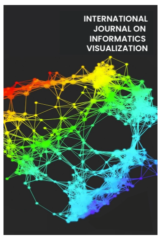
INTERNATIONAL JOURNAL
ON INFORMATICS VISUALIZATION

journal homepage : www.joiv.org/index.php/joiv



Overview of Software Re-Engineering Concepts, Models, and Approaches

Lim Fung Ji [a,*], Tan Bee Sian [b]

[a] *Department of Software Engineering and Technology, Tunku Abdul Rahman University of Management and Technology, Kuala Lumpur, Malaysia*
[b] *Department of Computer Science and Embedded Systems, Tunku Abdul Rahman University of Management and Technology, Kuala Lumpur, Malaysia*
*Corresponding author: *limfj@tarc.edu.my*

***Abstract*— Legacy systems face issues such as integrating new technology, fulfilling new requirements in the ever-changing environment, and meeting new user expectations. Due to the old complex system structure and technology, modification is hardly applied. Therefore, re-engineering is needed to change the system to meet new requirements and adapt to new technology. Software re-engineering generally refers to creating a new system from the existing one. Software re-engineering is divided into three (3) main phases: reverse engineering alteration and forward engineering. Reverse engineering examines, analyzes, and understands the legacy system in deriving the abstract representation of a legacy system; then, through necessary alterations such as restructuring, recording, and a series of forward engineering processes, a new system is built. This paper introduces the concepts of software re-engineering, including the challenges, benefits, and motivation for re-engineering. In addition, beginning with the traditional model of software re-engineering, this paper provides an overview of other models that provide different processes of software re-engineering. Each model has its unique set of processes for performing software re-engineering. Furthermore, re-engineering approaches show various ways of performing software re-engineering. Software re-engineering is a complex process that requires knowledge, tools, and techniques from different areas such as software design, programming, testing, et cetera. Therefore, monitoring the re-engineering process to meet the expectations is necessary.**

***Keywords*—Legacy system; software re-engineering models; reverse engineering; software re-engineering approaches.**

Manuscript received 25 Aug. 2024; revised 25 Oct. 2024; accepted 12 Nov. 2024. Date of publication 31 Jan. 2025.
International Journal on Informatics Visualization is licensed under a Creative Commons Attribution-Share Alike 4.0 International License.

I. Introduction

Software systems are dynamic and constantly evolving for various reasons, such as new user requirements, technological changes, and operation environments. However, legacy systems may require redesigning and reconstructing part(s) or the entire system to meet these evolving needs. As technology is developed speedily, for example, when smartphones become common, most application systems are built to be used in smartphone devices [1], and the rise of IoT (Internet of Things) technologies and their adoption in industries [2]. Therefore, software re-engineering comes into play for old systems to adapt to new technology environments. It involves studying the existing system to derive the abstract representation of the system and making the necessary changes to produce a new system that aligns with the evolving requirements. Not only does the software require reengineering, but hardware systems also require reengineering to improve their performance, for example, the renovation of the Ferrari 178H1 Load Handling Crane system to reduce the damage to the control unit [3].

The main motivation behind software re-engineering is to address issues such as lack of knowledge of the legacy system when the developer has left the company, the system is built using obsolete programming language, and the system's functionalities not being understood [4]. In addition, legacy systems may not be compatible with new technology; the monolithic architecture causes the creation of single-point failure, difficult to integrate with other systems [5].

These issues may arise due to the system's aging, making it challenging to maintain or update. Additionally, the need for re-engineering is also driven by enhancement purposes, such as improving the adaptability of web applications on different networks and devices [6]. This is particularly crucial in today's digital age, where users access web applications on various devices and platforms.

Moreover, software re-engineering may entail utilizing novel technologies like cloud technologies to improve the system's functionality [7]. Modernizing legacy systems also includes migrating system architecture, from monolithic to microservice architecture, to overcome and reduce the tight coupling between modules in monolithic architecture [5]. Modernization aims to prevent the legacy system from becoming obsolete and difficult to maintain [8].

A. Benefits, Challenges, and Risks in Software Re-engineering

The problems of legacy systems include using old technology, complex codes, and challenges to modify and expand. These characteristics of legacy systems may lead to higher maintenance costs, hiring of professionals to maintain old codes, and rarely providing the requirements to support the business [9]. Reengineering the existing software has various benefits compared to developing a new system from scratch. First, re-engineering reduces the maintenance cost of the legacy system and develops new software [10]. Secondly, re-engineering of databases to obtain the benefits of another type of database, for example, re-engineering conventional relational databases and migrating to NoSQL for storing unstructured data [11], and another example is adapting to fruitful computational resources in cloud-based servers [6]. Software re-engineering can also avoid cost and security risks caused by compliances with industry standards and outdated security measures [12].

Re-engineering software is a complicated and intricate process that deals with the intricacies of an existing system. This process can pose several challenges for the re-engineering team, for example, lack of knowledge about the legacy system, incomplete documentation, and the impact of system components [13]. One of the team's challenges is that users expect the new system to have functionality and performance similar to the old system, which requires additional effort in testing the software [14]. When upgrading a document information system, ensuring no data loss during the migration process is challenging, as the process is complex and requires careful planning and execution [15]. According to Zabidi et al. [13], the software community needs to deal with the modification of ten million legacy code statements for improvements, and the improvement depends on the knowledge and experiences of the developers in applying an ad hoc approach. It is also challenging for the development team to analyze the impact and risk of the modifications. Re-engineering is a complicated and time-consuming process that requires the consideration of downtime and data security for some industries, such as banks; this will lead to companies using outdated technology [16].

Embedded software re-engineering faces challenges in understanding its behavior. For instance, it is hard to comprehend a real-world implicit state machine from code. These are frequently accomplished by evaluating and setting static flags throughout the code so that changing a flag in one call alters the way the function behaves in a later call. Embedded control software shows much stateful behavior since discrete steps must be taken for every continuous process [17].

A software analyzer is required to perform software analysis during the reverse engineering stage of software re-engineering. Canfora et al. [18] studied reverse engineering challenges using reverse engineering tools such as software analyzers to perform static, dynamic, and historical analyses. The challenges are related to the ability requirements of the tools. In addition, the authors studied the challenges of developing software views. There are also challenges in transforming software architecture into modern architecture. The challenges of the re-engineering process are described in Table 1.

TABLE I
CHALLENGES FACED IN SOFTWARE RE-ENGINEERING

Tasks	**Challenges**
Static analysis	Ability to interpret non-compliant code and various language variations. Different characteristics in programming languages make it difficult to interpret the codes completely [18].
	Ability to extract information that's helpful for reverse engineering projects. In some cases, there is no need to have a complete abstract syntax tree representation of the program. It only requires relevant information for reverse engineering [18].
	Ability to retrieve the semantics of a program. The analyzer must be able to perform a set of analyses, which is absolutely required in reverse engineering [18].
Dynamic Analysis	It is crucial to be able to compile and execute a program when analyzing an intermediate version of it. This may not always be possible, especially if the program is incomplete and grabbed from a versioning system. Therefore, it is necessary to ensure that the tools and resources are available to analyze the program [18] comprehensively.
	Choosing the right inputs to execute a system is crucial when performing dynamic analysis, as the results heavily depend on the inputs used. Therefore, it is essential to carefully consider the approach taken when reverse engineering [18].
	Filtering and extracting relevant information from significant execution traces is a crucial challenge for better understanding [18].
	The reverse engineer may not have access to the system's code for certain types of systems but can only run the application. This is known as a black-box analysis, and the tool is required to perform it [18].
Historical Analysis	The tools should be able to integrate heterogeneous software repositories by linking changes and issues. This may face challenges such as the accuracy of information [18].
	The tools should be able to analyze and differentiate changes made to software artifacts [18].
	Grouping related changes together would enhance historical analysis by providing a comprehensive view of the development/maintenance activity [18].

