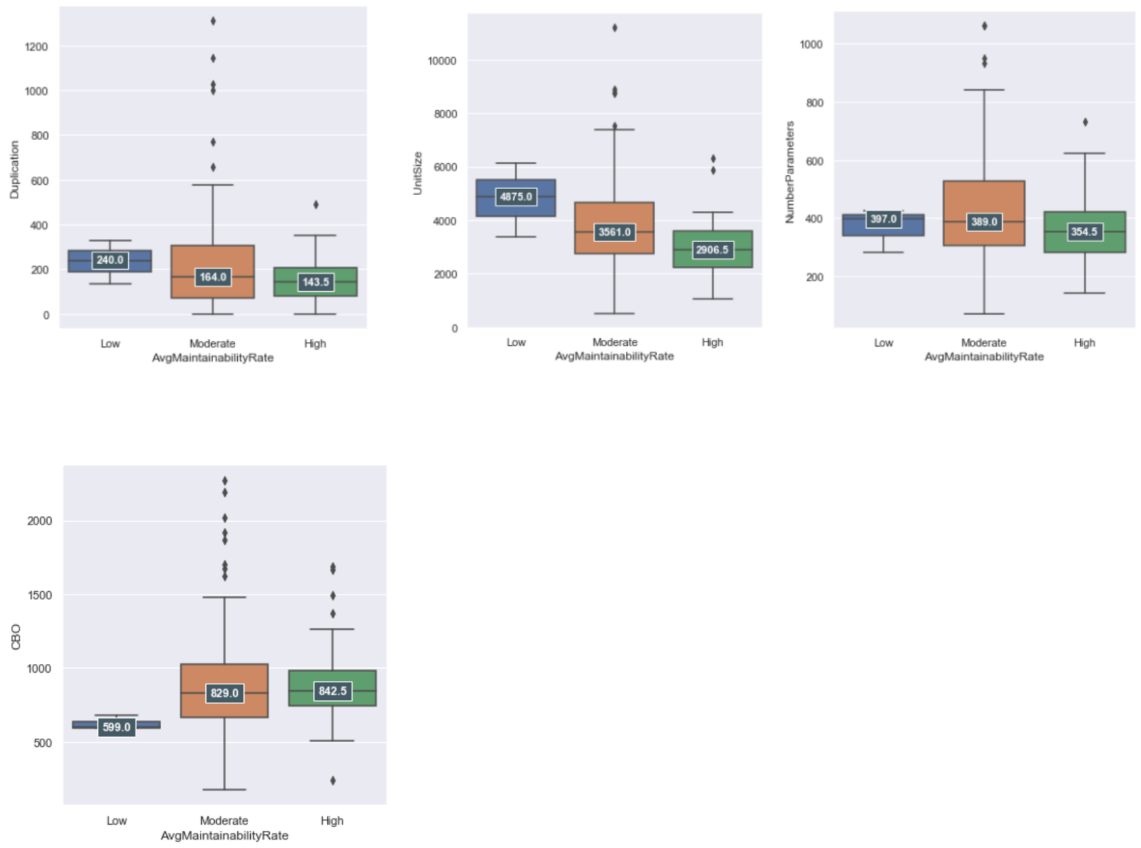


Fig. 21. Boxplot for the relationship between AvgMaintainabilityRate and each code metric in SIG Maintainability

Fig. 22 represents the relationship AvgMaintainabilityRate and each code metric in Sqale. The boxplot shows that only WMC metric influences the rate of AvgMaintai-nabilityRate, which the number of WMC touches the highest point when it ranks in the lowest rate of AvgMaintainabilityRate, and it will decrease gradually until the highest rate.

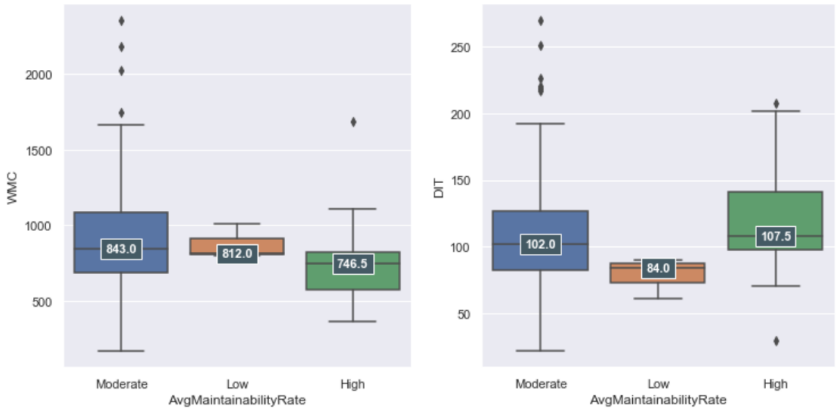


Fig. 22. Boxplot for relationship between AvgMaintainabilityRate and each code metric in Sqale

Moreover, the boxplot in Fig. 23 shows the relationship between AvgMaintainabilityRanking and each code metric in MI. The lowest ranking of AvgMaintainabilityRanking (2) has the

highest number in each code metric in terms of code metric TotalOp, TotalOpr, TotalUnqOp, and TotalUnqOpr compared to other rankings of AvgMaintainabilityRanking. Then these code metrics will decrease gradually as an increase of the AvgMaintainabilityRanking.

