Assessing the Impact of Re-engineering on Software Security

Yogesh K [1.]
Department of Computer Science, University of Texas at Arlington United States of America yxk9640@mavs.uta.edu

Shreya Reddy Gade [2]
Department of Computer Science, University of Texas at Arlington United States of America gade.shreyareddy@gmail.com

Sivathejeswara K[3]
Department of Computer Science, University of Texas at Arlington United States of America sivatheja.k@gmail.com

Nalini Pasupuleti[4]
Department of Computer Science, University of Texas at Arlington United States of America nalini5435@gmail.com

Abstract—*In today's technology-driven world, software is ubiquitous, and security is a primary concern for software developers, users, and organizations. Software reengineering, the process of modifying and improving existing software, is often necessary to keep up with changing technologies, user needs, and business requirements. However, reengineering can also introduce new security vulnerabilities, which can have severe consequences for software systems and their users. This project aims to assess the impact of software reengineering on security and survey best practices and guidelines for mitigating security risks during the reengineering process.*

Index Terms—*Software Re-engineering, Software Security, Case-Study, Software Methodology, Security Assessment*

1. INTRODUCTION

Software systems are vulnerable to various security threats, including malware, viruses, hacking, and cyberattacks. These threats can cause significant damage to software systems, result in data breaches, and compromise user privacy and security. Therefore, it is essential to ensure that software systems are secure and resilient to potential security threats. Software reengineering is an essential process for maintaining and improving the quality and functionality of software systems. However, the process of reengineering can also introduce new security vulnerabilities that can compromise the security of software systems. For example, software reengineering may involve modifying and restructuring the software code, which can result in the introduction of new security vulnerabilities that were not present in the original code. Therefore, it is essential to assess the impact of software reengineering on security and survey best practices and guidelines for mitigating security risks during the reengineering process.

1.1. Significance:

1.1.1. Understanding Reengineering:

To improve its functionality, maintainability and reliability software reengineering is a process of changing or redesigning the current software. Software reengineering may be necessary when business needs change, technology advances, or software needs to be migrated to new platforms and architectures. The goal of software reengineering is to prepare for enhancements. Take into consideration areas where existing software can be improved and new features can be added to meet future requirements. Software reconfiguration will allow developers to prepare for future development, ensuring that the software can always meet the requirements of businesses and their users. In addition, software reengineering is aimed at improving maintenance. Over time, changes in technology and business needs can make software maintenance difficult and expensive. Software reengineering helps reduce maintenance costs by improving the structure and organization of code, making it easier to understand and change. On a more superficial level, software reengineering also helps when migrating to a new platform or architecture. This goal focuses on ensuring a smooth migration process by identifying potential issues and providing a migration roadmap. Software reengineering can therefore help improve the reliability of existing software, including identifying and fixing bugs, improving error handling, and optimizing performance.This allows companies to be sure that their software is reliable, efficient and cost effective.

From our point of view, software reengineering is a useful process for improving the functionality, maintainability, and reliability of existing software. It helps companies meet changing demands and stay competitive in an ever-evolving technological environment.

1.1.2. Software security:

All software is potentially vulnerable to attack, and to minimize the bugs in software that make it more vulnerable to attack, proactively design, build, and test security software. approach is required. The methodology used is known as software security [2]. The software has security flaws, such as implementation bugs such as buffer overflows and design flaws such as inconsistent error handling. Malicious intruders can exploit these vulnerabilities to hack your system, cause damage, or steal sensitive data. As a result, not only is there a growing need to mitigate these risks, but sound software engineering practices must be followed to prioritize security from the start. To this end, a comprehensive objective risk analysis and testing of all software components is performed. This allows developers to identify potential vulnerabilities early in the development cycle and proactively address them before they become manifest. Threats that affect systems are language-based bugs such as buffer overflow attacks. To prevent such attacks, developers should use secure coding techniques such as input validation and output encoding.

1.2. Problem:

From the papers [2], [3] and the sections above, we can see that security risks exist in software, especially older versions. Receive required security updates and patches from vendors[3]. This can leave your system vulnerable to known vulnerabilities and bugs that can be exploited by attackers. Additionally, older software may not be compatible with the latest security protocols and updates. For example, outdated cryptographic algorithms may not provide adequate protection against modern attacks and may compromise sensitive data. Apart from the risks present in legacy systems, there are potential security risks in reworked systems.