TABLE I
CHALLENGES FACED IN SOFTWARE RE-ENGINEERING (CONTINUED)

Building Software Views	Powerful query languages are necessary to build different views from the information base populated by analyzers [18]. The ability to rebuild high-level information from low-level artifacts is crucial. Correctly visualizing abstracted information is crucial for software visualization as it significantly affects the effectiveness of program analysis and design recovery techniques for developers [18].
Architecture Modelling	When the requirements are not compatible with the design, it is time-consuming. This is due to the developer's insufficient understanding [19].
Architecture Refactoring	This activity is time-consuming, and it isn't easy to map the structures between different architectural patterns [19].
Planning on architecture migration	Planning is crucial in migrating software architecture, especially in determining the order of the migration process/activities [19].
Evolutionalization	If the architecture transformation process fails, a more complex recovery process will be met [19].
Process Support	The consideration of process, tools, and human decision support to transform legacy systems into mobile applications [20].
Evaluation on Usefulness	The evaluation of the effectiveness and usability of mobile applications (after conversion from legacy system) [20].

Transforming software architecture into modern architecture involves possible risks that need to be considered and evaluated before the re-engineering process. The risks are shown in Table 2.

TABLE II
RISK OF RE-ENGINEERING

Risks	Description
Costs	Monetary-related risks such as high maintenance costs, backup costs, and insignificant profit [21]
Time Delay	The reengineering process is slow due to insufficient knowledge of the existing system, budget run-off, management decisions, and mismatched architecture [21]. Testing consumes time and costs in reengineering [22].
Performance	Performance-related risks include inappropriate use of a reengineering approach, unmatched results, and inappropriate data restructuring [21].
User Satisfaction	Challenging in meeting user requirements in the competitive business environment [21].
Bugs	Users may suffer from bugs in the re-engineered product due to insufficient or inappropriate testing [14].

II. Material and Method

A. Software Re-Engineering Models

Re-engineering involves interpreting and analyzing the existing system code, implementing required changes, integrating revised modules, and testing them. The existing software system serves as the input for the re-engineering process, while the re-engineered and better-quality software serves as the output of re-engineering. The process of software re-engineering involves three main processes: reverse engineering, alteration, and forward engineering. Therefore, the re-engineering process can be illustrated and represented using a diagram shown in Fig. 1 [23].

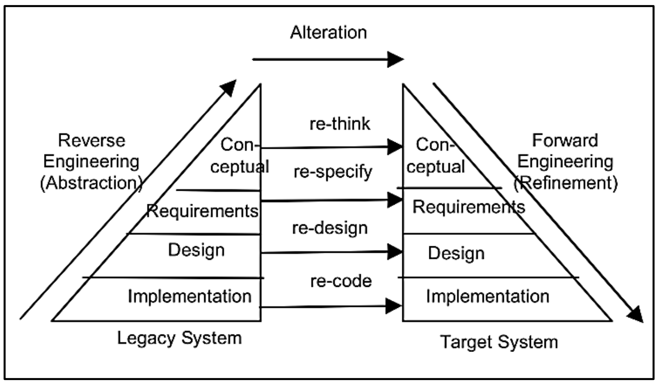


Fig. 1 General model of software re-engineering [23]

Reverse engineering is a technological procedure that involves the reverse study of a target product to extract design components like the flow of process, functional specifications, and organizational structure [24]. The system codes are analyzed during reverse engineering to produce a higher-level representation of the current system through a series of analytical works. The abstraction of system representation will be increased throughout the reverse engineering process. Then, forward engineering will continue to produce the new system through the four (4) levels of process, from conceptual to implementation [23]. During the re-engineering process, necessary alterations are applied to the system according to the requirements, which include re-coding, re-specifying, and re-design [25].

There are strategies to implement the changes to improve the system. One of the strategies is software revamping, which includes updating antiquated technological infrastructure to improve functionality, performance, and conformity with modern standards [26], for example, to overcome the limitations of monolithic gaming software architecture with cloud-oriented distributed engines [27]. Software restructuring or re-modularization is a regularly used strategy for system changes during software re-engineering. Software restructuring is to ameliorate the existing internal quality of the system without affecting the system's external behavior [28]. Another strategy is rearchitecting and modifying an application's architecture using more contemporary computer theories. For example, to change the software architecture to work in heterogeneous hardware architecture [29]. Replace strategy substitutes the entire system altogether [30]. These modification strategies enhance the legacy system in terms of codes, data structures, and architecture to meet new user requirements and organizational objectives.

Another rehosting strategy is to migrate the legacy system into a new environment with minimal code changes and functionality changes. Refactoring is reorganizing and enhancing already-written code to make it more readable and efficient without compromising its essential features. Transferring legacy programs to an alternative platform or infrastructure is known as re-platforming. When the legacy system cannot be modernized, a complete system replacement is considered [12].

Forward engineering produces a new system based on the output of reverse engineering. The forward engineering process starts from a high-level system representation and produces the new system's implementation-level codes. As the forward engineering process progresses, the system representation's abstraction level decreases as more information is added, moving down to the implementation level. This process involves creating a new system designed to meet the new requirements and specifications while ensuring that it is efficient, reliable, and maintainable.

The generic model shows the software re-engineering process based on the level of system representation. A software re-engineering process model indicates six (6) main processes [31], as shown in Fig. 2.

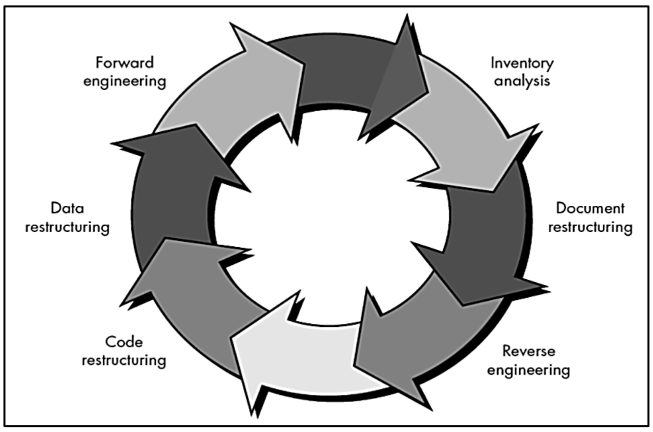


Fig. 2 Software re-engineering process model [31].

According to [32], the description of each process is as follows:

1) Inventory Analysis: Identify and analyze the information of the software program items based on a list of the required data to be collected.

2) Document Restructuring: Re-documentation is performed on an existing document. Normally, the existing document for a legacy system is weakly documented.

3) Reverse Engineering: Extract information regarding the existing system's architecture, design, and data.

4) Code Restructuring: Restructure the existing codes for maintainability, testability, and ease of understanding.

5) Data Restructuring: The existing data structure and model are analyzed and defined. Identify data attributes are identified.

6) Forward Engineering: Improve the quality of the existing system by recovering existing design information and performing necessary modifications or organization. This is also known as reclamation or renovation.

Another model for software re-engineering is the horseshoe process model. The horseshoe model depicts a software re-engineering process where reverse engineering derives the existing system's representations of different abstract levels through horizontal transformation. These representations act as input for generating new source code in the forward engineering process [33]. The model shows the process of re-engineering legacy systems into new ones. This model has been used as a reference for several works in re-engineering software architecture, user interface, and source code. Fig. 3 [33] shows the horseshoe re-engineering process model diagram.

