2022 IEEE 2nd Mysore Sub Section International Conference (MysuruCon)

Issues and Challenges in Existing Re-engineering Methodologies of Object Oriented Systems

1st Wavda Aniva Zabidi
Asia Pacific University of Technology and Innovation
Technology
Kuala Lumpur, Malaysia
wavdaniva@outlook.com

2nd Muhammad Ehsan Rana
Asia Pacific University of Technology and Innovation
Technology
Kuala Lumpur, Malaysia
muhd_ehsanrana@apu.edu.my

3rd Chandra Reka A/P Ramachandiran
Asia Pacific University of Technology and Innovation
Kuala Lumpur, Malaysia
chandra.reka@apu.edu.my

2022 IEEE 2nd Mysore Sub Section International Conference (MysuruCon) | 978-1-6654-9790-9/22/$31.00 ©2022 IEEE | DOI: 10.1109/MysuruCon55714.2022.9972365

***Abstract*—Software products in use today need to be updated frequently to stay alive in the market and meet the increasing needs of their customers. In general, a software system experience constant modification and extension in its life cycle. Most enhancements are performed based on developers' knowledge and experience using ad-hoc approaches. Re-engineering a software system is an economical way to provide a much-needed boost to alter the current software. In this paper, researchers examine the existing methodologies in terms of their inherent quality attributes like flexibility, simplicity, and effectiveness. This study also intends to address the limitations that software engineering teams are currently facing when they do not use a specific methodology in executing the re-engineering process.**

Keywords*— *Object-Oriented System, Software Re-engineering, Software Process, Software Methodology.

I. INTRODUCTION

An ideal feature of the software nowadays is its ability to evolve. Software change is very important to keep the software alive. This is in line with one of the principles of program evolution that supports continual improvement. Improvement in software over time is critical for managing change requests such as correcting bugs, program upgrades and satisfying client's changing needs or new requirements [1]–[3] and so on.

Currently, the demand of system enhancement in the industry has several forms, which involves (1) updating software to support new interface modes e.g. web-based applications or cloud-based provisioning; (2) migration of legacy applications to modern technologies from outdated/unsupported technologies; (3) transforming applications to take on new regulatory requirements; (4) rearchitecting systems to make them better suited to the business process [4].

The importance of maintaining software applications stems from the fact that nowadays a large amount of money has been invested by several software companies or on their systems. Hence it is essential to ensure that such systems 0p are maintained effectively, rather than developing new systems. Although, changes do not come without costs [3]. Especially considering that the software industry has evolved into an area where its survival and growth are directly linked to the effective utilization of its products within strict limits of resources, time and budgets [5], [6]. Delivering on these constraints require the delivery of robust software that is resilient and adaptable [6].

However, this continuously demanding maintenance could decrease the quality of the software and increase its maintenance cost. But all these changes in the system's structure are making the system more complex and difficult to understand over time. Software re-engineering is one of the important approaches that could be taken to improve software quality and increase software maintainability. Improving software quality and software maintainability can only be achieved if we could determine whether it is the right time to conduct maintenance or re-engineering on the system [7].

In a study conducted by Khadka et al. [8], it has been identified that the main consideration for the majority of software system practitioners to determine if a system is a legacy depends on whether the system functionalities still aligned with the organization business process. Meanwhile, according to Sneed [9], a system is considered a legacy system when it has been running for more than five years [10]. Most organizations' software systems must evolve to meet these complex requirements. Meanwhile, developing new software which would have a higher risk, costs and require a longer time. A systematic re-engineering is required to reduce the overall cost as well as increase the maintainability and quality of the system [3], [11], [12].

Throughout the software re-engineering process, the system's components need to be examined in detail, determine its inter-relationships and proceed with designing the new system based on the components of the legacy system with the new requirements. The re-engineering effort must be able to be tailored into a different type of business process and function with the help of basic guideline to effectively carry out the implementation of the re-engineering.

