# A COMPARATIVE STUDY OF SOFTWARE QUALITY USING THE HYBRID AGILE SOFTWARE DEVELOPMENT LIFECYCLE AND THE PLAN-DRIVEN DEVELOPMENT MODEL

BY

ABDUL MUIZZ BIN ABDUL SALAM

A thesis submitted in fulfilment of the requirement for the degree of Master of Computer Science and Information Technology

Kulliyyah of Information and Communication Technology
International Islamic University Malaysia

FEBRUARY 2025

# ABSTRACT

Software development methodology is a series of processes that is necessary for software development to achieve a good quality software. Software development methodology can be divided into three general established approaches: the plan-driven development model, agile methodologies, and hybrid agile model. Agile methodologies are difficult to adapt for businesses that have strict timeline from its clients; thus, companies, especially software development houses, prefer the plan-driven development model or hybrid agile model as the models to be referred to in a software project. However, which software model between the plan-driven development model and the hybrid agile model which is widely used by software development houses will result in higher-quality software? To answer the question, this research investigates the plan-driven development model and hybrid agile model for comparison of the software quality produced by the software engineers. This research is empirical research adopting an experimental study to compare the internal quality of a plan-driven development model with a hybrid agile model involving a group of software engineers who were divided into two groups; one group using the Hybrid Agile model, WaterScrumFall while the other be using a plan-driven development model, the Waterfall model. The outcomes of this study shows that the complexity of the code, measured by the Average Cyclomatic Complexity (ACC); the lines of codes indicates the size and scale of the codebase; and the dependencies between objects specifically addressing structural quality through metrics like Coupling between Objects (CBO) and Lack of Cohesion in Methods (LCOM). The less complex project with low coupling between objects and cohesion in methods will contribute to high-quality source code and easier project management and maintenance, and it will also be cost effective. As a result, the team that implements Hybrid Agile model produce lower ACC and CBO, but higher LCOM compared to the team that applies plan-driven methodology. This research concludes that the Hybrid Agile model produces better software quality than the plan-driven methodology. This research will contribute to the benefit of knowing which model will produce better internal software quality and provide better insights for developers on which software development methodology may lead to software that is less complexity with a minimal line of codes, less coupling between methods, and more lack cohesion method.

# ملخص البحث

منهجية تطوير البرمجيات هي سلسلة من العمليات الضرورية لتطوير البرمجيات لتحقيق جودة برمجيات جيدة. يمكن تقسيم منهجية تطوير البرمجيات إلى ثلاثة مناهج عامة راسخة: نموذج التطوير القائم على التخطيط، والمنهجيات المرنة، والنموذج المرن الهجين. يصعب على الشركات التي لديها جدول زمني صارم من عملائها تكييف المنهجيات المرنة؛ لذلك، تفضل الشركات، وخاصةً دور تطوير البرمجيات، نموذج التطوير القائم على التخطيط أو النموذج المرن الهجين كنماذج يتم الرجوع إليها في مشروع برمجي. ومع ذلك، أي نموذج برمجي بين نموذج التطوير القائم على التخطيط والنموذج المرن الهجين المستخدم على نطاق واسع من قبل دور تطوير البرمجيات سيؤدي إلى برمجيات عالية الجودة؟ للإجابة على السؤال، يبحث هذا البحث في نموذج التطوير القائم على التخطيط والنموذج المرن الهجين للمقارنة بين جودة البرمجيات التي ينتجها مهندسو البرمجيات. هذا البحث هو بحث تجريبي يتبنى دراسة تجريبية لمقارنة الجودة الداخلية لنموذج تطوير قائم على التخطيط مع نموذج مرن هجين يضم مجموعة من مهندسي البرمجيات الذين تم تقسيمهم إلى مجموعتين؛ مجموعة واحدة تستخدم النموذج المرن الهجين، WaterScrumFall، بينما تستخدم المجموعة الأخرى نموذج تطوير قائم على التخطيط، نموذج الشلال. تُظهر نتائج هذه الدراسة أن تعقيد التعليمات البرمجية، الذي يُقاس بمتوسط التعقيد الدوري (ACC)؛ تشير أسطر التعليمات البرمجية إلى حجم ونطاق قاعدة التعليمات البرمجية؛ والاعتمادات بين الكائنات التي تُعالج تحديدًا الجودة الهيكلية من خلال مقاييس مثل الاقتران بين الكائنات (CBO) ونقص التماسك في الأساليب (LCOM). سيساهم المشروع الأقل تعقيدًا مع الاقتران المنخفض بين الكائنات والتماسك في الأساليب في الحصول على تعليمات برمجية مصدرية عالية الجودة وإدارة وصيانة أسهل للمشروع، كما أنه سيكون فعالًا من حيث التكلفة. نتيجة لذلك، فإن الفريق الذي يُنفذ النموذج المرن الهجين يُنتج ACC و CBO أقل، ولكن LCOM أعلى مقارنةً بالفريق الذي يُطبق منهجية قائمة على التخطيط. يخلص هذا البحث إلى أن النموذج المرن الهجين يُنتج جودة برمجيات أفضل من المنهجية القائمة على التخطيط. سيُساهم هذا البحث في فائدة معرفة أي نموذج سيُنتج جودة برمجيات داخلية أفضل وتوفير رؤى أفضل للمطورين حول منهجية تطوير البرمجيات التي قد تؤدي إلى برمجيات أقل تعقيدًا مع الحد الأدنى من أسطر التعليمات البرمجية، وأقل اقتران بين الأساليب، والمزيد من نقص أسلوب التماسك.

# APPROVAL PAGE

I certify that I have supervised and read this study and that in my opinion, it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a thesis for the degree of Master of Computer Science and Information Technology.

…….
Norzariyah binti Yahya
Supervisor

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a thesis for the degree of Master of Computer Science and Information Technology.

…….
Noor Azura
Internal Examiner

…….
Siti Sarah Binti Maidin
External Examiner

This thesis was submitted to the Department of Computer Science and is accepted as a fulfilment of the requirement for the degree of Master of Computer Science and Technology.

…….
Asst. Prof. Dr. Amir 'Aatieff Bin Amir Hussin
Head, Department of Computer Science

This thesis was submitted to the Kulliyyah of Information and Communication Technology and is accepted as a fulfilment of the requirement for the degree of Master of Computer Science and Information Technology.

…….
Prof. Dr. Murni binti Mahmud
Dean,
Kulliyyah of Information and Communication Technology

# DECLARATION

I hereby declare that this dissertation is the result of my own investigations, except where otherwise stated. I also declare that it has not been previously or concurrently submitted for any other degrees at IIUM or other institutions.

Abdul Muizz bin Abdul Salam

Signature:..... Date:.....

**INTERNATIONAL ISLAMIC UNIVERSITY MALAYSIA**

**DECLARATION OF COPYRIGHT AND AFFIRMATION OF FAIR USE OF UNPUBLISHED RESEARCH**

**A COMPARATIVE STUDY OF SOFTWARE QUALITY USING THE HYBRID AGILE SOFTWARE DEVELOPMENT LIFECYCLE AND THE PLAN-DRIVEN DEVELOPMENT MODEL**

I declare that the copyright holder of this thesis/dissertation are jointly owned by the student and IIUM.



By signing this form, I acknowledged that I have read and understand the IIUM Intellectual Property Right and Commercialization policy.

Affirmed by Abdul Muizz bin Abdul Salam

…….. …….

Signature Date

**INTERNATIONAL ISLAMIC UNIVERSITY MALAYSIA**

**DECLARATION OF COPYRIGHT AND AFFIRMATION OF FAIR USE OF UNPUBLISHED RESEARCH**

**A COMPARATIVE STUDY OF SOFTWARE QUALITY USING THE HYBRID AGILE SOFTWARE DEVELOPMENT LIFECYCLE AND THE PLAN-DRIVEN DEVELOPMENT MODEL**

I declare that the copyright holder of this thesis/dissertation is International Islamic University Malaysia.



No part of this unpublished research may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording or otherwise without prior written permission of the copyright holder except as provided below

1. Any material contained in or derived from this unpublished research may only be used by others in their writing with due acknowledgement.

2. IIUM or its library will have the right to make and transmit copies (print or electronic) for institutional and academic purpose.

3. The IIUM library will have the right to make, store in a retrieval system and supply copies of this unpublished research if requested by other universities and research libraries.

By signing this form, I acknowledged that I have read and understand the IIUM Intellectual Property Right and Commercialization policy.

Affirmed by Abdul Muizz bin Abdul Salam

…….. …….

Signature Date

**INTERNATIONAL ISLAMIC UNIVERSITY MALAYSIA**

**DECLARATION OF COPYRIGHT AND AFFIRMATION OF FAIR USE OF UNPUBLISHED RESEARCH**

**A COMPARATIVE STUDY OF SOFTWARE QUALITY USING THE HYBRID AGILE SOFTWARE DEVELOPMENT LIFECYCLE AND THE PLAN-DRIVEN DEVELOPMENT MODEL**

I declare that the copyright holder of this thesis/dissertation is Abdul Muizz bin Abdul Salam



Affirmed by Abdul Muizz bin Abdul Salam

…….. …….

Signature Date

# ACKNOWLEDGEMENTS

All glory is due to Allah, the Almighty, whose Grace and Mercies have been with me throughout the duration of my programme. Although, it has been tasking, His Mercies and Blessings on me ease the herculean task of completing this thesis

I am most indebted to my supervisor, Dr. Norzariyah binti Yahya, whose enduring disposition, kindness, promptitude, thoroughness and friendship have facilitated the successful completion of my work. I put on record and appreciate her detailed comments, useful suggestions and inspiring queries which have considerably improved this thesis. Her brilliant grasp of the aim and content of this work led to her insightful comments, suggestions and queries which helped me a great deal. Despite her commitments, she took time to listen and attend to me whenever requested. The moral support she extended to me is undoubtedly a boost that helped build and write the draft of this research work.

Lastly, my gratitude goes to my beloved wife; for their prayers, understanding and endurance while away.

Once again, we glorify Allah for His endless mercy on us, one of which is enabling us to successfully round off the efforts of writing this thesis. Alhamdulillah.

# TABLE OF CONTENTS

# LIST OF TABLES

# LIST OF FIGURES

# CHAPTER ONE
# INTRODUCTION

## 1.1 OVERVIEW

The software development lifecycle (SDLC) is a systematic framework that outlines the phases of software development, starting from conception to deployment. It includes multiple phases, each with specific goals and outputs, ensuring that the result satisfies user needs and quality benchmarks. Bender RBT Inc. (2003) defines the software development life cycle as a methodology for creating high-quality software with clearly defined phases and processes of software development. In general, the software development lifecycle can be divided into three general established approaches which are the plan-driven development model, agile methodologies, and hybrid agile model (Imani, 2017). Plan-driven development is a methodology that prioritises planning over implementation. Plan-driven development follows the process sequentially and does not turn back. For example, in Waterfall methodology, the process will be sequential from planning, requirement gathering, design and implementation and the final phase is testing. While, agile methodology prioritises work done over plans. There are various types of Agile such as Scrum, eXtreme Programming, Lean and Dynamic System Development Method (DSDM). On the other hand, the hybrid agile model combines a plan-driven development model with an agile. An example of a hybrid agile model is the combination of Scrum and Waterfall as a model that is also known as, ScrumFall for short, and some call the combination as WaterScrumFall.

Each software project requires the software engineering team to refer to a software development lifecycle or known as a model, starting from the project initiation until the project completion. There are those who prefer plan-driven development models; some choose Agile, and recently, many started combining the plan-driven development model with agile methodologies, which led to the existence of a new model known as Hybrid Agile. However, does the new preference, the Hybrid Agile model, produce good software quality? Among the models, plan-driven development

and hybrid agile, which software model produces less complexity, Lines of Code (LOC), Lack of Cohesion in Methods (LCOM) and Coupling between Objects (CBO)?

This study compares the software quality between plan-driven development and hybrid agile models used by two software engineering teams in a software project. One of the teams used the hybrid agile model, the WaterScrumFall while the other team referred to a plan-driven methodology, the Waterfall model.

## 1.2 PROBLEM STATEMENT

In a software project, a software engineering team have to refer to a model for the completion of the project. Royce (1987) highlighted that one of the oldest software development life cycle models is the waterfall model that is categorised as a plan-driven development model. Later in 2001, Agile methodologies were introduced by Beck & al. (2001), and software engineering teams started using agile development models and approaches such as Scrum, eXtreme Programming and other development techniques.

However, the hybrid model such as Hybrid Agile has also been investigated in recent development initiatives. The hybrid agile is the combination of Agile approaches with plan-driven development model in a software project (Vijayasarathy & Butler, 2015). Hybrid Agile is considered the ideal model that best meets the needs of a project by leveraging the strengths of both plan-driven development models, such as sequential plan-driven development and Scrum with iterative and rapid delivery. However, besides combining two models in one project for better project flow, monitoring and to best fit the needs of a project, another angle needs to be taken care of, especially the code quality. The code quality leads to a better software quality, which is crucial throughout the entire software development process. Higher quality software generally leads to greater customer satisfaction, as it better meets the needs and expectations of those who request it. This study examines the software quality that results from projects using plan-driven development model compared to hybrid agile models. The results of this research will form the basis for the software engineering team to choose between the scheduled development and the Hybrid Agile model in their software projects.