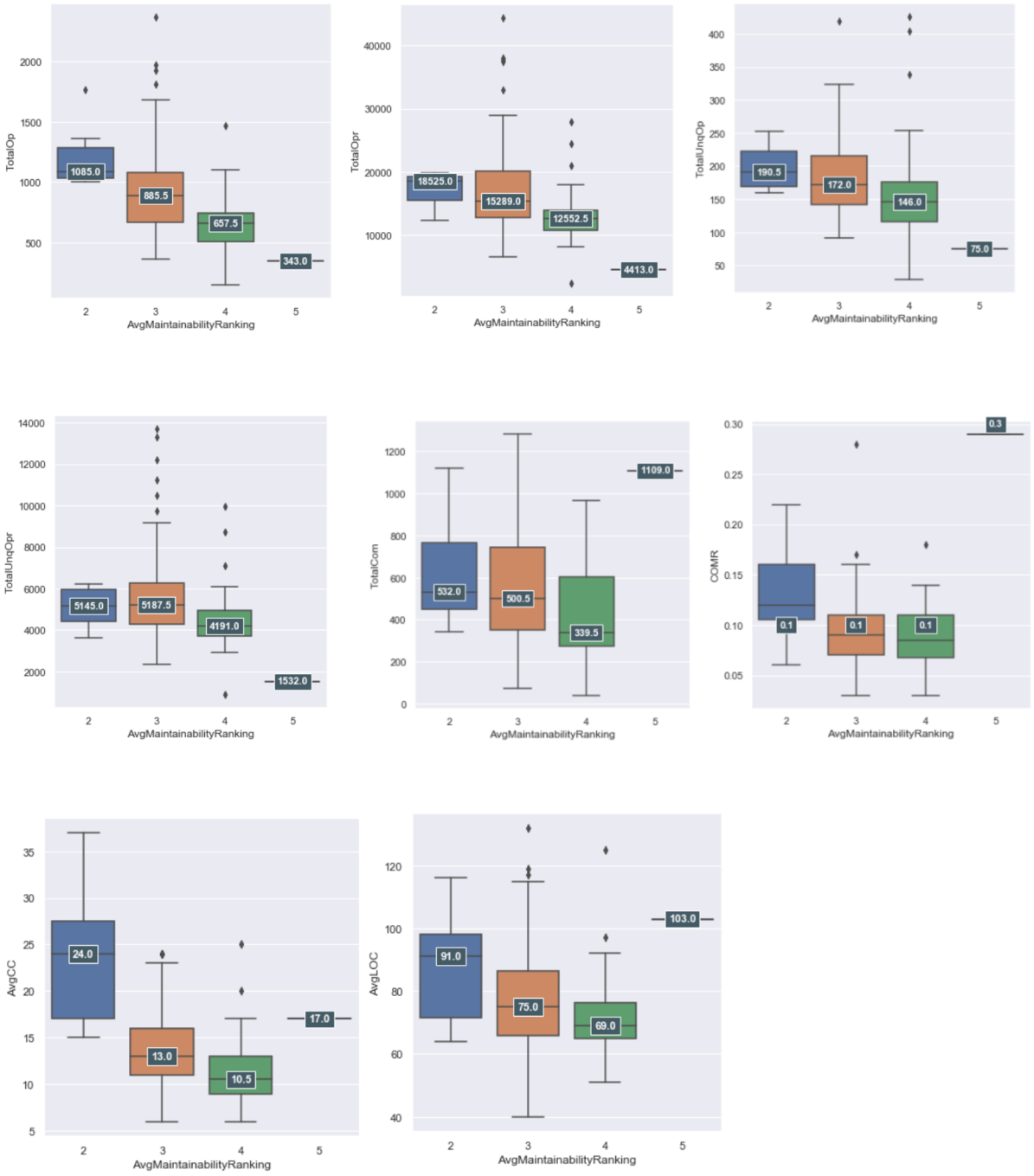


Fig. 23. Boxplot for AvgMaintainabilityRanking and each code metric in Maintainability Index

Fig. 24 shows the relationship between AvgMaintainabilityRanking and code metric in the Sqale method. WMC touches the highest point when AvgMaintainabilityRanking is lowest and will decrease gradually as the increase of AvgMaintainabilityRanking.

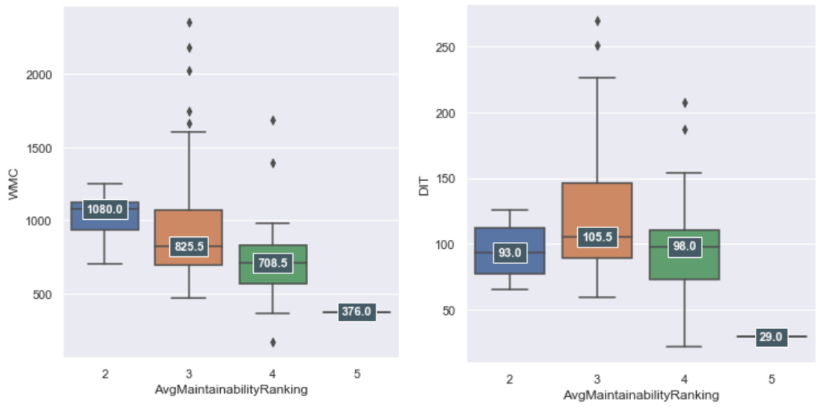


Fig. 24. Boxplot for relationship between Average Maintainability Ranking and Each Code Metric in Sqale

Fig. 25 describes the correlation between each code metric in MI and the rate in SqaleRating. The boxplot presents that the amount of code metric of TotalOp, TotalOpr, TotalUnqOp, TotalUnqOpr ranks in the lowest point when the SqaleRating is in the highest position. Furthermore, the amount will gradually increase as the decreasing of SqaleRating.