Most enhancements in the software industry are performed based on developer knowledge and experience using the ad-hoc approach due to the lack of methodologies that are available to support an effective systematic re-engineering. This ad-hoc approach might lead to failure in re-engineering processes. Therefore, a systematic methodology is needed to find an appropriate solution in order to prevent re-engineering project failures. The overall aim of this research is to examine the existing methodologies in terms of its issues when it comes to software re-engineering an object-oriented system from flexibility, simplicity, and effectiveness point of view. This study also intends to mainly address the limitations and shortcomings that development teams are currently experiencing in re-engineering process when no specific methodology are being used or when they use an existing methodology.

II. Literature Review

A. Overview of Software Re-engineering

Re-engineering is the process of redesigning and transforming an existing software system into a new form. Re-engineering defined as the transforming and evaluating an existing system by reconstructing it into a new form through continuous delivery of the new form [13]. One of the terms related to re-engineering is reverse engineering which is a process that needs to be done before the re-engineering process. Reverse engineering defined as the process of evaluating a specific system to identify its components and interconnections, as well as to construct the system into another form or a more complex system. This process is about understanding the system, whereas re-engineering is about restructuring / refactoring the system [14], [15].

Hammer and Champy identify system re-engineering as a revolutionary approach to restructuring business processes to achieve significant changes in terms of efficiency, quality, cost, service, and speed for an organization. System re-engineering is revolutionary to some degree because besides enhancing process, it also aims to get into the heart of problems though reinventing the system itself and build competitive advantages centered on advanced technology [16].

Re-engineering comes as part of a software modernization process. During the modernization of the legacy system, software developers dedicate most of their efforts to understanding the function and behavior of the system. This could be more challenging in the case of object-oriented code since multiple dispersed objects are assigned to the same function. Re-engineering or renovating process could increase the lifetime of a legacy system [17], [18].

B. Maintenance and Re-engineering

Re-engineering and maintenance are closely related to each other. Maintenance is the last stage of the system development life cycle which begins after the deployment of the system to fix errors, enhance performance and other software attributes. Maintenance plays a important role in a software system's life cycle. When maintenance exhaust, re-engineering is commenced. Maintenance raises the software age, and re-engineering brought the software system a fresh age period. As depicted in the following figure, T is the transition point, below point T; maintaining the system is not feasible. The system must be reengineered at point T as it is the only way to avoid new development costs [19].

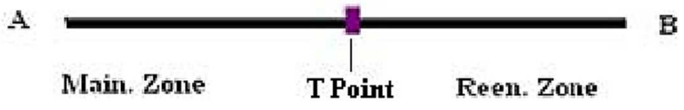

Fig. 1. System Maintenance and Re-engineering Timeline [19]

Systematic re-engineering is required to reduce the maintenance cost of the system. At this point, consider re-engineering or retiring is necessary for the system. If the system is retired, the company will have to spend more on the cost of new software. New software costs considerably higher than re-engineering does. Maintenance beyond point T raises the system's complexity, reduces the quality of software, and increases the cost whereas re-engineering increases software quality, control maintenance costs through reusing legacy system [20] and increases the software system lifetime as depicted in the following figure [19].

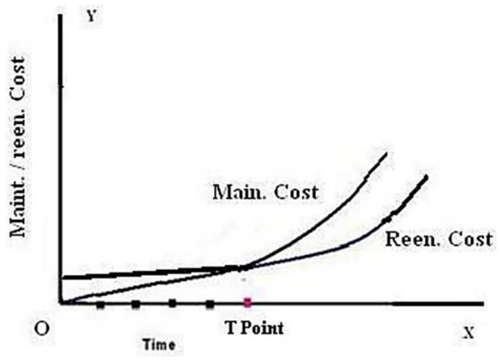

Fig. 2. System Maintenance and Re-engineering Cost Over Time [19]

C. Software Re-engineering Processes

It is very important to follow the set of tasks as specified below during the re-engineering process [21], [22].

- Establishing the re-engineering team
- Analyzing project feasibility
- Analyzing and planning
- Implementation of re-engineering
- Testing and transition

According to Kumar & Gill [19], re-engineering has the following three stages.