## 1.3 RESEARCH QUESTIONS

The research questions are as follows:

1. Which model between the Plan-driven development model, the Waterfall model and the Hybrid Agile model, WaterScrumFall produces better internal quality software?
2. What are the differences in internal quality software metrics score compared with the threshold for the Plan-driven development model, the Waterfall model and the Hybrid Agile model, WaterScrumFall?

## 1.4 RESEARCH OBJECTIVES

The objectives of the research are as follows:

1. To identify the software development lifecycle model between the Plan-driven development model, the Waterfall model and the Hybrid Agile model, WaterScrumFall will produce better internal quality software.
2. To examine the outcome of internal quality metrics for the Plan-driven development model, the Waterfall model and the Hybrid Agile model, WaterScrumFall with the threshold as the comparison.

## 1.5 RESEARCH SCOPE

This study examines software quality in the context of two development methodologies: the Hybrid Agile model and the Plan-Driven Development model. It focuses on comparing key quality attributes such as reliability, maintainability, efficiency, and overall project success. The study aims to provide insights into which model is more effective for clinic system.

The scope of this research includes a comparative analysis of software quality between Hybrid Agile and plan-driven model. Furthermore, evaluation of key quality

metrics like code complexity and maintainability. In addition, this research adopted experimental research approach that required the participant to develop a project of the clinic system demonstrating the application of both models.

This study does not cover the financial costs of implementing either development model and the security vulnerabilities and risk management associated with the software development. The scope is designed to ensure a focused and manageable research study within the given time and resource constraints. By limiting the analysis to software quality attributes, the research can provide precise insights without being too broad. The exclusion of security and financial aspects allows for a deeper examination of software process effectiveness rather than operational costs or risks.

This study will compare the Hybrid Agile and Plan-Driven development models concerning software quality, emphasizing reliability, maintainability, and efficiency. The research will focus on experimental data while excluding financial and security concerns to maintain a well-defined scope.

## 1.6 RESEARCH SIGNIFICANT

WaterScrumFall is a model implemented in an organisation that has been adopting the Waterfall model in their software project. It occurs when a software engineering team migrating from the Waterfall model to Scrum consists of the first few phases (such as feasibility studies, funding exercises, requirements gathering, etc) are executed using a Waterfall model. Then, the development and testing phases are executed using Scrum and broken into small sprints. The actual deployment, however, is then executed using the Waterfall Methodology again (Butgereit, 2018).

This research holds significant value in the field of software engineering as it provides empirical insights into the internal quality of software developed using two widely adopted methodologies: the Plan-Driven model (Waterfall) and the Hybrid Agile model (WaterScrumFall). A systematic literature review by Almeida & Bálint (2024) highlight that hybrid methodologies have emerged as organizations seek to efficiently utilize existing capabilities while exploring new opportunities. Additionally, Klünder et

al. (2021) study found that companies utilize multiple frameworks, methods and practices, and combine these into hybrid methods, indicating a prevalent trend towards hybrid approaches in software development. Given the increasing adoption of hybrid models in organizations transitioning from traditional methodologies, understanding their impact on software quality is critical for informed decision-making.

By analysing key quality metrics—code complexity, coupling between methods, and cohesion in methods—this study will determine which approach results in more maintainable, efficient, and cost-effective software. The findings offer software developers and project managers practical guidance on choosing the most suitable development methodology to achieve higher software quality with lower maintenance costs.

Ultimately, this research contributes to the broader software development community by bridging the knowledge gap between traditional and hybrid methodologies, enabling organizations to adopt optimal development practices that enhance software quality and sustainability.

# CHAPTER TWO
# LITERATURE REVIEW

## 2.1 INTRODUCTION

This chapter introduces the topics arranged in sequence, starting with an elaboration of quantitative study, followed by the explanation of software quality in Section 2.2. The literature review in Section 2.3 defines internal quality, plan-driven, agile and hybrid agile, followed by tools for analysis such as PHPMetrics. The literature review is continued with the explanation of existing research in comparing Agile and Hybrid Agile using experiments, followed by a controlled experiment to assess the hybrid software development process from projects. In addition, this chapter also discusses the Object Oriented (OO) metrics, and the focus are *Lines of Codes* (LOC), Cyclomatic Complexity (CC), Coupling between Objects (CBO) and Lack of Cohesion in Methods (LCOM). Lastly, Section 2.11 focuses on a diagram that describes the summary of the literature review.

## 2.2 SOFTWARE QUALITY

International Standard Organization (1991) addressed the ISO 9126 for information technology which is a framework for reviewing software quality. The ISO 9126 framework outlines six key quality metrics which are functionality, reliability, usability, efficiency, maintainability and portability, as depicted in Figure 2.1. Suppose the

software or a software project achieves all these quality metrics. In that case, it can be labelled as a high-quality software.

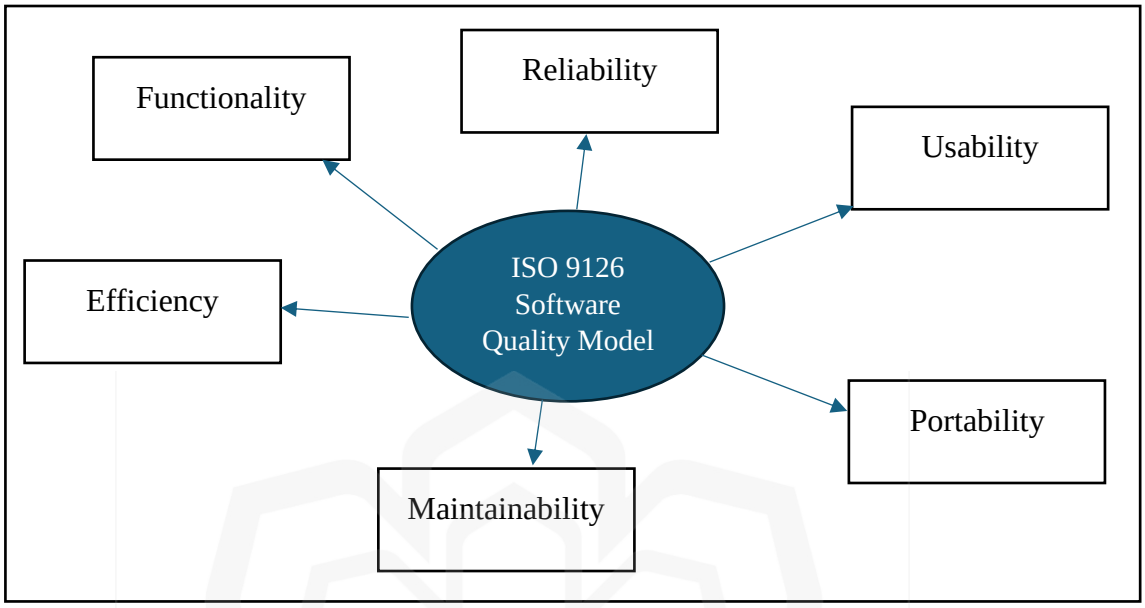



Figure 2.1: The ISO 9126 Software Quality Model.

The ISO 9126 framework also states the definition for each key quality metrics. Firstly, functionality can be defined as the extent to which the software meets the required functions and requirements. Furthermore, reliability is a requirement to determine the extent of the software's reliability. A software is easy to use when it meets the usability criteria. The efficiency of a software will be good when it can handle what it needs to handle. Moreover, the high-quality software is easy to maintain which defines its maintainability. Alomar et al. (2022) analysed 1,828 open-source projects, identifying 1,957 commits where developers explicitly mentioned refactoring for reusability. They extracted and labelled these as reusability refactorings. The study found that such refactorings often involve design-level changes affecting high-level code elements like packages, classes, and methods. These modifications can lead to improved software quality by enhancing code reusability and maintainability. Lastly, portability in software quality refers to the ease with which a software system or component can be transferred from one environment to another. This includes different operating systems, hardware platforms, or software dependencies. The portability of a software is achieved when it can adapt to multiple or different environments.

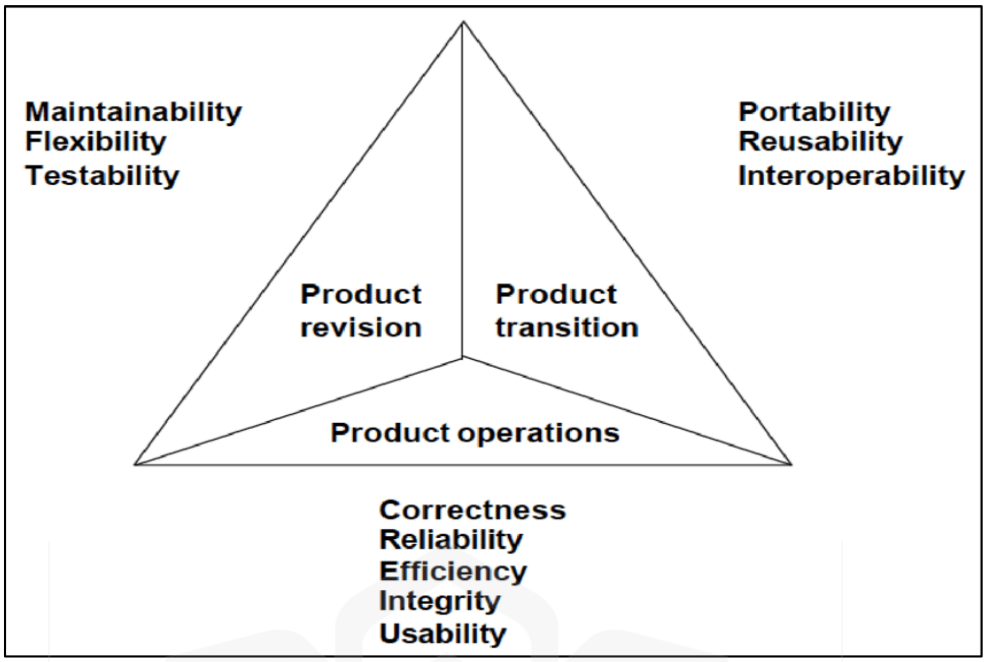



Figure 2.2: Mc Call's Quality Model (Cavano & McCall, 1978)

Another software quality model that has been established is the Mc Call's Quality Model as illustrated in Figure 2.2. This model is a well-known software quality framework that addresses software quality concerns. The model consists of three major perspectives, which are Product Revision, Product Transition and Product Operations, with each containing multiple quality factors and showed in Figure 2.2. The factors represent high-level attributes that determine the overall quality of a software product.

Al-Qutaish (2010) highlights that the Product Revision is the degree of ease of modification and adaptability, consists of maintainability (the effort required to locate and fix a fault in the program within its operating environment), flexibility (the ease of making changes required by changes in the operating environment) and testability (the ease of testing the program, to ensure that it is error-free and meets its specification).

Next, Al-Qutaish (2010) also mentioned that the Product Transition is the measurement of how well a software adapt to new environments and it breaks down to three quality factors which are portability (the effort required to transfer a program from one environment to another), reusability (the ease of reusing software in a different context) and interoperability (the effort required to couple the system to another system).

Lastly, Singh & Kannojia (2013) reviewed that the Product Operations is how good the software operates, and it has five factors which are correctness (the extent to which a program fulfils its specification), reliability (the system's ability not to fail), efficiency (further categorized into execution efficiency, storage efficiency and generally mean use of resources such as processor time or storage), integrity (the protection of the program from unauthorized access) and usability (the ease of the software).

This research uses internal quality that needs to be measured for each software engineering's team. The metrics from internal quality that need measured are maintainability, flexibility, reusability, readability and testability.

## 2.3 INTERNAL QUALITY

Internal quality is a type of quality that relies on clean code, complexity, duplication and component reuse. Based on ISO/IEC 25010:2023, there are nine characteristics of internal quality which are functional suitability, performance efficiency, compatibility, usability, reliability, security, maintainability, portability and accessibility (International Organization for Standardization, 2023).

Lines of Code (LOC) is a source code of a programmer that instructs the computer on how the program should work. LOC is a software metric that measures the size of a software program by counting the number of lines in its source code. It serves as a straightforward indicator of the program's length and complexity. LOC for every programming language varies. High-level programming languages can produce less LOC compared to assembly language. While LOC can provide a general sense of a program's size, it doesn't account for code quality or complexity. Petersson & Zhang (2013) found that by analysing LOC alongside other metrics, they could identify methods that were too lengthy or complex, leading to recommendations for refactoring to enhance maintainability.

Coupling between Objects (CBO) measures the coupling between classes based on class usage. CBO measures the degree of interdependence between classes in an object-oriented system. Specifically, it counts the number of classes to which a

particular class is coupled. High coupling indicates a class is heavily dependent on other classes, which can reduce modularity and complicate maintenance. Conversely, low coupling suggests a more modular and maintainable design. Chidamber and Kemerer introduced this metric in 1991 to assess the quality of object-oriented designs. Two classes are coupled when methods declared in one class use methods or instance variables defined by the other class (Chidamber & Kemerer, 1991). Milić & Makajić-Nikolić (2022) assess the architectures of microservices and monoliths based on coupling, testability, security and complexity. The study found that microservices reduce coupling but increase complexity in communication and security risks, while monoliths maintain simpler integration and security management but less efficient in large-scale applications. Thus, the study propose a hybrid solution for its optimization.