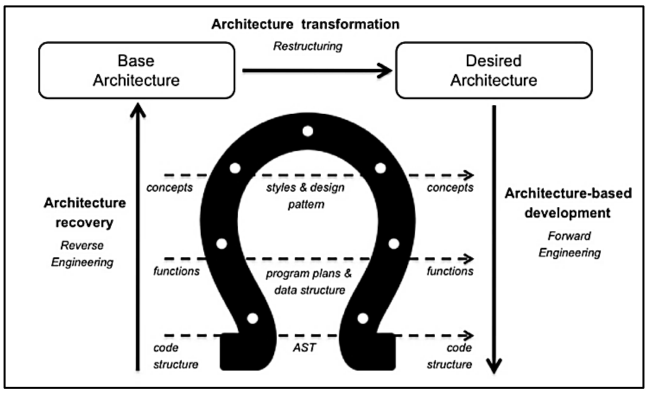


Fig. 3 Horseshoe model for re-engineering [33]

A phase model for re-engineering embedded automotive software is based on the SEI's horseshoe model but with additional phases: the preceding identification phase and a succeeding validation phase. The phase model concerns the function and source levels [34]. Fig. 4 [34] shows the phase model for re-engineering.

Based on the complexity index, the first stage identifies the functions with the lowest maintainability quality. The identified artifact then needs to be understood for its complexity. The bad smell pattern is used to detect structural deficiencies. Design principles and patterns are applied to create new solutions with a comprehensible software structure. The last stage is to verify the new solution against the requirements [34].

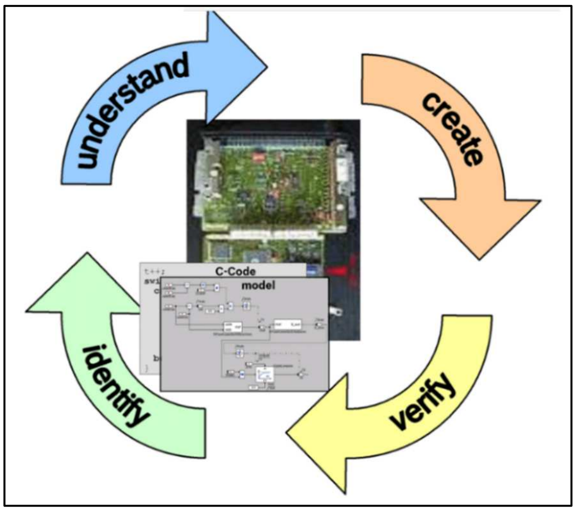


Fig. 4 Phase model for re-engineering [34]

In the hybrid re-engineering model [35], reverse and forward engineering are performed in parallel. For example, after the feasibility study, the next stage is requirement specification, where new requirements are gathered and mapped with the existing software requirement specification for requirement re-specification. Then, the new software specification will be used to map with the existing design for software re-design purposes. The process will then proceed to the following phases until the new system is derived. This model performs the whole re-engineering process (reverse engineering, modification, forward engineering) at every phase. The hybrid model is depicted in Fig. 5 [35].

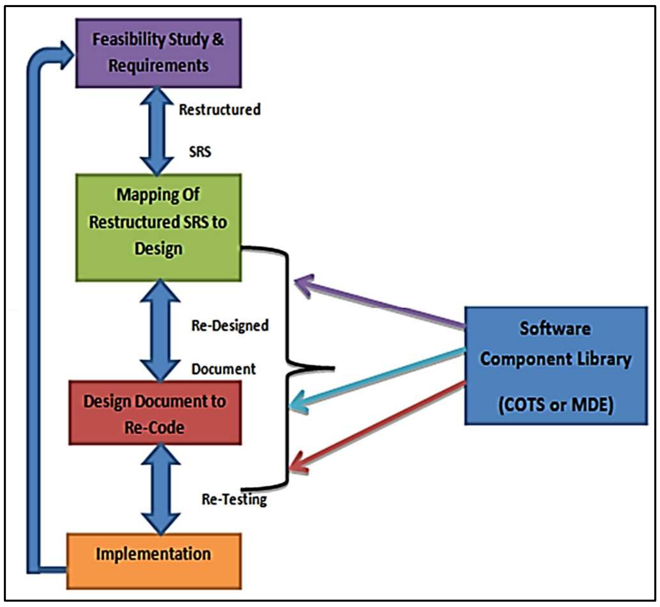

Feasibility Study & Requirements
Restructured
SRS
Mapping Of Restructured SRS to Design
Re-Designed
Document
Design Document to Re-Code
Re-Testing
Implementation
Software Component Library
(COTS or MDE)

Fig. 5 Hybrid re-engineering model [35]

Nowadays, the website is used for different purposes, such as online learning, shopping, and job seeking [36]. In addition, the web application for electronic banking provides more convenience to users in performing bank transactions [37]. Web re-engineering integrates new web applications while partially replacing existing legacy components. Dhiman has proposed a V model for web re-engineering [38]. The V model comprises various re-engineering processes such as web page, transaction, application migration, and graphic design re-engineering. These processes help reconstruct, refactor, and reengineer legacy web systems. In the V model, the product must go through the whole web development life cycle, passing the entire testing cycle. Fig. 6 [38] shows the V model for web re-engineering.

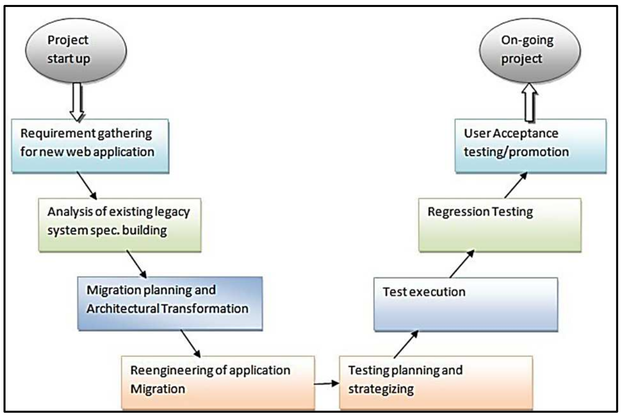

Project start up
Requirement gathering for new web application
Analysis of existing legacy system spec. building
Migration planning and Architectural Transformation
Reengineering of application Migration
Testing planning and strategizing
Test execution
Regression Testing
User Acceptance testing/promotion
On-going project

Fig. 6 V model for web re-engineering [38]

The left side of the V model describes the design and coding stages, while the right side shows the necessary validation stages for the products. The idea is based on the traditional V model of software development. The stages involved in the V model for web re-engineering (starting from top left) are as follows [38]:

1) Requirements gathering for a new web application: This stage collects the new requirements for the intended new web application through various techniques, such as interviews, discussions, model sites, etc. In addition, the new system objectives are determined through requirement analysis.

2) Analysis of existing legacy system/specification building: Analyze the existing system structure and functions and identify possible new requirements. Identify the possibility of component reuse.

B. Migration planning and architectural transformation

Perform planning on the resources and implementation plan for the migration and transformation to the target system.

1) Re-engineering of application migration: This is the stage where re-engineering works, such as architecture restructuring and code changes, are applied to reengineer the old system.

2) Test planning & strategizing: Created a test plan and determined the test strategy to evaluate the system's meeting of the new functional and non-functional requirements.

3) Test execution: Perform various tests according to the test plan.

4) Regression testing: Regression tests are performed to check the effects of change on the system.

5) User Acceptance testing: The system is being tested to meet users' expectations.

