Communications in Mathematics and Applications
Vol. 17, No. 1, pp. 235–248, 2026
ISSN 0975-8607 (online); 0976-5905 (print)
Published by RGN Publications
DOI: 10.26713/cma.v17i1.3513

http://www.rgnpublications.com

Review Article

Assessing Legacy Software Systems: Existence, Challenges, Impact, and Migration Strategies in Organizational Contexts

Sufian Khamis and Wajdi Aljudebi*

Department of Computer Science, Faculty of Computing & Information Technology, King Abdulaziz University, Jeddah, Saudi Arabia
***Corresponding author:** waljedaibi@kau.edu.sa

Received: September 12, 2025 **Revised:** November 5, 2025 **Accepted:** December 24, 2025

Abstract. Legacy software systems remain deeply embedded in modern organizations, often serving as the backbone of mission-critical operations. Despite their importance, these systems introduce significant challenges related to security, scalability, maintainability, and integration with modern technologies. This study provides an empirical examination of legacy software systems based on responses from 200 IT specialists across public and private organizations. The findings reveal high dependency on aging systems, widespread technical and organizational challenges, and substantial impact on business workflows and digital transformation efforts. The study also highlights modernization strategies currently adopted by organizations, including cloud migration, reengineering, and incremental refactoring. Based on these insights, this research proposes the need for structured evaluation models and future application of the Goal–Question–Metric (GQM) paradigm to measure the depth and breadth of legacy software system dependency. The results contribute to a deeper understanding of legacy software system realities and offer guidance for sustainable modernization practices.

Keywords. Legacy Software Systems (LSSs), Modern Software Systems (MSSs), Technological Challenges, Organizational Impact, System Modernization, Migration Strategies, Goal-Question-Metric (GQM)

Mathematics Subject Classification (2020). 97N80, 68T35

1. Introduction

Legacy software systems have been defined in multiple ways throughout the literature; however, these definitions largely complement one another rather than conflict. According to this survey, a legacy software system is defined as any software system that is five years old or more, has not received any major updates or replacements during this period, or is written in an outdated programming language. Although such systems may continue to satisfy core business functions, they frequently fail to meet evolving organizational and operational requirements. Understanding the current state of legacy software systems is critical for organizations seeking operational effectiveness across domains such as banking, healthcare, transportation, and manufacturing. The number of legacy software systems continues to grow as technological advancement accelerates the obsolescence of older platforms (Irani *et al*. [10]). Their complexity is often attributed to extensive codebases, heterogeneous environments, and intricate dependencies (Crotty and Horrocks [5]). Legacy software systems may also pose significant risks due to incompatibilities, security vulnerabilities, outdated languages like COBOL, and unsupported hardware (Hasan *et al*. [8]). Many fail to interoperate with modern applications, leading to inconsistent data formats, communication barriers, and integration challenges (Langer [14]).

Given these challenges, assessing the prevalence, nature, and organizational impact of legacy software systems has become critically important. Contemporary software solutions frequently depend on foundational legacy components, making these systems indispensable despite technical limitations (Dedeke [6]). This study examines the depth and breadth of legacy software systems within organizations, the extent to which modern software systems depend on them, and the challenges arising from their continued use. To achieve this, a survey was distributed to 200 IT professionals, capturing insights into legacy software system realities, operational concerns, and modernization strategies.

Most respondents worked in large public or private organizations, reflecting environments with complex IT infrastructures and high-volume operations. Participants had significant professional experience—most with over ten years in the IT field ensuring reliable perspectives. Their roles included executive management, IT directors, CIOs, project managers, system and network administrators, database administrators, software engineers, developers, and consultants.

This paper is structured as follows: Section 2 provides background for the research work, Section 3 presents the findings of the study, followed by the discussion in Section 4. Finally, Section 5 offers the conclusions and future work.

2. Background

2.1 Conceptual Definitions of Legacy Software Systems

The term '*legacy software system*' became widely adopted as organizations began distinguishing modern platforms from older mainframe-based systems (Bennett [5], and Sneed [22]). The literature provides a range of definitions that clarify the conceptual foundations of legacy software systems. Legacy software systems are defined across the literature as long-standing systems that continue to support essential business operations while relying on

outdated technologies, unsupported hardware, and obsolete programming environments (Sommerville [23], and Ali *et al*. [1]). Their age alone does not determine legacy status—lack of vendor support, integration limitations, stability concerns, and architectural rigidity are equally important factors (Dedeke [4]).