Lack of Cohesion of Methods (LCOM) quantifies the lack of cohesion within a class by evaluating the dissimilarity of methods based on the instance variables they access. A high LCOM value indicates that methods within a class are not related, suggesting the class may be trying to achieve multiple objectives and could benefit from refactoring. Husein and Oxley discussed this metric in 2009, emphasizing its role in identifying classes that lack a single, well-defined purpose (Husein & Oxley, 2009). LCOM measures the level of cohesion based on the field usage by the methods of a class. Albeladi et al. (2014) utilizes LCOM to assess software quality. The findings indicated that classes with high LCOM were more prone to defects, underscoring the importance of cohesion in class design.

Cyclomatic Complexity, introduced by Thomas McCabe in 1976, measures the number of linearly independent paths through a program's source code. Average Cyclomatic Complexity (ACC) defines the quantitative measure of the number of linearly independent paths in it. It also indicates the complexity of the program. ACC provides insight into the complexity of a program by counting decision points like loops and conditionals. A higher ACC suggests more complex code, which can be harder to test and maintain. Keeping ACC within reasonable limits is crucial for ensuring code quality. Munawar & Naveed (2022) analyses how the syntax of a programming language affects the complexity of its programs. The study uses Cyclomatic Complexity (CC) and other software metrics to compare Java and Python.

Weighted Methods per Class (WMC) measures the sum of complexity of the methods in a class. A predictor of the time and effort required to develop and maintain a class we can use the number of methods and the complexity of each method. A higher WMC indicates that a class has more complex methods, which can lead to increased development and maintenance efforts. This metric helps in identifying classes that might be doing too much and could benefit from refactoring. Saini et al. (2014) emphasized the relevance of WMC in assessing class complexity. The study found that classes with high WMC were often associated with higher defect rates, suggesting a need for design improvements.

Response for Class (RFC) defines a set of methods that can be executed in response and messages received by Object of that class. RFC measures the number of methods that can potentially be executed in response to a message received by an object of that class. A higher RFC indicates a greater number of methods that can be invoked, which can complicate testing and increase the likelihood of defects. It also helps estimating the time needed for testing the class. Managing RFC is essential for maintaining class simplicity and predictability. Stavtsev & Bugayenko (2024) investigates the correlation between Cyclomatic Complexity (CC) and Response for Class (RFC) in Java classes. The hypothesis is that smaller classes with fewer methods should be less complex, leading to better maintainability. The study concludes that a strong correlation exists between CC and RFC, reinforcing the idea that keeping classes small and method count low leads to lower complexity. This insight provides practical guidance for writing more maintainable Java code.

Alenezi (2021) applied a software metric approach to examine the method of evolution of the internal quality of object-oriented open-source software systems. The result from the approach shows that Lines of Code (LOC) variable gives a slight difference between different systems, all pairwise comparisons of internal quality attributes have considerable correlation, and the complexity and inheritance give effect on the LOC was positive and considerable while the effect of Coupling and Cohesion was not significant. In summary, software metrics like LOC, LCOM, CBO, and ACC directly impact internal quality attributes such as maintainability, reusability, understandability, modifiability, and testability. Optimizing these metrics helps in building a more robust and high-quality software system.

## 2.4 PLAN-DRIVEN DEVELOPMENT MODEL

A plan-driven approach is an approach from software development where tasks are allocated according to where they are visible in the software development lifecycle (Marinho et al., 2019). A plan-driven development model promotes software engineering activities in a well-planned manner wherein the progress of each phase is measured against the designed plan. Plan-driven development also known as heavyweight methodologies, are performed linearly, whereas operation of the activities such as specification, development, validation and evolution must be conducted in sequential order, which means that an activity must be done before the next one starts (Costa et al., 2022). Waterfall Model, V-Model, Iterative Model, Incremental Model, Incremental Prototyping Model, Spiral Model and Rapid Application Development (RAD) Model are the examples of the plan-driven Software Development Life Cycle (SDLC) models (Garg et al., 2022).

Before starting to work on a project, software engineers usually need to plan out the system requirements and when the work needs to be delivered. This means laying out the system needs, getting a sense of the project's scope, and making a reasonable delivery schedule. This kind of planning is necessary to keep risks to a minimum and make sure the development part goes smoothly. The plan has several positive aspects, but it also has some problems that can be listed. Putting a lot of weight on detailed paperwork and planning time ahead, and assuming that all needs can be fully understood and stated at the start of the project. These things can make it harder for team members to work together and talk to each other, and they make software goods take longer to get to market because they need more planning.

## 2.5 AGILE DEVELOPMENT MODEL

Agile practices are a development approach based on agile manifestos that value individuals and interactions, working software, customer collaboration and responding to change. Agile is a development approach based on four agile manifestoes and guided by twelve agile principles (Agile Alliance, 2015b).

Generally, agile can be defined as the ability to create and responsive to any change. It is a method that is used when dealing with an uncertain and unstable environment and is mostly successful. “Agile” label was chosen by the authors of Agile Manifesto for the whole idea because the word covers the adaptiveness and changes responsiveness, which is necessary to their approach (Agile Alliance, 2015a). The iterative development method is the basis of agile software development where the requirements and solutions change through the cooperation of self-organising cross-functional teams. It encourages repetitive persistent inspection and adaptation by promoting regulated project management processes. Furthermore, it also has leadership beliefs that encourage teamwork, self-organisation, and responsibility. In addition, it is also a set of engineering best practices to allow for fast delivery of high-quality software. Moreover, the agile methodology also refers to a business approach where it aligns development with client needs and company targets (Rizwan Jameel Qureshi & Hussain, 2008).

One of the advantages of agile software development is that it enables teams to deliver value faster, with more quality and the ability to expect and enhance skills to respond to change (Alsaqqa et al., 2020). Alsaqqa et al. also mention that feature delivery in increments can reduce the non-stagnant increasing cost, opening chances to other changes without loss in cost and time.

Apart from its advantages, it also comes with the disadvantages. There is a challenge of managing complex projects. Agile practitioners can also neglect non-functional requirements such as security and performance (Jarzebowicz & Weichbroth, 2021). Furthermore, emphasis on collaboration and communication in agile teams can sometimes lead to inefficiencies. In addition, the reliance on customer involvement can lead to scope creep.

## 2.6 HYBRID AGILE DEVELOPMENT MODEL

In software engineering, the plan-driven development model has long been a conventional method distinguished by thorough planning, documentation, and a step-by-step procedure. However, this model has been critically examined as a result of the quick advancement of technology and the growing complexity of software projects.

Plan-driven approaches' inefficiencies, especially when it comes to adjusting to shifting requirements, led to a move towards more adaptable and agile procedures.

Barnard (2006) highlights that the traditional plan-driven models are often inadequate in addressing the dynamic nature of software development, suggesting that integrating agile principles can enhance project and technical management. Numerous researchers have extensively studied the transition from plan-driven to agile methodologies to understand its impact on project success, adaptability, and efficiency. Studies have explored factors such as team collaboration, risk management, and customer satisfaction, finding that while Agile enhances flexibility and responsiveness, challenges remain in scaling and integrating Agile within large, complex organizations. Petersen and Wohlin (2010) provide empirical evidence that moving from a plan-driven approach to an incremental one with agile practices can significantly alter perceptions of bottlenecks and unnecessary work within software projects. The transition from plan-driven development to agile not only improves responsiveness to change but also fosters a more collaborative environment among development teams, aligning closely with the principles of agile development that emphasise customer collaboration over contract negotiation (Boehm, 2002).

The hybrid agile development model is a viable solution to the limitations of both plan-driven and agile methodologies. Hybrid techniques can successfully alleviate the drawbacks of both traditional models and agile practices by combining the structured planning and documentation of the traditional model with the flexibility and responsiveness of the latter. Agile Alliance et al. (2017) state that Hybrid Agile is the combination of Agile methods with other non-Agile techniques. This approach may involve, for instance, conducting a thorough requirements analysis before proceeding with sprints for incremental delivery. Additionally, it can include iterative prototyping of a design while adhering to a single plan-driven implementation. In essence, Hybrid Agile integrates non-Agile methodologies as a foundation, followed by the application of Agile techniques.

Agile Alliance et al. (2017) also state that there are two scenarios on using Hybrid Agile, which are Hybrid as Fit-for-Purpose and Hybrid as Transition-to-Agile. Hybrid as Fit-for-Purpose means using a hybrid model for the purpose of the project risk. For instance, if a project manager is dealing with a high-risk project, the project manager should consider using incremental techniques to ensure customer engagement. Agile techniques may have high initial cost, but it is best on the overall result. Hybrid

as Transition-to-Agile means a process of changing the methodology of software development from traditional to agile by using a hybrid technique. This is best implemented if the organisation is large and wants to use Agile as the methodology to build software.

One of the primary advantages of hybrid agile models is their ability to adapt to the specific needs of a project. Yahya and Sarah Maidin (2023) emphasise that hybrid agile is often preferred by software engineering teams because it allows for a tailored approach that can accommodate the unique complexities of different projects. The adaptability from hybrid agile is particularly beneficial in environments where requirements are likely to evolve, as it combines the predictability of plan-driven methods with the iterative nature of agile practices. Moreover, hybrid models can enhance communication and collaboration among team members. By incorporating agile practices such as daily stand-ups and iterative feedback loops within a structured framework, teams can foster better interaction and alignment on project goals. This is crucial in overcoming the silos that often characterise traditional plan-driven approaches, as highlighted by Rahim et al. (2018) who discovered that their application of a hybrid model enhanced team dynamics and project results in intricate settings (Rahim et al., 2018).

The hybrid model also addresses the challenge of managing requirements effectively by considering the successful integration of systems modules. The integration of plan-driven and agile methods allows teams to maintain a clear structure for requirements management while still being able to respond to changes as they arise (Imani, 2017). The dual capability of maintaining a clear structure for requirement management and responding to changes ensures that projects can remain aligned with user needs without sacrificing the rigor of documentation and planning that is often necessary for compliance and quality assurance. Furthermore, the hybrid agile model can significantly reduce time-to-market by enabling teams to deliver functional increments of software more rapidly. By leveraging agile iterations within a plan-driven framework, organisations can achieve a balance between thorough planning and quick delivery, thereby enhancing their competitive edge in fast-paced markets. Kuhrmann et al. support this notion by illustrating how hybrid approaches allow for the integration of agile practices into traditional frameworks, leading to improved efficiency and project success (Kuhrmann et al., 2017). In addition, hybrid models can facilitate better risk management. By combining the structured risk assessment processes of plan-driven

methodologies with the adaptive risk mitigation strategies of agile practices, teams can proactively address potential issues throughout the development lifecycle. This comprehensive approach to risk management is essential for ensuring project stability and success, particularly in large-scale and complex software projects (Walrave et al., 2022).

In conclusion, the hybrid agile development model effectively addresses the shortcomings of both plan-driven and agile methodologies. By providing a flexible yet structured framework, hybrid approaches enhance adaptability, communication, requirements management, time-to-market, and risk management. This makes them particularly suitable for modern software development environments that demand both rigor and responsiveness.

## 2.7 ANALYSIS TOOL

There's a lot of analysis tool that can evaluate the metrics for PHP language such as PHPDepend, PHPMetrics and many more. PHPMetrics is an open-source library package developed by Lépine (2013). For the purpose of research, PHPMetrics was used because the availability of metrics is wide. The metrics that are available are shown in the Table 2.1

Table 2.1 Metrics available in PHPMetrics (Lépine, 2013)

| Metrics type | Metrics | Formula |
|---|---|---|
| Halstead complexity measures | Program vocabulary | $n = n1 + n2$ |
| | Program length | $N = N1 + N2$ |
| | Calculated program length | $N' = n1 * log2(n1) + n2 * log2(n2)$ |
| | Volume | $V = N * log2(n)$ |

| | | |
|---|---|---|
| | Difficulty | $D = \left(\frac{n1}{2}\right) * \left(\frac{N2}{n2}\right)$ |
| | Effort | $E = D * V$ |
| | Time required to program | $T = \frac{E}{18seconds}$ |
| | Number of delivered bugs | $B = V/3000$ |
| Cyclomatic complexity number and weighted method count | Cyclomatic complexity | $(CCN) = E - N + 2P$<br>Where:<br>$P =$ number of disconnected parts of the flow graph<br>$E =$ number of edges<br>$N =$ number of nodes |
| | Weighted Method Count | `sum m(w') over (w' in w)` |
| Kan's defect | Kan's defect | $kanDefect = 0.15 + 0.23 * \text{number of do ... while()} + 0.22 * \text{number of switch()} + \mathbf{0.07} * \text{number of if()}$ |
| Maintainability index | Maintainability index without comments | $MIwoc = 171 - 5.2 * \ln(aveV) - 0.23 * aveG - 16.2 * \ln(aveLOC)$ |
| | Maintainability index comment weights | $MIcw **= 50 * sin(sqrt(2.4 * perCM))$ |
| | Maintainability index | $mi = MIwoc + MIcw$ |
| Lack of cohesion of methods | | |
| Card and Agresti metric | Structural complexity | $SC = fan - out^2$ |

| | | |
|---|---|---|
| | Data complexity | $DC = v/(fan - out + 1)$ |
| Length | $loc$: lines count<br>$cloc$: lines count without multiline comments<br>$lloc$: lines count without empty lines | |
| Methods | | |
| Coupling | Afferent coupling $(Ca)$ | The number of classes in other packages that depend upon classes within the package is an indicator of the package's responsibility. |
| | Efferent coupling $(Ce)$ | The number of classes in other packages that the classes in a package depend upon is an indicator of the package's dependence on externalities. |
| | Instability | $I = Ce/(Ce + Ca)$ |
| Depth of inheritance tree | | Measures the length of inheritance from a class up to the root class. |

## 2.8 AN EXPERIMENT COMPARING AGILE AND HYBRID AGILE

Three issues were brought to light by Kavitha and Neelu (2021) in their research. These issues include the failure of a software project due to quality attributes that have not been optimised, the difficulty of delivering a high-quality project on schedule while

adhering to a limited budget, and the inability to consolidate the strengths of each agile technique.