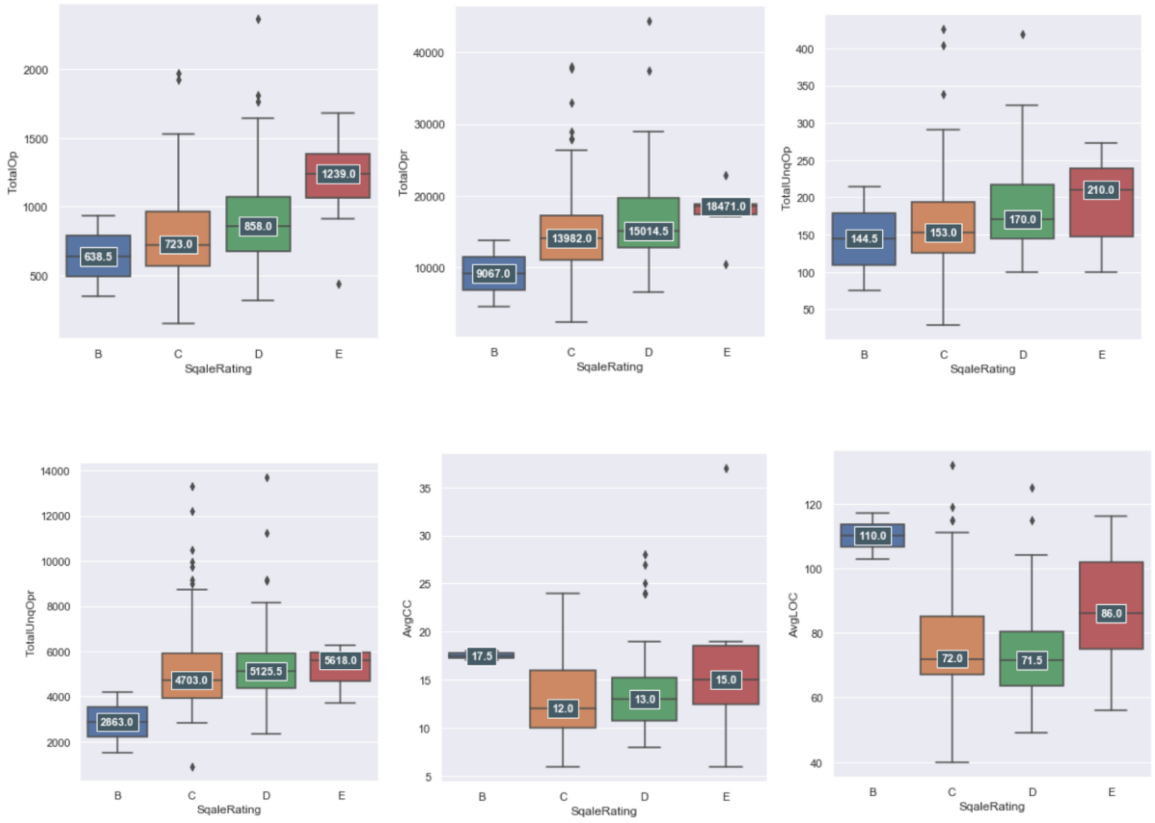


Fig. 25. Boxplot for relationship between Sqale Rating and Each Code Metric in MI

Lastly, Fig. 26 describes the correlation between SqaleRating and code metric in SIG Maintainability. We only display the code metric of UnitSize because only the code metric of UnitSize

is not utilized by the Sqale method among all metrics SIGMaintainability used. As the boxplot presented, the incline of the number UnitSize touches in the lowest point when it ranks at the lowest rate of SqaleRating, and the number will increase as the decrease rate of SqaleRating.

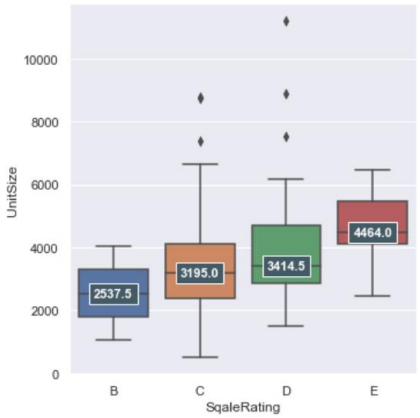


Fig. 26. Boxplot for relationship between SqaleRating and Each Code Metric in SIG Maintainability

RQ3: How is the microservice approach scalable for implementation?

We state some hypotheses regarding this research question which consisted of:

Hypothesis 1: The number of directory projects will decrease the performance of Microservice Platform.

Hypothesis 2: The different amount of directory projects will influence the performance of Microservice Platform.

Hypothesis 3: The frequency of running the microservice platform will influence the performance of the Microservice Platform.

We see the scalability of microservice through the time of query executed. We tested the microservice platform to conduct the calculation of technical debt in some scenario tests. Firstly, we tried the microservice platform to some projects simultaneously in 5, 10, 20, 50, 80, 100, and 124 directories. The test was conducted in Postman by running the API URL "/measurements/calculate" in the microservice platform.

This API has a function to run the calculation in all technical debt identification methods. From the experiment, we gain the result as follows.

Table. 20. Time for Query Executed of the microservice platform based on the amount of directory

Amount of directory	Time for Query Executed
5 directories	1 m 40.38 s

10 directories	4 m 34.26 s
20 directories	5 m 57.54 s
50 directories	10 m 44.46 s
80 directories	18 m 39.30 s
100 directories	23 m 28.74 s
125 directories	34 m 20.70 s

We can conclude based on Table 20. that when the number directory is rising, the time of query executed will increase. The time for a query executed started in 1m 40.38 s for calculating 5 directories and continuously improving until 34m 20.70 s for 125 directories. Therefore, based on Table 20, the average time for several different amounts of the directory is as follows.

Table. 21. Time for Query Executed of the microservice platform based on the amount of directory

Difference amount of directory	Time for Query Excecuted
5 directories	2m 53.38 s
10 directories	1m 23.38 s
20 directories	4m 49.44 s
25 directories	10 m 51.96 s
30 directories	6m 20.88 s

Table 21 shows that the time for a query executed is varied, not depending on the different amount of directory.

Secondly, we run the microservice platform in a collection in Postman with an iteration. We set the iteration until seven times. This test aims to see the performance of the microservice if we run the microservice many times. The amount of time executed in the iteration is presented in the Table. 22.

Table. 22. Time for Query Executed of the microservice platform based on the amount of directory

Number of Iteration	Time for Query Executed
1	1675054 ms
2	34633143 ms
3	6984211 ms
4	788496 ms
5	9868730 ms

6	21293434 ms
7	49124488 ms

The load test results in Table 23 show no correlation between the frequency of running micro-service platform and time for a query executed because the time for a query executed is variated and not increasing continuously as the increment of the number of iteration.

6. Conclusion

This research aims to develop a microservice-based platform that implemented some technical identification methods and provides the analysis result from the data collected from the platform result. The microservice platform implemented the calculation TD by three TD identification methods consisted of MI, SIG Maintainability and Sqale and calculated 124 java directory projects from Masaryk University.

Kendal τ method for analyzing the data usage platform shows a weak correlation between the technical debt identification method of MI & SIG Maintainability, SIG Maintainability & Sqale, and MI & Sqale. The scatter plot used also shows no correlation between these three technical debt identification methods.

Regarding the correlation between all technical debt identification methods and each code metric utilized, the increasing number of code metrics of TotalOp, TotalOpr, CC, Duplication, UnitSize, and NumberParameters, WMC will decrease AvgMaintainabilityRate, AvgMaintainabilityRanking, and SqaleRating.

We measure the microservice scalability by running a load test for the main API implemented in Postman. The load test results show the increment amount of directory projects will decline the performance of the microservice platform. However, the increase of different number of project and the frequency of running microservice platform are not leveraging the performance of microservice platform.

Future works can focus on implementing other TD identification methods in the microservice platform to gain more interpretive knowledge regarding the metric that influences specialized debt increment. Then, some metrics found in this study that have influenced the rising of TD can be used as a reference to arrange a new formula for calculating the TD.

REFERENCE

Avgeriou, P *et al.* (2016) 'Technical Debt: Broadening Perspectives Report on the Seventh Workshop on Managing Technical Debt (MTD 2015)', *ACM SIGSOFT Software Engineering Notes*, 41(2), pp. 38–41. doi: 10.1145/2894784.2894800.

Avgeriou, P *et al.* (2020) 'An Overview and Comparison of Technical Debt Measurement Tools', *IEEE Software*. Available at: https://ieeexplore.ieee.org/document/9200792.