Using the V-model with the re-engineering process improves website maintainability and effectiveness through better validation, verification, and production. The Re-V model [8] was proposed for re-engineering a legacy system. The proposed model is based on the software development V-model, which shows the application of tests at different levels during software development. This model relates different levels of testing during the re-engineering process to test and validate the system. The model consists of two main parts: the deconstruction of the legacy system and the reconstruction of the new system.

Deconstruction of legacy system. The developers will try to understand the existing system's features, user stories, and functionalities by deconstructing the legacy system. The process is conducted until the code level of the existing system. Reconstruction is performed, which includes restructuring of the code. The legacy code will be transformed into new codes. Unit test cases are generated and performed during this stage to ensure the function unit meets the expectations. Furthermore, integration and acceptance tests validate the new system's functionalities. Fig. 7 [14] depicts the process in Re-V model.

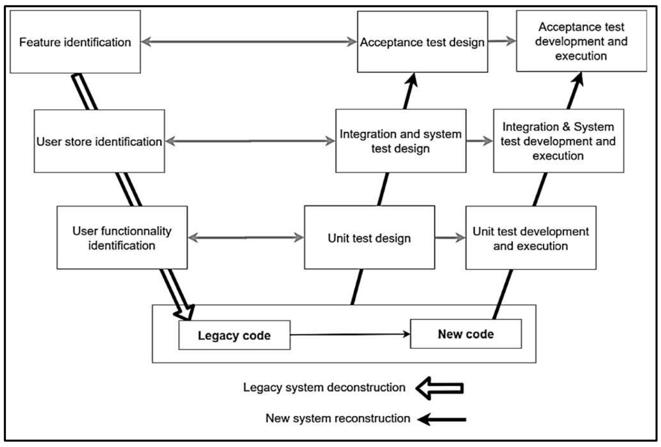

Feature identification
Acceptance test design
Acceptance test development and execution
User store identification
Integration and system test design
Integration & System test development and execution
User functionnality identification
Unit test design
Unit test development and execution
Legacy code
New code
Legacy system deconstruction
New system reconstruction

Fig. 7 Re-V model for legacy system re-engineering [14]

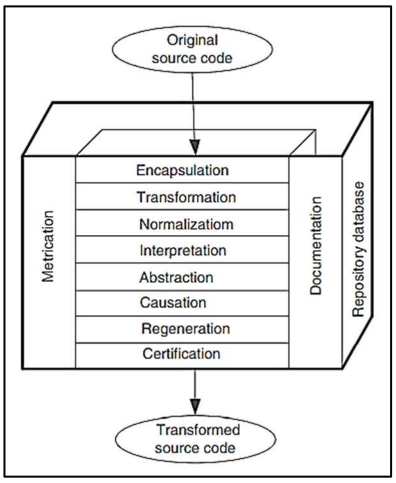


Fig. 8 Source code re-engineering reference model [39]

The Source Code Re-engineering Reference Model (SCORE/RM) in Fig. 8 [39] defines the framework for the process of software re-engineering. The framework defined by SCORE/RM (SCORE/RM framework) consists of four (4) elements, as in Table 3.

TABLE III
SCORE/RM ELEMENTS [39]

Elements	Description
Function	Represents the process of reengineering. Eight (8) functions form the eight (8) layers of SCORE/RM.
Metrication	A set of metrics is applied before and after each function layer is executed. These metrics are used to evaluate software improvement.
Documentation	Consists of information regarding the old and new systems, such as specifications, constraints, and implementation information
Repository Database	Stores items involved in re-engineering process such as source code, metrication, and documentation.

The eight (8) layers of the SCORE/RM function perform the re-engineering tasks to deliver the intended new system. The top six (6) layers are the reverse engineering process, and the bottom three (3) are the forward engineering process. The layers of functions are described in Table 4.

TABLE IV
SCORE/RM FUNCTION LAYERS [39]

Function Layer	Description
Encapsulation	This procedure is used to capture and manage the source code so that it may consistently serve as the base of reference for the subsequent layers
Transformation	This process alters the source code's control flow to organize and structure it.
Normalization	The stage where data and its structures are examined.
Interpretation	The starting process is to derive the first portion of the software.
Abstraction	Determine and identify the object hierarchy from the annotated and rationalized source code.
Causation	The forming of a hierarchy structure is examined to serve as the foundation for the requirements specification.
Regeneration	Implementation of a new system based on the functional specification and requirements.
Certification	Analysis of the new software demonstrates that it is compliant with the source code, meets the requirements, and operates as intended.

SCORE/RM systematically rationalizes and rebuilds software by comprehending its functions and requirements according to established software engineering practices.

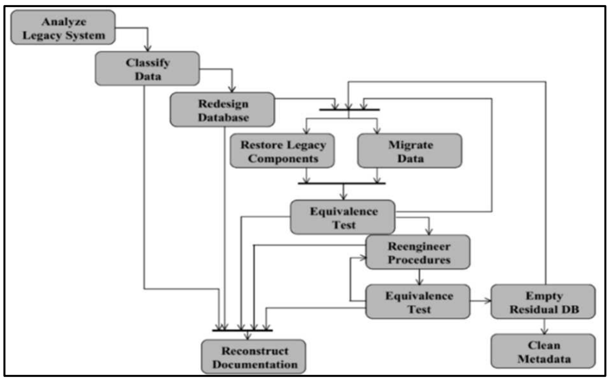


Fig. 9 Iterative re-engineering process model [25]

An iterative re-engineering process model is proposed by Bianchi et al. [40]. The re-engineering process in this model is predicated on the progressive evolution of a legacy system by the sequential re-engineering of the system's components, ensuring its coexistence through a series of transitional stages. The legacy and new systems can coexist during the re-engineering process. The legacy and new systems share metadata, data, and operative environment. Fig. 9 [40] shows the process model of iterative re-engineering.

Fig. 10 [40] shows the architecture during the reengineering process. The legacy components still exist and operate while the new components are built. With the coexistence of legacy and new systems as one architecture during reengineering, the iterative process model guarantees the continuity of system operation to satisfy user needs. In addition, the freezing time due to the reengineering process will be reduced.

