University of Salford MANCHESTER

Software Quality Management

Name: Jack Travis

Student ID: @00609342

Username: AGD991

Assessment Tile: Assignment 2: (software reuse)

CRN: 35537

Deadline Date: 21st April 2023

Word Count: 2856

EXECUTIVE SUMMARY

Software reuse is the process of reusing assets, either internally or through third-party means. This may be accomplished by reusing whole applications or systems, or by reusing components, objects or functions.

Reusing software is very common within the industry, due to research sufficiently suggesting that it can improve software quality, since already created software has already been tested and therefore more likely to be stable, and can reduce the total development time, since some features or components now only need to be integrated instead. Specialists in a company may also be better utilized, focusing on developing reusable software for future convenience.

However, there are some challenges involved with implementing software reuse. The aim of this report is to identify such challenges, analyse suggested solutions for their ability to deal with the aforementioned challenges, and to discuss and recommend a way in which Valtech, a company that creates Information Systems for the Healthcare sector, can utilize software reuse.

This was accomplished by reviewing scientific publications on this field, found on the IEEE and ScienceDirect databases. A focus on using publications during or surpassing 2019 was taken, to provide relevant and up-to-date information and recommendations.

The conducted research found that the three main challenges for software reuse, in terms of software quality and productivity, were related to security, maintainability and the organisation of the repository. These challenges were further explored to identify why that pose such a challenge, and what effects they can have if not handled correctly.

These challenges were used to evaluate many proposed solutions for implementing software reuse, considering how successful each is at alleviating said challenges. The report concludes and recommends that Valtech should use a structured implementation of software reuse, specifically a model-driven approach, which was deemed the most suitable for the nature of their work.

TABLE OF CONTENTS

1 INTRODUCTION

Software reuse is a development practice whereby utilizing existing resources is favoured, rather than developing said resources from scratch. (Gaber, 2023) This may be done at an object or component level or may involve reusing entire systems.

Likewise, any development of code that is identified as something that might be reused at a later point, is developed with software reuse in mind, creating the necessary interfaces and documentation to allow this to happen.

Successfully using software reuse should allow for developers to focus on other areas, where software reuse isn't applicable or currently possible, with the aim of reducing the overall development time and associated cost. (Santana, 2019) Reused software may also be more stable, in comparison with rewriting the same feature or function, since it would have already been tested. Reusing the same resources multiple times, and fixing any found issues, should result in more reliable software overall. (Dano, 2019)

However, there are challenges that are associated with software reuse. This report aims to investigate such challenges, specifically in terms of software quality and the company's productivity, and analyse proven solutions to mitigate such challenges. The analysis and conclusions of this report are based off extensive research on this matter, utilizing publications found on IEEE and ScienceDirect databases, focusing on those with a publication date during or surpassing 2019.

This report will begin with an overview of what software reuse, software quality, and productivity is, before exploring the associated challenges. Solutions of implementing software reuse, specifically regarding information systems in the healthcare sector, will be critically analysed regarding how well they deal with the aforementioned challenges, before a discussion of the findings of this report. Finally, recommendations on how Valtech may proceed with utilizing software reuse is covered, followed by concluding statements about this report.

2 AN OVERVIEW OF SOFTWARE REUSE

2.1 WHAT IS SOFTWARE REUSE

The concept of software reuse was first proposed in 1968 and has since seen many techniques and methods used to accomplish this goal. (Sommerville, 2016) An empirical study, conducted in 2000 (Rine & Nada, 2000), showed that many large companies, such as Hewlett-Packard, Boeing and branches of NASA, had already been utilizing software reuse for more than 7 years, reporting success in creating better quality products for reduced total effort. Since then, the popularity of software reuse has continued to grow. (Li et al., 2021)

Its success is believed to be due to the benefits gained from this method. Instead of always developing everything that's needed, requiring extensive testing, existing resources may instead be utilized, only requiring implementation and basic testing. The existing resource is often already thoroughly tested, reducing the likelihood of bugs, therefore making it more stable on average.

Successfully utilizing software reuse can yield faster development times, greater quality software and reduced and more accurate cost estimates, since the cost of already developed resources is known. (Santana, 2019)

Software reuse comes in many forms, such as internal reuse, making use of the companies own resources, or external reuse, utilizing third-part components, such as open-source or COTS (commercial-off-the-shelf) software.