Bijlsma, D. *et al.* (2012) 'Faster issue resolution with higher technical quality of software', *Software Quality Journal*, 20(2). doi: 10.1007/s11219-011-9140-0.

Brown, N *et al.* (2010) 'Managing Technical Debt in Software-Reliant Systems'. Available at: https://dl.acm.org/doi/10.1145/1882362.1882373.

Curtis, B, Sappidi, J, and Szynkarski, A (2012) 'Estimating the Principal of an Application's Technical Debt'. doi: 10.1109/MS.2012.156.

Fucci, G *et al.* (2021) 'Waiting around or job half-done? Sentiment in self-admitted technical debt', *Eindhoven University of Technology*. Available at: https://www.win.tue.nl/~aserebre/MSR2021.pdf.

Griffith, I, Izurieta, C, and Huvaere, C. (2017) 'An Industry Perspective to Comparing the SQALE and Quamoco Software Quality Models', *ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. doi: 10.1109/ESEM.2017.42.

Halepmollası, R. (2020) 'A Composed Technical Debt Identification Methodology to Predict Software Vulnerabilities', *IEEE/ACM 42nd International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*. doi: 10.1145/3377812.3381396.

Hazelcast (no date) 'Six Advantages of Microservices'. Available at: https://media.bitpipe.com/io_15x/io_151716/item_2149538/6_Advantages_of_Microservices.pdf.

Hegeman, J, H. (2011) 'On the Quality of Quality Models', *University of Twente*. Available at: http://essay.utwente.nl/61040/1/MSc_J_Hegeman.pdf.

Heitlager, I, Kuipers, T., and Visser, J (2007) 'A Practical Model for Measuring Maintainability', *6th International Conference on the Quality of Information and Communications Technology (QUATIC).* Available at: https://ieeexplore.ieee.org/abstract/document/4335232.

Irudayaraj, Prathap and P, Saravanan (2019) 'Adoption Advantages Of Micro-Service Architecture In Software Industries'. Available at: https://www.researchgate.net/publication/339749608_Adoption_Advantages_Of_Micro-Service_Architecture_In_Software_Industries.

Kruchten, P, Nord, R, and Ozkaya, I (2012) 'Technical Debt: From Metaphor to Theory and Practice', *IEEE Software*. Available at: https://ieeexplore.ieee.org/document/6336722.

Letouzey, J (2012) 'The SQALE method for evaluating Technical Debt'. doi: 10.1109/MTD.2012.6225997.

Letouzey, J (2016) ‘The SQALE Method for Managing Technical Debt Definition Document’. Available at: http://www.sqale.org/wp-content/uploads/2016/08/SQALE-Method-EN-V1-1.pdf.

Liu, J *et al.* (2020) ‘Is Using Deep Learning Frameworks Free? Characterizing Technical Debt in Deep Learning Frameworks’. doi: 10.1145/3377815.3381377.

M.A.M.Najm, N (2014) ‘Measuring Maintainability Index of a Software Depending on Line of Code Only’, *IOSR Journal of Computer Engineering (IOSR-JCE).*

Mamun, M, A, A, Berger, C, and Hansson, J (2019) ‘Effects of Measurements on Correlations of Software. Empirical Software Engineering’, *Empirical Software Engineering*, 24(8). Available at: https://www.researchgate.net/publication/332072045_Effects_of_Measurements_on_Correlations_of_Software_Code_Metrics.

McConnell, S, C (2013) ‘Managing Technical Debt’, *ICSE Conferences*.

Nugroho, A, Visser, J, and Kuipers, T (2011) ‘An Empirical Model of Technical Debt and Interest’. doi: 10.1145/1985362.1985364.

Oppedijk, F, R (2008) ‘Comparison of the SIG Maintainability Model and the Maintainability Index’, *Ph.D. Dissertation. Master’s thesis, University of Amsterdam*.

Peffers, K, Rothenberger, M, A, and Tuunanen, T (2008) ‘A Design Science Research Methodology for Information Systems Research Journal of Management Information Systems’. doi: 10.1.1.535.7773.

Rios, N *et al.* (2018) ‘The Most Common Causes and Effects of Technical Debt: First Results from a Global Family of Industrial Surveys’, *ESEM*. Available at: https://www.researchgate.net/publication/328083791_The_most_common_causes_and_effects_of_technical_debt_first_results_from_a_global_family_of_industrial_surveys.

‘SIG/TÜViT Evaluation Criteria Trusted Product Maintainability: Guidance for producers Version 11.0’ (2019) *Software Improvement Group*. Available at: https://www.softwareimprovementgroup.com/wp-content/uploads/2019/11/20190919-SIG-TUViT-Evaluation-Criteria-Trusted-Product-Maintainability-Guidance-for-producers.pdf.

Strečanský, P, Chren, S, and Rossi, B (2020) ‘Comparing Maintainability Index, SIG Method, and SQALE for Technical Debt Identification’, *Hindawi Scientific Programming*. doi: 10.1145/3234152.3234193.

Theodoropoulos, T, Hofberg, M, and Kern, D (2011) ‘Technical Debt from the Stakeholder Perspective’, *MTD ’11: Proceedings of the 2nd Workshop on Managing Technical Debt*. doi: 10.1145/1985362.1985373.

Thomas, P, Escalona, M, J, and Mejías, M (2013) ‘Open source tools for measuring the Internal Quality of Java software products’, *Computer Standards & Interfaces*, 36(1), pp. 244–255.

Viggiato Markos *et al.* (2018) ‘Microservices in Practice: A Survey Study’. Available at: https://www.researchgate.net/publication/326610698_Microservices_in_Practice_A_Survey_Study.

Yuepo, G and Carolyn S (2011) ‘A portfolio approach to technical debt management’, *MTD ’11: Proceedings of the 2nd Workshop on Managing Technical Debt*. doi: 10.1145/1985362.1985370.

