Assessor name should be included as S. M. D. T. H. Dias

Introduction

The link between software complexity and maintainability has long piqued interest in the IT industry due to the critical role it plays in developing dependable and high-quality software solutions. Software complexity is frequently defined in terms of how difficult it is to understand the codebase. The maintainability of the code has a complicated relationship with the complexity of the codebase. While simpler codebases generally have higher maintainability, very sophisticated codebases frequently have worse maintainability. Numerous factors, such as code organization, modularity, documentation, design and architecture, testing, and automation, have a significant impact on the link between complexity and maintainability. Despite prior research on the subject, it aims to adopt a distinct viewpoint and investigate some of the possibilities that have not yet been covered.

Modern software systems usually include a large number of interrelated components and stakeholders, which adds to their inherent complexity and serves as a substantial development roadblock. This complexity may have a substantial impact on the system's maintainability, making future updates and upgrades more challenging. As a result, a thorough analysis of the connection between maintainability and software complexity is necessary. Knowing how these two things are related can help in the development of strategies that result in software solutions that are simpler to maintain, more efficient and easier to use.

The major goal of this research article is to investigate the connection between software complexity and maintainability. In particular, the study will look at how factors like code size, architectural complexity, design patterns, and code smells affect maintainability. Additionally, the study will look at how various software development tools and practices, such as code reviews, refactoring, and automated testing can be used to improve the ability to maintain complex software systems. The study's goal is to offer perceptions and suggestions that help guide software development procedures and aid in the creation of more maintainable software solutions.

The findings of this study will have a substantial impact on the software development industry since they will make it easier for decision-makers to weigh complexity and maintainability in their choices. The research will offer insights into the best techniques for maintaining intricate yet manageable software systems, assisting in the creation of more solid and dependable software solutions. The results of the study will be useful to software developers and project managers who will be better able to weigh the advantages and disadvantages of software complexity and maintainability and decide what to do with their software development projects. In the long run this research will advance software engineering, enhancing the quality, and durability of software systems.

Due to the expanding complexity of software, which presents considerable obstacles during maintenance, the function of the comparison between software complexity and maintainability in the software development industry is regarded as significant. This study emphasizes the significance of software complexity during the development process and offers developers insight into the crucial criteria for lowering code complexity and consequently, maintenance expenses. The industry can benefit from better software maintainability by having a competitive edge. By determining the correlation between software complexity and maintainability; efficient testing procedures may be created. Further studies in this field may be advanced by this research and it may contribute to the body of knowledge among researchers.

The software development business faces a sizable issue from complications brought on by software complexity during maintenance. This is

a result of developers' propensity to place a higher value on programming languages, frameworks, and technical stacks during the development process than on the complexity of the software. Software maintainability may suffer if this important area is ignored. The goal of this research paper is to examine the connection between software complexity and maintainability by reviewing prior studies in the area, assessing various software complexity metrics and their effects on maintainability, using actual software projects as examples and case studies, and finally presenting suggestions and guidelines for increasing software maintainability by reducing complexity. Software systems' quality and endurance can be increased by taking software complexity into account during the development process, which will help to reduce maintenance problems.

Literature Review

The purpose of this review is to perform a detailed comparison analysis on the various studies that have been done on the topic of software complexity and software maintainability, along with the study that we have conducted.

Our aim is to utilize the data presented by previous studies, compare the data we have gathered using numerous methods and incorporate them together in order to originate a more thorough and accurate outcome for the research. The main aspects of several selected studies that were taken as references in our research is as follows.

When considering the metrics referred to make conclusions, many researchers have collected the source code of both older and newer versions of software compared the time the research was conducted, to identify the precise modifications that have been made and how such modifications have affected the maintainability of software by testing the code segments to measure the accuracy of the output.