Developing software also comes in multiple forms, such as opportunistic, which focuses on making components reusable when identified, and planned software reuse, where a clear plan of what will become made reusable is identified beforehand.

2.2 Software Quality

Software quality is defined as how well a product achieves its desirable attributes, outlined by the end-user requirements. A product of a higher standard will achieve the attributes with greater success. (Zhao et al., 2021)

There are standards, such as ISO/IEC 25010:2011 (ISO/IEC 25010:2011, 2019) , which defines a model consisting of the characteristics and sub-characteristics that should be used to justify the quality of the produced software, such as functional suitability, reliability and security.

Software quality is especially important with applications in the healthcare domain, since they may be working with sensitive information. (Smiari et al., 2022) The required quality of these systems, especially in terms of security and useability, may be high, requiring more time and resources to make sure the software is up to code.

2.3 Company Productivity

Company productivity refers to how efficiently the development teams of a given company can work, to produce software faster. A shorter production time will often result in a reduced cost of the system, allowing for more competitive pricing and deployment time, a reduced cost for the consumer and/or greater profit margins for the company in question. (Santana, 2019)

3 CHALLENGES ASSOCIATED WITH SOFTWARE REUSE

This section details the main three challenges identified during research, that can have an adverse effect on software quality and productivity. Failure to implement a solution that adequately deals with these issues could result in significant problems long term for the company. Below details the specifics of each challenge, and the affects that could be seen if nothing is done about them.

3.1 SECURITY

Security, especially for software that will be used in the healthcare sector, is of vast importance, since it may require handling personal data of the user. However, reliance on reusable code may prove to be an issue regarding security, since a vulnerability that may be exploited in the reusable code may propagate through every piece of software that uses it.

An Empirical study conducted in 2021 (Li et al., 2021) investigated security vulnerabilities in projects that were dependent on other reusable libraries. Of the 337,415 projects studied, it was found that 67,806 projects were dependant on libraries with now known exploits.

As such, time will need to be spent ensuring that reusable code remains secure, ensuring that the fix is propagated throughout. If left, and the exploits are used to gain user information, the company may be liable to lawsuits, potentially causing massive financial implications, and a negative image on their company.

3.2 ORGANIZATION OF REPOSITORY

Keeping an easy-to-use repository is necessary to not impede on the development productivity. (Getov, 2010) Without proper organisation, documentation, and diagrammatic representations, finding the reusable components and figuring out how to use their interface may become a very time-consuming task.

3.3 MAINTAINABILITY

The exact interface created for a reusable component may later be identified as not suitable for its use in another application, requiring alterations to be made. (Feitosa et al., 2020) This may be a time-consuming task, especially in the case of poor documentation or not having direct access to the source code, negatively affecting productivity.

4 CRITICAL EVALUATION OF SOFTWARE REUSE

This section will analyse the different suggested methods of software reuse, in terms of their capability to deal with the aforementioned challenges, specifically for a Medical Information System, in terms of software quality and productivity.

4.1 SOFTWARE QUALITY

Reusing existing software has shown to be successful in increasing the quality of software, since the reused software has been proven to work already. (Dano, 2019) This frees up development time for more important features, or features that there isn't an applicable reuse option.

However, since Valtech specialises in the development of information systems for the healthcare sector, it is imperative that all produced software is up to code, keeping the data it handles secure. To successfully achieve the standards that need to be met, adequate preplanning is required, requiring a full review of reusable code to ensure it is suitable, especially when considering third-party software.

This also means that any identified security flaws need to be addressed quickly. Reusing software with limited/no support would not allow this to be accomplished. This would make it inadvisable to opt for third-party software that cannot guarantee full software support, since COTS software will most likely not provide access to the source code, hindering or preventing the company from fixing the vulnerability themselves, or are open source, potentially making it easier to identify and spread awareness of the vulnerability. (Li et al., 2021)

Possible security risks that will only be fixed slowly, or not at all, will significantly dampen the quality of the software. As such, the Opportunistic approach to software reuse does not seem advisable, since it focuses on finding and reusing code when a case is identified (Mikkonen & Taivalsaari, 2019), which doesn't fit well with high-risk systems, such as ones dealing with healthcare.

