Software Reusability: Approaches and Challenges

Moko Anasuodei¹, Ojekudo, Nathaniel Akpofure²

¹Department of Computer Science and Informatics, Faculty of Science, Federal University Otuoke, Nigeria

²Department of Computer Science, Faculty of Natural and Applied Sciences, Ignatius Ajuru University of Education, Nigeria

Abstract: Software reuse is used to aid the software development process which in recent times can improve the resulting quality and productivity of software development, by assisting software engineers throughout various software engineering phases to enhance the quality of software, provide quick turnaround time for software development using few people, tools, and methods, which creates a good software quality by enhancing integration of the software system to provide a competitive advantage. This paper examines the concept of software reuse, the approaches to be considered for software reuse, which is broadly shared into three categories: component-based software reuse, domain engineering and software product lines, architecture-based software reuse and challenges that affect the software reuse development process.

Keywords: software, reuse, engineering, challenges, domain engineering, component-based

I. INTRODUCTION

Software reusability is seen as the use of existing software systems to develop new software systems which means re-use of software. This is an evolving field of software technology, which helps in reducing the time and effort needed to create software systems from the ground up.

Software reuse rules possess three simple principles: Firstly, the object must be reusable, then the object must be of value for the reusable object and then the user should know precisely how to use it. Two procedures are included in these three rules: development for reuse and development with reuse. If these problems are being fixed then software reuse can occur. The reuse of software is not only reuse of software, but also a reuse of any software development accomplishment like project proposal, reporting feasibility, request analysis, abstract design, detailed design, demo coding tests, documentation, user manual. Yang (2010)

All through the software engineering process, the software is available in different forms and are in different formats such as requirements specifications, the architectural design, and the source code. Software engineering is the systemic use of tools and knowledge, to achieve the stated technical, economic, and human goals of a comprehensive software system. Software results from performing a set of software engineering activities, including requirement gathering, design, implementation, testing, and maintenance and various tools have been developed and used to support these activities automatically. The aim has been to improve the software's resulting quality and productivity by assisting the engineer throughout the engineering processes and various

phases. Software engineering has been more centered on original development which gives an optimal software at a faster and less costly price, a design process based on systemic reusable is now recognized.

Software reuse reduces efforts and cost, because software development costs can be extremely high, but they are shared, such that the cost of reuse can be extremely low. One major advantage of software reuse suggested by Keswani et al (2014) explains that there is a significant need for the number of bugs to be reduced during software development process, such that instead of developing an entirely new product, legacy software proven reuse is recommended. Software reuse has as its objective to be recycling of the software product's design, code, and other components, thus cutting costs, time and improving product quality. This study's major motivation is at investigating the need for software reuse and why it should be adopted. The Research Questions are: Why the need for software reusability? What are the various technologies used? What developmental frameworks are in place for software reuse and finally the challenges faced.

Figure 1. Types of Software Reuse

Figure 1 above shows the 2 types of software reuse: Horizontal reuse which applies to software components that are used in a lot of areas, it also entails the reuse within a larger system of a commercial off-the-shelf application (COTS) or a third-party application such as an e-mail or a word processing program. While Vertical reuse relates to functional system areas or domains that are used by a family of similarly functioning systems. This vertical reuse gave rise to an additional discipline, namely domain engineering. Domain engineering is a thorough, iterative approach life cycle used by a company to achieve strategic business objectives. Jalender et al (2010).

Mäkitalo et al (2020) adopting Kruegers classic software reuse research paper implies that software reuse must fulfil the subsequent four "truisms":

1. To be efficient, the software reuse technique must decrease the cognitive spacing between the initial system concept and the final implementable system.
2. The reuse of objects by a software reuse technique should be simpler than the development of the software from start to finish.
3. You need to know what it does to select an object for reuse.
4. You must be able to "find it" rapidly than you can "build it" to reuse software object effectively.