Appendix

Table 1. Acronyms of Each Code Metric

Acronym	Explanation
Avg	Average
CC	Cyclomatic complexity
LOC	Line of code
TotalOp	Total operator
TotalOpr	Total operand
TotalUnqOp	Total unique operator
TotalUnqOpr	Total unique operand
TotalCom	Total comment
AvgCom	Average comment, Total comment in a directory divided by the total number of file
AvgLOC	Average LOC, Total LOC in a directory divided by the total number of file which contains LOC
AvgCC	Average CC, Total CC divided by the total number of file which contains CC
AvgCCRate	Rate of AvgCC
TotalProgVol	Total program volume (TotalProgLgth)*(log(TotalProgVc)/Math.log(2));
AvgProgVol	Average program volume, Total program volume in a directory divided by the total number of file which contains program volume
TotalProgVol	Total program volume
TotalProgVc	Total program vocabulary
TotalProgLgth	Total program length
TotaEstProgLgth	Total of calculation program vocabulary TotalUnqOp*(Math.log(TotalUnqOp) / log(2)) + TotalInqOpr*(Math.log(TotalUnqOpr) /log(2));
TotalTimeReqToProg	Total time request to progress
TotalDiff	Total difficulty
TotalEff	Total effort
TotalNumOfDelBug	Total number of delivered bugs
MI	Maintainability Index
AvgMaintainability	Average maintainability, Calculation from AvgCC, AvgProgVol, AvgLOC, and AvgCom
AvgMaintainabilityRate	Rate of average maintainability
Volume	Same with LOC, Definition for SIG Maintainbility Method
UnitSize	Total line of code in every unit
CBO	Coupling Between Object Classes, Total coupling between module, a count of the number of other classes to which it (a class) is coupled.

	(Qureshi, 2012).
Duplication	Total duplication in a unit or between the class in a directory
NumberParameters	Number parameters, Total number of parameters from every unit
VolumeRanking	Volume ranking, Ranking of volume in a directory
CCRanking	CC ranking, Ranking of cc in a directory
UnitSizeRanking	Ranking of unit size in a directory
CouplingRanking	Ranking of CBO in a directory
DuplicationRanking	Ranking of duplication in a directory
UnitInterfacingRanking	Ranking of unit interfacing in a directory
AvgAnalisabilityRanking	Average analisability ranking
AvgChangeabilityRanking	Average changeabilityranking
AvgStabilityRanking	Average stability ranking
AvgTestabilityRanking	Average testability ranking
DIT	Deep of inheritance, This count is the maximum length / depth from the node to the root of the tree. (Qureshi, 2012).
WMC	Weighted Methods for Class
COMR	CommentRatio, Total Comment dividend by Total LOC in a file
TotalFileCC	Total file in a directory that has CC >= 6
TotalFileCBO	Total file in a directory that has CBO >=7
TotalFileCOMR	Total file in a directory that has comment ratio (COMR) <= 35% in a file
TotalFileNumParameters	Total file in a directory that has Number of parameters in a module call <6
AllIndex	Total from index in each characteristic in Sqale Method
SqaleRating	The rating in the sqale method which is obtained from AllIndex dividend by working hours.

Table 2. Data resulted from MIService

Directory Name	TotalOp	TotalOpr	TotalUnqOp	TotalUnqOpr	Total-Com	COMR
01	1023	14969	182	4982	761	0.1
02	901	12843	193	4266	422	0.08
03	693	9520	151	3004	289	0.06
04	1179	21463	164	4874	381	0.06
05	1020	13237	182	4582	978	0.12
06	1765	19847	208	5145	689	0.19
07	1169	17262	204	5166	740	0.1
08	1358	18471	159	4465	532	0.12
09	1159	16012	177	4661	221	0.04

10	968	14021	131	4000	534	0.08
11	768	10995	151	3344	264	0.08
12	591	6457	149	2360	71	0.03
13	925	13919	117	3749	305	0.05
14	778	11219	176	3902	333	0.07
15	514	8121	123	2933	227	0.08
16	597	12786	152	4667	348	0.09
17	532	12482	117	3969	163	0.04
18	992	16973	200	4963	339	0.04
19	934	13721	214	4194	443	0.06
20	1063	20804	178	5884	416	0.06
21	940	15213	224	5499	1211	0.14
22	862	12213	211	3947	519	0.08
23	591	8821	153	2846	457	0.1
24	612	10990	157	3551	283	0.06
25	1143	15083	152	3822	724	0.09
26	520	10808	103	3283	563	0.09
27	343	4413	75	1532	1109	0.29
28	1085	12541	147	3626	407	0.13
29	506	8934	118	3313	446	0.1
30	716	11654	129	4001	326	0.07
31	541	10287	101	3659	318	0.07
32	998	12240	173	4370	344	0.06
33	518	10993	130	4125	221	0.05
34	611	13301	118	4520	512	0.09
35	705	15204	171	5245	538	0.1
36	716	13982	161	5004	177	0.05
37	1527	25442	256	7376	927	0.1
38	1045	15190	213	5042	364	0.12
39	517	10354	146	3988	314	0.07
40	527	11557	115	4196	354	0.07
41	909	18683	210	6260	663	0.11
42	1215	18790	252	6093	840	0.12
43	488	10588	153	4170	694	0.13
44	739	15475	167	6108	768	0.12
45	458	10821	108	3843	96	0.03
46	878	16558	156	5281	891	0.17
47	952	18843	176	6193	713	0.12
48	664	11377	135	4100	348	0.11
49	893	17037	195	5556	617	0.12
50	150	2324	29	909	40	0.03
51	461	11479	99	4176	143	0.03
52	1013	17224	183	5421	478	0.08
53	587	11203	141	3969	257	0.08
54	1683	22782	137	4891	405	0.04
55	1178	20164	217	6482	709	0.12
56	511	10673	83	3316	265	0.07
57	989	15695	215	5516	516	0.09

58	1239	17731	226	5618	346	0.1
59	940	13479	198	4846	603	0.09
60	1171	26291	229	8323	809	0.11
61	613	12989	138	4703	228	0.05
62	818	11813	149	4360	435	0.07
63	1067	18525	228	6231	492	0.09
64	969	22098	206	7902	1106	0.13
65	925	15377	188	5117	274	0.04
66	764	14961	186	5745	594	0.1
67	742	12417	172	4545	793	0.14
68	514	10989	106	3833	257	0.05
69	700	12623	142	4490	255	0.07
70	737	13911	151	5440	503	0.09
71	620	9493	151	3762	283	0.08
72	1034	15068	192	5035	409	0.06
73	878	16110	167	5813	257	0.05
74	760	13677	176	5021	606	0.11
75	1084	17241	234	6012	322	0.06
76	674	13317	146	4407	479	0.09
77	1811	28922	323	9170	787	0.09
78	1081	22603	319	8160	498	0.05
79	1414	16978	273	5846	1027	0.14
80	723	13701	175	5028	415	0.1
81	1230	20326	224	6243	698	0.11
82	664	11258	131	4098	297	0.05
83	645	14375	166	5209	602	0.12
84	723	14012	142	4522	391	0.07
85	664	12343	143	4691	313	0.09
86	361	10125	92	4269	379	0.09
87	1000	19625	168	5841	1121	0.22
88	838	14306	139	4620	331	0.09
89	387	10797	103	3689	744	0.14
90	802	16075	158	5554	1165	0.28
91	317	9224	111	3730	197	0.07
92	957	16123	196	5685	608	0.12
93	855	19585	179	7447	440	0.05
94	435	10340	99	3726	361	0.14
95	590	15365	118	5307	456	0.09
96	725	18040	136	5944	865	0.15
97	896	27954	221	8978	929	0.12
98	763	17939	149	5795	338	0.06
99	663	14707	168	5175	494	0.09
100	671	13982	158	4738	386	0.09
101	407	10947	77	4198	417	0.1
102	766	16929	161	5799	608	0.09
103	382	9950	113	3622	319	0.08
104	524	11671	145	4378	932	0.18
105	1290	23113	226	7158	614	0.11

