International Journal of Advanced Research in Engineering Technology & Science
ISSN: 2349-2819
Email: editor@ijarets.org Volume-10 Issue-11 November-2023 www.ijarets.org

STUDY ON METHODS AND TRIALS OF SOFTWARE REUSABILITY

Praveer Saxena,
Research Scholar, Glocal School of Technology and Computer Science,
Glocal University, Mirzapur Pole Saharanpur (U. P.) India.

Dr. Amit Singla,
Research Supervisor, Glocal School of Technology and Computer Science,
Glocal University, Mirzapur Pole Saharanpur (U.P) India.

ABSTRACT:

The process of software reuse, which entails creating new software systems and applications from pre-existing software components, is covered in the article. Because it may increase the productivity and quality of software development by supporting software engineers at different stages of the process, this strategy has gained importance in recent years. Component-based software reuse, domain engineering and software product lines, and architecture-based software reuse are the three primary methods of software reuse that are discussed in this article. Domain engineering and software product lines involve identifying and documenting the commonalities and variabilities of a set of related software systems, and using this information to develop reusable components. Component-based software reuse involves developing software components that can be reused across different software applications or systems. Developing a software architecture that can be used to other software applications or systems is known as architecture-based software reuse. Software reuse may have advantages, but there are drawbacks as well that may interfere with the development process. These obstacles include discovering, choosing, and modifying pre-existing software components as well as organisational and cultural barriers that may make it challenging to successfully apply software reuse techniques.

INTRODUCTION

Software reuse is the practice of creating new software systems utilising older software artefacts. Reusing software helps to cut down on the expense, effort, and time needed to construct software systems. Due to the components prior testing, validation, and performance optimization, reuse of software components can result in increased productivity, improved quality, and lower development costs. The concept of software reuse has been around since the late 1960s, but it took several decades for the practice to become widely adopted. One of the early efforts to promote software reuse was the Ada programming language, which was designed to facilitate the development of reusable software components. However, the adoption of Ada and other early approaches to software reuse was limited by the lack of standardization and the complexity of developing and

managing reusable components. Today, there are many software reuse techniques and practices that have been developed and refined over the years. Some of these include domain analysis, software product lines, component-basedsoftware engineering, service-oriented architecture, and open-source software. Each of these approaches has its own strengths and weaknesses, and choosing the right approach depends on the specific needs and requirements of the software development project. Despite the potential benefits of software reuse, there are still obstacles and restrictions that preventit from being widely used. One of the most difficult issues is finding and selecting relevant reusable artefacts, which necessitates a thorough grasp of the system requirements, available components, and project applicability. Another problem is managing the complexity of reusable software systems, which can be difficult to understand and troubleshoot due to the interplay of numerous components. Furthermore, establishing compatibility between distinct software components can be difficult since they may have different interfaces, data formats, or implementation details. In assumption, software reuse is asignificant study and development subject in software engineering. While the concept of software reuse has been around for decades, much work remains to be done in order to fully realise its potential. With the development of new approaches and methods, as well as the rising availability of reusable software components, software reuse is anticipated to continue to play a major role in software development in the coming years.

REVIEW OF RELATED WORKS:

Kaur and Sohal (2021) investigated how to construct a QR code library using software reuse approaches. They created the library utilising Android and other modern technologies, and their approach included applying design patterns, aspect-oriented integration, and other object-oriented programming structures to boost productivity, save time, and save development costs. They were able to accomplish these benefits by reusing code rather than beginning from scratch. Thapar et al (2022) proposed a quality model for assessing software components based on reusability. They identified three essential factors that are relevant in software selection and development: the quality properties preferred by stakeholders,the necessary improvements to software component reuse, and the integration of these properties into a proposed model. Their model helps to ensure that only quality properties that are necessary for improving software reuse are integrated into the development process. Ahmer et al (2019) conducted a literature review to gain a better understanding of the concept, benefits, and factors of software reusability. They identified 11 methods for software reuse, including design patterns, component-based design, application frameworks, legacy systems wrapping, service-driven systems, application product lines, COTS integration, and program libraries. Their review helped to consolidate existing knowledge on software reuse and provided a framework for future research. Varnell-Sarjeant and Amschler Andrews (2019) analyzed empirical studies to compare reuse results in embedded and non-embedded