Therefore, a system rebuild or redesign is required, which can be achieved by a system rebuild. However, the methods adapted and implemented during reengineering can make the system vulnerable, so it is essential to follow the reengineering and security model chosen. This article examined several software reengineering and software security models, and based on that research, evaluated and explored combinations of software reengineering and security models.

2. RELATED WORK

Recent research on software reengineering includes an empirical study on Legacy System Reengineering Plan, Legacy System Redesign. Further research has revealed various methods that can be used to refactor the system.

2.1. Reengineering Models:

2.1.1. Waterfall model:

The waterfall model is a linear, sequential approach to software development consisting of phases such as requirements gathering, design, implementation, testing, and maintenance. It is popular as a reengineering model because of its simplicity, clarity, and predictability. However, there are also some limitations. For example, once a phase is complete, it's often difficult to make changes to the project, slowing the feedback loop. Moreover, this model is not suitable for complex and large-scale projects. This is because it can be difficult to define all the requirements up front.

2.1.2. Spiral model:

The spiral model is an iterative approach to software development where each iteration goes through the same phases as the waterfall model, but with a greater emphasis on risk management and flexibility. It is seen as a more realistic approach to software development because it allows feedback and adjustments through each iteration. However, it can become difficult to manage and the cost of each iteration can add up quickly.

2.1.3. Agile model:

The Agile model is a highly collaborative approach to software development that emphasizes teamwork, flexibility, and customer satisfaction. It's an iterative, incremental model focused on delivering working software frequently. Agile models are popular with software reengineering teams because they promote adaptability and responsiveness to change. However, they can be difficult to manage and require highly skilled and experienced team members.

2.1.4. DevOps model:

DevOps is a software development approach that emphasizes collaboration and communication between development and operations teams. It's an ongoing, iterative process with a focus on automation and monitoring. The DevOps model is popular with reengineering teams because it allows software changes to be deployed quickly and frequently. However, implementation can be difficult and may require significant changes to existing organizational cultures and processes.

2.1.5. Model-driven model:

A model-driven model is an approach that uses a model as the central artifact to represent a software system. Emphasis is placed on the use of modeling techniques to automate the reengineering process. Model-driven models are popular with reengineering teams because they save time and reduce the risk of human error. However, it can be complex and requires advanced expertise in modeling techniques.

2.1.6. Architecture-driven model:

The architecture-driven model is an approach that emphasizes the importance of software architecture in the reengineering process. It focuses on the analysis and design of software architecture with the goal of improving the overall quality and maintainability of software systems. Architecture-driven models are popular with reengineering teams because they help reduce complexity and improve scalability. However, it can be time consuming and require advanced expertise in software architecture.

2.1.7. Component based model:

A component-based model is an approach that emphasizes the use of reusable software components in the reengineering process. It focuses on developing software components that can be incorporated into large-scale systems. Component-based models are popular with reengineering teams because they save time and reduce development costs. However, identifying and designing suitable components can be difficult, and the quality of the system can be affected by the quality of the components used.

2.1.8. Iterative reengineering model:

We analyze and modify the entire system and redevelop it in a new form. The goal is to gradually refactor each program over a short period of time to assess the assets represented by the legacy system, the familiarity of system administrators and users with the legacy system, and the continuity of current operations execution during the refactoring process. to maintain.

Through a comprehensive re-engineering approach, developers can address issues such as software bugs, outdated technology, and scalability issues. This approach helps keep the system viable, secure, and effective over time, even as technology and business needs evolve.

2.1.9. Service-oriented software reengineering model:

The Service-Oriented Software Reengineering (SoSR) methodology is an approach developed to help organizations modernize their legacy software systems. This methodology focuses on transforming legacy systems into a set of loosely coupled services using an architecture-centric, service-oriented, role-specific, and model-driven approach.

By adopting a service-oriented approach, organizations can break down legacy systems into smaller, more manageable components that are easier to update and maintain. This approach also helps foster communication and collaboration between different parts of an organization by creating a common language and set of standards for service design and implementation.

The SoSR methodology is also role-specific. H. Provide clearly defined tasks and responsibilities for each participant in the reengineering process. This ensures that all participants are on the same page and working towards a common goal. The SoSR methodology describes the use of web services in modernization efforts. Web services provide a standardized way for different systems to communicate with each other, regardless of the underlying technology or programming language used.