106	675	12679	125	4184	418	0.11
107	652	13204	133	4580	362	0.09
108	1925	38047	426	13317	1156	0.09
109	1639	37432	253	11227	954	0.09
110	923	27933	145	9155	979	0.11
111	1321	28906	290	9739	637	0.07
112	816	17622	226	7054	390	0.06
113	728	20966	128	7085	832	0.13
114	668	13904	218	5106	455	0.09
115	1100	24447	254	8717	786	0.09
116	2367	44436	419	13704	1283	0.11
117	1382	33011	259	10507	1180	0.1
118	1966	37707	404	12211	844	0.07
119	1073	25010	271	9141	705	0.07
120	709	23598	96	8100	1171	0.16
121	375	8419	102	3246	266	0.12
122	439	13392	96	4944	341	0.08
123	1467	27881	338	9956	964	0.12
124	1076	20105	290	6934	955	0.11

Table 3. Data resulted from MIService (cont')

Directory Name	TotalLOC	AvgLOC	CC	AvgCC	AvgMain-tainability	AvgMain-tainabil-ityRate
01	7827	101	941	17	75.94	Moderate
02	5005	87	978	19	75.01	Moderate
03	4507	83	476	13	75.7	Moderate
04	6370	115	1213	24	64.47	Low
05	8315	88	845	13	82.2	Moderate
06	3579	73	1609	27	84.26	Moderate
07	7782	119	1222	22	70.46	Moderate
08	4556	116	1545	37	67.53	Moderate
09	4933	93	768	13	68.37	Moderate
10	7114	107	837	15	72.17	Moderate
11	3413	94	769	20	72.83	Moderate
12	2458	52	576	12	79.61	Moderate
13	5658	125	929	25	61.72	Low
14	5004	92	924	20	73.33	Moderate
15	2760	72	521	14	80.44	Moderate
16	4059	71	501	9	82.7	Moderate
17	4577	84	358	8	72.17	Moderate
18	7602	115	1259	24	62.2	Low
19	7301	117	900	18	67.31	Moderate
20	7222	90	943	14	72.34	Moderate
21	8556	87	1094	12	84.65	Moderate

22	6560	111	637	14	71.93	Moderate
23	4729	115	597	17	73.51	Moderate
24	4672	97	627	14	71.82	Moderate
25	7690	132	1025	23	67.34	Moderate
26	5973	96	616	12	77.08	Moderate
27	3844	103	339	17	91.84	High
28	3059	92	908	28	75.96	Moderate
29	4339	81	598	17	80.31	Moderate
30	4491	68	492	12	80.57	Moderate
31	4644	80	592	15	77.09	Moderate
32	5815	64	961	16	80.03	Moderate
33	4295	61	580	13	79.81	Moderate
34	5737	91	640	16	76.61	Moderate
35	5498	75	779	17	80.76	Moderate
36	3771	59	697	14	79.04	Moderate
37	9604	99	1279	16	75.04	Moderate
38	2934	59	808	14	87.12	High
39	4792	72	499	10	80.56	Moderate
40	5106	76	459	10	79.32	Moderate
41	6052	86	757	12	79.85	Moderate
42	7047	91	1252	18	78.68	Moderate
43	5462	72	515	9	88.04	High
44	6529	68	754	11	86.91	High
45	3540	72	432	13	72.27	Moderate
46	5231	93	735	14	83.12	Moderate
47	5804	95	1025	17	77.69	Moderate
48	3053	62	674	15	86.12	High
49	5124	73	1003	15	83.4	Moderate
50	1238	65	120	12	76.8	Moderate
51	4576	67	369	9	74.85	Moderate
52	5968	84	976	20	75.23	Moderate
53	3190	61	495	11	84.04	Moderate
54	9507	113	434	6	66.5	Moderate
55	5980	83	1230	17	80.14	Moderate
56	3815	86	351	10	76.35	Moderate
57	5920	71	713	11	82.08	Moderate
58	3638	74	789	15	79.71	Moderate
59	6381	77	1101	16	80.67	Moderate
60	7120	76	1629	22	79.78	Moderate
61	4382	66	458	8	78.73	Moderate
62	5874	80	627	12	78.17	Moderate
63	5428	70	1005	15	80.85	Moderate
64	8540	73	837	9	86.6	High
65	6535	70	523	11	75.66	Moderate
66	5760	64	581	11	85.65	High
67	5841	75	760	15	85.94	High
68	4856	67	399	10	78.9	Moderate
69	3427	56	499	10	84.18	Moderate

70	5566	66	765	11	84.43	Moderate
71	3606	59	490	10	85.98	High
72	7053	83	931	19	73.74	Moderate
73	4972	62	561	8	79.57	Moderate
74	5632	68	621	9	86.45	High
75	5067	61	875	12	80.74	Moderate
76	5090	72	622	17	80.89	Moderate
77	8863	73	1516	15	79.54	Moderate
78	9255	73	1014	10	76.22	Moderate
79	7236	76	1489	19	84.65	Moderate
80	4241	63	545	10	86.08	High
81	6502	69	912	13	83.86	Moderate
82	5461	82	612	16	73.3	Moderate
83	4943	66	723	12	86.63	High
84	5273	82	884	22	74.25	Moderate
85	3607	40	556	9	94	High
86	4095	56	409	8	88.62	High
87	5108	104	1086	24	81.36	Moderate
88	3696	56	789	16	85.11	High
89	5344	66	225	6	91.37	High
90	4161	77	769	13	93.83	High
91	2984	53	380	8	86.85	High
92	5192	71	979	16	83.81	Moderate
93	8503	69	909	10	77.69	Moderate
94	2606	56	482	13	91.34	High
95	4814	63	493	9	84.58	Moderate
96	5792	75	647	15	85.6	High
97	7875	70	838	10	84.72	Moderate
98	5353	66	634	11	79.09	Moderate
99	5480	76	826	15	79.89	Moderate
100	4465	65	450	8	84.21	Moderate
101	4300	68	359	9	85.25	High
102	6598	78	729	14	79.48	Moderate
103	4063	75	393	11	80.69	Moderate
104	5325	78	393	8	90.2	High
105	5436	70	1690	24	79.9	Moderate
106	3837	72	583	13	83.75	Moderate
107	4166	65	513	9	84.35	Moderate
108	13264	72	1800	11	81.22	Moderate
109	10920	78	1159	11	78.97	Moderate
110	8890	68	494	6	85.85	High
111	9363	69	1190	12	79.41	Moderate
112	6600	49	644	8	86.77	High
113	6622	67	614	8	87.88	High
114	4885	62	533	8	86.18	High
115	9143	70	1143	11	82.41	Moderate
116	11789	81	1909	13	79.76	Moderate
117	11741	83	1035	11	79.76	Moderate

