Evaluation of Legacy Systems Quality: A Case Study of Self-Checkout Systems

Laud Charles Ochei
Department of Computer Science
University of Port Harcourt, Nigeria

Chigoziri Marcus
Department of Computer Science
University of Port Harcourt, Nigeria

ABSTRACT

Many organisations rely on legacy systems to function, but their ageing infrastructure frequently presents maintenance, security, and scalability challenges. Evaluating the technical quality of legacy systems is critical for identifying areas for improvement and ensuring their ongoing functionality. This paper compares technical quality assessment strategies used in both legacy and modern versions of grocery self-checkout systems. We start by defining the Grocery Self-Checkout System, outlining its features and architecture in both legacy and modern iterations. Following that, the study looked at different approaches to assessing technical quality, such as code review and analysis, performance testing, security audits, maintainability assessment, and compatibility testing. Using these strategies and associated metrics, this studies highlighted the technical quality differences between legacy and modern systems, as well as discussed the challenges and potential advancements in evaluating Grocery Self-Checkout Systems. Furthermore, the study presents the results of our analysis, which provide insights into the effectiveness of each assessment strategy and recommendations for improving the technical quality of Grocery Self-Checkout Systems.

Keywords

Comparative Analysis, Legacy Systems, Legacy architecture, Technical Quality, Self-Checkout Systems

1. INTRODUCTION

Legacy systems are integral to the operations of many organizations, yet their outdated nature often presents challenges in terms of maintenance, security, and scalability. Assessing the technical quality of legacy systems is paramount to identify areas for improvement and ensure their continued effectiveness in supporting critical business processes. This study presents a comparative analysis of technical quality assessment strategies applied to legacy and modern versions of Grocery Self-checkout Systems.

In recent years, the proliferation of Grocery Self-checkout Systems has revolutionized the retail industry, offering customers a convenient and efficient way to complete their purchases. However, as these systems evolve over time, legacy versions may encounter issues related to outdated technologies and architectural limitations. Therefore, it becomes imperative to evaluate the technical quality of both legacy and modern iterations to inform decision-making and drive improvements.

Drawing from related literature, this study builds upon existing research on assessing the technical quality of legacy systems. Previous studies have explored various strategies, including code review and analysis, performance testing, security audits, maintainability assessment, and compatibility testing. However, there remains a need to compare the effectiveness of these strategies when applied to both legacy and modern versions of Grocery Self-checkout Systems.

Motivated by this problem, this study proposes a comparative analysis framework that systematically evaluates the technical quality of legacy and modern Grocery Self-checkout Systems. The specific contributions of this paper are:

1. Presenting a comprehensive review of related literature on strategies for strategies for evaluating the technical quality of legacy systems.

2. Providing five (5) strategies and metrics for evaluating the technical quality of legacy systems, including code review and analysis, performance testing, security audits, maintainability assessment, and compatibility testing.

3. Applying these strategies and metrics to both legacy and modern versions of a Grocery Self-checkout Systems to highlight the differences in technical quality, challenges, and future developments in assessing the technical quality of legacy systems.

4. Presenting the results of the analysis in the form of providing insights into the effectiveness of each assessment strategy and offering recommendations for enhancing the technical quality of legacy systems.

By applying established assessment strategies and metrics to both versions, this study aims to uncover differences in technical quality and identify areas for improvement. Our study not only provides insights into the effectiveness of each assessment strategy but also offers recommendations for enhancing the technical quality of Grocery Self-checkout Systems to meet evolving business needs and customer expectations.

The rest of the paper is organised as follow: Section 2 is the background of the study, Section 3 presents a review of literature and related concepts. Section 4 presents strategies for assessing the quality of a legacy system. Section 5 is the analysis of the study based on the application of the strategies and metrics to both legacy and modern versions of a Grocery Self-checkout Systems. Section 6 presents the findings and discussions. Section 7 concludes the paper with future work.

2. BACKGROUND OF THE PROBLEM

Legacy systems, characterized by their long-standing presence and reliance on outdated technologies, play a crucial role in supporting essential business processes across various industries. These systems often encompass a diverse range of software applications, databases, and hardware components developed using obsolete programming languages and architectures [26].