According to research done by several university students in USA, the best method to reduce the software complexity and increase the ease of software maintainability is using good coding practices such as meaningful comment and variable declarations and reusing code to avoid redundancy. The importance of using good coding practices had been highlighted in the conversations that had taken place between our team and several industry experts as well, but there were also multiple other instances that were brought to light as well, including using proper code formatting, proper file naming, organization of code files and folders etc. Therefore, all the ideas that were newly introduced were incorporated into our study with the purpose of including all possible solutions in favor of reducing software complexity and increasing software maintainability

Methodology

Getting data from a variety of sources was essential for drawing conclusions about the connection between software complexity and maintainability. A variety of approaches, such as literature reviews, online databases, scholarly publications, and industry reports, were used to gather pertinent data. These resources provide insightful analysis of program complexity metrics and recommended procedures for software system maintenance. To further understand how these topics are taught in academic contexts and how they may be improved, interviews with students studying software engineering were also performed. Using a range of research approaches, this study was able to provide an in-depth investigation of the connection between software complexity and maintainability. To create more productive and efficient software development techniques in the industry, it also emphasizes the importance of teaching software engineering students on these concepts.

In the current study, forms were used to collect data about participants' experiences in the Information Technology (IT) industry, with a

focus on the complexity and maintainability of code. The questionnaire was composed of several simple-to-answer questions, the first of which focused on the participants' prior IT experience. (Writing Survey Questions | Pew Research Center)The interaction of the participants with the code and their opinions on complexity and maintainability was then the focus of the questions. Participants were questioned toward the end of the form about the strategies they used to deal with the code complexity. In addition the participants had the chance to voice their perspectives on how complexity and maintainability relate to one another. Forms were considered an effective way to collect data due to their widespread use as a tool for information gathering from people. The study's findings are reported in this essay along with some discussion of the implications for further investigation into the complexity and maintainability of IT systems. We investigated the impact of this program complexity on the software development process, including interviews with experienced software engineers. After speaking with multiple other software engineers, our team was able to arrange a discussion with one of the top experts in this IT field. The importance of factoring in system software complexity was brought up during this discussion. We identified the best procedures for businesses to adopt in order to avoid issues brought on by complicated code. In addition to that, developers said their personal observations of how code complexity can be affected the development process. During this discussion and interview, the idea of maintainability in connection to code complexity was also analyzed, looking at how those two are related. Finally, the potential consequences of different industrial methods for assessing code complexity in software systems were also questioned.

As part of their conversation with Mr. D.B. Wijesinghe, a seasoned business expert with more than three years of software maintenance expertise; our team members learned important information that helped them comprehend the difficulties involved in software maintenance. We created a list of questions and Mr. Wijesinghe presented a number of crucial ideas that are crucial to our investigation. He outlined a number of difficulties that arise during maintenance while examining code to ascertain its functionality. The issues experienced during maintenance have been attributed primarily to poor coding methods. It was emphasized how crucial it is for each developer to have a comprehensive understanding of software maintenance and the efforts that may be taken to reduce its complexity. Mr. Wijesinghe noted that it is essential to make code readable for others in order to make modifications or maintenance easier. It was found that typically, illogical variables, poor documentation and poor commenting make it difficult for maintainers to analyze the code, resulting in delays. For the purpose of considerably; simplifying software maintenance the significance of reducing such circumstances was underlined.

In order to look at the relationship between software complexity and maintainability, a study was done with university students who were just starting out in the field of software development. Since those students had a different viewpoint on software complexity and maintainability than professionals in the business, they were thought to be the best group to learn from. Since we are software engineering students we could easily relate with these students because as third-year students who are studying software engineering, we were also participating in software development projects. During the study the students were questioned about the techniques employed to lessen the complexity of the software, such as the usage of meaningful variables and adding commenting, as well as if complex functions were divided into smaller sub-problems to lessen the complexity.(readability - Hiding away complexity with sub functions - Software Engineering Stack Exchange) They were also questioned about their experiences with software maintenance and the difficulties they encountered as a result of the lack of attention paid to lowering software complexity during

development. Due to the fact that these students were also engaged in software maintenance, they were able to get invaluable insights into the challenges encountered during this stage. Through this study, a deeper knowledge of how software complexity is affecting to software maintainability, particularly in the initial stages of software development was attained.