1) Reverse engineering: This project involves the project team to thoroughly examine the system's function and behavior. The business process is improved, and the requirements are updated. Objects are added or deleted following the planned new system. Reverse engineering aims to recover missing/undetermined information detecting side effects and apply the reuse through understanding the system [21], [22]. In this stage, the process begins from the code level to higher-level abstraction. It is a vertically upward stage as displayed in Figure 3 [19].

2) *Architecture transformations:* This stage is also known as the refactoring and restructuring process. The system's code, data, design plan, model, requirement structure, document, etc. [10], [11] are updated and enhanced to suit with new technology and environment [19] by also ensuring that the changes on the internal structure of the system will not affect its external behavior, but enhance the code readability [10], [11]. Some features and modules from the existing system may be included or excluded on the architecture of the re-engineered system depending on the customer demand [12].

3) Forward engineering : Forward engineering is the opposite side of reverse engineering. In other words, it can also be defined as the traditional software development in which conceptual design is elaborated and presented in more detail into detailed design and physical implementation [22].

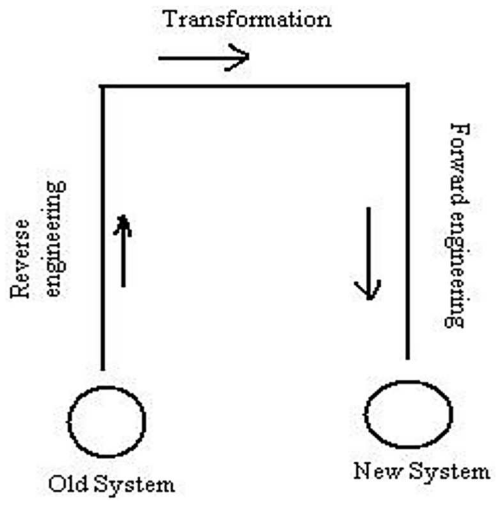

Fig. 3. Software Re-engineering Process [19]

4) *Re-engineering Methodologies:* Technological innovations and market demands are significantly changing in the last years. Companies need to be able to quickly adapt to these changes in order to be able to improve companies' performance. However, this fast pace changes have led the many businesses to face challenges in their daily operations, such as continuously increasing competition as well as late deliveries. A business reengineering project is one of the solutions that many companies could take to improve companies' performance including profits, cost reduction, its responsiveness towards competitive pressure and customer satisfaction. Past studies have proposed several methodologies that could support a system reengineering process.

Hammer & Champy's [23] methodology focuses on project analyzing and planning for a successful implementation based on the company's situation. They believe that unclear objective, lack of efficient management and lack of resistance to change are the main problems of a project failure. Davenport's methodology emphasizes innovating business process using information technology [24].

Hammer & Champy's methodology was synthesized in the following phases (1) business processes re-engineering introduction, (2) identification of business processes, (3) selecting the business process to be redesigned, (4) understand the selected business process, (5) redesigning the selected business process, and (6) implementation of the redesigned business process [24].

An agile unified methodology was proposed by Sahoo et al. [25] which involves two phases, the first one is planning phase which is divided into two tasks (1) identification and prioritization of the new requirements by applying information collection technique, and (2) release iterations planning as a guide for the iterative reengineering activities. Secondly is the iterative reengineering phase which consists of a series of iterations (reverse engineering, reincarnation and validation). An agile reengineering process framework named PARFAIT described by Cagnin et al. [26] uses the static structure of a rational unified process (RUP). It explains how to rapidly deliver and evolved version of the legacy system to the end user.

AbdEllatif et al. [27] explain ontology-based knowledge using a map methodology approach to overcome the business process reengineering obstacles.