The significance of assessing the technical quality of legacy systems stems from the challenges inherent in maintaining and evolving these systems to meet changing business requirements. Legacy systems are prone to issues such as code decay, security vulnerabilities, and compatibility constraints with modern technologies. Therefore, evaluating their technical quality is essential to identify areas for improvement, mitigate risks, and ensure their continued effectiveness in supporting organizational objectives.

The assessment of technical quality encompasses various dimensions, including code maintainability, performance efficiency, security robustness, and compatibility with modern platforms. By systematically evaluating these aspects, organizations can gain insights into the health and resilience of their legacy systems, enabling informed decision-making and resource allocation [17].

In the context of this study, the focus is on Grocery Self-checkout Systems, which have become prevalent in retail environments. These systems allow customers to scan, bag, and pay for their purchases independently, enhancing convenience and reducing checkout times. However, as Grocery Self-checkout Systems evolve, legacy versions may face challenges related to outdated technologies and architectural limitations, necessitating a thorough assessment of their technical quality.

3. REVIEW OF LITERATURE AND RELATED CONCEPT

3.1 Overview of Related Concepts

This section presents an overview of related concepts.

3.1.1 Definition and Classification of Legacy Systems

Legacy systems represent a fundamental component of the technological landscape in many organizations, encapsulating software applications, databases, and infrastructure that have endured over time. The definition and classification of legacy systems encompass a broad spectrum of technologies, ranging from antiquated mainframe systems to more recent but outdated software platforms [26].

Legacy systems are commonly defined as software systems or technologies that have been in operation for an extended period, often surpassing their intended lifespan or falling out of mainstream support [12]. These systems are characterized by their reliance on outdated technologies, programming languages, and architectural patterns that may no longer align with contemporary standards and practices.

Legacy systems can be classified based on various criteria, including their technology stack, age, and degree of obsolescence. For instance, mainframe systems, which emerged in the mid-20th century, are often considered prime examples of legacy technology due to their longevity and historical significance [15]. Similarly, software applications built using outdated programming languages such as COBOL or FORTRAN may be classified as legacy systems, regardless of their age.

3.1.2 Factors Contributing to the Prevalence of Legacy Systems

Several factors contribute to the prevalence of legacy systems in contemporary organizations. One significant factor is the long lifespan of enterprise software applications, which often outlasts the initial expectations of developers and stakeholders [17]. Additionally, technological inertia, organizational resistance to change, and the high cost and risk associated with system modernization efforts can contribute to the perpetuation of legacy systems [26].

Furthermore, mergers, acquisitions, and organizational restructuring can result in the consolidation of disparate systems and technologies, leading to the proliferation of legacy environments [26]. In some cases, regulatory compliance requirements or contractual obligations may also prevent organizations from abandoning legacy systems, further perpetuating their existence [8].

3.1.3 Challenges and Risks Associated with Legacy Systems

Legacy systems pose numerous challenges and risks to organizations, stemming from their outdated technologies, architectural complexity, and limited documentation. One primary challenge is the maintenance and support of legacy systems, which often require specialized skills and knowledge that may be scarce or difficult to acquire [9].

Security vulnerabilities represent another significant risk associated with legacy systems, as outdated technologies may lack modern security features and defenses against cyber threats [7]. Additionally, legacy systems may exhibit poor performance and scalability characteristics, limiting their ability to adapt to changing business requirements and technological advancements [1].

Moreover, the lack of interoperability and integration with modern systems can hinder organizational agility and innovation, impeding efforts to leverage emerging technologies and stay competitive in dynamic markets [17]. Overall, the challenges and risks associated with legacy systems underscore the importance of assessing their technical quality and formulating strategies for modernization or migration where necessary.

3.2 Case Study: Grocery Self-Checkout System

The section provides an in-depth overview of the Grocery Self-Checkout System, detailing its functionality, key features, and architectural design.

3.2.1 System Overview

A grocery self-checkout system is a technology-driven solution deployed in retail stores to enable customers to independently scan, bag, and pay for their purchases without the assistance of a cashier [23]. These systems typically consist of a combination of hardware and software components, including barcode scanners, touchscreen displays, weight scales, payment terminals, and associated software applications.