II. REVIEW OF RELATED WORKS

Kaur and Sohal (2013) evaluated how a QR code library can be built using new technologies such as Android and software reuse. In the QR code library, how to reuse software works. During the development of a software reuse system, certain approaches such as design patterns, aspect-oriented integration, reuse generator, object-oriented programming structures and software reuse libraries, framework integration etc. are considered. This approach increases production, saves time, and reduces software development costs and reduces overruns of the schedule. Thapar et al (2014) paper present a quality model, based on reusability, which can be used for software component assessing. Three essential factors are relevant in software selection and software development. Only quality properties preferred by stakeholders and necessary to improve the reuse of software components are integrated into the proposed model. Ahmer et al (2014) examine the literature on the software reusability concept (SR). A systematic review of the literature was studied to understand the concepts, approaches, benefits, levels of reusability, factors, and software reusability adaptations. The study on the content of software reusability was systematically reviewed, and many databases were scanned for literature. Studies of 1977-2013 published were considered and identified if they defined an evaluation of the software reusability intervention in information and communication technology. Further highlights were explained on the re-usability concept of software, stating that it comprises of eleven methods, which are covers design patterns, component-based design, application frameworks, legacy systems wrapping, service-driven systems, application product lines, COTS integration and program libraries. Varnell-Sarjeant and Amschler Andrews (2015) literature survey shows empirical reuse studies to compare reuse results in embedded systems with non-embedded systems. Reuse success and failure in embedded versus non-embedded systems is analyzed and compared to attain reuse results which include reuse amount, effort, quality, performance, and overall achievement, to distinguish between development approach for determining its influence on

reuse success or failure. Xin and Yang (2017) discussing on the engineering management software reuse framework, demonstrates how to select types of reuse and what to do in the management of processes. The study discussed four types of software reuse and how reuse feasibility should be analyzed. Different types of re-use processes and key points were linked, and a case study was illustrated, which shows that in each type of software reuse engineering management, there will be significant benefits from the re-use factors and approaches together which will provide reasonable processes. Furthermore, key point management plays a key role in the reuse of software. Mateen et al (2017) proposed a reuse approach that showed how software reuse improves software quality. The Verification and Validation (V&V) method used for this purpose is an aspect of the quality control process for software, which controls quality and accuracy during the software life cycle. A QUESTIONAIRE survey to identify the impact on quality attributes, specifications and design specifications for the reuse approach was conducted and the study checked on the existing software industries on quality improvements techniques such as ad hoc, CBSE, MBSE, product line, COTS reuse. The analysed results with the help of the MATLAB tool, provided and efficient data management, better output organization to monitor the reusability and quality improvement technology.

III. SOFTWARE REUSE APPROACH

Software reuse technology is a software engineering method that aims at reusing and amplifying existing software, its reuse is of different forms such as System reuse, Application reuse, Component reuse, object and function reuse:

- a. *System Reuse*: This approach includes full selection, which consists of several applications that can be reused in a system. A system of systems is used to freely monitor and control two or more component systems. Conceptual design, architectural designs, system selections, interface development and integration and development are part of the development process and work parallel to governance and management policies.
- b. *Application Reuse*: This form of reuse involves the adaption of a software system to address the need of different customers, such that the source code is not affected. It is developed basically for the general market purpose, not for specific customers. The system products are sometimes known as COTS (Commercial off-shelf system) products, whereby the products are adaptable because it uses a built-in configuration mechanism that enables a system to be designed to satisfy different customers.
- c. *Component-Based Reuse*: Explains that different software is divided into atomic components, which shows that different software systems have some

common components. This form of reuse uses a repository of these common atomic components to build a new software system by selecting the appropriate component from the repository, every time a new component is needed.

- d. *Object and Function Reuse*: Software components that execute a specific task can be reused, such as the mathematical function or class object. For the past several decades, this method of reuse seems to have been prevalent across standard libraries. Many functional and class libraries are freely accessible. By connecting them with a recently designed application code, you re-use the classes and functions in these libraries. Reuse is notably profitable in areas like mathematical algorithms and graphics in which is specific, costly skill is required for the development of efficient objects and functions. (Sommerville, 2016)

Figure 2: Software Reuse Development frameworks

Figure 2 shows the software developmental frameworks to be considered for software reuse, explaining the terms in Figure 2 above: Application frameworks are when collections of abstract and concrete application systems classes are adjusted and enlarged before software design. The link-up of shared services, which may be supplied externally, develops service-oriented systems. Designs are generic abstractions occurring throughout applications are portrayed as design patterns that display abstract and specific objects and interactions. Software Engineering based on components System integration (object collections) that comply with the component-model standards is created for system development. Configurable Application Systems are domain-specific systems designed to be tailored to specific system customer requirements.