However, a more structured approach may still be suitable. (Oliveira et al., 2021) By identifying possible cases for reusing ahead of time, a full investigation can take place, both for internal and third party solutions. The level of support can be adequately identified, allowing for a decision to be made to commit to a given solution, or to proceed with the redevelopment of a given feature. The reusing, or the need to create reusable resources can also be identified beforehand, with development taking place with this goal in mind. This should help to achieve a high level of software quality throughout the lifecycle of the software.

4.2 PRODUCTIVITY

Being able to develop software quicky is a useful ability for software development, since being able to get software of the same quality as your competitors on the market first gives you a competitive edge. This remains the case for medical information systems. (Oliveira et al., 2021)

The opportunistic approach suggests that it can achieve greater development speeds, (Mikkonen & Taivalsaari, 2019) since little-to-no time is spent preplanning, which may make it seem favourable. However, this may also become a shortfall of this solution. Developers attempt to use often unrelated software to achieve their goal, without attempting to understand how components work. This approach will most likely lead to trade-offs of software quality for development speed, which itself could be lost due to bugs, based off incompatibility, or misunderstanding how a function works. Since little effort was made to learn how the tools they are using work, attempting to fix the issues could prove to be very costly, in terms of time and money.

Whereas a more structured approach will take longer than opportunistic development on paper, although still faster than not using software reuse at all. (Smiari et al., 2022) However, the time spent preplanning and investigating will allow for detailed research into the possible solutions, helping to pick and identify cases for reusable code, as well as possible areas to create reusable code. This will help with later development, whilst ensuring software quality is met throughout development.

Proper pre-planning will also help significantly with organising the repository, as the documents and diagrammatical representations can be identified beforehand, to allow for ensuring their completion at a later stage. This would not be the case for Opportunistic development, since a lack of understanding of how the code works will not result in detailed and accurate documentation. (Mikkonen & Taivalsaari, 2019) Likewise, the lack of understanding and documentation will be the downfall of maintainability, which would not be the case with a structured solution.

5 Discussion and Recommendations

Based off the conducted research, I would deem an Opportunistic reuse approach to not be advisable, due to its poor planning, which is not compatible with the structure required to make healthcare information system, and since software quality is usually hindered with this approach. (Mikkonen & Taivalsaari, 2019) Attempting to restructure the company to facilitate this solution would potentially be costly, yielding indifferent or negative effects on software quality and productivity. Therefore, this solution is highly advised against.

On the other hand, opting for a structured solution shows promise for creating software of a greater quality, whilst also reducing the development time, thus increasing the productivity of the company. Research found that the two most promising solutions for implementing this were using Production Line Engineering (Santana, 2019) and Model-driven engineering. (Oliveira et al., 2021)

Based on reviewing the two options listed above, both appear to be suitable for the application of medical information systems. However, it is my belief that a model-driven approach is deemed the best solution to proceed with, based off the empirical data that shows its pervious successful deployments in this field.

Therefore, I would recommend Valtech would proceed with the following:

- Undergo staff training for at least one team, teaching them the concept of Model-driven engineering and of software reuse. This should be accomplished within 2-3 months.
- Hire 1-2 individuals who specialize in developing systems using Model-driven engineering. If deemed appropriate, these may also be the individuals who perform the training outlined in the first point. They should be hired and available to work by the end of staff training.
- Undergo a pilot test for Model-driven engineering on a small-to-medium sized project, to test the effectiveness of this solution. Strengths and weaknesses should be identified throughout this project, as well as a final review of the success overall success and expected ROI of the developed software. This should last several months.
- The data collected during the pilot test should be analysed, deciding how successful the model-driven engineering was. If it's deemed to be advantageous, weaknesses should be investigated further to improve upon found shortcomings. Another pilot's test on a larger scale should be conducted to confirm the solutions success, before fully committing to this solution. If this method is deemed unsuitable, further investigation should be conducted regarding why it failed, and what other methods may prove to be more suitable.

Following the above recommendations should provide proof-of-concept that this solution works, whilst identifying areas where the solution could further improve for the specific needs of Valtech.

Alternatively, it should identify the solution as not viable, if that turns out to be the case, without having a significant effect on the company, since the company wasn't completely restructured, nor was their current method of development completely abandoned.