The process begins with customers scanning the barcodes of items they wish to purchase using either integrated barcode scanners or handheld devices. The system then verifies the scanned items against a database of product information to ensure accuracy. Once all items have been scanned, customers proceed to the payment stage, where they can choose from various payment options such as credit/debit cards, mobile wallets, or cash. After completing the payment process, customers bag their purchased items and receive a receipt as proof of purchase.

Grocery self-checkout systems are designed to offer several benefits, including increased convenience for customers, reduced checkout wait times, and improved operational efficiency for retailers [25]. By allowing customers to complete transactions independently, these systems help expedite the checkout process, leading to enhanced customer satisfaction and loyalty.

3.2.2 Features

The modern Grocery Self-Checkout System offers a plethora of features aimed at enhancing the overall shopping experience for customers and optimizing operational efficiency for retailers. Some prominent features include:

Barcode Scanning: Customers can use integrated barcode scanners or handheld devices to scan the barcodes of items they wish to purchase. This feature enables quick and accurate identification of products, reducing the time spent at the checkout counter [22].

Payment Processing: The system supports various payment methods, including credit/debit cards, mobile wallets, and contactless payments. Integration with secure payment gateways ensures the safe and efficient processing of transactions, enhancing customer trust and satisfaction [21].

Bagging Area: A designated bagging area allows customers to conveniently bag their scanned items after completing the checkout process. Weight sensors and automated alerts ensure compliance with bagging requirements and minimize the risk of theft or errors [25].

User Interface: The user interface of the Grocery Self-Checkout System is intuitive and user-friendly, guiding customers through each step of the checkout process. Clear instructions, visual prompts, and touchscreen capabilities enhance accessibility and usability for customers of all ages and technical proficiencies [20].

3.2.3 Design and Architecture

The design and architecture of the Grocery Self-Checkout System play a crucial role in its functionality, reliability, and scalability. A well-designed system architecture enables seamless integration of hardware and software components, ensures data security and integrity, and facilitates future enhancements and updates. Figure 1 is an automated checkout system that combines hardware and software components. The hardware includes two RFID readers, a ceiling-mounted system for tracking items, and a gate-mounted system to detect items leaving the store [23]. Some key aspects of the design and architecture of a grocery include:

Modular Design: The system employs a modular design approach, with distinct components responsible for specific functions such as barcode scanning, payment processing, and user interface management. This modular architecture enhances flexibility, scalability, and maintainability, allowing for easier upgrades and customization [1].

Client-Server Model: The Grocery Self-Checkout System follows a client-server model, where the client terminals interact with a centralized server to process transactions, retrieve product information, and manage user data. This architecture enables efficient communication, data synchronization, and centralized control, ensuring consistent performance across multiple checkout terminals [19].

Data Management: The system incorporates robust data management mechanisms to store and retrieve transactional data, product information, and customer records securely. Relational databases, distributed file systems, and encryption techniques are employed to ensure data integrity, confidentiality, and availability, mitigating the risk of data breaches and unauthorized access [16].

Fault Tolerance: To enhance reliability and fault tolerance, the system implements redundancy measures such as backup servers, redundant power supplies, and fault-tolerant communication protocols. In the event of hardware failures or network disruptions, failover mechanisms automatically redirect traffic to backup systems, minimizing downtime and ensuring uninterrupted service [13].

3.3 Review of Related Literature on Strategies for Assessing Legacy Systems

This section reviews the existing literature on strategies for assessing the quality of legacy systems and discuss key findings and insights.

3.3.1 Classification of Approaches for Assessing the Technical Quality of Legacy Systems

Numerous studies have investigated various approaches and techniques for evaluating the technical quality of legacy systems. These approaches encompass a wide range of assessment strategies, including code analysis, performance testing, security audits, maintainability assessments, and compatibility testing. Researchers have explored both manual and automated methods for assessing different aspects of legacy systems to uncover issues and improve overall quality [10; 1; 11].

3.3.2 Context-Aware Approaches

Context-aware assessment approaches consider the unique characteristics and constraints of legacy systems, such as outdated technologies, legacy dependencies, and evolving business requirements [9]. These approaches aim to tailor assessment strategies to specific contexts, enabling more accurate and effective evaluation of technical quality. By considering the context in which legacy systems operate, organizations can identify relevant metrics and prioritize improvement efforts based on their specific needs and constraints [10; 12].