2.2. Security:

Each model mentioned in the white paper has its own strengths and weaknesses, but model selection depends on factors such as project size and complexity, organizational culture, and available resources. After examining and understanding all the models, we found that the model detailed in this document showed the most promising results in achieving a secure and revised software system. We also did some research on security models to gain a deeper understanding. We found that there is no one size fits all model and each re-engineering model depends on the specific techniques and tools used in each model. However, security models that can be applied to software reengineering projects include:

2.2.10. Threat modelling:

This involves identifying potential security threats to software systems and analysing their potential impact. This helps identify areas where security measures need to be implemented. Security Testing: Software systems are tested for vulnerabilities such as buffer overflows, injection attacks, and cross-site scripting. This helps identify areas where security measures need to be implemented.

Encryption: This encrypts sensitive data in transit and at rest, protecting it from unauthorized access.

Access Control: This includes implementing appropriate access controls to limit access to sensitive data and functions within software systems.

Authentication and Authorization: This includes implementing appropriate authentication and authorization mechanisms to ensure that only authorized users can access software systems.

Secure Coding Practices: This includes using secure coding practices such as input validation and error handling to prevent vulnerabilities from being introduced into software systems. Security reviews: This includes regular reviews of software systems to identify security vulnerabilities and to ensure that security measures are properly implemented and maintained.

These security measures can be applied to any of the reengineering models described above. This ensures that the reengineering process does not introduce new security vulnerabilities into the software system and that existing vulnerabilities are identified and fixed.

We also examined how well traditional metrics predict software security and whether they correspond to specific metrics.

A software reengineering model makes the system more secure. There is increasing research to achieve quality and robust reengineering. One of the problems is that the whole system is retrofitted, redesigned, and the lack of precise method of the system. As a result, several proposals were put forward to more clearly support the redesign. Although the software has been independently and fairly actively researched, the models referred to in our research papers are unique promising.

3. STRUCTURE OF PAPER

In our paper we proceed further by giving an idea of each model we selected. In Section IV we explain the 3 models in-detail by describing case-studies for Model-Driven Reengineering, Architecture-Centric Re-engineering and Component-Based Re-engineering (CBRE) for a Legacy Customer Relationship Management and we describe the models. Further in Section V, we shared our findings after the survey. Further in Section VI we have given __ for our results. Finally in Section VII our conclusions and future work have been summarized.

4. APPROACH

4.1. Case Study on Model-Driven Re-engineering (MDRE)

4.1.11. Background:

An established bank wanted to improve the security of its legacy online banking system. The system had been developed years ago using older technologies and programming practices, which resulted in multiple vulnerabilities and security issues. The bank decided to adopt Model-Driven Re-engineering (MDRE) to modernize the system and enhance its security.

Problem: The online banking system faced several security challenges, including:

a) Insecure communication channels

b) Weak authentication and authorization mechanisms

c) Vulnerabilities in the application code

d) Insufficient data encryption

These vulnerabilities exposed the system to various risks, such as unauthorized access, data breaches, and financial fraud.

The bank followed a systematic MDRE process to tackle these security challenges:

Reverse Engineering: The existing online banking system was reverse engineered to create an abstract representation using models. This step allowed the team to understand the structure, behavior and interactions within the system.

Security Analysis: The team performed a thorough security analysis using threat modeling and risk assessment techniques to identify key vulnerabilities and potential system impact.

Model Transformation: Based on our security analysis, the team updated the model with appropriate security enhancements, including: B. Secure communication protocols, multi-factor authentication, and role-based access control mechanisms. The team also incorporated security patterns and best practices into the model.

Forward Engineering: The improved model was used to generate new code for an online banking system. The team used automated tools and frameworks to ensure that the generated code adhered to the desired security properties.

Validation and Validation: The team performed rigorous security testing, including static analysis, dynamic analysis, and penetration testing, to validate and validate the effectiveness of the security enhancements introduced through the MDRE process.

By following a systematic MDRE process, the bank was able to modernize and secure its system, reducing the risk of security incidents and ensuring compliance with industry standards and regulations.

Here are some real-time examples illustrating the application of re-engineering techniques to improve software security and functionality in various industries:

European Space Agency (ESA)

The European Space Agency (ESA) adopted re-engineering techniques to modernize and improve the security of its satellite control software systems. Legacy systems were developed using outdated programming languages and architectural patterns, making them vulnerable to security threats and difficult to maintain. By applying reengineering techniques, ESA was able to upgrade the system, improve security, and facilitate maintenance and updates.

Federal Aviation Administration (FAA) – Mode-Based Reengineering

The Federal Aviation Administration (FAA) faced the challenge of maintaining and protecting legacy air traffic control systems designed with outdated technology. The FAA undertook a redesign to modernize the system, improve safety, and ensure it can meet the growing demands of air traffic management. The revised system has better security mechanisms such as encrypted communication channels and secure authentication protocols.