In their research Kavitha and Neelu (2021) were studying three objectives which are to propose a new hybrid agile methodology, to ensure timely delivery of software projects to clients with low cost and to help software engineers to understand the key characteristics of the agile methods.

The experiment by Kavitha and Neelu (2021) employed a comparative examination of existing methodologies across four critical dimensions: team size, number of teams, volatility, and team dispersion. The experiment comprises sample size of 10 individuals, including a hybrid model specialist, a product owner, and 8 production team members. According to Figure 2.1, the model proposed from the experiment is initiated with requirement collection through a feasibility study. Customers will be engaged as soon as feasible after the requirements have been collected to participate in the design process, incorporating prototyping and risk analysis. Once the design is finalised and devoid of issues, the subsequent step involves the technical personnel responsible for the creation, testing, and deployment of the product. The customer will conduct a review of the generated product to verify the requirements. These processes are executed within the designated time window.

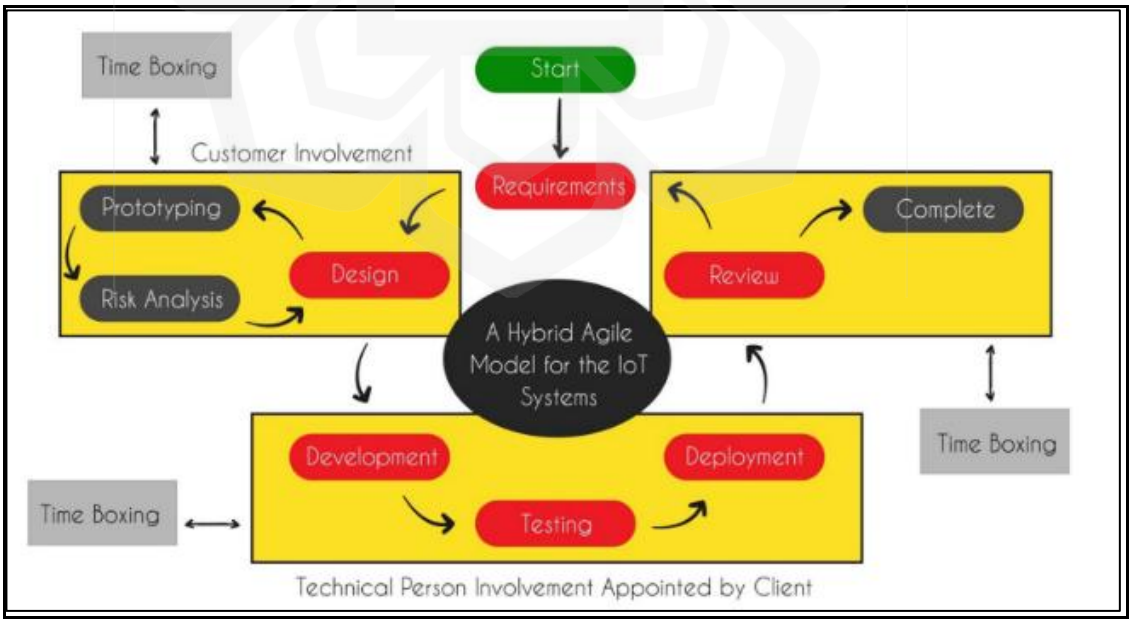



Figure 2.3: Proposed software development model by Kavitha & Neelu (2021)

Kavitha & Neelu (2021) finds out that without using a popular quality model, they evaluate the proposed model with Hybrid Agile Quality Parameter Analysis (HAQPA) based on 6 quality parameters: availability, scalability, flexibility, performance, usability and understandability. These parameters were evaluated throughout the Software Development Lifecycle phase with point 1 (No Impact) to 5 (Very High Impact). From the evaluation matrix, understandability has the greatest impact, followed by flexibility, usability, performance and availability. The lowest impact is scalability.

## 2.9 A CONTROLLED EXPERIMENT ON ASSESSMENT OF A HYBRID SOFTWARE DEVELOPMENT PROCESS

Less evaluation of the effectiveness of hybrid methods in academic contexts is the problem that has been addressed by Włodarski et al. (2021). The objective of the research conducted by Włodarski et al. (2021) is to assess the impact of a given development method on the success of students computing assessment from three aspects: product quality, team productivity and human factors.

The students are responsible on building a system to keep track of and share expenses with others using PHP, HTML and CSS. Requirement gathering was done by instructors and a Backlog of user stories were given to students. Any changing needs were not permissible throughout the experiment.

According to the research conducted by Włodarski et al. (2021), there are three groups, each of which will handle different development approaches: iterative, sequential and hybrid. The details of the processes by each of the development approaches are shown in Figure 2.2. Activities and project phases as well as work strategy and monitoring methods are the key points among the development approach. First, iterative approach is an approach consisting of a few iterations which in this case study, lasts for 2 weeks. An iteration consists of designing, coding and testing phases. Consequently, the outcome of the iteration will be a functionality that will be added to the system. Next, sequential approach following the sequence of development phases starting from planning, designing, coding (after design was approved) and testing. However, if a bug is found during the testing phase, bug fixing, and application quality

assurance are executed in about two weeks. On the other hand, a hybrid approach is a mix of different methods which depend on the organisation throughout the development process. One of the approaches namely Waterfall-Agile-Approach. The baseline using the Waterfall model and the implementation is in iterative approach.

The activities consist of three processes which are planning, work progress follow up and testing. Planning in the context of this research involves choosing the functionalities, identifying subtasks, and delegating among team members. For iterative and hybrid approaches, they will plan for the following two weeks while for sequential approaches, the plan consists of the entire implementation phase which is 6 weeks and must set delivery targets for every follow-up class. Assignment tasks were done in GitHub Issues. Furthermore, the user story and its translated tasks were provided to them.

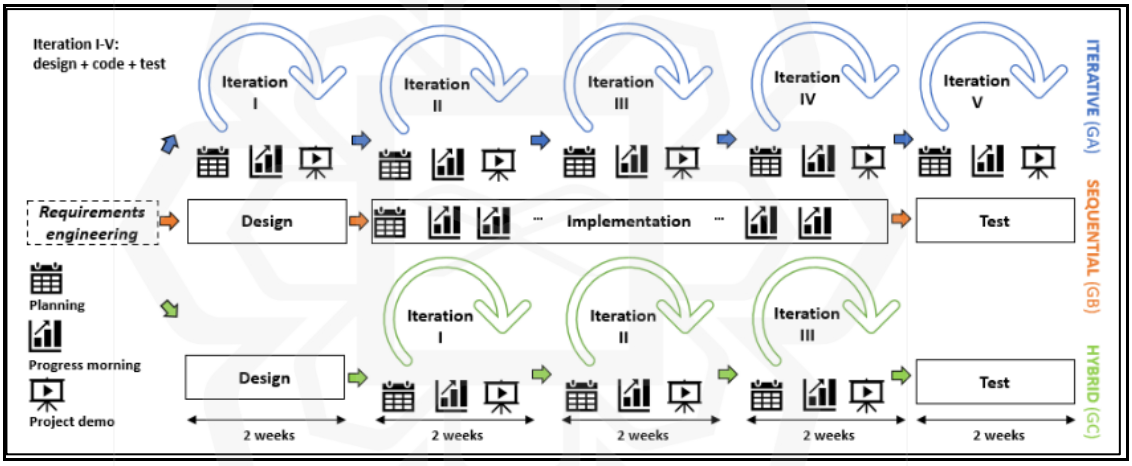



Figure 2.4: Processes of the three models (Włodarski et al., 2021)

Work progress follow up means each group needs to organise a meeting every week with agendas of update of the task statuses on GitHub issues, project summary report and calculation of work progress Key Performance Indicators (KPI)s. The first two agendas are the same for all approaches. A meeting report also needs to be provided which contains advancements made since the last class, tasks to be done before next class, problems faced, and risks found. The third agenda are explained as follows:

- Iterative - using simplified version of Team Velocity-like, tracked amount of completed tasks against total number of anticipated for a given iteration

- Sequential - using classic project management, Schedule Performance Index (SPI) - ratio of earned and planned value at a given time. In this study, the number of tasks done is divided by the number of tasks decided in the planning phase. During every team meeting and follow-up, students compare the current SPI score to the initial value they set.
- Hybrid - using Work In Progress, a metric originating from Lean manufacturing method - corresponds to the amount of ongoing tasks at a given time. Each team member will get two tasks. A Kanban board used for task tracking purposes which is available in GitHub. Task delegation summaries were given at the end of the team meeting.

Testing for sequential and hybrid approaches has a dedicated phase while the iterative method has testing for each iteration. Regardless of any processes, students need to do 3 processes which are: write and document test cases that link to the user story, execute the test cases, update Requirement Traceability Matrix (RTM) based on the output of the tests.

The experiment conducted by Włodarski et al. (2021) has affected the students' learning outcomes with five technical skills, and five soft skills have been evaluated. Hybrid teams acquire the highest score among the others: iterative and sequential teams That proven hybird teams have the highest soft skills followed by sequential and iterative. Technical skill for all teams evaluated much higher rates of positive feedback as compared to soft skills.

Hybrid approach has made improvement on team productivity in terms of functionalities completed that makes the first hypothesis to be true. Functional Completeness metric for the sequential and hybrid approaches is higher than the iterative approach. But hybrid approaches provide more gain in team productivity compared to the sequential approach.

The experiment partially confirms the second hypothesis on Functional Adequacy (FA). The FA metric for hybrid teams is the highest, which is 5% more than the other two teams. Having a dedicated testing phase is not the only success factor, the combination of iterative development and a dedicated testing phase results in the best outcome in the study.

Concerning the practice “Align team structure with system architecture” (hybrid approach characteristic), this method scored highest in aspect of HTML code quality, but it gives a small difference between the other two methods. In addition, hybrid teams scored lowest for PHP code quality because of inability to handle supporting tools.

## 2.10 CONCLUSION

This chapter explained the background study for this research. It explains the definition of quantitative study and its implications in this research. Moreover, experiment method been derived from the quantitative study that been used for this research. Furthermore, software quality definition was elaborated, and its key quality metrics been discussed. The key quality metrics has been shortlisted to four metrics as the analysis tool, PHPMetrics, is available to track those metrics. The four metrics are the core characteristics of internal quality of software.

From the software quality explanation, there is also explanation on the internal quality and its uses for this research. To make this research clearer, the definition of each of the software development methodologies: plan-driven, agile, and hybrid agile are introduced in Section 2.5, Section 2.6 and Section 2.7, respectively. Its application throughout the software development process is explained thoroughly. The comparison of software quality using plan-driven and Hybrid Agile model is the best combination as many practitioners in organizations prefer those models. The latter one was being used by organization that transition from the plan-driven model to the agile model.

Section 2.8, the analysis tool to use was explained and the metrics that are available to be captured from the case study for each developers’ team. Existing research study are explained in Section 2.9 and Section 2.10, which are about the comparison between agile and hybrid agile and a controlled experiment to assess hybrid software development process, respectively. No research study currently emphasises the plan-driven development strategy in conjunction with a Hybrid Agile paradigm through experimental methods. Consequently, this research addresses the gap by comparing internal quality through the application of those models. Figure 2.5 illustrates the structure of the literature review.