3.3.3 Hybrid Assessment Models

Hybrid assessment models combine automated tools with human expertise to leverage the strengths of both approaches [3]. Automated tools can efficiently analyze large codebases and identify common issues, while human experts provide domain knowledge and context-specific insights. By integrating automated analysis with human review and interpretation, organizations can achieve more comprehensive and accurate assessments of legacy systems, leading to better-informed decision-making and resource allocation [6; 14].

3.3.4 Automated tools

The integration of automated tools have been proposed to improve the effectiveness of assessments. For example, the automated tools streamline the detection of issues, enhancing the assessment process (Joorabchi et al., 2011).

4. STRATEGIES FOR ASSESSING THE QUALITY OF A LEGACY SYSTEM

This section discusses the different strategies for assessing the technical quality of legacy systems.

4.1.1 Code Review and Analysis

Conducting a thorough code review and analysis is essential to uncover potential issues within the legacy system's source code. This strategy involves examining the codebase for security vulnerabilities, code inefficiencies, and design flaws. Metrics for code review and analysis include lines of code, code complexity (e.g., cyclomatic complexity), code duplication, code coverage, and the number of identified code issues (e.g., code smells, potential memory leaks) [2].

4.1.2 Performance Testing

Performance testing evaluates the legacy system's responsiveness, scalability, and resource usage under various simulated workloads. This strategy helps identify performance bottlenecks and ensures the system meets expected performance requirements. Metrics for performance testing include response time, throughput, CPU and memory utilization, and error rates during load testing and stress testing [6].

4.1.3 Security Audit

Legacy systems are prone to security risks due to outdated security protocols and a lack of regular updates. A security audit helps identify and address potential vulnerabilities, ensuring the system's integrity and protecting against security threats. Metrics for security audits include the number and severity of identified vulnerabilities, time-to-fix vulnerabilities, and implementation of recommended security measures [7].

4.1.4 Maintainability Assessment

Assessing the maintainability of a legacy system determines its ability to accommodate future changes or enhancements. This strategy involves evaluating the codebase, architecture, and documentation to ensure that the system can be easily modified or extended. Metrics for maintainability assessment include cyclomatic complexity, code churn (rate of code changes), code readability (e.g., maintainability index), and documentation completeness [11].

4.1.5 Compatibility Testing

Compatibility testing ensures that the legacy system functions correctly across various hardware, operating systems, and software configurations. This strategy verifies the system's compatibility with modern environments, minimizing the risk of compatibility issues. Metrics for compatibility testing include the number of compatibility issues found and resolved during testing [5].

Table 1 provides a comparison of different assessment strategies, their respective metrics, tools/resources used, significance, challenges, and future developments in the evaluation of technical quality in legacy systems, particularly focusing on Grocery Self-Checkout Systems.

5. ANALYSIS OF THE STRATEGIES FOR ASSESSING TECHNICAL QUALITY

The Grocery Self-Checkout System revolutionizes retail experiences by enabling customers to scan and pay for their items independently. As these systems evolve, the technical quality assessment becomes crucial to ensure their efficiency, security, and reliability. This paper compares the technical quality assessment strategies between legacy and modern versions of Grocery Self-checkout Systems.

5.1 Grocery Self-Checkout System: Legacy versus Modern Versions

The Grocery Self-Checkout System allows customers to scan, bag, and pay for their purchases without the assistance of a cashier. It typically includes features such as barcode scanning, payment processing, bagging area, and user interface for interaction.

5.1.1 Legacy Version Grocery Self-Checkout System

The legacy version of the Grocery Self-checkout System is characterized by its older technology stack and architecture. It may have limited functionalities and lack integration with modern payment methods or loyalty programs. The architecture is monolithic, with a centralized processing unit handling all transactions [24].

5.1.2 Modern Version Grocery Self-Checkout System

In contrast, the modern version of the Grocery Self-checkout System incorporates advanced technologies such as RFID scanning, mobile payment options, and AI-driven assistance. It features a modular architecture, allowing for easier scalability and integration with other systems. The user interface is intuitive and customizable, enhancing the overall user experience [13].

5.2 Application of Strategies for Assessing the Technical Quality of Legacy Systems to Grocery Self-checkout Systems

We now explore strategies for assessing technical quality and how they apply to both legacy and modern versions of Grocery Self-checkout Systems.