IV. CHALLENGES

B.Jalender et al (2010) divided the challenges faced by software reuse into technical and non-technical obstacles, such that the technical challenges involve missing systematic component requirements, inability to certify the accuracy of a component, remarkably bad presentation of reusable parts and generally is the absence of a software reconstitution methodology, the complete nonexistence of reusable and reliable resources for software and absence of software tools and techniques to achieve software reuse. The non-technical obstacles are the inability to engage, encourage, train and reuse software. The absence of organizational support for software reuse institutionalization. The problem in evaluating reuse benefits and there is a need to address intellectual property rights and the software reuse contractual problems.

Sajjah and Ali (2014) identified 8 challenges associated with software reuse during software application developments in a systematic study retrieved from 36 chosen studies, the problems highlighted by the study includes Domain analysis and modelling with 83%, Lack of reuse skills and knowledge 75%, Lack of management support 33%, High reuse cost 33%, Lack of component storage 25%, Lack of documentation 20%, Lack of proper IT infrastructure 11%, Lack of team awareness 6%. Making domain analysis and modelling to have the highest form of challenge faced during software reuse.

Charles (2014) describes the difficulties and business advantages of reuse, which is extremely prevalent and problematic, the author categorized the challenges of software reuse into technical, organizational, economical, and legal impediment.

The study listed the major technical obstacles faced as, finding codes and design which are tough to comprehend when having to deal with classes, especially classes which have no clearly defined focus, such that to identify reuse opportunities, software developers require a similar, easy to understand and extremely seamless transition and complexity to evaluate and reuse resources, which are issues in understanding architecture of reference, frameworks, models, classes, etc. It really challenging to find codes and design that can be reuse when dealing with complex classes. The organizational barriers highlighted are lack of coordinated reuse from organization because organizations consider a reuse approach to measure, manage, study, and finally adapt to the technical domain engineering as time intensive and costly for engineers and programmers. Many organizations processes do not take reuse into account; they do not have clear directives and processes that describe when and how to use existing software in conformity with software development strategies, such as agile, waterfalls and others. The economical hurdles are the cost of the manufacturer to supply reuse component is viewed as an investment

combination with "re-user" cost to locate, integrate, and check reusable component. The legal issues are broken down into four parts: firstly, trade secret protection is a problem that developers of software face when switching the software or any part thereof to another, as most software tailored to the business needs, converting it into another one means you will pass on the business secrets or expertise of the company. Secondly Patents are presented for new and inventive technical innovations. For methods or processes describing how the products function, patents are granted. Copyright protection is the third legal challenge, only helps protect the software itself not the fundamental principles and concepts. Finally, Ethics, responsibility and obligations are usually reuse of software is at users own risk. It is in the utmost importance of the "reuser" to have a quality assurance of the software to inspect if the reused software complies with quality standards to prevent damage and breakdowns.

Schenkelberg, (2016) highlighted that the most likely use of reuse software is for structured and modular programming because it uses top-down analysis for problem-solving, modularization for program structure and organization, and structured code for the individual modules such that it simplifies the task of programming by making use of a highly structured organizational plan and reduce the complexity by improving programmer productivity. Software reuse challenges involve maintenance cost increase, software tools needing longer support and become outdated, "Not invented here" attitude which decreases acceptance, there is operating cost of producing & sustaining a component library, uses a lot of time to select reusable software components, more knowledge and training is essential, necessitates more skillset. With the above-listed challenges faced by software reusability being addressed, it is safe to say software reuse is possible and can be adopted.

Capilla et al (2019) survey shows that the most popular quality properties when adopting a reuse strategy are readability, functional stability, performance interoperability, security, privacy, portability, efficiency, and modularity, in that if these listed qualities are not put in place, software reuse becomes challenging.

Mäkitalo et al (2020) identified software reuse downsides to be problems in compatibility which can lead to technical debt and reuse copy-paste causing problems of traceability, dependencies of snowball can affect reuse of code, reusable software assets often lack maintenance, Open package repositories, such as Node Package Manager (NPM), may cause security concerns and General Public Licences (GPL) are tough to understand.