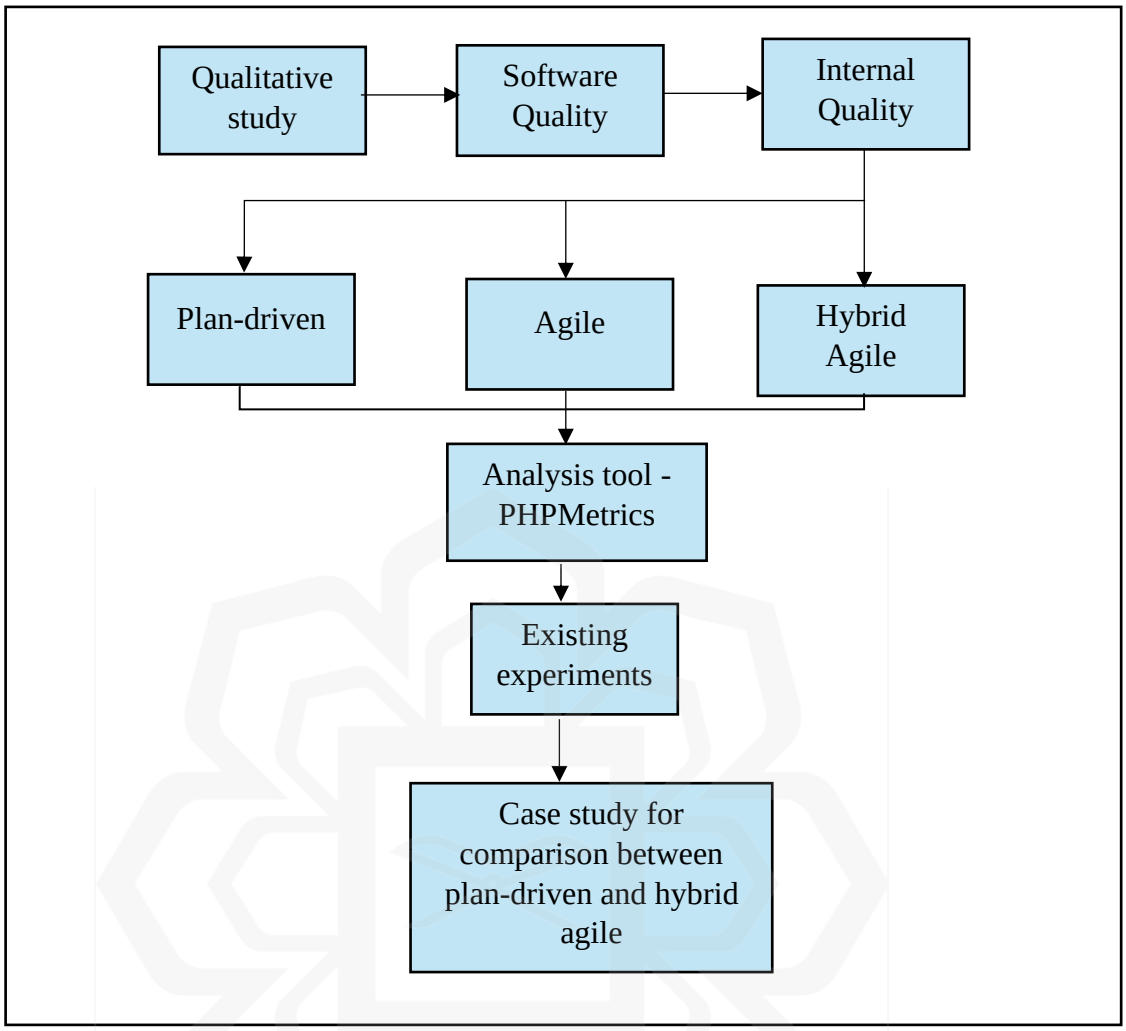



Figure 2.5: The literature review outline

# CHAPTER THREE
# RESEARCH METHODOLOGY

## 3.1 INTRODUCTION

This chapter outlines the research methodology and descriptively discusses on the design of the research. The definition of quantitative study and its uses are elaborated in Section 3.2. The flow of the research will be elaborated in Section 3.3. Moreover, the experiment setup and its measured metrics are explained in Section 3.4. In the last section, this chapter collectively explains the research flow, the metrics analysis approach and highlights the expected results in Section 3.4.

## 3.2 QUANTITATIVE STUDY

Quantitative study focuses on collecting and analysing numerical data such as surveys and experiments that can produce a researchable product (Bhandari, 2020). This method can ensure researchers know the concepts behind the topic they did.

The quantitative approach in this research involves a comparative analysis of two distinct software development models: the Plan-driven Development model (Waterfall) and the Hybrid Agile model (WaterScrumFall). By collecting and analyzing numerical data, this method evaluates which model produces better internal software quality, focusing on measurable metrics. This comparative focus ensures that the research systematically explores the strengths and weaknesses of both approaches, providing actionable insights into their effectiveness.

## 3.3 RESEARCH DESIGN

The methodology consists of a structured sequence of steps to ensure a systematic approach to the research, as shown in Figure 3.1. It begins with a literature review, where existing studies and publications are analysed to identify gaps or unexplored

areas in the research domain, forming the basis for defining the study's focus. Following this, a problem is identified based on the uncovered research gaps, and specific research questions and objectives are formulated to address the problem.

The evaluation criteria are then chosen, outlining the selected metrics from the tool that has been selected. The metrics chosen are based on the type of internal quality where code complexity and maintainability are highlighted. The next step involves designing experiment instruments, including tools, protocols, and processes, with careful consideration of participant selection which are software engineers with 2–3 years of experience, the experiment's duration, and how the data will be collected (Wohlin et al., 2024).

A pilot test is conducted to evaluate the effectiveness of these instruments and their alignment with the research objectives, allowing for refinement before the main experiment. The actual experiment is then carried out, starting with a participant briefing to ensure they understand their roles and the procedures. The experiment is closely monitored to ensure consistency, reliability, and validity of the data collected. After completion, the data is analysed using appropriate methods to extract meaningful insights and answer the research questions. Finally, the findings are compiled into a detailed report, documenting the entire process and results to contribute to the body of knowledge and facilitate future research.

The methodology is structured to enable a comparative study between the two process models under investigation. After identifying the research gap through a literature review, the study formulates research questions that specifically aim to compare the Plan-driven Development model and the Hybrid Agile model. Experiment instruments are designed to ensure consistent data collection across both groups, facilitating a direct comparison of their performance. The pilot test ensures that the instruments are effective in capturing the data necessary for this comparison. The main experiment involves two teams, each adopting one of the models, with their development activities monitored and recorded for analysis. By analysing the data collected from both teams, the study systematically evaluates which model leads to better internal software quality, as evidenced by key metrics.

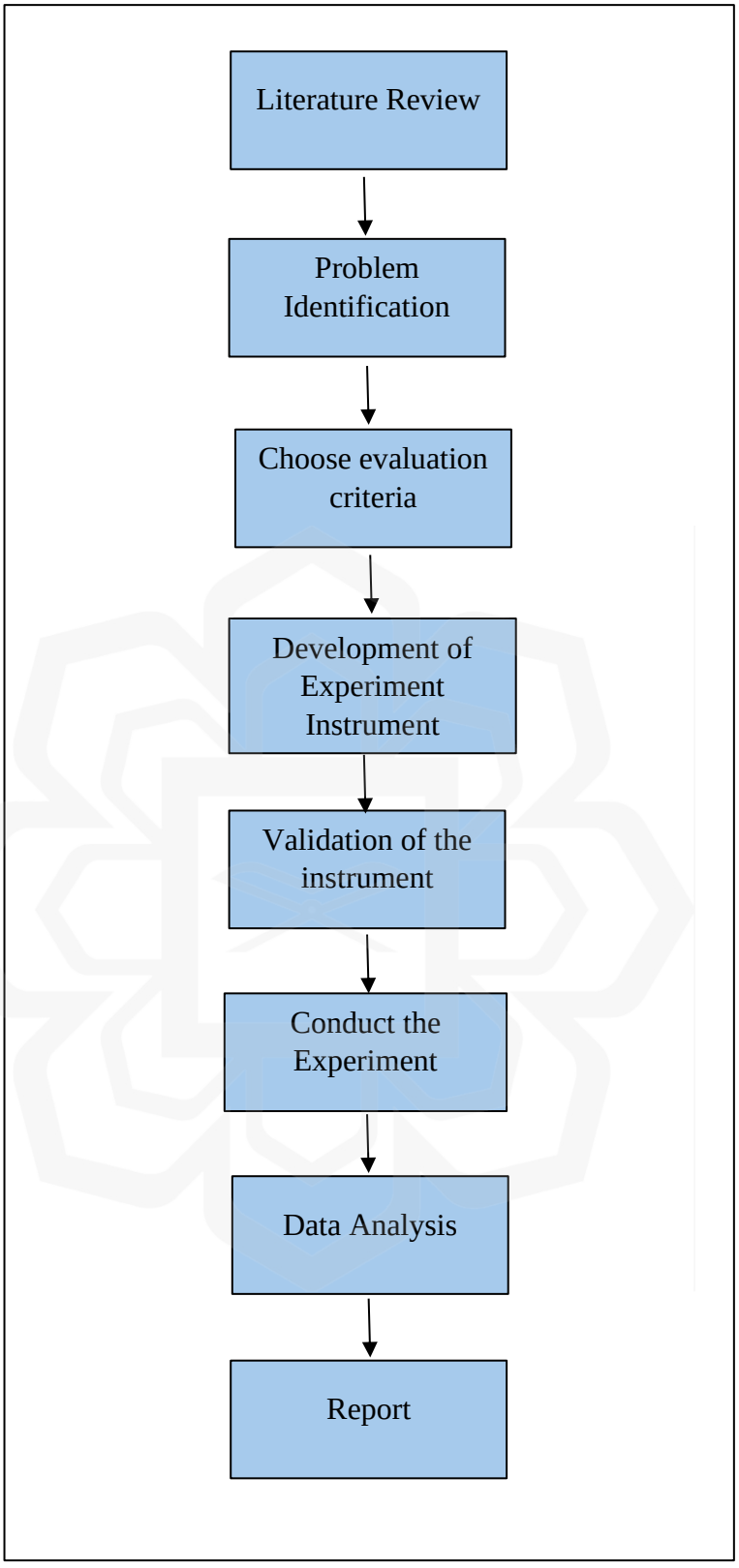



Figure 3.1: Research methodology flow

## 3.4 EXPERIMENT SETUP

The experiment is explicitly designed as a comparative analysis of the Plan-driven Development model (Waterfall) and the Hybrid Agile model (WaterScrumFall). The modules that are referred by the participants is a clinic system which targets the codes of the project developed. The choice of a clinic system as the focus of this research is motivated by its suitability as a case study for comparing software development methodologies, particularly the plan-driven development model and the Hybrid Agile model. Clinic systems are complex, data-intensive, and mission-critical applications that demand a high level of accuracy, security, and user-friendliness. These characteristics make them an excellent candidate for exploring how different development methodologies address real-world challenges in software development.

In a plan-driven development context, the rigid structure and detailed upfront planning align with the healthcare industry's need for compliance with regulations, such as data privacy and security standards. On the other hand, a Hybrid Agile model offers the flexibility to adapt to evolving requirements, which is particularly beneficial for addressing the dynamic and diverse needs of healthcare providers and patients.

By using a clinic system as the focal point, this research can effectively evaluate how each methodology impacts project outcomes, such as development efficiency, system quality, user satisfaction, and the ability to accommodate changes. This comparison not only highlights the strengths and weaknesses of both methodologies in a healthcare setting but also provides practical insights for selecting the most appropriate development approach for similar projects in the future.

The independent variables for the experiment are the Plan-driven Development model and Hybrid Agile model. Meanwhile, the dependent variables are the internal quality metrics are LOC, LCOM, CBO and ACC.

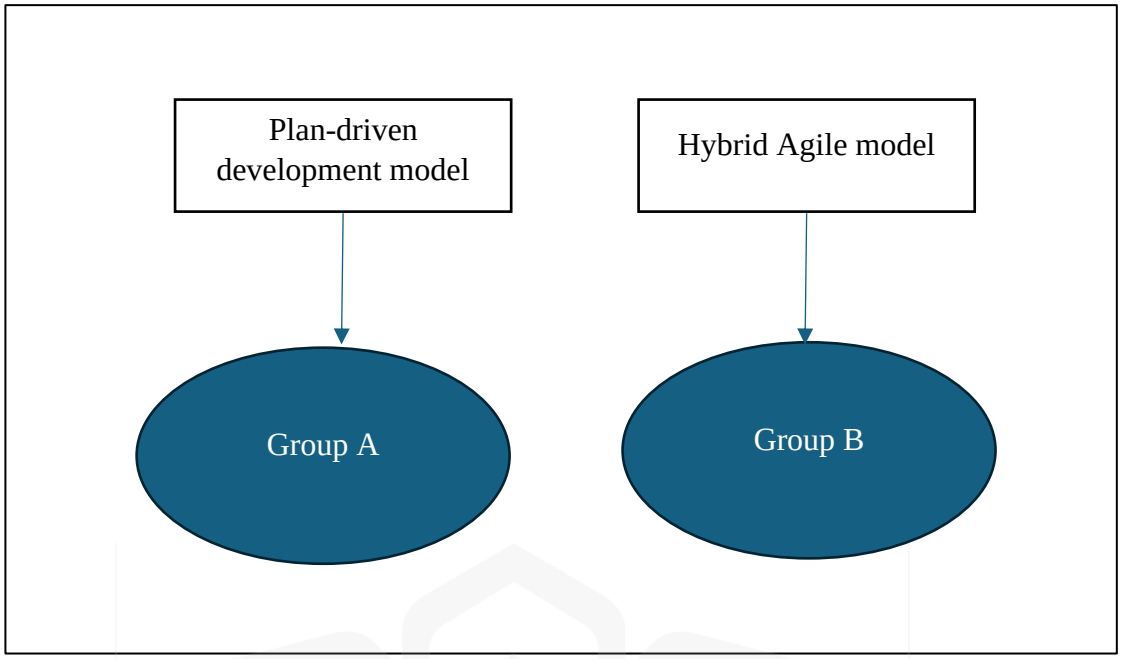



Figure 3.2: The categories of developers with its referring model

The experiment consists of two categories of developers, as depicted in Figure 3.2. These categories consist of software engineers with between 2 to 3 years of working experience. One of the teams will be using plan-driven development model, the Waterfall model as shown in Figure 3.3 while the other one will be using hybrid agile model, WaterScrumFall. Activities from the two teams is recorded throughout the development. The data that will be collected are: Internal Quality Metrics: Lines of Codes (LOC), Lack of Cohesion in Methods (LCOM), Coupling between Objects (CBO), and Average Cyclomatic Complexity (ACC).