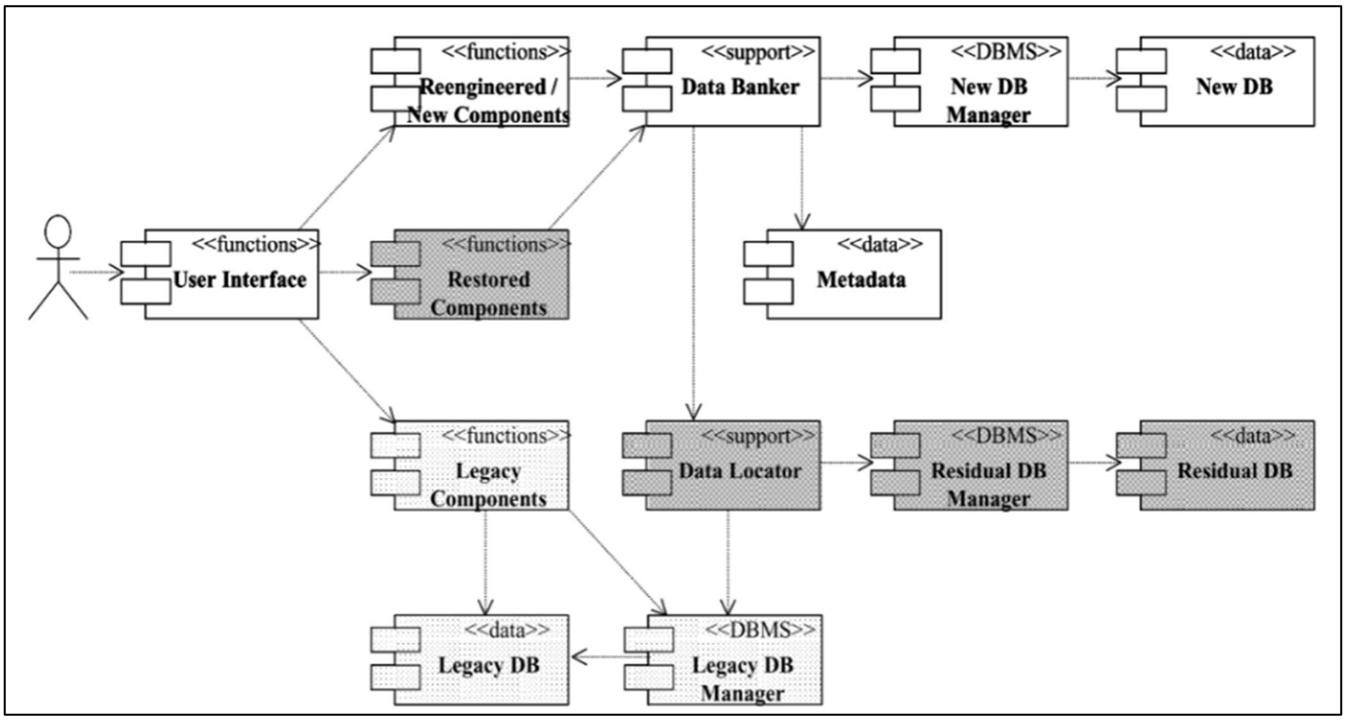


Fig. 10 System architecture during the re-engineering process [40]

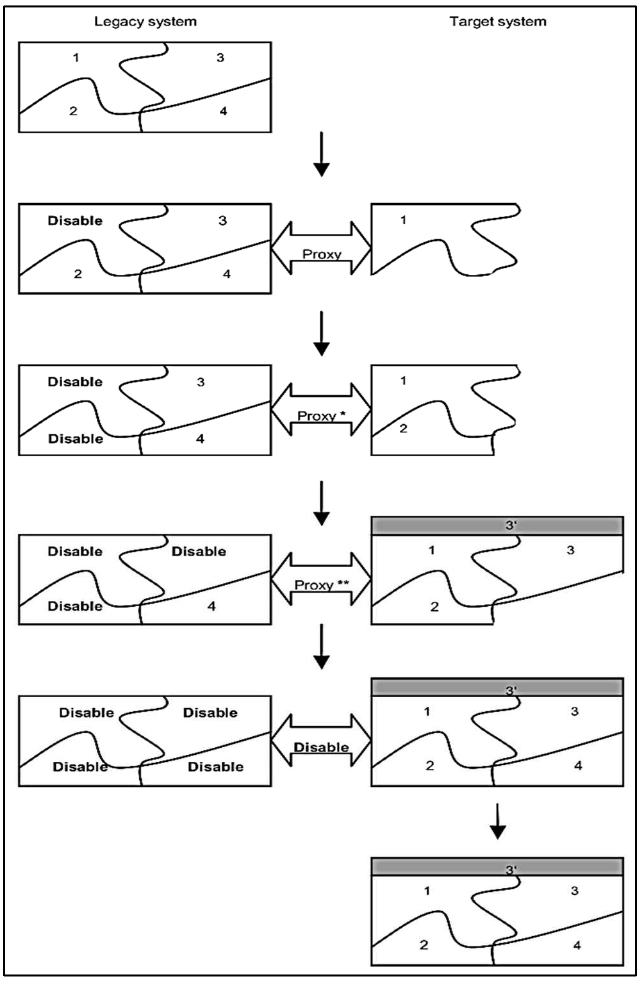


Fig. 11 Dual spiral re-engineering model [23]

Another software re-engineering model is the dual spiral re-engineering model. The dual spiral model's re-engineering process requires both old and new systems to work together, with the old system's functionalities decreasing. In contrast, the new system's functionalities increase throughout the process. Proxy is applied for integration purposes of both systems during the re-engineering process. The increasing and decreasing functionalities of the old and new systems are depicted in Fig. 11 [26], and the incremental and decremental spiral process is shown in Fig. 12 [23].

The dual spiral re-engineering model consists of three (3) main steps: the division of functionalities, the start of the spiral procedure, and the termination of the spiral procedure. In the first step, the functionalities of the old system are divided into sets of functionality. Then, these functionalities will be put into the decremental spiral process to remove functionalities in the old system and, simultaneously, the incremental spiral process to add new functionalities. This represents the transition of functionalities from the old to the new system.

The transition of functionalities may not be in the way of a one-to-one functionality transition; it could be a one-to-many or one transition. The re-engineering process (spiral process) will stop until all functionalities are transmitted into a new system and users are approved.

A cloud migration metamodel is proposed for migrating legacy systems to the cloud. This metamodel is created through a series of processes that gather knowledge from the software development cycle and different re-engineering models from multiple sources through a literature review. Then, the activities and tasks for various phases in the migration life cycle are analyzed, categorized, and validated [41]. The metamodel provides a framework for planning and managing the methodical, situation-specific system migration to the cloud.

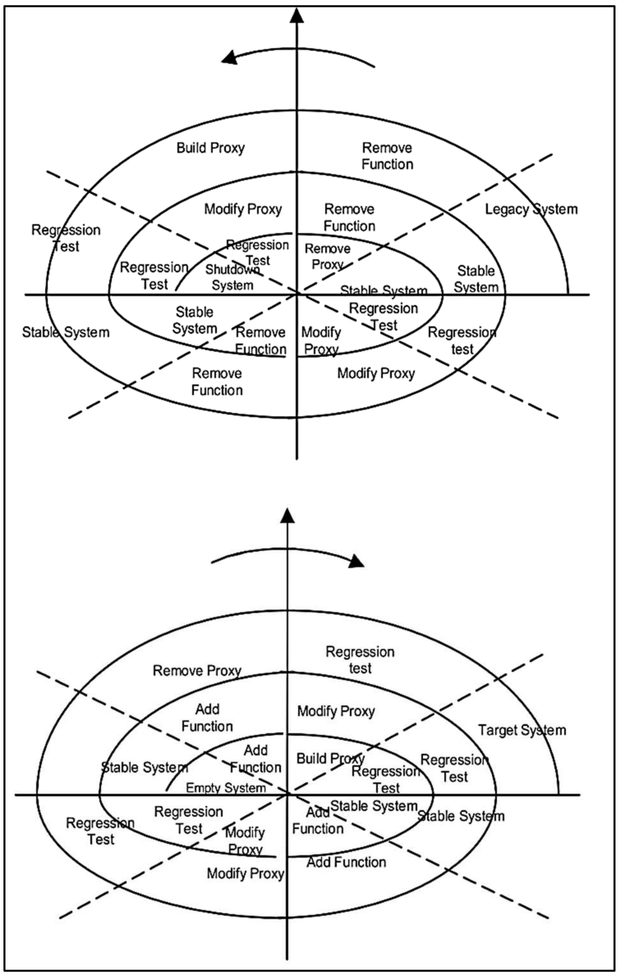


Fig. 12 Incremental and decremental spiral process [23]