Major financial institution

A large financial institution needed to improve the security of its outdated online banking system. By using model-driven reengineering (MDRE) techniques, the institution is able to modernize its systems, improve its security posture, and remediate various vulnerabilities such as insecure communication channels and weak authentication mechanisms. I was. The revised system includes secure communication protocols, multi-factor authentication and role-based access control mechanisms.

Telecommunications company

A telecommunications company faced the challenge of maintaining and securing its legacy billing system. The system was developed using an old programming language and lacked proper security measures, exposing it to various risks such as unauthorized access and invasion of privacy. The company applied reengineering techniques such as component-based reengineering (CBRE) to modernize the system, improve security, and facilitate maintenance and updates.

Health institution

Healthcare organizations needed to improve the security of their electronic health record (EHR) systems to comply with privacy regulations and protect patient information. Using reengineering techniques, the organization was able to upgrade its EHR system and implement better data encryption, secure authentication, and access control mechanisms. The revised system is now compliant with data protection regulations, enhancing protection of sensitive patient data.

These real-time examples demonstrate the effectiveness of re-engineering techniques to address software security and functionality challenges across multiple industries. By applying these techniques, organizations can modernize legacy systems, improve their security posture, and better protect sensitive data and assets.

4.2. Case Study: Re-engineering the U.S. Federal Aviation Administration (FAA) Legacy Air Traffic Control System

Background:

The U.S. Federal Aviation Administration (FAA) is responsible for managing air traffic control (ATC) across the United States. The ATC system is a complex infrastructure that coordinates the movement of thousands of aircraft daily. However, the FAA faced challenges in maintaining and securing its legacy ATC systems, which were developed using outdated technologies and programming practices. These systems were difficult to maintain, suffered from performance limitations, and posed security risks.

Problem:

The legacy ATC system faced several challenges, including:

Inefficient communication between various subsystems

Limited scalability to handle the increasing volume of air traffic

Outdated security measures that exposed the system to potential cyberattacks

Difficulty in maintaining and upgrading the system due to its monolithic architecture

Approach:

The FAA decided to adopt a re-engineering approach to modernize its ATC system and address these challenges:

System Analysis: The FAA performed a thorough analysis of the existing ATC system to understand its architecture, subsystems, and the interactions between them. This analysis helped us identify areas that needed improvement, especially regarding communications, scalability, and security.

Modular Design: The FAA has redesigned the system using a modular architecture that divides the system into smaller, more manageable components. This modular design makes the system more serviceable and easier to upgrade and expand.

Secure Communication: The revised ATC system implements secure communication protocols such as Transport Layer Security (TLS) to transfer sensitive information between subsystems and external entities such as aircraft and other air traffic control centers. Protect your transmission.

Scalability and Performance: The FAA has introduced new technologies and methods, such as cloud computing and parallel processing, to improve system scalability and performance. This allows ATC systems to better handle increased air traffic without compromising safety or efficiency.

Validation and Validation: Rigorous tests including functional, performance and safety tests were conducted to validate and verify the effectiveness of the revised ATC system. The FAA also hired independent cybersecurity experts to conduct penetration tests and identify potential vulnerabilities.

4.3. Case Study: Architecture-Centric Re-engineering (ACRE) for a Retail E-commerce Platform

Background:

A rapidly growing retail company operated a successful e-commerce platform that allowed customers to browse and purchase products online. However, the platform's monolithic architecture, developed using older technologies, was becoming increasingly difficult to maintain, scale, and secure. The company decided to adopt Architecture-Centric Re-engineering (ACRE) to modernize the platform, improve its performance, and enhance security.

Problem:

The e-commerce platform faced several challenges, including:

Limited scalability to handle increased traffic and customer demands

Difficulty in maintaining and updating the platform due to its monolithic architecture

Inadequate security measures, exposing the platform to cyber threats and data breaches

Inefficient integration with third-party services, such as payment gateways and shipping providers

Approach:

The retail company followed a systematic ACRE process to address these challenges:

Architectural Analysis: The company conducted a comprehensive analysis of the existing e-commerce platform to understand its architecture, components, and their interactions. This analysis helped identify the primary issues and areas that required improvement.

Architectural design: Based on the analysis, the company developed a new microservices-based architecture for its e-commerce platform. This architecture decomposed a monolithic platform into smaller, independent services that can be developed, maintained, and scaled independently.