118	12680	79	1556	12	76.28	Moderate
119	10032	61	852	8	83.49	Moderate
120	7226	66	574	8	90.56	High
121	2294	52	232	6	93.15	High
122	4086	51	354	9	88.32	High
123	7882	64	1389	11	86.9	High
124	8445	82	1140	15	81.06	Moderate

Table 4. Data resulted from SIG Maintainability Service

Directory Name	Unit-Size	CBO	Duplica-tion	NumberPa-rameters	AvgMaintainabilityRanking
01	3895	794	148	287	3
02	3195	631	77	229	3
03	2399	498	130	205	4
04	6135	676	330	427	3
05	2861	829	366	311	3
06	5697	707	208	483	2
07	4513	766	210	354	3
08	5319	601	433	340	2
09	3874	615	127	319	3
10	3925	549	146	251	3
11	3006	558	72	205	3
12	1506	456	191	131	3
13	3381	589	240	282	4
14	2941	644	102	294	4
15	1921	508	51	175	4
16	2831	784	187	332	3
17	2746	659	89	361	4
18	4875	599	134	397	3
19	4038	591	0	287	3
20	5543	893	450	481	3
21	3928	842	771	503	3
22	3284	546	154	201	3
23	2360	432	48	202	3
24	2441	606	34	259	4
25	4572	553	173	265	3
26	2962	521	69	265	3
27	1037	241	0	159	5
28	3699	555	133	306	2
29	1975	561	114	315	3
30	2386	718	123	323	3
31	2206	669	38	382	3
32	3040	736	442	306	2
33	1914	738	226	305	3
34	3086	787	63	419	3
35	3293	890	154	514	3

36	2991	847	41	527	3
37	5972	1174	304	616	3
38	3332	872	93	393	3
39	2208	659	64	324	3
40	2423	748	373	389	3
41	3952	1050	1313	490	3
42	5618	995	1029	586	2
43	2253	719	78	309	4
44	2913	986	167	393	4
45	2161	654	0	296	3
46	3843	852	104	505	3
47	5115	1008	177	472	3
48	2389	756	184	361	3
49	4097	895	43	584	3
50	511	175	0	72	4
51	2087	667	187	314	3
52	3669	883	164	482	3
53	2465	727	27	368	3
54	6463	751	1145	293	3
55	4734	1081	235	620	3
56	2296	562	21	261	4
57	3469	969	504	475	3
58	4253	870	363	528	3
59	3495	848	64	366	4
60	6607	1429	191	614	3
61	2695	752	51	273	3
62	3004	713	258	309	3
63	4485	1033	279	533	2
64	4255	1489	141	624	3
65	3337	912	79	414	4
66	3008	932	224	390	3
67	2818	831	219	419	3
68	2161	682	48	374	4
69	2564	837	54	390	4
70	3045	943	231	374	3
71	2024	716	64	255	4
72	3546	799	553	423	3
73	3246	1035	155	382	3
74	3550	845	49	403	4
75	3843	1113	77	505	3
76	2949	739	44	386	4
77	6153	1672	336	932	3
78	4731	1442	440	689	3
79	4464	1049	1002	605	3
80	2987	941	146	348	3
81	5040	1058	313	467	3
82	2359	668	294	327	3
83	3190	840	236	486	3

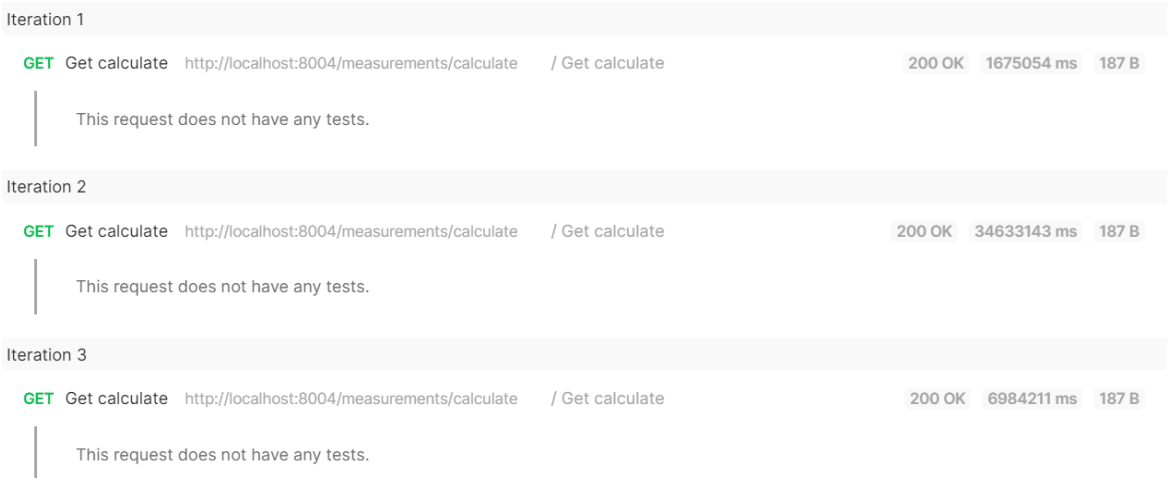
84	3561	735	73	459	3
85	2614	961	8	257	3
86	1738	808	0	295	3
87	4726	956	320	570	2
88	3634	742	199	342	3
89	2220	684	81	218	4
90	3628	936	260	425	3
91	1615	706	155	276	4
92	3868	941	126	487	3
93	3957	1386	174	725	3
94	2457	675	208	283	3
95	2929	923	68	341	3
96	3826	999	492	480	3
97	6537	1480	94	566	3
98	3872	974	10	461	4
99	3677	859	161	488	4
100	2649	860	67	306	4
101	2039	805	0	285	4
102	3598	945	453	425	3
103	2110	587	86	243	4
104	1921	771	168	305	4
105	7545	1063	150	841	3
106	2707	731	308	344	4
107	2901	793	184	406	4
108	8756	2272	249	1062	3
109	8882	1867	576	796	3
110	5881	1668	225	408	3
111	6638	1702	178	772	3
112	3448	1260	353	491	3
113	4091	1203	108	516	4
114	2870	801	163	303	3
115	5705	1475	249	559	4
116	11214	2188	657	951	3
117	7378	1918	334	598	3
118	8784	2017	177	1063	3
119	4694	1620	394	793	3
120	4283	1371	128	509	3
121	1503	505	30	142	4
122	2900	883	86	194	4
123	6302	1690	133	732	4
124	4601	1184	275	543	3