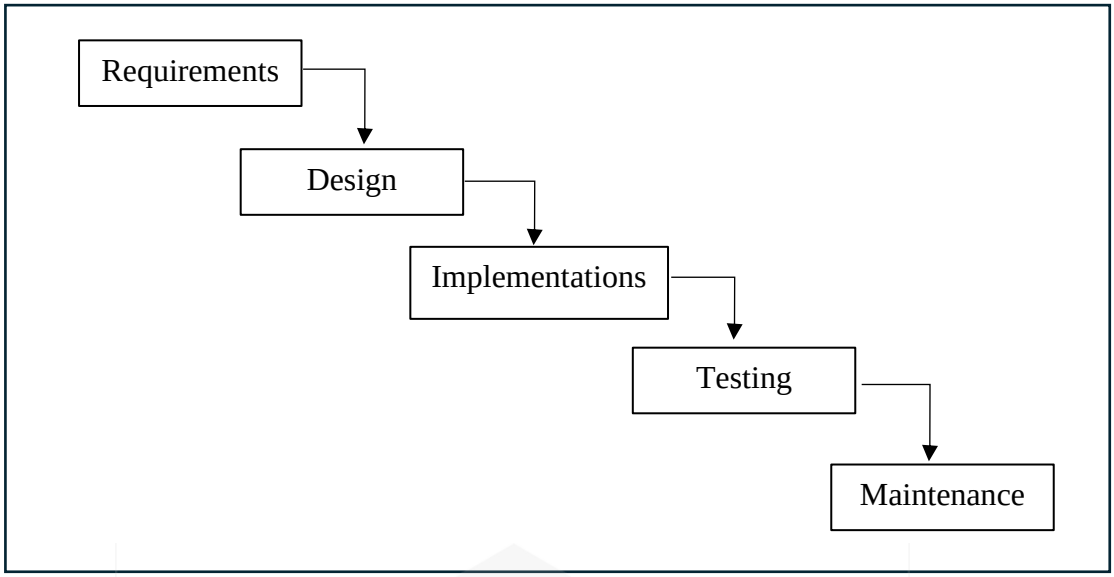



Figure 3.3: The plan-driven development model (Pfleeger, 2001)

The Hybrid Agile model illustrated in Figure 3.4 that used in this research is the model developed by a graduate student of a project sponsored by the Fundamental Research Grant of 2019. This project is the continuity of the first phase of the grant.

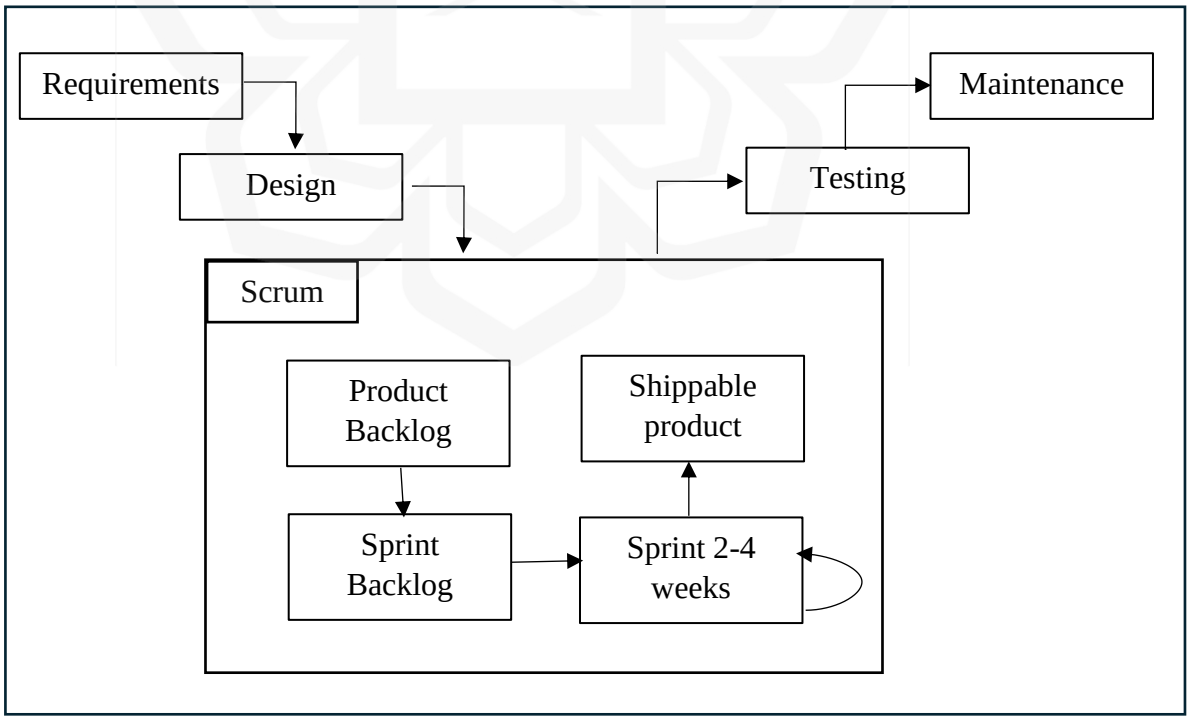



Figure 3.4: The WaterScrumFall model

The project that needs to be built by the software development team is a clinic system. This system is a medium-sized project, but they will build half of the system modules despite this research time and cost constraint. Both teams build components of the clinic system. The programming language that they will be using is PHP, which is the most common language for building a web-based system. Use cases of the system already targeted for the purpose of this experiment. Output from this research will be the quality of the software built by both teams. The output will be tested using PHPMetrics whether the system achieves the standard software quality.

## 3.5 CONCLUSION

This research uses a quantitative approach to evaluate software quality by comparing two development models: the Plan-driven Development model (Waterfall) and the Hybrid Agile model (WaterScrumFall). Two teams of software engineers, each with two to three years of experience, work on developing components of a clinic system, focusing on internal quality metrics such as Lines of Code (LOC), Lack of Cohesion in Methods (LCOM), Coupling Between Objects (CBO), and Average Cyclomatic Complexity (ACC). Using PHP as the programming language, the teams build half of the system's modules due to time and cost constraints. The quality of the developed software is analyzed with PHPMetrics to assess whether it meets standard quality benchmarks. By capturing and comparing the results, the study aims to provide valuable insights into the effectiveness of these development approaches in producing high-quality software.

# CHAPTER FOUR
# RESULTS AND ANALYSIS

## 4.1 INTRODUCTION

This chapter discusses the findings from the experiment conducted involving two development teams using two different methodologies: 1) Plan-driven development model, and 2) Hybrid agile model, . Both models referred by the development teams are illustrated in Figure 3.2 and Figure 3.3 respectively. Both teams develop the same system known as clinic system as described in Chapter 3.1. The following subchapters will discuss the experiment and the results from the data analysis. In the last section, this chapter explains the overall comparison in terms of the metrics stated in Chapter 3.

## 4.2 DATA COLLECTED FROM BOTH MODELS

The data are collected from two development teams which are assigned to the plan-driven development model and hybrid agile model team. The data collected are in the form of source code, which is the complete and working clinic system. There are five metrics collected however, only four metrics to measure software quality is selected and reported in this research. These metrics are chosen because it is the core of the internal quality attributes and PHPMetrics can give a concise and precise measurement of the metrics. The exclusion of WMC from the selected metrics is because there is a limited time frame when doing this research. The four metrics are presented in Table 4.1 below.

Table 4.1: Comparison table between plan-driven development model and Hybrid Agile model

| | Plan-driven development model | Hybrid Agile model |
|---|---|---|
| Average cyclomatic complexity | 2.08 | 1.33 |
| Coupling between Object | 4.49 | 3.14 |
| Lack of cohesion in method | 1.27 | 1.86 |
| Lines of Codes | 811 | 1169 |

## 4.3 DATA ANALYSES

Table 4.1 shows that the plan-driven development model has a higher average cyclomatic complexity than the Hybrid Agile model. The coupling between object is higher in the plan-driven development model than in the Hybrid Agile model. The Hybrid Agile model has a bit more of a lack of cohesion in method than the plan-driven development model. Furthermore, the Hybrid Agile model also produces more lines of codes than the plan-driven development model.

However, to compare the outcomes of the data collected, this research represents the comparison with the threshold for each metrics.

Table 4.2: Comparison of each methodology with suggested threshold

| | Plan-driven development model | Hybrid agile model | Suggested threshold |
|---|---|---|---|
| Average cyclomatic complexity | 2.08 | 1.33 | 10 |
| Coupling between Object | 4.49 | 3.14 | 3 |
| Lack of cohesion in method | 1.27 | 1.86 | 2 |

Lack of Cohesion in Method value is in the range of Ferreira et al. (2012) who defined the cohesion threshold at 0 as good, 1-20 as regular, and cohesion greater than 20 as bad. Moreover, Chidamber & Kemerer (1994) defines as a cohesive class.

Stevens et al. (1974) defined coupling as the measure of the strength of association established by a connection from one module to another. Highly coupled classes are not expected, as it is considered bad design and can lead to difficulty in understanding classes. In addition, higher coupling increases the complexity of the codes. Two classes are coupled if a method of one class uses an attribute or method of the other class (Chidamber & Kemerer, 1991). Benlarbi et al. (2000) derived that the threshold for Coupling between Objects is 5.

Lines of Codes is correlated with Cyclomatic Complexity (Gill & Kemerer, 1991), hence more Lines of Codes means a more complicated project and the complexity will be higher.

McCabe (1976) defined the upper bound for complexity to be 10, and when the complexity exceeded 10 the developer had to either recognise and modularise the sub-functions or redo the software. Moreover, based on the complexity presented in the findings, all projects can be concluded as simple and no reworking is needed. Additionally, Lopez & Habra (2005) stated that most of the methods (95%) have a complexity of 1, which can be considered as simple, and reworking is not required.

The team from the plan-driven development model developed the system of 811 Lines of Code (LOC), with an average Lack of Cohesion in Methods (LCOM) of 1.27, and an average Coupling between Objects (CBO) of 4.49, ranging from a minimum of 1 to a maximum of 15. The Average Cyclomatic Complexity (ACC) is 2.08, with a minimum of 1 and a maximum of 9. The team employing the Hybrid Agile methodology built a system including 1169 lines of code (LOC), with an average Lack of Cohesion in Methods (LCOM) of 1.86 and an average Coupling Between Objects (CBO) of 3.14, ranging from a minimum of 0 to a maximum of 16. The mean of ACC is 1.33, with a minimum of 1 and a high of 4.

Therefore, in can be concluded that the Average Cyclomatic Complexity for both models the Plan-driven development and hybrid is below the recommended threshold of 10, with Plan-driven at 2.08 and Hybrid Agile at 1.33 with hybrid agile proven produce a lower ACC in comparison to Plan-driven development. The low in

ACC indicated that the source codebase that is more comprehensible, debuggable, and extensible, hence enhancing the efficiency and sustainability of software development techniques (Håkansson & Badran, 2016).

In terms of the Coupling Between Objects, the Plan-driven model has a score of 4.49, which is higher than the threshold of 3. On the other hand, the Hybrid Agile model has a score of 3.14, which is closer to meeting the barrier but is still somewhat above it. High Coupling Between Objects denotes a scenario in which classes, modules, or components exhibit significant interdependence. The system's design compromises maintainability, reusability, and testability, leading to challenges in adapting to new requirements or technologies. The situation obstructs simultaneous development, introduces complexities in refactoring, and may lead to a series of changes that become challenging to oversee. To tackle these issues, loose coupling is typically favoured, as it facilitates the creation of more modular, flexible, and maintainable systems (Osbakk, 2015). The lack of cohesion in methods means that both models exhibit values that approach the recommended threshold of 2, with the Plan-driven model registering at 1.27 and the Hybrid Agile model at 1.86. Cohesion metrics assess the degree to which the methods within a class are interconnected and relevant to one another. A cohesive class executes a singular function, whereas a non-cohesive class engages in two or more disparate functions. A class that lacks cohesion might benefit from being reorganised into two or more smaller groups. Thus, both models indicate that the models have no issues on LCOM.

Table 4.2 compares the Plan-driven development model and the Hybrid Agile model against suggested thresholds for three key software metrics. For Average Cyclomatic Complexity, both models (Plan-driven: 2.08, Hybrid Agile: 1.33) perform well below the suggested threshold of 10. In terms of Coupling Between Objects, the Plan-driven model (4.49) exceeds the threshold of 3, while the Hybrid Agile model (3.14) is closer but still slightly above the limit. Lastly, for Lack of Cohesion in Methods, both models show values near the suggested threshold of 2, with Plan-driven at 1.27 and Hybrid Agile at 1.86.

## 4.4 OVERALL COMPARISON

Based on the analyses presented in Section 4.3, the Hybrid Agile team produce more LOC than from the plan driven with the difference of 358 LOC. However, Hybrid Agile model produces lower CBO and less ACC. This indicates that both models generally align with the thresholds, though Hybrid Agile performs better in terms of complexity and coupling.