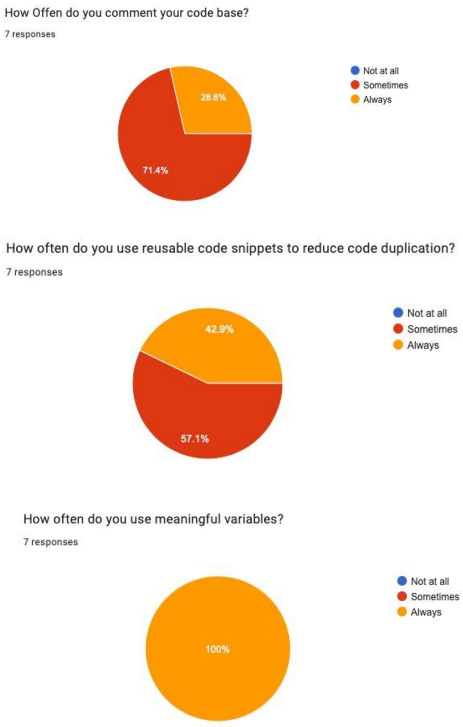
Results

In a study on the association between code complexity and maintainability, form-based surveys, a highly effective technique, were used. The poll reached more people than was previously thought possible, including students, interns, and full-time employees. All participants, regardless of their level of professional experience, agreed on the importance of minimizing code complexity. Furthermore, the vast majority of participants underlined the significance of adding comments to code to help other developers understand the code base. As the complexity of the development increases, comments were thought vital for developers to review their own code. The analysis demonstrated that the use of meaningful variable names is a de facto industry standard. Leveraging code snippets could assist reduce complexity and the time and effort needed to write new code, according to 66% of respondents, who also agreed that this could help with code reuse. Cyclomatic complexity, lines of code (LOC), and function point analysis were found to be the most often utilized metrics for measuring code complexity by industry professionals, according to the survey. The majority of survey participants in the study generally agreed with the idea that less complex code is easier to maintain.

In a discussion about software development, the importance of well-structured code with different methods and classes was emphasized by a developer. Furthermore, the proverb "Code should have clear purpose" was quoted, and the concept of single responsibility, which requires each class or function to have a specific role or accountability, was explained. (SOLID: The First 5 Principles of Object Oriented Design | DigitalOcean) The potential for methods or classes to have multiple functionalities was acknowledged by the developer, and the need for comments to minimize code complexity was emphasized. In team-based software development, comments play a crucial role in enabling the code to be understood by every team member, necessary modifications to be identified, and effective collaboration to occur. (How to Effectively Collaborate on a Development Project (4 Tips) - ManageWP) Additionally, the use of reasonable variable names was recommended by the developer to further simplify the codebase. In summary, these practices - well-structured code, clear purpose, single responsibility, comments, and meaningful variable names - promote clarity, readability, and maintainability in software development.

The process of figuring out code and its functionalities, especially when the original software engineers were not involved in the system's development, is time-consuming and difficult, as we recently learned from a software maintenance engineer with whom we had a conversation. To solve this, it is thought crucial to follow coding best practices that simplify code by making it easier to read and understand. The importance of this code simplification may be seen in the fact that it makes it easier for fresh software engineers to alter or correct faults during maintenance chores. Maintaining complex code can be difficult for even seasoned software engineers. It is further evidence of the need for code simplification because altering a small portion of code in a complicated system can have a variety of unintended repercussions. The problem of software maintenance is made even more onerous by the fact that keeping adequate documentation is tough when working with complicated systems. The conversation revealed that all major aspects of software maintenance, including debugging, testing, understanding code, and documentation, become increasingly challenging when code complexity increases.

The study done on Information Technology at university students produced useful insights into the ways to reduce software complexity and the difficulties encountered during program maintenance. These students often follow the best coding standards because it is necessary for their coursework when creating software for academic objectives. It is simpler for other developers to understand the codebase since students who are working on a group project frequently use comments. These students understand the value of decomposing complicated functionality into smaller sub-functions in addition to appropriate variable naming and other recommended practices for coding. However, it was discovered that assigning software maintenance to those who were not involved in the creation process can make it more difficult, particularly if the product is complicated and best coding techniques weren't followed. This can make the maintenance stage take up more time. From the study we conducted through the survey, the below mentioned results were obtained about software complexity and software maintenance.



Discussion

To describe the findings of our study in this discussion part, a variety of methods were used. An appropriate explanation is given in the part that follows. This analysis was a significant contribution to our discussion. In order to learn more about their extensive professional expertise in our field, consultations were made with licensed software developers. In the part that follows, the conclusions drawn from these data are also thoroughly covered.