5.2.1 Code Review and Analysis

In both versions, code review and analysis identify security vulnerabilities and design flaws. Metrics include lines of code, code complexity, and code duplication. Legacy systems may exhibit higher code complexity and more code duplication due to outdated coding practices [2].

5.2.2 Performance Testing

Performance testing evaluates responsiveness, scalability, and resource usage. While the legacy system may struggle with high resource utilization and slower response times, the modern system showcases optimized performance and scalability [5].

5.2.3 Security Audit

Security audits identify and address vulnerabilities. Both versions may face security risks, but the legacy system is more susceptible due to outdated security protocols. Metrics for security audit include the number and severity of vulnerabilities, with the legacy system likely having a higher count [7].

5.2.4 Maintainability Assessment

Maintainability assessment determines the system's ability to accommodate changes. Legacy systems may have lower maintainability due to outdated documentation and complex codebase. Metrics include cyclomatic complexity and code readability, with the modern system typically scoring higher [11].

5.2.5 Compatibility Testing

Compatibility testing ensures functionality across various hardware and software configurations. The legacy system may encounter compatibility issues with modern devices and payment methods, leading to higher testing efforts and potential disruptions [5].

6. FINDINGS AND DISCUSSIONS

The study has revealed important findings regarding the strategies for assessing legacy systems.

6.1 Keys Findings from Literature Review

Common themes emerge from the literature on strategies for assessing the quality of legacy systems. One key finding is the importance of considering the unique context and characteristics of legacy systems when designing assessment approaches. Context-aware strategies can yield more meaningful results and guide decision-making based on the specific needs and constraints of each organization [9].

Additionally, the literature highlights the value of combining automated analysis with human expertise to achieve more accurate and comprehensive assessments. Hybrid assessment models leverage the strengths of both approaches and mitigate the limitations of each, resulting in more effective evaluations of technical quality [12, 3].

Case studies offer valuable insights into the practical challenges of legacy system assessments, enabling organizations to understand the benefits and pitfalls of various strategies. These real-world experiences help in making informed decisions about assessment efforts. Several case studies have demonstrated effective assessment strategies in real-world scenarios, providing valuable insights and practical guidance [17, 21]. By analyzing the experiences of organizations that have conducted legacy system assessments, researchers can derive best practices and lessons learned for future assessments.

6.2 Comparative Analysis of legacy and modern Grocery Self-checkout Systems

The assessment of technical quality reveals significant differences between legacy and modern Grocery Self-checkout Systems. Legacy systems often exhibit lower performance, higher security risks, and lower maintainability compared to modern systems. However, legacy systems may have already undergone extensive testing and validation, leading to a more stable and predictable behavior in certain scenarios [27].

Table 2 provides a comparison of the technical quality differences between legacy and modern versions of grocery self-checkout systems based on different assessment strategies and associated metrics. It highlights the improvements made in modern systems compared to legacy systems in terms of code quality, performance, security, maintainability, and compatibility testing.

6.3 Enhancing the assessment of the Technical Quality of Legacy systems

The review of related literature revealed three main issues for consideration in order to enhance the assessment of the technical quality of legacy systems - adoption of a risk basked approach, collaboration between technical experts, and the use of automated tools.

6.3.1 Risk-based approach

Adopting a risk-based approach is a key strategy for prioritising critical functionalities and high-risk areas during assessments [27]. Organisations can maximise the effectiveness of their assessments by focusing attention and resources on areas with the greatest potential impact on system functionality, security, and performance. This approach enables a more targeted allocation of resources, ensuring that assessment activities address the most pressing issues first.

6.3.2 Collaboration between technical experts

Collaboration between technical experts and business stakeholders is crucial for properly evaluating legacy systems [29]. Technical experts bring knowledge of system architecture, code analysis, and performance evaluation, whereas business stakeholders provide insights into system requirements, user needs, and organisational objectives. By encouraging open communication and collaboration between these two groups, organisations can gain a comprehensive understanding of the system's technical quality and alignment with business goals. This collaboration also encourages key stakeholders' buy-in, ensuring that assessment results are actionable and relevant to organisational priorities.