# CHAPTER FIVE
# CONCLUSION AND FUTURE WORKS

## 5.1 INTRODUCTION

In this thesis, the two experiments were successfully conducted, and the results were taken to answer the research questions as listed in Section 1.3. While Hybrid Agile model is considered as ideal model apart from its model combination, most of the software engineering team never compare the software quality they develop with the existing software development methodology.

In a software project, mentioned earlier in Chapter 1, companies such as software development houses and even software engineering teams will refer to a software development model to ensure their software project is developed and delivered systematically. Two commonly used models in software projects are the plan-driven development model, specifically the Waterfall model, and the Hybrid Agile model, known as WaterScrumFall, which combines elements of the Waterfall model and Scrum. These models have been chosen are based on industry relevance. Many software development houses operate under strict client timelines, making pure Agile difficult to implement due to its flexible and evolving nature. As a result, organizations often prefer plan-driven or Hybrid Agile approaches to balance structure and adaptability. Besides, these models provide contrasting approaches. For example, plan-driven model follows a sequential, well documented process with clearly defined stages, making it suitable for projects with fixed requirements and strict governance while the Hybrid Agile model integrates structured planning with iterative development, aiming to combine the rigor of plan-driven methods with the flexibility of Agile.

This research focuses on the internal quality of a software. The internal quality is chosen to be studied, and the focus is on the source of the software developed. It is expected that the results of this will provide the software engineering team with a view from the internal quality perspective on which model between the plan driven development model, the Waterfall model and the Hybrid Agile model, may lead to a software that is less complexity with a minimal line of codes, low coupling between

objects, and high lack cohesion in methods which resulting in easier to maintain software and lower software maintenance and upgrading costs. The exclusion of WMC from the selected metrics is because there is a limited time frame when doing the research.

Based on the analyses presented in Section 4.4, the Hybrid Agile team produce more LOC than from the plan-driven with the difference of 358 LOC. However, Hybrid Agile model produces lower CBO and less ACC. Hybrid Agile model implements Scrum method that require coding and testing back and forth, which lead to the difference in LOC. Furthermore, Hybrid Agile model getting feedback early from the Scrum method that requires refactoring of the code, causes the code to be less difficult to maintain and turns out low in CBO. Additionally, Hybrid Agile model produces a system that is easier to understand and less risk when modification is needed. As an implication, Hybrid Agile model tend to produce lower in ACC.

Table 5.1: Summary of the research from Research Question with its Research Objective

| Research Question | Research Objective | Output |
|---|---|---|
| Which model between the Plan-driven development model, the Waterfall model and the Hybrid Agile model, WaterScrumFall produces better internal quality software? | To identify the software development lifecycle model between the Plan-driven development model, the Waterfall model and the Hybrid Agile model, WaterScrumFall will produce better internal quality software. | In average, Hybrid Agile model is better than the plan-driven development model. |
| What are the differences in internal quality software metrics score compared with its | To examine the outcome of internal quality metrics for the Plan-driven development | Difference in internal quality metrics score between the Waterfall |

| | | |
|---|---|---|
| threshold for the Plan-driven development model, the Waterfall model and the Hybrid Agile model, WaterScrumFall? | model, the Waterfall model and the Hybrid Agile model, WaterScrumFall with the threshold as the comparison | model and WaterScrumFall model.<br>LOC: 358<br>ACC: 0.75<br>LCOM: 0.59<br>CBO: 1.35 |

Based on Table 5.1, this research summarise the software quality metrics collected from the experiment. Software developers and project managers can choose the best software development methodologies before initiating software development execution to achieve high quality software. Thus, the practitioners can minimize the risks on maintaining the codes.

## 5.2 RESEARCH CONTRIBUTION

This research holds significant value in the field of software engineering as it provides empirical insights into the internal quality of software developed using two widely adopted methodologies: the Plan-Driven model (Waterfall) and the Hybrid Agile model (WaterScrumFall).

By analysing key quality metrics—code complexity, coupling between methods, and cohesion in methods—this study will determine which approach results in more maintainable, efficient, and cost-effective software. The findings offer software developers and project managers practical guidance on choosing the most suitable development methodology to achieve higher software quality with lower maintenance costs.

Ultimately, this research contributes to the broader software development community by bridging the knowledge gap between traditional and hybrid methodologies, enabling organizations to adopt optimal development practices that enhance software quality and sustainability.

## 5.3 CHALLENGES AND LIMITATIONS OF THE RESEARCH

One of the primary challenges of this study is the competence level of developers. The research assumes that all developers have similar skill sets and experience, but variations in technical expertise can significantly influence the quality of the software produced. Developers with more experience may write cleaner, more efficient code, while those with less experience may struggle with complexity, leading to higher coupling and lower cohesion. Since the study does not account for these differences, it is difficult to determine whether the observed results are due to the development methodology itself, or the skill level of the individuals involved. Future research could incorporate a broader range of developer experience levels to assess how this factor impacts software quality.

Another limitation is the narrow scope of the system type used in the study. The research focuses exclusively on a clinic system, which, while useful for comparison, may not fully represent the complexity of other types of software applications. For instance, e-commerce platforms, financial systems, or enterprise-level software often have more intricate dependencies, security concerns, and scalability requirements that could influence how well a particular development model performs. The results from a clinic system may not generalize to these more complex applications. Expanding future research to include multiple types of systems would provide a more comprehensive understanding of how different methodologies impact software quality across various domains.

The small sample size is another critical limitation. With only one developer assigned per methodology, the study lacks statistical significance, making it challenging to draw conclusive insights. The software quality metrics observed could be more reflective of individual performance rather than the inherent advantages or disadvantages of the Plan-driven or Hybrid Agile methodologies. A more robust approach would involve testing the methodologies with a larger sample of 20 to 30 developers per model, ensuring that the results are more reliable and applicable to real-world software development teams. Additionally, incorporating team-based projects rather than individual assignments could provide insight into the collaborative aspects of these methodologies, which play a crucial role in large-scale software development.

## 5.4 FUTURE WORK

To provide better understanding to the implications of the results, future studies could address the competence level of developers who are doing the development of a system. The competence level can affect the completion rate of the software project (Licorish et al., 2022). Thus, it is important to consider for the competence level of the developers.

Furthermore, future work should consider using other type of system than the clinic system. For example, an e-commerce system that used by nationwide. This is to ensure that whether Hybrid Agile model or the plan-driven development model is suitable for the development of a complex system like the e-commerce system that is using more database tables.

Lastly, future studies should consider involving a wide range of developers rather than just one for each of the software development models. A suggested range could be 20-30 developers for both the Hybrid Agile model and the plan-driven development model. This will strengthen the results by allowing for comparisons among numerous developers, thus providing a more robust hypothesis and statistical analysis.

# REFERENCES


Agile Alliance. (2015a, June 29). *Agile Manifesto for Software development*. Agile Alliance. https://www.agilealliance.org/agile101/the-agile-manifesto/

Agile Alliance. (2015b, June 29). *What is Agile Software Development?* Agile Alliance. https://www.agilealliance.org/agile101/

Agile Alliance, Hartman, B., Griffiths, M., Rothman, J., Fewell, J., Kauffman, B., Matola, S., & KiwiHoria. (2017, January 4). *What is Hybrid Agile, Anyway?* Agile Alliance. https://www.agilealliance.org/what-is-hybrid-agile-anyway/

Albeladi, A., Abdalkareem, R., Agwaeten, F., Altoum, K., Bennis, Y., & Nasereldine, Z. (2014). *Toward Software Measurement and Quality Analysis of MARF and GIPSY Case Studies a Team 13 SOEN6611-S14 Project Report*.

Alenezi, M. (2021). Internal Quality Evolution of Open-Source Software Systems. *Applied Sciences*, *11*(12), 5690. https://doi.org/10.3390/app11125690

Almeida, F., & Bálint, B. (2024). Approaches for Hybrid Scaling of Agile in the IT Industry: A Systematic Literature Review and Research Agenda. *Information*, *15*(10), 592. https://doi.org/10.3390/info15100592

Alomar, E. A., Wang, T., Raut, V., Mkaouer, M. W., Newman, C., & Ouni, A. (2022). Refactoring for reuse: an empirical study. *Innovations in Systems and Software Engineering*, *18*(1), 105–135. https://doi.org/10.1007/s11334-021-00422-6

Al-Qutaish, R. E. (2010). Quality models in software engineering literature: an analytical and comparative study. *Journal of American Science*, *6*(3), 166–175.

Alsaqqa, S., Sawalha, S., & Abdel-Nabi, H. (2020). Agile Software Development: Methodologies and Trends. *International Journal of Interactive Mobile Technologies (IJIM)*, *14*(11), 246. https://doi.org/10.3991/ijim.v14i11.13269

Barnard, E. M. (2006). 4.1.2 Project-driven adaptation of software life cycle model. *INCOSE International Symposium*, *16*(1), 509–522. https://doi.org/10.1002/j.2334-5837.2006.tb02761.x

Beck, K., & al., E. (2001). *Manifesto for Agile Software Development, (URL http://agilemanifesto.org/)*.

Bender RBT Inc. (2003). *Systems Development Lifecycle: Objectives and Requirements*. Bender RBT Inc.

Benlarbi, S., El Emam, K., Goel, N., & Rai, S. (2000). Thresholds for object-oriented measures. *Proceedings 11th International Symposium on Software Reliability Engineering. ISSRE 2000*, 24–38.

Bhandari, P. (2020, June 12). *What is quantitative research?* Scribbr. https://www.scribbr.com/methodology/quantitative-research/

Boehm, B. (2002). Get ready for agile methods, with care. *Computer*, *35*(1), 64–69. https://doi.org/10.1109/2.976920

Butgereit, L. (2018). Five Lessons Learned Using Water-Scrum-Fall in South Africa. *2018 International Conference on Multidisciplinary Research*, *2018*, 175–182. https://doi.org/10.26803/MyRes.2018.14

Cavano, J. P., & McCall, J. A. (1978). A framework for the measurement of software quality. *Proceedings of the Software Quality Assurance Workshop on Functional and Performance Issues* -, 133–139. https://doi.org/10.1145/800283.811113

Chidamber, S. R., & Kemerer, C. F. (1991). Towards a metrics suite for object oriented design. *Conference Proceedings on Object-Oriented Programming Systems, Languages, and Applications*, 197–211.

Chidamber, S. R., & Kemerer, C. F. (1994). A metrics suite for object oriented design. *IEEE Transactions on Software Engineering*, *20*(6), 476–493.

Costa, A. A. M., Ramos, F. B. A., Valadares, D. C. G., Albuquerque, D. W., Filho, E. D., Gomes, A. B., Perkusich, M. B., & Oliveira de Almeida, H. (2022). Disciplined Teams vs. Agile Teams. In *Research Anthology on Agile Software, Software Development, and Testing* (pp. 40–55). IGI Global. https://doi.org/10.4018/978-1-6684-3702-5.ch003

Ferreira, K. A. M., Bigonha, M. A. S., Bigonha, R. S., Mendes, L. F. O., & Almeida, H. C. (2012). Identifying thresholds for object-oriented software metrics. *Journal of Systems and Software*, *85*(2), 244–257. https://doi.org/10.1016/j.jss.2011.05.044

Garg, A., Kumar Kaliyar, R., & Goswami, A. (2022). PDRSD-A systematic review on plan-driven SDLC models for software development. *2022 8th International Conference on Advanced Computing and Communication Systems (ICACCS)*, 739–744. https://doi.org/10.1109/ICACCS54159.2022.9785261

Gill, G. K., & Kemerer, C. F. (1991). Cyclomatic complexity density and software maintenance productivity. *IEEE Transactions on Software Engineering*, *17*(12), 1284–1288.

Håkansson, J., & Badran, S. (2016). *Evaluating cyclomatic complexity on functional JavaScript*.

Husein, S., & Oxley, A. (2009). A Coupling and Cohesion Metrics Suite for Object-Oriented Software. *2009 International Conference on Computer Technology and Development*, 421–425. https://doi.org/10.1109/ICCTD.2009.209

Imani, T. (2017). Does a Hybrid Approach of Agile and Plan-Driven Methods Work Better for IT System Development Projects? *International Journal of Engineering Research and Applications*, *07*(03), 39–46. https://doi.org/10.9790/9622-0703043946

International Organization for Standardization. (2023). *ISO/IEC 25010:2023: Systems and software engineering — Systems and software quality requirements and evaluation (SQuaRE) — System and software quality models*. Geneva, Switzerland: ISO. https://www.iso.org/standard/78176.html

International Standard Organization. (1991). *ISO/IEC 9126: Information Technology - Software Product Evaluation - Quality characteristics and guidelines for their use*. ISO/IEC.