Cagnin et al. [28] describes a segmentation reengineering process. The analysis model from the procedural legacy C code is recovered and then partially transformed using design pattern into a java object-oriented code. Migrating legacy object-oriented system using another reengineering process to component-based systems is described in [29], which suggests on how to improve code granularity and reusability using process metrics.

a) *Issues Identified in Existing Methodologies:* In a software system, about 70% of the resources are generally designated towards maintenance tasks. These are a huge challenge for the software community, in which tens of millions of lines of legacy code need to be modified during enhancement. Furthermore, most improvements are done depending on the experience and knowledge of the developer using an ad hoc approach. The issue becomes even serious when there are lack of documentation of the legacy system and the engineers involved in the re-engineering project do not have enough knowledge of the legacy system [30]. Software re-engineering intends to provide an engineering approach for improving the software.

Other factors that put negative effects on productivity are lack of team experience and high project staff loading [31]. These issues can lead to tasks overlap which can last for an indefinite time. Development projects generally manage the single of the software system within a specific duration of time, but maintenance is more challenging as it involves separate software system versions with customer involvement. The customer starts enhancement and every change made would involve a cycle that includes several steps such as to request for change, planning, implementation, verification, validation, as well as redocumentation. Software development process success is defined using estimated costs by the successful completion of the scope within the specified period. The same procedure to develop software cannot be used to maintain a system [32].

Besides that, Nasiri et al. [33] have stated in his study that a rational analysis needs to be carried out towards the organizational processes considering that failure in re-engineering processes before the new product delivery is a fact. The weight and impact of the system components and organization business processes make it challenging for managers and analysts to determine the best solution to be taken. Therefore, developing a new methodology that could support organizing the re-engineering process needs to be done, since the selection of a methodology before the implementation of the project could facilitate the issues above.

When it comes to Object-Oriented Programming (OOP), Cockburn and Highsmith argue that the practice of OOP Software Engineering offers a feasible solution to achieve a reliable system architecture. However, from the agile methodologies' iterative and incremental lightweight point of view, OOP would only effective at the program level. the development steps and the releasable functionalities must be aligned with the agile philosophy of the software development stakeholders [34].

The following table shows a comparison of issues identified from existing re-engineering methodologies.

TABLE I. RE-ENGINEERING ISSUES IDENTIFIED IN THE EXISTING METHODOLOGIES

Methodology	Issues Identified	Source
Ad-Hoc	Only 3-11% of the needed enhancements could be completed. Less than 20% of the code was running	[25]
PARFAIT (Rational Unified Process)	Only 0-40% of the needed enhancements could be completed.	[25], [26]
Agile Unified Methodology	Only 25-50% of the needed enhancements could be completed.	[25]
Agile Object-Oriented Re-engineering Methodology	Re-engineering processes are done manually without any engineering toolsets. Low quality of the system architecture.	
Agile Business Process Reengineering (Scrum)	Not suitable for all types of organization and structure.	[34]
Extreme Programming (XP)	Re-engineering processes are done manually without any engineering toolsets. Lack of software metrics related to XP projects Low Capability Model Integration Model Integration (CMMI) level	[35]
Waterfall	It does not allow constant update and iterations which are necessary for re-engineering, especially when developers need to familiarize with the system under consideration.	[36]

Software development companies that emphasize consumer interests, product quality, teamwork and individual adoptions that utilizes intriguing agile methodologies have gained profitability through their software product development practices [34]. However, a recent literature surveys shows that there is lack of a systematic re-engineering methodology [25].

III. CONCLUSION

In conclusion, re-engineering could address issues such as the software backlog problem, reduce software investment in organizations and increase the age of the software. This research examined the issues when it comes to software re-engineering an object-oriented system in terms of flexibility, simplicity, and effectiveness. It also addressed the limitations that software engineering teams generally face when there is no specific methodology being used in executing the re-engineering process. These ad-hoc approaches might lead to failure in re-engineering processes. Therefore, a systematic methodology is needed to find an appropriate solution in order to prevent re-engineering project failures.

REFERENCES

[1] S. A. Bohner, “Extending Software Change Impact Analysis into COTS Components,” in 27th Annual NASA Goddard / IEEE Software Engineering Workshop, SEW 2002, 2003, pp. 175–182.