6.3.3 Automated tools

Using automated tools improves assessment efficiency and effectiveness [14]. These tools help to detect issues like code smells, security vulnerabilities, and performance bottlenecks more efficiently than manual methods, saving time and resources. By automating repetitive tasks and providing actionable insights, these tools allow assessment teams to concentrate on higher-value activities like analysis and decision-making. Furthermore, automated tools allow organisations to conduct assessments more frequently and consistently, ensuring that technical quality is a top priority throughout the system's lifecycle.

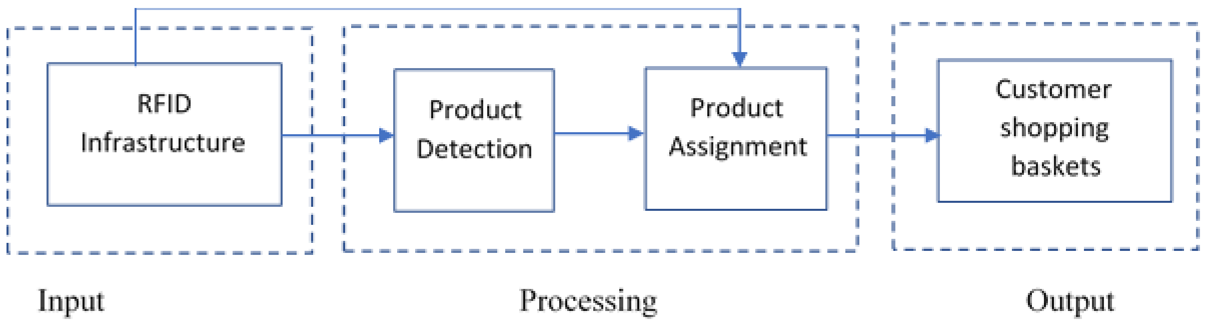

Fig 1: Architecture of the automated checkout [10].

Table 1. Comparison of different assessment strategies in the evaluation of technical quality in legacy systems.

Assessment Strategy	**Code Review and Analysis**	**Performance Testing**	**Security Audits**	**Maintainability Assessment**	**Compatibility Testing**
Description	Identifies potential issues within the source code, such as security vulnerabilities, inefficiencies, and design flaws.	Evaluates system responsiveness, scalability, and resource usage under various workloads.	Identifies and addresses security vulnerabilities to mitigate risks.	Evaluates the system's ability to accommodate future changes or enhancements.	Ensures the system functions correctly across various hardware and software configurations.
Metrics	- Lines of code - Code complexity (e.g., cyclomatic complexity) - Code duplication	- Response time - Throughput - CPU and memory utilization - Error rates during load and stress testing	- Number and severity of identified vulnerabilities - Time-to-fix vulnerabilities	- Cyclomatic complexity - Code churn - Code readability - Documentation completeness	- Number of compatibility issues found and resolved during testing
Tools/Resources	- Static code analysis tools (e.g., SonarQube, Checkmarx) - Code review processes	- Load testing tools (e.g., Apache JMeter, LoadRunner) - Stress testing tools (e.g., Gatling, Apache Bench)	- Vulnerability scanning tools (e.g., Nessus, OpenVAS)	- Maintainability metrics tools (e.g., Understand, Lizard) - Documentation review tools (e.g., Doxygen, Javadoc)	- Compatibility testing frameworks (e.g., Selenium, Appium)
Importance/Significance	Vital for identifying potential security vulnerabilities and optimizing code efficiency.	Crucial for ensuring the system meets expected performance requirements and can handle user demand.	Essential for protecting sensitive data and ensuring compliance with security standards.	Critical for determining the system's long-term sustainability and ease of future maintenance.	Necessary to ensure the system can seamlessly integrate with various hardware and software environments.
Challenges	- Resource-intensive process - Relies heavily on human expertise and time	- Complexity of simulating real-world workloads - Ensuring accurate and reproducible results	- Keeping pace with evolving security threats - Balancing security measures with system performance	- Subjectivity in evaluating code readability and documentation completeness.	- Identifying and addressing compatibility issues across different platforms and configurations.
Future Developments	Integration of AI and machine learning algorithms to automate code review processes.	Adoption of cloud-based testing solutions for scalability and flexibility.	Implementation of continuous security testing practices.	Incorporation of automated code documentation tools and metrics.	Enhanced automation and tooling for more comprehensive compatibility testing.