systems. They compared the success and failure of software reuse in these two types of systems and looked at factors such as reuse amount, effort, quality, performance, and overall achievement. Their findings help to identify the factors that contribute to successful software reuse and to understand the impact of the development approach on reuse success or failure.Xin and Yang (2017) discussed an engineering management software reuse framework that provides guidance on how to select types of reuses and how to manage the reuse process. They identified four types of software reuse and explained how reuse feasibility should be analysed. Their study highlighted the importance of key point management in the reuse of software and demonstrated how a careful approach to software reuse can provide significant benefits. Mateen et al. (2021) developed a reuse strategy for increasingsoftware quality. They employed a verification and validation (V&V) method to control quality and accuracy throughout the software life cycle, followed by a questionnaire survey to determine the influence of their methodology on quality attributes, specifications, and design specifications. They discovered that using ad hoc, CBSE, MBSE, product line, and COTS reuse strategies resulted in considerable increases in software quality. Finally, these studies highlight the potential benefits of software reuse and provide guidance on how to achieve these benefits through careful planning, management, and selection of reuse techniques. While software reuse has been around for decades, there is still much to learn about how to make the most of this powerful tool for software development.

EXISTING SYSTEM:

Component-based development (CBD) is an existing system of software reuse. It is a software engineering approach that emphasizes the use of reusable software components. In CBD, software is built by assembling pre-existing software components, rather than writing code from scratch. CBD promotes the development of modular, reusable, and maintainable software systems.

CBD involves creating software components that can be reused in different applications. These components are designedto be self-contained and can be easily integrated with other components. CBD promotes software reuse by allowing developers to create software systems by assembling pre-existing components, rather than writing code from scratch. Thisreduces development time and cost while improving software quality.

CBD components can be developed in any programming language and can be implemented as standalone executables, dynamic link libraries, or web services. Examples of CBD frameworks include Microsoft's .NET Framework and Java Enterprise Edition.

CBD has several advantages, including:

- Reduced development time and cost
- Improved software quality

- Increased productivity
- Improved maintainability and scalability
- Improved interoperability
- Enhanced reuse of existing software components However, CBD also has some disadvantages, including:
- Increased complexity due to the need for component integration
- Limited availability of reusable components
- Increased overhead due to component management and version control

PROPOSED SYSTEM:

Since there are several proposed systems of software reuse that can be used to improve the software development process, Service oriented architecture is being proposed.

Service-oriented architecture (SOA) is a software architecture that emphasizes the use of loosely coupled services. SOA promotes software reuse by allowing developers to reuse existing services to build new software systems. SOA services canbe implemented in any programming language and can be accessed using standard protocols such as SOAP and REST.

SOA is an architectural style that encourages the use of services to enable communication between different software components. SOA can be presented as a technique to develop modular, reusable software components that can be shared and utilized in other applications when it comes to software reuse.

SOA proposes a system that is composed of a set of services, each of which has a well-defined interface and functionality. These services can be reused in different applications, allowing for greater flexibility and efficiency in software development.

SOA can be proposed as a way to create modular, reusable software components that can be shared and used in different applications. This approach promotes a flexible and efficient software development process that can save time and resources.

SOFTWARE REUSE APPROACH:

Software reuse technology is a software engineering approach that aims to reuse existing software components to create new software systems. There are several forms of software reuse, including system reuse, application reuse, component reuse, object reuse, and function reuse.

System reuse involves selecting multiple applications that can be reused within a system. This approach requires the conceptual design, architectural design, system selection, interface development, integration and development to work parallel to governance and management policies.

Application reuse entails changing a software system to meet the needs of several clients while preserving the source code. This form of reuse is created for wide market use and is sometimes referred to as commercial off-the-shelf (COTS) products. It employs a built-in configuration mechanism that allows a system to be developed to meet the needs of various customers.

Component-based reuse divides software into atomic components. A repository of these components is used to build a new software system by selecting the appropriate component from the repository every time a new component is needed.