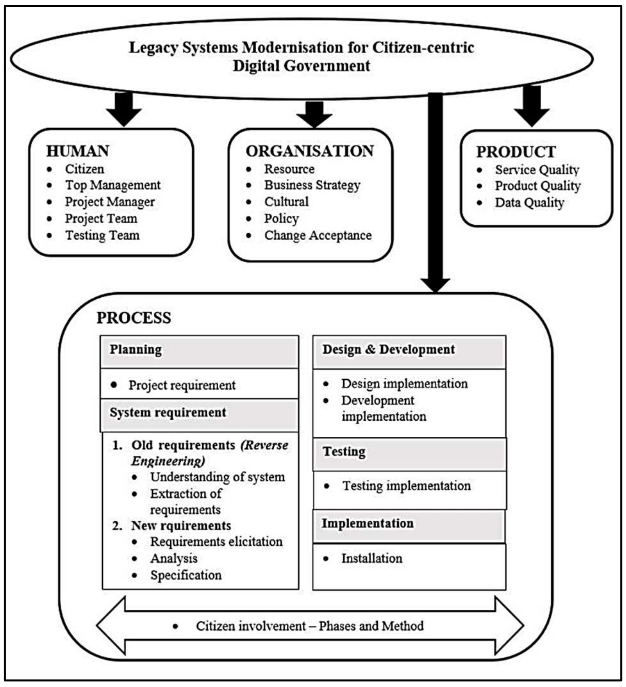


Fig. 13 The conceptual model of legacy systems' modernization for citizen-centric digital government [42]

Fig. 13 shows the conceptual model for modernizing the legacy system for citizen-centric digital government. This model is built based on the analysis of the concerned factors and provides guidelines for performing modernization [42]. The four primary aspects of the conceptual model are the organization, process, product, and human. Every aspect has some components and factors that are pertinent to its classification. The conceptual model has adopted four models, that are: Systems and Software Quality Model (ISO/IEC 25010:2011), Data Quality Model (ISO/IEC 25012:2008), Renaissance Method, and SERVQUAL Model to ensure the quality of the modernized system. SERQUAL is a research tool designed to assess service quality by gathering respondents' expectations and views along five service quality criteria [43]. The described models in this section have shared a common ground in the overall re-engineering process, which is to derive the representation of the existing system (reverse engineering), perform a necessary transformation (alteration), and produce a new system (forward engineering). However, with the fast advancement of technology, such as blockchain and artificial intelligence, these technologies affect how systems are built and maintained. Therefore, the discussed model may need to evolve in conjunction with the evolution of technology. The comparison of re-engineering models is shown in Table 5.

TABLE V
COMPARISONS OF SOFTWARE RE-ENGINEERING MODELS

Elements	Description	Limitations
Generic model	Showing the three (3) main processes of software re-engineering [23].	The re-engineering process involves the whole system; therefore, the target system is frozen during the re-engineering process [23].
Phase model	The model is derived from the SEI's horseshoe model but with additional phases: a preceding identification phase and a succeeding validation phase. It is for re-engineering automotive software [34].	This is different from the horseshoe model, which describes the re-engineering at the architecture, function, and source levels. The phase model focuses on the source and function levels [34].
Hybrid re-engineering model	A model where reverse engineering and forward engineering are performed in parallel [35].	No metrics are available to assess the performance and scalability of the new system [35].
V-model for re-engineering	V model for web re-engineering starts with requirement gathering for a new system, analyze, and transformation of the legacy system, followed by different types of tests [38]	The V-model of development has limitations, such as there is no iteration of the re-engineering process, and unsuitable for a system where requirements are rapidly changing [44]. Therefore, both

Elements	Description	Limitations
Re-V model for re-engineering	Based on the V-Model of development, the re-engineering of the legacy system with a relevant level of test design to validate the new system at different levels of testing [14]	the V-model and Re-V model for re-engineering may have similar limitations
SCORE/RM model	The model reengineers the legacy system from the source code through eight (8) layers of the process [39].	To incorporate supporting tools used in formal specifications into SOCRE/RM, they need to be improved [39].
Iterative re-engineering process model	The re-engineering model allows the coexistence of old and new systems during re-engineering [40].	Need to build a data locator and manage the residual database during the re-engineering process [40].
Dual spiral re-engineering model	It consists of a two-spiral process, which transmits the functions of the old system into the new system's functions [23].	The model divides the system based on functionality but not module [23]. Therefore, it may take more work to divide the legacy system's functionality with convoluted codes that deviate from design principles.
Conceptual model for legacy system modernization.	A conceptual model guiding the legacy systems' modernization towards the citizen-centric digital government. [42].	To give organizations consistent direction, the modernization of legacy systems must consider all pertinent elements and factors from many perspectives, including citizen-centric requirements [42].

III. Results and Discussion

A. *Software Re-Engineering Approaches*

The first approach discussed is the Big Bang approach. This approach replaces the existing system with the new one at once. The Big Bang approach is typically required when there is a need for direct problem-solving [45]. However, the strength of applying this approach is that it requires a huge upfront investment and a long time for return [46]. The incremental approach breaks the re-engineering process into different phases, which are executed gradually, and each phase is to re-engineer certain parts of the software.

The incremental approach shows a low-risk way of re-engineering but needs a well-defined way to ensure it is performed in a stable, less risky, and cost-efficient way [46]. The iterative approach splits the system into different packages, and re-engineering activities are performed on the parcel to derive the new system package. The process continues until all packages are re-engineered. This approach requires the existence of documentation and code [47].

Partial approach reengineering separates the system into part(s) to be re-engineered and part(s) not to be re-engineered. After the reengineering process, the reengineered parts are combined with the non-reengineered parts to become the new system [48]. The approaches discussed by several authors [46]-[48] reengineered the legacy system in a division-oriented way, which divided it into parts.

Wrapping [16] is another approach that encloses the current code within a contemporary interface. External entities communicate and connect to the system through the interfaces provided in the wrapper. A re-engineering case of Java-based Object-Oriented software is performed by applying the Scrum approach of the Agile framework. This approach performs the re-engineering tasks within different sprints to re-engineer the selected classes in the software program. For example, reverse engineering, alteration, and forward engineering tasks are performed in Sprint 1.0. The objectives of this approach are to reduce the complexity and improve the maintainability of the software [49].

A model-driven re-engineering approach, the MLSAC (Migration of Legacy Software Applications to the Cloud) framework, is applied in the re-engineering of software in cloud computing. MLSAC pulls together various cohesive, empirically validated cloud-specific approach fragments from research and application. From this collection, a meta-model (or meta-method) and matching instantiation guidelines are built. Metamodeling serves as a representational layer for re-engineering methodologies and is the foundation of MLSAC. The principles of the design science research approach guide the creation and assessment of MLSAC. In a particular case of re-engineering to cloud platforms, the metamodel can also be used to develop and manage custom re-engineering techniques [50].

Another approach modernized and transformed a legacy system into a mobile-based application through processes based on three crucial aspects: process activities, process view, and process support. These aspects include modernization activities such as planning, modeling, transforming, and evaluating [16].

A previous study proposed a re-engineering framework for open-source software using the decision tree approach as cited in [51]. This approach provides a guideline for identifying re-engineering requirements. Re-engineering approaches transform the existing legacy system into an enhanced new system. In other words, these approaches are considered part of the legacy system modernization approach. Re-engineering approaches can be merged to fulfill the modernization needs but must include consideration of risk, budget, and time [23].

IV. Conclusion

From the reengineering models and approaches discussed in this paper, software reengineering involves multiple processes and activities to transform the legacy system into a new system. The reengineering process is complex; it requires the developer/engineer to have knowledge and skills in different perspectives of technologies. In addition, software reengineering must consider costs, resources, risks, and organizational goals.