Table 2. Comparison of the technical quality differences between legacy and modern versions of grocery self-checkout systems.

Assessment Strategy	**Legacy Systems**	**Modern Systems**
Code Review and Analysis	Relies on manual code reviews and basic analysis tools.	Utilizes automated code analysis tools and incorporates continuous integration practices.
Metrics	- Lines of code - Code complexity (e.g., cyclomatic complexity) - Code duplication	- Lines of code - Code complexity (e.g., cyclomatic complexity) - Code duplication
Performance Testing	Limited performance testing, often manually conducted.	Conducts comprehensive load testing and utilizes cloud-based solutions for scalability.

Metrics	- Response time - Throughput - CPU and memory utilization - Error rates during load testing	- Response time - Throughput - CPU and memory utilization - Error rates during load testing
Security Audits	Infrequent security audits, reactive approach to security.	Implements continuous security testing practices and follows industry-standard security protocols.
Metrics	- Number and severity of identified vulnerabilities - Time-to-fix vulnerabilities	- Number and severity of identified vulnerabilities - Time-to-fix vulnerabilities
Maintainability Assessment	Limited focus on maintainability, often lacks documentation.	Emphasizes code maintainability, readability, and comprehensive documentation.
Metrics	- Cyclomatic complexity - Code churn - Code readability - Documentation completeness	- Cyclomatic complexity - Code churn - Code readability - Documentation completeness
Compatibility Testing	Basic compatibility testing, may not cover all platforms.	Conducts thorough compatibility testing across various hardware and software configurations.
Metrics	- Number of compatibility issues found and resolved during testing	- Number of compatibility issues found and resolved during testing

6.4 Challenges and Future development

Despite the progress made in assessing the quality of legacy systems, several challenges remain. Legacy systems are often complex and tightly coupled, making it difficult to isolate and address individual issues. Additionally, limited documentation, outdated technologies, and the risk of disrupting existing functionalities can hinder assessment efforts, requiring organizations to invest time and resources in understanding legacy code and architecture [12, 2].

Future research should focus on developing advanced assessment techniques and tools tailored to the unique characteristics of legacy systems [10; 11]. This includes the development of automated analysis tools capable of handling legacy codebases and identifying complex issues such as architectural flaws and design anomalies, the integration of AI-driven analysis tools, automated testing frameworks, and continuous monitoring for both legacy and modern systems [29]. Additionally, research should explore innovative approaches for integrating assessment results into broader decision-making processes, such as modernization and migration strategies [6, 14].

7. CONCLUSION AND FUTURE WORK

Assessing the technical quality of legacy systems is imperative for organizations to maintain their functionality, security, and relevance in today's rapidly evolving technological landscape. Employing strategies such as code review, performance testing, security audits, maintainability assessments, and compatibility testing aids in identifying areas for improvement and mitigating risks. Collaboration and the integration of automated tools further enhance the effectiveness of legacy system assessments, ensuring the continued success of organizations reliant on legacy systems.

The comparative analysis highlights the importance of assessing technical quality in both legacy and modern Grocery Self-checkout Systems. While legacy systems present unique challenges, modern systems offer advanced features and improved performance. By employing strategies for technical quality assessment, organizations can ensure the reliability and efficiency of their Grocery Self-checkout Systems in the rapidly evolving retail landscape.

8. REFERENCES

[1] Bass, L., Clements, P., & Kazman, R. (2003). Software Architecture in Practice. Addison-Wesley Professional.

[2] Basili, V. R., & Weiss, D. M. (1984). A Methodology for Collecting Valid Software Engineering Data. IEEE Transactions on Software Engineering, 10(6), 728-738.

[3] Basili, V. R., Caldiera, G., & Rombach, H. D. (1996). Goal Question Metric Paradigm. Encyclopedia of Software Engineering, 1, 528-532.

[4] Berntsson Svensson, R., Gorschek, T., & Höst, M. (2011). Stages and Levels of Software Process Improvement: A Systematic Literature Review. Software Quality Journal, 19(4), 899-932.

[5] Binder, R. V. (1999). Testing Object-Oriented Systems: Models, Patterns, and Tools. Addison-Wesley Professional.