Jarzebowicz, A., & Weichbroth, P. (2021). A Qualitative Study on Non-Functional Requirements in Agile Software Development. *IEEE Access*, *9*, 40458–40475. https://doi.org/10.1109/ACCESS.2021.3064424

Kavitha, D., & Neelu, L. (2021). Estimation of software quality parameters for hybrid agile process model. *SN Applied Sciences, August 2020*. https://doi.org/10.1007/s42452-021-04305-0

Klünder, J., Hebig, R., Tell, P., Kuhrmann, M., Nakatumba-Nabende, J., Heldal, R., Krusche, S., Fazal-Baqaie, M., Felderer, M., Bocco, M. F. G., Küpper, S., Licorish, S. A., Lòpez,

G., McCaffery, F., Top, Ö. Ö., Prause, C. R., Prikladnicki, R., Tüzün, E., Pfahl, D., … MacDonell, S. G. (2021). *Catching up with Method and Process Practice: An Industry-Informed Baseline for Researchers*. https://doi.org/10.1109/ICSE-SEIP.2019.00036

Kuhrmann, M., Diebold, P., Münch, J., Tell, P., Garousi, V., Felderer, M., Trektere, K., McCaffery, F., Linssen, O., Hanser, E., & Prause, C. R. (2017). Hybrid software and system development in practice: waterfall, scrum, and beyond. *Proceedings of the 2017 International Conference on Software and System Process*, 30–39. https://doi.org/10.1145/3084100.3084104

Lépine, J.-F. (2013). *PhpMetrics*. GitHub. https://phpmetrics.github.io/website/

Licorish, S. A., Galster, M., Kapitsaki, G. M., & Tahir, A. (2022). Understanding students' software development projects: Effort, performance, satisfaction, skills and their relation to the adequacy of outcomes developed. *Journal of Systems and Software*, *186*, 111156. https://doi.org/10.1016/j.jss.2021.111156

Lopez, M., & Habra, N. (2005). Relevance of the cyclomatic complexity threshold for the java programming language. *SMEF 2005*, 195.

Marinho, M., Noll, J., Richardson, I., & Beecham, S. (2019). Plan-Driven Approaches Are Alive and Kicking in Agile Global Software Development. *2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, 1–11. https://doi.org/10.1109/ESEM.2019.8870168

McCabe, T. J. (1976). A complexity measure. *IEEE Transactions on Software Engineering*, *4*, 308–320.

Milić, M., & Makajić-Nikolić, D. (2022). Development of a Quality-Based Model for Software Architecture Optimization: A Case Study of Monolith and Microservice Architectures. *Symmetry*, *14*(9), 1824. https://doi.org/10.3390/sym14091824

Munawar, K., & Naveed, M. S. (2022). The Impact of Language Syntax on the Complexity of Programs: A Case Study of Java and Python. *International Journal of Innovations in Science and Technology*, *4*(3), 683–695. https://doi.org/10.33411/IJIST/2022040310

Osbakk, M. (2015). *Architectural Design of Loosely Coupled Services: A Case Study*.

Petersen, K., & Wohlin, C. (2010). The effect of moving from a plan-driven to an incremental software development approach with agile practices. *Empirical Software Engineering*, *15*(6), 654–693. https://doi.org/10.1007/s10664-010-9136-6

Petersson, B., & Zhang, S. (2013). *An Approach to Improve Quality of Software Using Metrics and Technical Debt-A case study within Model-Driven Environment*.

Pfleeger, S. L. (2001). *Software Engineering: Theory and Practice* (2nd ed.). Prentice Hall PTR.

Rahim, S., Chowdhury, A. E., Nandi, D., & Rahman, M. (2018). ScrumFall: A Hybrid Software Process Model. *International Journal of Information Technology and Computer Science*, *10*(12), 41–48. https://doi.org/10.5815/ijitcs.2018.12.06

Rizwan Jameel Qureshi, M., & Hussain, S. A. (2008). An adaptive software development process model. *Advances in Engineering Software*, *39*(8), 654–658. https://doi.org/10.1016/j.advengsoft.2007.08.001

Royce, W. W. (1987). Managing the development of large software systems: concepts and techniques. *Proceedings of the 9th International Conference on Software Engineering*, 328–338.

Saini, N., Kharwar, S., & Agrawal, A. (2014). A Study of significant software metrics. *International Journal of Engineering Inventions*, *3*(12), 1–7.

Singh, B., & Kannojia, S. P. (2013). A Review on Software Quality Models. *2013 International Conference on Communication Systems and Network Technologies*, 801–806. https://doi.org/10.1109/CSNT.2013.171

Stavtsev, M., & Bugayenko, Y. (2024). *Evaluating the Dependency Between Cyclomatic Complexity and Response For Class*.

Stevens, W. P., Myers, G. J., & Constantine, L. L. (1974). Structured design. *IBM Systems Journal*, *13*(2), 115–139.

Vijayasarathy, L. R., & Butler, C. W. (2015). Choice of software development methodologies: Do organizational, project, and team characteristics matter? *IEEE Software*, *33*(5), 86–94.

Walrave, B., Dolmans, S., van Oorschot, K. E., Nuijten, A. L. P., Keil, M., & van Hellemond, S. (2022). Dysfunctional Agile–Stage-Gate Hybrid Development: Keeping Up Appearances. *International Journal of Innovation and Technology Management*, *19*(03). https://doi.org/10.1142/S0219877022400041

Włodarski, R., Falleri, J.-R., & Parvéry, C. (2021). Assessment of a Hybrid Software Development Process for Student Projects: A Controlled Experiment. *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET)*, 289–299. https://doi.org/10.1109/ICSE-SEET52601.2021.00039

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., & Wesslén, A. (2024). *Experimentation in Software Engineering*. Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-662-69306-3

Yahya, N., & Sarah Maidin, S. (2023). Hybrid agile development phases: the practice in software projects as performed by software engineering team. *Indonesian Journal of Electrical Engineering and Computer Science*, *29*(3), 1738. https://doi.org/10.11591/ijeecs.v29.i3.pp1738-1749

# APPENDIX A

# SOFTWARE REQUIREMENTS SPECIFICATIONS (SRS)

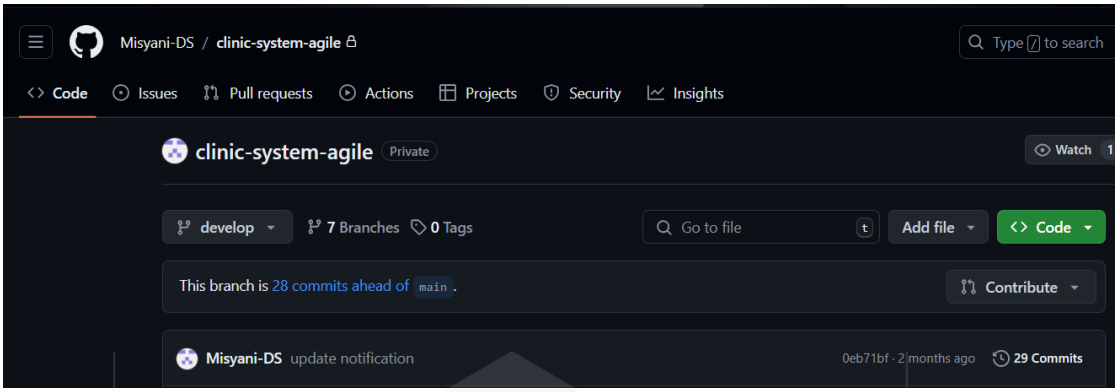
| REQ-ID | Requirement Description | Users involved | Remarks |
|---|---|---|---|
| **SYSTEM FEATURE 1.0 USER AUTHENTICATION** | | | |
| RF1-001 | The system shall provide the users with the ability to access the system by login using username and password. | All users | What are the role differences between patient and doctor? |
| RF1-002 | When the user is authenticated, the system should be able to verify each activity made by the user. | All users | |
| | | | |
| **SYSTEM FEATURE 2.0 USER REGISTRATION** | | | |
| RF2-001 | The system shall provide registration for patients. | Patient | - Public Page<br>- Field list:<br>1) Name<br>2) Email<br>3) Phone number<br>4) Password<br>5) Confirm Password |
| RF2-002 | The system must notify patient upon successful registration. | Patient | To verify the email used. |
| RF2-003 | The system shall provide user registration for administrator. | Admin | - After login |
| **SYSTEM FEATURE 3.0 APPOINTMENT** | | | |
| RF3-001 | The user can add appointments on a future date. | Patient | Field list:<br>1) Title<br>2) Date<br>3) Time<br>4) Remark |
| RF3-002 | The user can edit an existing | Patient | |

| | | | |
|---|---|---|---|
| | appointment. | | |
| RF3-003 | The user can cancel/remove an existing appointment. | Patient | |
| RF3-004 | The user can view the added appointment. | Patient | |
| RF3-005 | The system must notify patients of an upcoming appointment. | Patient | |
| RF3-006 | The system shall provide access to view all appointments created by patients. | Admin | |
| RF3-007 | The system shall provide access to edit any appointments created by patients. | Admin | |
| RF3-008 | The system shall provide access to cancel/remove any appointments created by patients. | Admin | |
| RF3-009 | The system must notify the patients if the appointment is cancelled/removed by the admin/doctor | Patients | |
| **SYSTEM FEATURE 4.0 CONSULTATION** | | | |
| RF4-001 | The system shall provide remarks for doctors on the medication. | Doctor | Field list:<br>1) Appointment ID<br>2) Doctor's remark |
| RF4-002 | The user can view the doctor's remark from the successful appointment. | Patient | After login and consultation |
| **SYSTEM FEATURE 5.0 PHARMACY** | | | |
| RF5-001 | The admin can view the list of medication provided by the doctor. | Admin | After login and consultation |
| RF5-002 | The system shall calculate the total cost of the medication. | Admin | Each medication has a cost value. |

| | | | |
|---|---|---|---|
| RF5-003 | The system shall generate an invoice for the consultation with the medication. | Admin | Cost of the consultation and medication (if any) |
| **SYSTEM FEATURE 6.0 PAYMENT** | | | |
| RF6-001 | The system shall provide access to payment gateway for patient | Patient | After pharmacy dispensary |
| RF6-002 | The system must notify the admin on the payment received from the patient | Admin | After patient made payment |

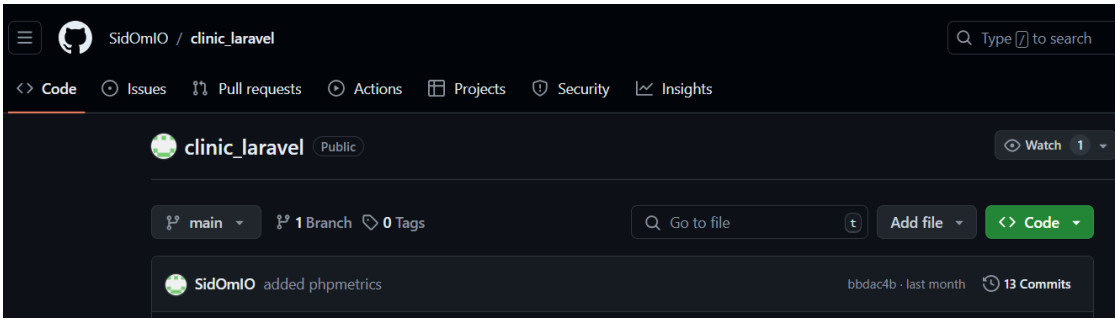
# APPENDIX B

# GITHUB FROM HYBRID AGILE DEVELOPMENT



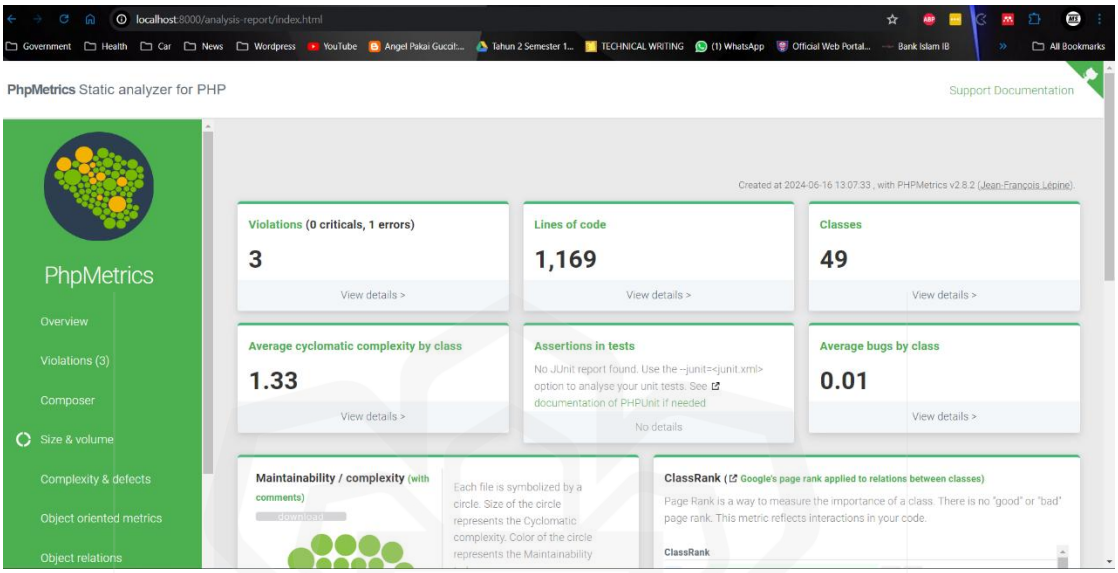
# APPENDIX C

# GITHUB FROM PLAN-DRIVEN DEVELOPMENT



# APPENDIX D

# PHPMETRICS RESULT FOR HYBRID AGILE DEVELOPMENT



# APPENDIX E

# PHPMETRICS RESULT FOR PLAN-DRIVEN DEVELOPMENT