6 CONCLUSION

Software reuse can be a very powerful tool, reducing development time whilst improving the overall quality of the code. Careful considerations should be made regarding whether to create and reuse internal assets, or to utilize third-party assets, such as COTS. Such considerations include functionality, total cost, level of support, and the level of security they provide. It is also imperative to keep an easy-to-use organisational system for reusable resources, including documentation, to allow for faster finding and implementation of said resources, to further aid in company productivity.

Although software reuse can be very useful, poor implementations or methods can lead to dampened performance, such as Opportunistic reuse, which would be unsuitable for an application in medical information systems.

This report successfully reviews modern literature, focusing on publications during or surpassing 2019, to analyse, discuss and recommend a method in which Valtech can proceed with, allowing them to use software reuse to gain a competitive edge in their field.

However, the proposed year restriction was also the shortfall of this report. There was a plethora of suitably useful publications pre-2019 that would have possibly suited Valtech better, whilst remaining as an up-to-date solution. For example, no detailed literature on Legacy System Wrapping was found, which could be very suitable for working with the old, outdated systems often found in the healthcare sector. Such an approach may have proven more suitable, producing greater quality software and productivity.

It is understandable that a 5-year limit would be imposed for most of the collected material, to ensure the chosen method is modern. However, it would also be worth noting the presence of COVID during this period may have reduced the amount of research conducted, and thus reduced the number of publications in this field. Therefore, surveying a longer time period may have been more suitable.

7 REFERENCES

- Gaber, T. (2023). Software Reuse [Review of Software Reuse].
- Sommerville, I. (2016). *Software engineering* (10th edition, pp. 17–31). Pearson Education Limited.
- Rine, D. C., & Nada, N. (2000). An empirical study of a software reuse reference model. *Information and Software Technology*, *42*(1), 47–65. https://doi.org/10.1016/S0950-5849(99)00055-5
- Li, Q., Song, J., Tan, D.-W., Wang, H., & Liu, J. (2021). PDGraph: A Large-Scale Empirical Study on Project Dependency of Security Vulnerabilities. *Dependable Systems and Networks*. https://doi.org/10.1109/dsn48987.2021.00031
- Zhao, Y., Hu, Y., & Gong, J. (2021). Research on International Standardization of Software Quality and Software Testing. *2021 IEEE/ACIS 20th International Fall Conference on Computer and Information Science (ICIS Fall)*. https://doi.org/10.1109/icisfall51598.2021.9627426

- *ISO/IEC 25010:2011*. (2019). Iso.org. https://www.iso.org/obp/ui/#iso:std:iso-iec:25010:ed-1:v1:en
- Mikkonen, T., & Taivalsaari, A. (2019). Software Reuse in the Era of Opportunistic Design. *IEEE Software*, *36*(3), 105–111. https://doi.org/10.1109/ms.2018.2884883
- Santana, E. (2019). Software Reuse and Product Line Engineering. *Handbook of Software Engineering*. https://doi.org/10.1007/978-3-030-00262-6_8
- Oliveira, D., Rui Gonçalves Miranda, Francini Abdul Hak, Abreu, N., Leuschner, P., António Abelha, & Machado, J. (2021). Steps towards an Healthcare Information Model based on openEHR. *Procedia Computer Science*, *184*, 893–898. https://doi.org/10.1016/j.procs.2021.04.015
- Getov, V. (2010, July 1). *Software Development Productivity: Challenges and Future Trends*. IEEE Xplore. https://doi.org/10.1109/COMPSAC.2010.91
- Smiari, P., Bibi, S., Ampatzoglou, A., & Arvanitou, E.-M. (2022). Refactoring embedded software: A study in healthcare domain. *Information and Software Technology*, *143*, 106760. https://doi.org/10.1016/j.infsof.2021.106760
- Feitosa, D., Ampatzoglou, A., Gkortzis, A., Bibi, S., & Chatzigeorgiou, A. (2020). CODE reuse in practice: Benefiting or harming technical debt. *Journal of Systems and Software*, *167*, 110618. https://doi.org/10.1016/j.jss.2020.110618
- Dano, E. B. (2019). Importance of Reuse and Modularity in System Architecture. *Innovations in Systems and Software Engineering*. https://doi.org/10.1109/isse46696.2019.8984472