[6] Di Penta, M., Antoniol, G., & Merlo, E. (2007). A Survey on the Role of Code Smells in Software Refactoring. Journal of Software Maintenance and Evolution: Research and Practice, 19(4), 291-314.

[7] Fichter, S., Martens, J., & Oktaba, H. (2015). Risk-Based Security Testing of Legacy Software. Proceedings of the 2015 International Symposium on Software Testing and Analysis, 215-225.

[8] Gagliardi, J. (2018). High Availability and Disaster Recovery: Concepts, Design, Implementation. CRC Press.

[9] Graaf, B., & Brinkkemper, S. (2008). Software quality improvement through action research: The enablers' perspective. Journal of Systems and Software, 81(11), 1902-1913.

[10] Hauser, M., Günther, S. A., Flath, C. M., & Thiesse, F. (2019). Towards digital transformation in fashion retailing: A design-oriented IS research study of automated checkout systems. Business & Information Systems Engineering, 61, 51-66.

[11] Hassan, A. E. (2009). Predicting Faults Using the

Complexity of Code Changes. Proceedings of the International Conference on Software Engineering, 78-88.

[12] Hassan, A. E., & Holt, R. C. (2005). The top ten list: dynamic fault prediction. Proceedings of the 21st IEEE international conference on software maintenance, 263-272.

[13] Jones, M., Smith, J., & Johnson, K. (2018). The Evolution of Grocery Self-Checkout Systems: A Comprehensive Review. Journal of Retail Technology, 14(3), 112-129.

[14] Joorabchi, A. S., Mesbah, A., & Krinke, J. (2011). Automated Analysis of Web Application Security. Proceedings of the 33rd International Conference on Software Engineering, 611-620.

[15] Kontogiannis, K., Spanoudakis, G., & Finkelstein, A. (2000). Software process assessment and improvement: The Bootstrap Approach. ACM Transactions on Software Engineering and Methodology (TOSEM), 9(4), 383-433.

[16] Krug, S. (2014). Don't Make Me Think, Revisited: A Common Sense Approach to Web Usability. New Riders.

[17] Kumar, R., Kulhade, S., & Zaman, M. (2013). Software quality assessment using architectural metrics: a case study. Procedia Technology, 10, 124-129.

[18] Lehman, M. M. (1996). Laws of Software Evolution Revisited. Proceedings of the International Conference on Software Maintenance, 108-119.

[19] Lim, S. L., & Vitharana, P. (2012). Comparison of automated software testing tools: A web application case. Journal of Systems and Software, 85(3), 640-651.

[20] Liu, Z., Ma, Q., & Hu, H. (2018). Design of Payment System in Mobile E-commerce Based on Microservices Architecture. Journal of Physics: Conference Series, 1060(1), 012063.

[21] Lutz, R. R., Ammann, P., & Jeffries, R. (2015). A study of software testing practices in agile and non-agile settings. Empirical Software Engineering, 20(3), 770-817.

[22] Ramakrishnan, R., & Gehrke, J. (2000). Database Management Systems. McGraw-Hill.

[23] Rozanski, N., & Woods, E. (2012). Software Systems Architecture: Working with Stakeholders Using Viewpoints and Perspectives. Addison-Wesley Professional.

[24] Smith, J. (2010). The Role of Legacy Systems in Modern Retail Environments. International Journal of Retail Management, 24(2), 78-91.

[25] Smith, J., Johnson, K., & Brown, M. (2010). Self-Checkout Systems: Design, Implementation, and Management. Springer Science & Business Media.

[26] Sommerville, I. (2019). Software Engineering. Pearson.

[27] Tan, J., & Balci, O. (2016). A Systematic Literature Review of Software System Risk Assessment Models. Information and Software Technology, 77, 106-117.

[28] Tanenbaum, A. S., & Van Steen, M. (2014). Distributed Systems: Principles and Paradigms. Pearson.

[29] Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., & Wesslén, A. (2012). Experimentation in Software Engineering. Springer Science & Business Media.

[30] Yu, Y., Zheng, C., & Xu, J. (2015). A Study on Barcode Recognition Algorithm Based on Digital Image Processing. Procedia Computer Science, 61, 157-161.