Table 5. Data resulted from Sqale Service

Directory Name	DIT	WMC	SqaleRating
01	109	933	C
02	72	714	C
03	66	519	C
04	90	1009	D
05	123	784	D
06	82	1133	D
07	94	1024	C
08	73	1249	E
09	82	761	C
10	86	700	C
11	65	684	C
12	83	473	D
13	61	812	D
14	74	846	C
15	56	589	C
16	89	601	D
17	71	669	C
18	84	803	C
19	80	705	B
20	105	916	D
21	123	1132	D
22	65	696	C
23	59	546	C
24	68	680	C
25	70	799	C
26	89	638	C
27	29	376	B
28	65	705	D
29	67	548	C
30	80	611	C
31	81	604	C
32	107	816	D
33	102	601	D
34	88	716	C
35	103	922	C
36	101	789	C
37	147	1214	C
38	98	903	D
39	82	599	C
40	94	697	D
41	112	939	E
42	118	1113	E
43	105	600	C
44	154	787	C
45	78	537	C

46	97	816	C
47	103	1091	C
48	80	729	D
49	120	977	C
50	22	172	C
51	92	522	D
52	103	1026	C
53	96	696	C
54	127	672	E
55	127	1160	D
56	66	563	C
57	120	907	D
58	75	890	E
59	116	984	C
60	165	1592	C
61	94	574	C
62	92	703	D
63	126	1056	D
64	178	1109	C
65	128	936	C
66	140	743	D
67	96	826	D
68	86	596	C
69	92	741	C
70	127	843	D
71	87	574	C
72	106	1007	D
73	124	855	D
74	110	795	C
75	146	1091	C
76	104	828	C
77	192	1664	D
78	185	1182	D
79	158	1405	E
80	97	750	D
81	161	1136	D
82	86	589	D
83	113	801	D
84	93	820	C
85	149	558	C
86	106	475	C
87	93	1080	D
88	104	732	D
89	109	417	C
90	102	902	D
91	104	492	D
92	106	1039	C
93	151	1064	C

94	77	679	E
95	115	696	C
96	128	927	D
97	167	1082	C
98	130	965	C
99	101	915	C
100	107	737	C
101	98	480	C
102	119	904	D
103	80	542	C
104	98	586	C
105	128	1601	D
106	88	761	D
107	96	759	D
108	270	2184	C
109	219	1744	D
110	202	802	C
111	189	1477	C
112	174	1053	D
113	141	950	C
114	113	825	D
115	187	1393	C
116	251	2356	D
117	220	1428	C
118	226	2023	C
119	217	1377	D
120	158	774	C
121	70	367	C
122	113	641	C
123	208	1690	C
124	146	1371	C



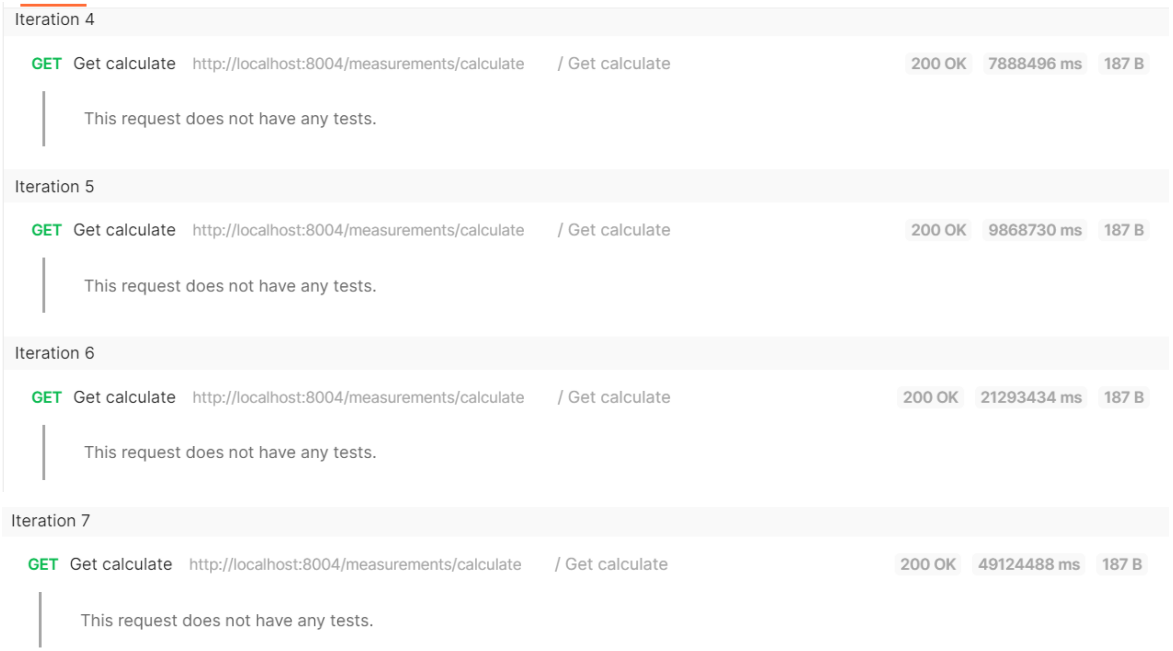


Figure 1. The perfomance test result from Microservice platform in the Postman

Non-exclusive licence to reproduce thesis and make thesis public

I, Octanty Mulianingtyas

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to

reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright,

A Microservice-based Platform for Software Technical Debt Analysis

supervised by Professor Ulrich Norbisrath, PhD, University of Tartu, Professor Bruno Rossi, PhD, Masaryk University

2. I grant the University of Tartu a permit to make the work specified in p. 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 3.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.

3. I am aware of the fact that the author retains the rights specified in p. 1 and 2.

4. I certify that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Octanty Mulianingtyas

4/08/2021