Reusing software components that perform a certain activity, such as mathematical functions or class objects, is an example of object and function reuse. This way of reusing has been used in standard libraries for decades. It is particularly advantageous in areas such as mathematical algorithms and graphics, where the production of efficient objects and functions necessitates a specialised, pricey skill.

CHALLENGES:

The difficulties encountered during software reuse: It cites three separate studies that revealed distinct impediments and issues experienced during software reuse.

B. Jalender and colleagues (2010) identified both technical and non-technical barriers to software reuse. Missing systematic component requirements, inability to guarantee the accuracy of a component, poor presentation of reusable pieces, and the absence of a software reconstitution technique are among the technical obstacles. The inability to engage, encourage, train, and reuse software, the lack of organisational support for software reuse institutionalisation, the difficulty in evaluating reuse benefits, and the need to address intellectual property rights and software reuse contractual issues are the non- technical barriers.

Sajjah and Ali (2014) conducted a systematic study of 36 chosen studies and identified 8 challenges associated with software reuse during software application developments. The challenges highlighted by the study include domain analysis and modeling, lack of reuse skills and knowledge, lack of management support, high reuse cost, lack of component storage, lack of documentation, lack of proper IT infrastructure, and lack of team

awareness. Making domain analysis and modeling the highest form of challenge faced during software reuse.

Charles (2014) categorized the challenges of software reuse into technical, organizational, economical, and legal impediments. The technical challenges include issues such as the lack of proper documentation and testing of reusable components, while the organizational challenges are related to the lack of support for reuse initiatives, inadequate training, and the absence of a reuse culture within an organization. The economic challenges refer to the cost of developing and maintaining a reusable library, and the legal challenges include issues such as intellectual property rights and software reuse contractual problems.

The study highlighted several technical, organizational, economical, and legal obstacles to software reuse. Technical obstacles include finding codes and designs that are difficult to comprehend, particularly complex classes, and understanding the architecture of reference, frameworks, models, and classes. Organizational barriers include a lack of coordinated reuse from organizations, as they often do not have clear directives and processes that describe when and how to use existing software in conformity with software development strategies. Economical hurdles include the cost of the manufacturer to supply reuse components, viewed as an investment, and the cost for the "re-user" to locate, integrate, and check reusable components. Legal issues are broken down into four parts: trade secret protection, patents for new and inventive technical innovations, copyright protection, and ethical responsibilities and obligations. The reuse has the responsibility to have a quality assurance of the software to inspect if the reused software complies with quality standards to prevent damage and breakdowns. Overall, software reuse can be challenging, but if the challenges are addressed, it can be a viable option.

Software reuse is the practice of utilizing existing software artifacts, such as code, documentation, and design, to develop new software systems. According to Schenkelberg (2016), structured and modular programming is the most likely use of software reuse, as it uses a top-down analysis approach for problem-solving, modularization for program structure and organization, and structured code for the individual modules, which simplifies the task of programming and reduces complexity, thus improving programmer productivity.

However, there are several challenges that come with software reuse. These challenges, as highlighted by Schenkelberg (2016), include increased maintenance costs, the need for longer software tool support, a "Not invented here" attitude that decreases acceptance, the operating cost of producing and sustaining a component library, the time required to select reusable software components, the need for more knowledge and training, and the necessity for a more diverse skillset.

To overcome these challenges, software developers must prioritize quality properties when adopting a reuse strategy. According to Capilla et al (2019), the most popular quality properties are readability, functional stability, performance interoperability, security, privacy, portability, efficiency, and modularity. If these properties are not put in place, software reuse can become challenging.

Mäkitalo et al (2020) identified some downsides to software reuse, such as compatibility problems that can lead to technical debt, reuse of copy-paste causing problems of traceability, dependencies of snowball that can affect the reuse of code, reusable software assets often lacking maintenance, open package repositories causing security concerns, and General Public Licenses (GPL) that can be challenging to understand.

In conclusion, software reuse is a positive practise that can increase productivity while decreasing complexity. However, it is not without difficulties, and developers must prioritise quality properties while also addressing drawbacks in order to make software reuse feasible and successful.