Re-engineering: The company redesigned the components of the platform while adhering to the new architectural design. This includes refactoring code, adopting modern development frameworks and languages, and implementing secure communication protocols such as HTTPS and OAuth2.

Integrations: The revamped platform featured efficient integrations with third-party services such as payment gateways, shipping carriers, and customer support systems. This improved the overall customer experience and streamlined the company's operations.

Validation and Validation: The company conducted extensive testing, including functional, performance and security testing, to verify and validate the effectiveness of the

revised platform. An independent security audit was also conducted to identify potential vulnerabilities and ensure compliance with industry standards and regulations.

4.4. Case Study: Component-Based Re-engineering (CBRE) for a Legacy Customer Relationship Management (CRM) System

Background:

A medium-sized software company specializing in CRM solutions faced challenges in maintaining and updating its legacy CRM system. The system, developed over a decade ago, was based on monolithic architecture and older technologies. As the company expanded, the CRM system struggled to accommodate new features and integrations, and maintaining the system became increasingly time-consuming and costly. To address these challenges, the company decided to adopt Component-Based Re-engineering (CBRE) to modernize the CRM system and improve its maintainability and scalability.

Problem:

The legacy CRM system faced several challenges, including:

Limited extensibility and adaptability to new requirements

Difficulty in integrating with modern third-party services and APIs

Performance and scalability issues due to the monolithic architecture

Increased maintenance costs and complexity

Approach:

The company followed a systematic CBRE process to modernize the CRM system:

System Decomposition: The company analyzed the legacy CRM system and decomposed it into smaller, more manageable components. Each component was responsible for specific functions such as: B. Customer data management, marketing automation or analytics.

Component evaluation and replacement: The team evaluated each component to determine if it could be replaced with a modern off-the-shelf solution or if it needed to be redesigned. For replaceable components, the team selected new solutions that fit the company's technical requirements and integration needs.

Component re-engineering: Components that could not be replaced were reworked by the team using the latest technology and best practices. This process involved redesigning components, updating the code base, and improving security and performance characteristics.

Integration and testing: The team integrated the revised off-the-shelf components into a cohesive CRM system, ensuring seamless interaction between components. The company then conducted extensive testing, including functional, performance and security testing, to verify the functionality and reliability of the revised system.

5. RESULTS

5.1. The reengineering approach taken by the FAA has resulted in a modernized and safer ATC system, with significant improvements in the following areas:

Efficient Communication: The revised system improves communication between subsystems, making air traffic control operations more efficient and reliable.

Improved security: The adoption of secure communication protocols and advanced security measures has reduced the system's vulnerability to cyber threats. Scalability and performance: The revised ATC system can handle more air traffic without compromising safety or efficiency.

Maintainability: The modular architecture facilitates maintenance and allows the FAA to respond more effectively to changing requirements and new technologies.

This case study demonstrates the effectiveness of reengineering to address the challenges the FAA faces in maintaining and protecting legacy ATC systems. By adopting a systematic reengineering approach, the FAA was able to modernize its ATC system, improve its safety posture, and ensure the safe and efficient management of U.S. air traffic.

5.2. The Architecture-Centric Reengineering (ACRE) approach has resulted in a modernized, scalable and secure e-commerce platform with significant improvements in the following areas:

Scalability: The microservices-based architecture has enabled the platform to handle increased traffic and customer demands more efficiently.

Maintainability: The modular architecture makes the platform easier to maintain and update, allowing you to respond more quickly to changing business needs and adopt new technologies.

Security: The revamped platform includes advanced security measures such as encrypted communication channels and robust authentication mechanisms that reduce the risk of cyber-attacks and data breaches.

Integrations: Improved integrations with third-party services have streamlined company operations and enhanced the customer experience.

5.3. A component-based reengineering approach has brought his CRM system up to date, improving:

Increased scalability: A new modular architecture makes it easier to add new features and integrations, allowing you to quickly respond to changing customer needs and market trends.

Improved scalability and performance: The revamped CRM system now handles increased workloads better and ensures consistent performance as the company expands.

Simplified maintenance: The modular design and use of off-the-shelf components reduced the complexity of CRM system maintenance, reduced maintenance costs, and streamlined updates.

Better integration capabilities: Modern CRM systems can easily integrate with third-party services and APIs, allowing businesses to take advantage of the latest technology and industry standards.