References

[1] P. Huriati, H. Azmi, Y. Wati, D. Meidelfi, and T. Lestari, "Black box testing on the online quiz application using the Equivalence Partitions method," *International Journal of Advanced Science Computing and Engineering*, vol. 2, no. 2, pp. 51–56, Aug. 2020, doi:10.30630/ijasce.2.2.48.

[2] A. Katiyar and P. Kumar, "A Review of Internet of Things (IoT) in Construction Industry: Building a Better Future," *International Journal of Advanced Science Computing and Engineering*, vol. 3, no. 2, pp. 65–72, Aug. 2021, doi: 10.30630/ijasce.3.2.53.

[3] L. Markis, P. S. Wardana, and A. Syawaldipa, "Renovation of Crane Control System of Reach Stalker Ferari 178h1 Using Avr Atmega2560," *International Journal of Advanced Science Computing and Engineering*, vol. 4, no. 2, pp. 113–120, Aug. 2022, doi:10.30630/ijasce.4.2.86.

[4] M. Nagl and B. Westfechtel, "Reverse and Reengineering for Old Systems is Seldom Complete," 2024, pp. 179–198. doi: 10.1007/978-3-031-51335-0_9.

[5] Q. Z. Ang, P. C. Yau, C. S. Sum, Q. Cao, and D. Wong, "Legacy Modernization: A Cloud Migration Strategy with Serverless Microservice Architecture," in *Proceedings - 2023 International Conference on Intelligent Computing and Control, IC and C 2023*, Institute of Electrical and Electronics Engineers Inc., 2023, pp. 59–63. doi: 10.1109/IC-C57619.2023.00017.

[6] K. An, "Enhancing Web App Execution with Automated Reengineering," in *The Web Conference 2020 - Companion of the World Wide Web Conference, WWW 2020*, Association for Computing Machinery, Apr. 2020, pp. 274–278. doi: 10.1145/3366424.3382087.

[7] O. Kernytskyy, A. Kernytskyy, and V. Teslyuk, "The Synthesis Method for Specifications and Requirements in the Process of IT Project Reengineering," in *International Scientific and Technical Conference on Computer Sciences and Information Technologies*, Institute of Electrical and Electronics Engineers Inc., 2023. doi:10.1109/CSIT61576.2023.10324175.

[8] N. Somogyi and G. Kovesdan, "Software Modernization Using Machine Learning Techniques," *SAMI 2021 - IEEE 19th World Symposium on Applied Machine Intelligence and Informatics, Proceedings*, Institute of Electrical and Electronics Engineers Inc., Jan. 2021, pp. 361–365. doi: 10.1109/SAMI50585.2021.9378659.

[9] R. Capuano and H. Muccini, "A Systematic Literature Review on Migration to Microservices: A Quality Attributes perspective," in *2022 IEEE 19th International Conference on Software Architecture Companion, ICSA-C 2022*, Institute of Electrical and Electronics Engineers Inc., 2022, pp. 120–123. doi: 10.1109/ICSA-C54293.2022.

[10] J. Singh, K. S. Dhindsa, and J. Singh, "Software quality improvement and validation using reengineering," *Journal of Engineering Research (Kuwait)*, vol. 9, no. 4 A, pp. 59–73, 2021, doi: 10.36909/jer.

[11] B. Namdeo and U. Suman, "Cost Model for Database Reengineering from RDBMS to NoSQL," in *4th International Conference on Recent Trends in Computer Science and Technology, ICRTCST 2021 - Proceedings*, Institute of Electrical and Electronics Engineers Inc., 2022, pp. 164–168. doi: 10.1109/ICRTCST54752.2022.9781890.

[12] T. Khan. "Legacy Application Modernization: A Comprehensive Approach to Modernize Your Business." IBM Blog. https://www.ibm.com/blog/legacy-application-modernization/ [accessed Aug. 24, 2024].

[13] W. A. Zabidi, M. E. Rana, and C. R. A. P. Ramachandiran, "Issues and Challenges in Existing Re-engineering Methodologies of Object Oriented Systems," in *MysuruCon 2022 - 2022 IEEE 2nd Mysore Sub Section International Conference*, Institute of Electrical and Electronics Engineers Inc., 2022. doi:10.1109/MysuruCon55714.2022.9972365.

[14] H. Khodabandehloo, B. Roy, M. Mondal, C. Roy, and K. Schneider, "A Testing Approach while Re-engineering Legacy Systems: An Industrial Case Study," in *Proceedings - 2021 IEEE International Conference on Software Analysis, Evolution and Reengineering, SANER 2021*, Institute of Electrical and Electronics Engineers Inc., Mar. 2021, pp. 600–604. doi: 10.1109/SANER50967.2021.00073.

[15] D. Ramos-Vidal, "Reengineering legacy document information systems: Challenges and solutions," in *ACM International Conference Proceeding Series*, Association for Computing Machinery, Jun. 2023, pp. 286–291. doi: 10.1145/3593434.3593436.

[16] N. Jomhari, N. A. A. Alias, A. A. A. Ellah, A. A. Magableh, and E. M. Ghazali, "A Multi-Criteria Decision-Making for Legacy System Modernization With FUCOM-WSM Approach," *IEEE Access*, vol. 12, pp. 48608–48619, 2024, doi: 10.1109/access.2024.3383917.

[17] W. Said, J. Quante, and R. Koschke, "Do Extracted State Machine Models Help to Understand Embedded Software?," in *2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC)*, 2019, pp. 191–196. doi:10.1109/ICPC.2019.00038.

[18] G. Canfora, M. di Penta, and L. Cerulo, "Achievements and challenges in software reverse engineering," *Communications of the ACM*, vol. 54, no. 4. pp. 142–151, Apr. 2011. doi: 10.1145/1924421.1924451.

[19] N. Chondamrongkul, J. Sun, and I. Warren, "Software Architectural Migration: An Automated Planning Approach," *ACM Trans. Softw. Eng. Methodol.*, vol. 30, no. 4, Jul. 2021, doi: 10.1145/3461011.

[20] A. Ahmad, A. Alkhalil, A. B. Altamimi, K. Sultan, and W. Khan, "Modernizing Legacy Software as Context—Sensitive and Portable Mobile-Enabled Application," *IT Professional*, vol. 23, no. 1, pp. 42–50, 2021, doi: 10.1109/MITP.2020.2975997.

[21] D. A. I. Kreedy, "Estimation of Risk in Software Re-engineering Projects," *International Journal of Scientific and Research Publications (IJSRP)*, vol. 10, no. 06, pp. 799–802, Jun. 2020, doi:10.29322/ijsrp.10.06.2020.p10293.

[22] I. Jovanovikj, E. Yigitbas, A. Nagaraj, A. Anjorin, S. Sauer, and G. Engels, "Validating Test Case Migration via Mutation Analysis," *Proceedings of the IEEE/ACM 1st International Conference on Automation of Software Test*, pp. 31–40, Oct. 2020, doi:10.1145/3387903.3389319.

[23] X. Yang, L. Chen, X. Wang, and J. Cristoforo, "A Dual-Spiral Reengineering Model for Legacy System," *TENCON 2005 - 2005 IEEE Region 10 Conference*, 2005, pp. 1–5. doi:10.1109/tencon.2005.301068.

[24] N. Team "Reverse Engineering," in Handbook for CTFers, Singapore: Springer Nature Singapore, 2022, pp. 295–427. doi: 10.1007/978-981-19-0336-6_5.

[25] D. Bouchiha, "Reengineering Legacy Systems Towards New Technologies," *Encyclopedia of Information Science and Technology*, Fifth Edition, IGI Global, 2020, pp. 1214–1230. doi: 10.4018/978-1-7998-3479-3.ch084.