[2] K. K. Holgeid, J. Krogstie, and D. I. K. Sjøberg, “Study of development and maintenance in Norway: Assessing the efficiency of information systems support using functional maintenance,” Inf. Softw. Technol., vol. 42, no. 10, pp. 687–700, 2000.

[3] I. Bassey, “Enhancing Software Maintenance via Early Prediction of Fault-Prone Object-Oriented Classes,” Int. J. Softw. Eng. Knowl. Eng., vol. 27, no. 4, pp. 515–537, 2017.

[4] K. Lano and H. Haughton, “Software Modernisation and Re-engineering,” in Financial Software Engineering, I. Mackie, Ed. Brighton: Springer Cham, 2019, pp. 117–130.

[5] Sujan, L. J. L., Vishwanath D. Telagadi, C. G. Raghavendra, B. M. J. Srujan, R. B. Vinay Prasad, B. D. Parameshachari, and K. L. Hemalatha. "Joint reduction of sidelobe and PMEPR in multicarrier radar signal." In Cognitive Informatics and Soft Computing, pp. 457-464. Springer, Singapore, 2021.

[6] N. Goyal and R. Srivastava, “Changeability Evaluation Model for Object-Oriented Software,” Int. J. Comput. Sci. Inf. Technol., vol. 9, no. 4, pp. 29–37, 2017.

[7] J. Singh, A. Gupta, and J. Singh, “Identification of requirements of software reengineering for JAVA projects,” in IEEE International Conference on Computing, Communication and Automation (ICCCA 2017), 2017, pp. 931–934.

[8] R. Khadka, B. V. Batlajery, A. M. Saeidi, S. Jansen, and J. Hage, “How do professionals perceive legacy systems and software modernization?,” in 36th International Conference on Software Engineering, 2014, no. 1, pp. 36–47.

[9] H. M. Sneed, Software Renewal: A Case Study, vol. 1, no. 3. 1984, pp. 56–63.

[10] W. Lin, “Web-based Software Reengineering - A case study on next-generation product-selection system,” KTH Royal Institute of Technology, 2017.

[11] Ijjina, Earnest Paul, and C. Krishna Mohan. "One-shot periodic activity recognition using convolutional neural networks." In 2014 13th International Conference on Machine Learning and Applications, pp. 388-391. IEEE, 2014.

[12] M. Muzammul, N. A. Prince, M. Awais, and A. Alvi, “Software re-engineering role in human-computer interaction (HCI) with quality assurance,” Int. J. Comput. Sci. Netw. Secur., vol. 18, no. 9, pp. 51–55, 2018.

[13] E. J. Chikofsky and J. H. Cross, “Reverse Engineering and Design Recovery: A Taxonomy,” IEEE Softw., vol. 7, no. 1, pp. 13–17, 1990.

[14] S. Demeyer, S. Ducasse, and O. Nierstrasz, Object-oriented Reengineering Patterns. Lulu.com, 2009.

[15] W. K. G. Assunção, R. E. Lopez-Herrejon, L. Linsbauer, S. R. Vergilio, and A. Egyed, “Reengineering legacy applications into software product lines: a systematic mapping,” Empir. Softw. Eng., vol. 22, no. 6, pp. 2972–3016, 2017.

[16] P. Quiroz-Palma, A. Suárez-Alarcón, A. Santamaría-Philco, W. Zamora, V. Garcia, and E. Vera-Burgos, “ITSIM : Methodology for Improving It Services. Case Study CNEL EP-Manabi,” in 2019 International Conference on Information Technology & Systems (ICTS). Advances in Intelligent Systems and Computing, 2019, vol. 918, pp. 199–209.

[17] A. Gade, S. Patil, S. Patil, and D. Pore, “Reverse Engineering of Object Oriented System,” Int. J. Sci. Res. Publ., vol. 3, no. 4, pp. 1–7, 2013.

[18] A. Jatain and D. Gaur, “Reverse engineering of object oriented system using hierarchical clustering,” Int. Arab J. Inf. Technol., vol. 15, no. 5, pp. 857–865, 2018.