This case study demonstrates the effectiveness of component-based reengineering (CBRE) in modernizing a traditional CRM system while addressing scalability, maintainability, and integration issues. By adopting his systematic CBRE approach, the software company successfully updated his CRM system, reducing maintenance costs and complexity while increasing adaptability and efficiency.

6. DISCUSSION

This section provides an analysis of the survey results, discussing the factors that contributed to the overall success of these methodologies in improving software security, the difficulties encountered during implementation, and the reasons behind the effectiveness of particular re-engineering methodologies.

(a) Effectiveness of Re-engineering Methodologies:

According to the findings of the survey, a number of factors determine how effective re-engineering methodologies are at increasing software security, including:

The approach's comprehensiveness: Approaches that address different parts of programming security, like engineering, parts, and code-level weaknesses, will generally yield more huge upgrades in security.

The blending of security measures: It is more likely that a re-engineering process that incorporates security best practices and patterns will result in long-term security enhancements.

The flexibility of the technique: Procedures that are adaptable and can be customized to the particular setting and necessities of a product framework have a higher probability of progress in upgrading security.

(b) Challenges in Re-engineering for Security:

The findings of the survey highlight a number of obstacles encountered during the implementation of software security re-engineering techniques:

System complexity of software: During the re-engineering process, it is difficult to identify and address all potential security flaws due to the complexity of modern software systems.

Lack of security expertise: Re-engineering methodologies and techniques are frequently hampered by software developers' and engineers' lack of security expertise.

Limitations on resources: It can be difficult to allocate sufficient resources for security enhancements because the re-engineering process can be time-consuming and resource-intensive.

(c) Factors Contributing to Success:

Re-engineering methodologies for improving software security are successful when they are successfully implemented, according to the survey.

Inclusion of safety specialists: During the re-engineering process, engaging security experts can assist in more effectively identifying and addressing potential security vulnerabilities.

Continuous validation and testing of security: During the re-engineering process, security testing and validation activities are included to guarantee that security enhancements work and do not create new vulnerabilities.

Adoption of security best practices: Following laid out security best practices and rules can essentially upgrade the adequacy of re-designing philosophies in further developing programming security.

In conclusion, the survey's findings suggest that, when used correctly, re-engineering can significantly improve software security. However, achieving long-term security enhancements necessitates overcoming obstacles and making use of success factors. The findings of this survey and a plan for future research in this area are presented in the following section.

7. CONCLUSION

This survey paper has provided a comprehensive assessment of the impact of refactoring on software security by analyzing different refactoring methods, techniques, and tools. Survey results indicate that refactoring can have a significant impact on software security when implemented effectively, with specific improvements depending on methods, techniques, tools, and system contexts. system is selected.

However, the survey also highlights some of the challenges encountered during the refactoring, such as software system complexity, limited security expertise, and resource constraints. To overcome these challenges and achieve lasting security improvements, it is essential to engage security professionals, perform ongoing security testing and validation, and apply established security best practices.

Future work in this area may focus on the following directions:

Develop new refactoring methods and techniques to address specific challenges faced during refactoring, such as scalability, automation, and integration of emerging security practices . Conduct more research and empirical evaluations to evaluate the effectiveness of refactoring approaches in different contexts and fields, thereby providing more realistic insights into their impact on software security.

Investigate the integration of refactoring methods with other software development processes, such as DevSecOps, to ensure

a more holistic approach to improving software security throughout the development lifecycle.

By exploring these future research directions, we can improve our understanding of the impact of refactoring on software security and develop more effective strategies for maintaining and improve the security status of legacy software systems.

REFERENCES

[1] E. J. Chikofsky and J. H. Cross, "Reverse engineering and design recovery: a taxonomy," in IEEE Software, vol. 7, no. 1, pp. 13-17, Jan. 1990, doi: 10.1109/52.43044..

[2] H. M. Sneed, "Planning the reengineering of legacy systems," in IEEE Software, vol. 12, no. 1, pp. 24-34, Jan. 1995, doi: 10.1109/52.363168.

[3] A. Zerouali, V. Cosentino, T. Mens, G. Robles and J. M. Gonzalez-Barahona, "On the Impact of Outdated and Vulnerable Javascript Packages in Docker Images," 2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER), Hangzhou, China, 2019, pp. 619-623, doi: 10.1109/SANER.2019.8667984.

[4] R. Xiong and B. Li, "Accurate Design Pattern Detection Based on Idiomatic Implementation Matching in Java Language Context," 2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER), Hangzhou, China, 2019, pp. 163-174, doi: 10.1109/SANER.2019.8668031.