Code comments and complexity reduction make the codebase simpler for maintenance engineers to understand and evaluate, which improves maintenance effectiveness.(https://users.cs.utah.edu/~germain/PPS/Topics/commenting.html#:~:text=Commenting%20involves%20placing%20Human%20Readable,that%20other%20people%20will%20use.) When the software system reaches the maintenance stages, it will be useful when relevant variable names are used. The significance of adhering to version control systems was discovered to be essential in the study of industry professionals. An advantage mentioned was the ability to track down the changes made to the codebase during maintenance. It allows developers to effectively manage and organize code versions. (https://reqtest.com/requirements-blog/what-are-benefits-of-version-control/) Developers also reuse code fragments and components to avoid code duplications, to reduce lines of code and prevent code bloat, which is defined as unnecessarily lengthy, resource-intensive code. Less time is spent on maintaining the codebase when this technique is applied.(The Art of Clean Code: Naming, Commenting, and Code Organization | Fajarwz)

From the study conducted it was identified that several students struggled to understand how software complexity affects software maintainability. In order to produce a new generation of software engineers that are capable of producing high-quality software products, it is

critical to prioritize software engineering education that highlights the significance of readability, maintainability, and software complexity metrics. (What Should We Teach New Software Developers? Why? | January 2010 | Communications of the ACM)

In order to increase the maintenance that follow up with the reduction in software complexity, different complexity measuring methods are used. One of the popular techniques for assessing software complexity and code's difficulty is Halstead complexity. This weighs program length, vocabulary length, volume, and difficulty.

This helps to identify portions of the code that are complex and challenging to maintain.(Software Engineering | Halstead's Software Metrics - javatpoint). Another techniques used is the Cyclomatic Complexity, quantifies the number of linear independent paths within a code. (Cyclomatic Complexity | Calculation | Examples | Gate Vidyalay) The software program becomes easier to understand and maintain when the cyclomatic complexity value is low

V(g) = e – n + 2 (e->number of edges, n->number of nodes)

V(g) = bd + 1 (bd-> number of binary decisions)

From the above discussion, we were able to analyze the relationship between software complexity and maintainability. After going through the data gathered from the perspective of software complexity and the data gathered from the perspective of software maintainability, we were able to emerge with an adequately comprehensible image of what practices should be and should not be implemented to contribute to the strengthening the maintainability and the depletion of the complexity of software.

Conclusion

The relationship between software complexity and maintainability was examined in this study utilizing a variety of data collection techniques. To gain a thorough grasp of the experiences and viewpoints of industry professionals and software development students on these themes, surveys and interviews with experts in the field were undertaken. The results showed striking commonalities across the many data sources, highlighting the significance of software developers paying strict attention to the complexity of their code to increase its maintainability. It was argued that assuring long-term maintainability may be achieved by simplifying the code. The knowledge acquired from this research is especially beneficial for students studying software engineering because it offers insightful advice for their potential jobs in the sector. Overall, the significance of addressing

- η_1 = the number of distinct operators
- η_2 = the number of distinct operands
- N_1 = the total number of operators
- N_2 = the total number of operands

From these numbers, several measures can be calculated:

- Program vocabulary: $\eta = \eta_1 + \eta_2$
- Program length: $N = N_1 + N_2$
- Calculated estimated program length: $\hat{N} = \eta_1 \log_2 \eta_1 + \eta_2 \log_2 \eta_2$
- Volume: $V = N \times \log_2 \eta$
- Difficulty : $D = \frac{\eta_1}{2} \times \frac{N_2}{\eta_2}$
- Effort: $E = D \times V$

software complexity in the pursuit of achieving better maintainability of software systems is highlighted by this study.

Abstract

This paper aims to conduct a systematic analysis of the relationship between software complexity and software maintainability. The research uses a variety of methods, including surveys and questionnaires, in order to gather data of higher accuracy. Viewpoints and opinions of industry professionals, entry-level employees, and university students were also taken as inputs to further pursue the topic. The outcome of the study depicts a significant negative correlation between the complexity and maintainability of software. This research has implications for personnel in the IT industry to implement measures toward a significant decline in software complexity and the elevation of software maintainability.