[19] A. Kumar and B. S. Gill, “Maintenance vs. Reengineering Software Systems,” Glob. J. Comput. Sci. Technol., vol. 11, no. 23, pp. 58–64, 2011.

[20] P. V Nguyen, “The Study and Approach of Software Re-Engineering,” arXiv, p. 9, 2011.

[21] L. H. Rosenberg, “Software engineering.”

[22] M. Majthoub, M. H. Qutqui, and Y. Odeh, “Software Re-engineering: An Overview,” in 2018 8th International Conference on Computer Science and Information Technology (CSIT), 2018, pp. 266–270.

[23] M. Hammer and J. Champy, Reengineering the Corporation: A Manifesto for Business Revolution. Harper Business, 2006.

[24] L. P. Lopez-Arredondo, C. B. Perez, J. Villavicencio-Navarro, K. E. Mercado, M. Encinas, and P. Inzunza-Mejia, “Reengineering of the software development process in a technology services company,” Bus. Process Manag. J., vol. 26, no. 2, pp. 655–674, 2019.

[25] A. Sahoo, D. Kung, and S. Gupta, “An agile methodology for reengineering object-oriented software,” in International Conference on Software Engineering and Knowledge Engineering (SEKE), 2016, pp. 638–643.

[26] M. I. Cagnin, J. C. Maldonado, F. S. R. Germano, and R. D. Penteado, “PARFAIT: Towards a framework-based agile reengineering process,” in Agile Development Conference (ADC), 2003, pp. 22–31.

[27] M. AbdEllatif, M. S. Farhan, and N. S. Shehata, “Overcoming business process reengineering obstacles using ontology-based knowledge map methodology,” Futur. Comput. Informatics J., vol. 3, no. 1, pp. 7–28, 2018.

[28] Kumar, Thanikodi Manoj, Kasarla Satish Reddy, Stefano Rinaldi, Bidare Divakarachari Parameshachari, and Kavitha Arunachalam. "A low area high speed FPGA implementation of AES architecture for cryptography application." Electronics 10, no. 16 (2021): 2023.

[29] E. Lee, B. Lee, W. Shin, and C. Wu, “A Reengineering Process for Migrating from an Object-oriented Legacy System to a Component-based System,” in IEEE Computer Society’s International Computer Software and Applications Conference, 2003, pp. 336–341.

[30] S. Christa, V. Madhusudhan, V. Suma, and J. J. Rao, “Software Maintenance: From the Perspective of Effort and Cost Requirement,” in 2016 International Conference on Data Engineering and Communication Technology, 2016, vol. 2, pp. 603–611.

[31] C. Singh, N. Sharma, and N. Kumar, “Analysis of software maintenance cost affecting factors and estimation models,” Int. J. Sci. Technol. Res., vol. 8, no. 9, pp. 276–281, 2019.

[32] F. U. Rehman, B. Maqbool, M. Q. Riaz, U. Qamar, and M. Abbas, “Scrum Software Maintenance Model: Efficient Software Maintenance in Agile Methodology,” in 21st Saudi Computer Society National Computer Conference, 2018, pp. 1–5.

[33] S. Nasiri, M. J. Nasiri, and A. S. Azar, “Towards A Methodology for the Pre-Stage of Implementing a Reengineering Project,” J. Information, Knowledge, Manag., vol. 11, pp. 215–234, 2016.

[34] K. Bhavsar, V. Shah, and S. Gopalan, “Scrum: An Agile Process Reengineering in Software Engineering,” Int. J. Innov. Technol. Explor. Eng., vol. 9, no. 3, pp. 840–848, 2020.

[35] N. Ramadan Darwish, “Improving the Quality of Applying eXtreme Programming (XP) Approach,” Int. J. Comput. Sci. Inf. Secur., vol. 9, no. 11, pp. 16–22, 2011.

[36] J. Krüger and T. Berger, “Activities and Costs of Re-Engineering Cloned Variants Into an Integrated Platform,” in 14th International Working Conference on Variability Modelling of Software-Intersive Systems, 2020, p. 10.