[26] H. Hadawale, "Revamp Your Existing Software to Save Infrastructure Cost," LinkedIn, [Online]. Available: https://www.linkedin.com/pulse/revamp-your-existing-software-save-infrastructure-cost-hadawale/. [Accessed: Aug. 24, 2024].

[27] L. de Giovanni et al., "Revamping Cloud Gaming With Distributed Engines," *IEEE Internet Computing*, vol. 26, no. 6, pp. 88–95, 2022, doi: 10.1109/MIC.2022.3172105.

[28] B. G. Varghese R, K. Raimond, and J. Lovesum, "A novel approach for automatic remodularization of software systems using extended ant colony optimization algorithm," *Information and Software Technology*, vol. 114, pp. 107–120, 2019, doi:10.1016/j.infsof.2019.06.002.

[29] H. Andrade, C. Berger, I. Crnkovic, and J. Bosch, "Principles for Re-architecting Software for Heterogeneous Platforms," *2020 27th Asia-Pacific Software Engineering Conference (APSEC)*, 2020, pp. 405–414. doi: 10.1109/apsec51365.2020.00049.

[30] Luvina Software. Software Re-Engineering: A Lifesaver for Legacy Systems. https://luvina.net/en/software-re-engineering [accessed Aug. 23, 2024].

[31] A. S. Abbas, W. Jeberson, D. W. Jeberson, and V. V Klinsega, "Proposed Software Re-engineering Process That Combine Traditinal Software Reengineering Process With Spiral Model Proposed Software Re-engineering Process That Combine Traditinal Software Re-engineering Process With Spiral Model," *International Journal of Advanced Research in Computer Science*, vol. 4, no. 2, 2013.

[32] A. Kumar, "Software Re-engineering Process Model," *International Journal of Science and Research*, pp. 2319–7064, 2019, doi:10.21275/SR21101152601.

[33] C. J. Fernández Candel, J. García Molina, F. J. Bermúdez Ruiz, J. R. Hoyos Barceló, D. Sevilla Ruiz, and B. J. Cuesta Viera, "Developing a model-driven reengineering approach for migrating PL/SQL triggers to Java: A practical experience," *Journal of Systems and Software*, vol. 151, pp. 38–64, May 2019, doi: 10.1016/j.jss.2019.01.068.

[34] A. Thums and J. Quante, "Reengineering embedded automotive software," *2012 28th IEEE International Conference on Software Maintenance (ICSM)*, 2012, pp. 493–502. doi:10.1109/ICSM.2012.6405312.

[35] S. Tarar, "Design Paradigm and Risk Assessment of Hybrid Re-engineering with an approach for development of Re-engineering

Metrics," *International Journal of Software Engineering & Applications*, vol. 3, no. 1, pp. 27–36, Jan. 2012, doi:10.5121/ijsea.2012.3103.

[36] M. Madiah, K. X. Ng, Y. W. Tan, Z. H. Tan, Z. T. Chong, and J. X. Chan, "Wix for Web Development and the Application of the Waterfall Model and Project Based Learning for Project Completion: A Case Study," *Journal of Informatics and Web Engineering*, vol. 3, no. 2, pp. 212–228, Jun. 2024, doi: 10.33093/jiwe.2024.3.2.16.

[37] Y. H. Tay, S. Y. Ooi, Y. H. Pang, Y. H. Gan, and S. L. Lew, "Ensuring Privacy and Security on Banking Websites in Malaysia: A Cookies Scanner Solution," *Journal of Informatics and Web Engineering*, vol. 2, no. 2, pp. 153–167, Sep. 2023, doi: 10.33093/jiwe.2023.2.2.12.

[38] P. Dhiman, "Unified V- Model Approach of Re-Engineering to reinforce Web Application Development," *IOSR Journal of Computer Engineering*, vol. 15, no. 6, pp. 09-17, 2013, doi:10.9790/0661-1560917.

[39] A. Colbrook, C. Smythe, and A. Darlison, "Data abstraction in a software re-engineering reference model," *Proceedings. Conference on Software Maintenance 1990*, 1990, pp. 2–11. doi:10.1109/ICSM.1990.131314.

[40] A. Bianchi, D. Caivano, V. Marengo, and G. Visaggio, "Iterative reengineering of legacy systems," *IEEE Transactions on Software Engineering*, Mar. 2003, pp. 225–241. doi:10.1109/TSE.2003.1183932.

[41] P. Pamami, A. Jain, and N. Sharma, "Cloud Migration Metamodel : A framework for legacy to cloud migration," *2019 9th International Conference on Cloud Computing, Data Science & Engineering (Confluence), 2019*, pp. 43–50. doi:10.1109/confluence.2019.8776983.

[42] H. Abu Bakar, R. Razali, and D. Jambari, "Legacy Systems Modernisation for Citizen-Centric Digital Government: A Conceptual Model," *Sustainability*, vol. 13, p. 13112, Nov. 2021, doi:10.3390/su132313112.

[43] A. Rolo, R. Alves, M. Saraiva, and G. Leandro, "The SERVQUAL instrument to measure service quality in higher education – A case study," *SHS Web of Conferences*, vol. 160. EDP Sciences, Les Ulis, 2023. doi:10.1051/shsconf/202316001011.

[44] D. Kumar. "Software Engineering | SDLC V-Model - GeeksforGeeks."GeeksforGeeks. https://www.geeksforgeeks.org/software-engineering-sdlc-v-model/ [accessed Aug. 22, 2024].

[45] S. Mauluddin and R. Sidik, "Reverse Engineering in Student Mark Recapitulation Application," *IOP Conference Series: Materials Science and Engineering*, Institute of Physics Publishing, Nov. 2019. doi: 10.1088/1757-899X/662/2/022097.

[46] G. Zhang, L. Shen, X. Peng, Z. Xing, and W. Zhao, "Incremental and iterative reengineering towards Software Product Line: An industrial case study," *2011 27th IEEE International Conference on Software Maintenance (ICSM)*, 2011, pp. 418–427. doi:10.1109/ICSM.2011.6080809.

[47] V. Durelli, R. Penteado, S. Borges, and M. Viana, "An Iterative Reengineering Process Applying Test-Driven Development and Reverse Engineering Patterns," *INFOCOMP*, vol. Special Edition, pp. 1–8, Jan. 2010.

[48] E. J. Byrne and D. A. Gustafson, "A software re-engineering process model," *[1992] Proceedings. The Sixteenth Annual International Computer Software and Applications Conference*, 1992, pp. 25–30. doi: 10.1109/CMPSAC.1992.217608.

[49] J. Singh, K. S. Dhindsa, and J. Singh, "Performing Reengineering using Scrum Agile Framework," in *2020 Indo – Taiwan 2nd International Conference on Computing, Analytics and Networks (Indo-Taiwan ICAN)*, 2020, pp. 33–35. doi:10.1109/Indo-TaiwanICAN48429.2020.9181328.

[50] M. Fahmideh, J. Grundy, G. Beydoun, D. Zowghi, W. Susilo, and D. Mougouei, "A model-driven approach to reengineering processes in cloud computing," *Information and Software Technology*, vol. 144, p. 106795, 2022, doi: 10.1016/j.infsof.2021.106795.

[51] J. Singh, K. Singh, and J. Singh, "Reengineering framework for open source software using decision tree approach," *International Journal of Electrical and Computer Engineering*, vol. 9, no. 3, pp. 2041–2048, Jun. 2019, doi:10.11591/ijece.v9i3.pp2041-2048.