DISCUSSIONS:

Software reuse is an essential component of software engineering because it can result in significant benefits such as reduced development time, lower costs, greater software quality, and increased productivity. To enjoy these benefits, software architectures, design patterns, requirements specifications, and design documents must be rigorously documented and designed. The reuse of software components may not always be possible due to various factors such as technical obstacles, organizational barriers, economic hurdles, and legal issues.

However, by employing the proper tactics, such as structured and modular programming, reusing theories and frameworks that provide the software's backbone, and reusing theories and frameworks that provide the software's backbone, these obstacles can be solved.

To address the challenges connected with software reuse, software development techniques should include software reuse as an integral component of the development process. This includes meticulously documenting software components and architectures, creating libraries of reusable components, and encouraging reuse within the organisation. By executing these measures, software development teams can reduce maintenance costs, shorten development time, improve software quality, and increase productivity.

In conclusion, software reuse is a vital aspect of software engineering that can lead to significant benefits. It is essential to document and systematically design software components, architectures, and design patterns to

reap the full benefits of software reuse. While there are challenges associated with software reuse, these can be overcome by adopting the right strategies and incorporating software reuse into the software development process.

CONCLUSION:

In conclusion, software reuse implemented a service-oriented architecture which consists of various benefits, including modular and reusable components, flexibility, loose coupling, service discovery, interoperability, and scalability. Finding, selecting, and adapting existing components, as well as organisational and cultural aspects, provide obstacles. To reap the benefits of SOA for software reuse, which can considerably enhance software development quality and productivity, careful planning is required. Overall, SOA allows for the development of modular, reusable software components that increase flexibility and efficiency. Its weak coupling allows for simple upgrades and replacements without affecting other components, and service discovery protocols make interoperability possible. The modular nature of SOA enables scalability, but challenges in finding and adapting components and organizational factors need to be addressed. Careful consideration and planning are crucial to harness the benefits of software reuse with SOA, resulting in enhanced quality and productivity in software development.

REFERENCES

1. Ahmar, I., Abualkishik, A., & Yusof, M. (2014). Taxonomy, Definition, Approaches, Benefits, Reusability Levels, Factors and Adaption of Software Reusability: A Review of The Research Literature. Journal of Applied Sciences, 14. https://doi.org/10.3923/jas.2014.2396.2421

2. B. Jalender, N. Gowtham, Kumar, K. Praveen, K. Murahari, & K. Sampath. (2010). Technical Impediments to Software Reuse. International Journal of Engineering Science and Technology, 2(11).

3. Capilla, R., Gallina, B., Cetina, C., & Favaro, J. (2019). Opportunities for Software Reuse in An Uncertain World: From Past to Emerging Trends. Journal of Software: Evolution and Process, 31(8). https://doi.org/10.1002/smr.2217

4. Gacek, C. (Ed.). (2002). Software Reuse: Methods, Techniques, And Tools: 7th International Conference, ICSR-7 Austin, TX, USA, April 15–19, 2002 Proceedings (Vol. 2319). Springer Berlin Heidelberg. https://doi.org/10.1007/3-540-46020-9

5. Jalender, B., Govardhan, D., & Premchand, P. (2010). A Pragmatic Approach to Software Reuse. Journal of Theoretical and Applied Information Technology, 14, 10.

6. Kaur, A., & Sohal, H. (2013). QR Code Library on The Base of Software Reuse Approach. International Journal of Science and Engineering Applications, 2, 44–48. https://doi.org/10.7753/ijsea0203.1002

7. Keswani, R., Joshi, S., & Jatain, A. (2014). Software Reuse in Practice. 2014 Fourth International Conference on Advanced Computing & Communication Technologies, 159–162. https://doi.org/10.1109/acct.2014.57

8. Kim, Y., & Stohr, E. A. (1992). Software Reuse: Issues and Research Directions. Proceedings of the Twenty-Fifth Hawaii International Conference on System Sciences, 612–623 Vol.4. https://doi.org/10.1109/HICSS.1992.183360

9. Mäkitalo, N., Taivalsaari, A., Kiviluoto, A., Mikkonen, T., & Capilla, R. (2020). On Opportunistic Software Reuse. Computing, 102(11), 2385–2408.