V. DISCUSSION

Software reuse is needed to reuse all components and software systems with basic functions so that they can be reused to follow the motto "Waste not, want not." To deal

with our research questions. These systems or components are however sometimes so precise, that modifying them for a new situation is very cost-effective. So, therefore the theories which provide the framework of the software can be reused instead of code reuse. Which is the reuse concept, reuse a software component cannot be used but instead there is a reuse of idea, a working approach, or an algorithm. (Sommerville,2016)

Therefore, this study proposes that software reuse is important and should be considered in software engineering process, such that code reuse, software architectures, design patterns, requirements documents, and design documents are well documented and systematically designed. The main way to tackle the essential complexity of software development remains in a systematic software reuse, which can create exceptional impacts for software reuse possibilities. Bearing in mind the benefits that come with software reuse and paying key attention to its associated drawbacks during software development process.

VI. CONCLUSION

In conclusion, procedures and processes for software reuse should be integrated into the existing software development process, such that software asset archives should be created and sustained so they will help software assets to be designed and reused. Individuals also should be trained in software reuse skills and despite the initial cost there are significant benefits to software reuse if strategic sustainability majors are followed and the necessary planning is done, which will improve product quality and dependability and project development time and project-related costs can also be reduced.

REFERENCES

- [1] Ahmar, I., Abualkashik, 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] Al-Badareen, A., Selamat, M., A. Jabar, M., Din, J., & Turaev, S. (2010). *Reusable Software Components Framework*.
- [3] 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).
- [4] 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>
- [5] Charles, B. (2014). *Reuse: Challenges and Business Success*. <https://www.Academia.Edu>.
- [6] 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>
- [7] Jalender, B., Govardhan, D., & Premchand, P. (2010). A Pragmatic Approach to Software Reuse. *Journal Of Theoretical and Applied Information Technology*, 14, 10.
- [8] John, L. (2019, May 6). (5) *What Are the Basic Issues in Software Reuse?* - Quora. <https://www.Quora.Com/What-Are-The-Basic-Issues-In-Software-Reuse>

- [9] 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>
- [10] 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>
- [11] 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>
- [12] Konda, B. M., & Mandava, K. K. (N.D.). *A Systematic Mapping Study on Software Reuse*. 93.
- [13] Mäkitalo, N., Taivalsaari, A., Kiviluoto, A., Mikkonen, T., & Capilla, R. (2020). On Opportunistic Software Reuse. *Computing*, 102(11), 2385–2408. <https://doi.org/10.1007/S00607-020-00833-6>
- [14] Mateen, A., Kausar, S., & Sattar, A. (2017). A Software Reuse Approach and Its Effect on Software Quality, An Empirical Study for The Software Industry. *International Journal of Management, It & Engineering*, 7(2), 14.
- [15] Mcclure, C. (1997). *Software Reuse Techniques: Adding Reuse to The System Development Process* (1st Edition). Prentice Hall.
- [16] Sajjah, M., & Ali, A. Z. (2014). *Challenges Of Adopting Software Reuse: Initial Results*. 85–89.
- [17] *Considerations*. Accendo Reliability. <https://accendoreliability.com/software-reuse-considerations/>
- [18] Sommerville, I. (2016). *Software Engineering* (10th Ed.). Pearson. Software-Engineering-Book.Com
- [19] Teofilo Gonzalez, & Jorge Diaz-Herrera. (N.D.). *Computing Handbook Computer Science and Software Engineering* (3rd Ed.). CRC Press.
- [20] Thapar, S. S., Singh, P., & Rani, S. (2014). Reusability-Based Quality Framework for Software Components. *Acm Sigsoft Software Engineering Notes*, 39(2), 1–5. <https://doi.org/10.1145/2579281.2579298>
- [21] The Education Channel. (2017, October 6). *Software Reuse, Types, Benefits of Reuse, Reusable Components*. <https://www.youtube.com/watch?v=Lozztwrxoj8>
- [22] Varnell-Sarjeant, J., & Amschler Andrews, A. (2015). Comparing Reuse Strategies in Different Development Environments. In *Advances in Computers* (Vol. 97, Pp. 1–47). Elsevier. <https://doi.org/10.1016/Bs.Adcom.2014.10.002>
- [23] Xin, T., & Yang, L. (2017). A Framework of Software Reusing Engineering Management. *2017 Ieee 15th International Conference on Software Engineering Research, Management and Applications (Sera)*, 277–282. <https://doi.org/10.1109/Sera.2017.7965738>
- [24] Yang, Q. (2010). Application Research on Software Reuse Technology. *2010 3rd International Congress on Image and Signal Processing*, 3712–3714. <https://doi.org/10.1109/Cisp.2010.5647100>