A number of authors further emphasize that legacy software systems are typically mission-critical platforms that have been in place for many years, still fulfil core business needs, and are often tightly bound to specific operating-system versions or hardware configurations, frequently beyond their official end-of-life (Ali *et al*. [1], Crotty and Horrocks [5], and Ransom *et al*. [21]). Other studies describe legacy software systems more broadly as previous-generation computer systems and applications whose underlying technologies have become outdated, even though they remain in active use within organizations (Ali *et al*. [1], and Sommerville [23]). Some authors also propose time-based thresholds —such as systems older than five or ten years — as practical indicators that a system is likely to exhibit legacy characteristics (de Lucia *et al*. [15], and Sneed [22]). More generally, legacy environments are seen as encompassing not only application code but also associated hardware, programming languages, data stores, embedded business processes, and organizational policies inherited from earlier technological eras (Dedeke [6], and Warren [25]).

2.2 Importance and Characteristics of Legacy Software Systems

Despite their limitations, legacy software systems often remain indispensable. They store critical business logic accumulated over decades, making replacement risky and costly (Crotty and Horrocks [5]). Organizations also face operational and cultural resistance when attempting modernization efforts (Irani *et al*. [10]). Skills shortages further complicate maintenance, as few professionals are trained in older languages like COBOL, PASCAL, or FORTRAN (Sommerville [23]).

In addition to these, numerous studies highlight that legacy software systems are not merely old technologies but mission-critical assets upon which entire business operations depend. Because core functionalities are tightly coupled with specific hardware or software versions, replacing even small components can be difficult and disruptive (Ali *et al*. [1], and Sommerville [23]). It is also well documented that a significant proportion of system-replacement projects fail, reinforcing organizations' reluctance to retire functioning legacy environments (Marchant *et al*. [17]). Hardware constraints further contribute to persistence, as older platforms often lack compatibility with modern technologies, and replacement parts may no longer be available (Bakar *et al*. [3]). Personnel-related factors likewise amplify dependency: specialists familiar with legacy environments frequently retire or leave, and insufficient documentation makes onboarding new staff extremely difficult (Sommerville [23]). Moreover, younger IT professionals tend to prefer modern technologies, reducing the talent pool capable of maintaining legacy software systems (Khadka *et al*. [12]).

Beyond technical challenges, psychological and organizational factors reinforce continued reliance on legacy software systems. Employees and managers often prefer familiar systems – a dynamic commonly described as '*resistance to change*' (Bennett [4], and Dedeke [6]). Furthermore, in many organizations, legacy software systems still successfully meet business needs, creating little internal pressure to modernize (Bakar *et al*. [3]). As a result, some

companies continue operating legacy platforms for 25 to 30 years and expect them to remain in service to achieve organizational goals (Bennett [4], and Langer [14]). Vendor support for certain legacy hardware and middleware further prolongs system lifespan (Warren [25]).

Building on this persistent organizational dependence, the literature also classifies legacy software systems into several distinct types and highlights technical characteristics that contribute to their long-term survival and associated challenges.

Legacy software systems manifest in several forms, including:

- *End-of-life systems* — software no longer supported by vendors (Ali *et al*. [1]).
- *Unscalable platforms* — systems unable to accommodate growing data or user demands.
- *Heavily patched applications* — systems modified repeatedly over time, often introducing security vulnerabilities (Sommerville [23]).
- *Systems lacking maintainability expertise* — where knowledgeable staff have retired or left (Bakar *et al*. [3]).

Common characteristics include obsolete technologies, outdated programming languages, architectural inflexibility, integration barriers, performance degradation, code rot, and outdated user interfaces (Hussain *et al*. [7], and Ramos *et al*. [20]).

Further illustrating these dependencies and characteristics, empirical evidence shows that legacy software systems remain deeply embedded across modern industries, with notable examples including COBOL-based financial applications (Hussain *et al*. [9]), Microsoft's designation of WINDOWS 7 as a legacy platform in 2020 (MICROSOFT), and mainframe systems that persist due to high migration cost and operational risk (Langer [14]). Empirical studies show that legacy software systems negatively affect digital transformation efforts, increase maintenance costs, and reduce productivity (Irani *et al*. [10], and Mallidi *et al*. [16]).

2.3 Evaluation and Modernization Strategies for Legacy Software Systems

Effective legacy software system management requires evaluating both the business value and the technical value of each system. To support decision-making, legacy software systems are commonly categorized into four quadrants — low/low, low/high, high/low, and high/high — based on system quality and business importance. This classification helps determine whether a system should be maintained, reengineered, or fully replaced (de Lucia *et al*. [15], and Ransom *et al*. [21]).

Figure 1 illustrates the relationship between technical value and business value, showing how organizations position their legacy software systems within these quadrants to inform modernization strategies.

Building on such evaluation frameworks, modernization approaches offer practical pathways for transitioning legacy software systems toward more sustainable architectures. Modernization strategies include cloud migration, incremental refactoring, complete reengineering, virtualization, and API-based integration (Hasan *et al*. [8], and Ponce *et al*. [18]). Each approach presents trade-offs related to cost, risk, and organizational readiness. Microservices-based modernization has become increasingly popular due to its modularity and scalability (Khadka [12]).

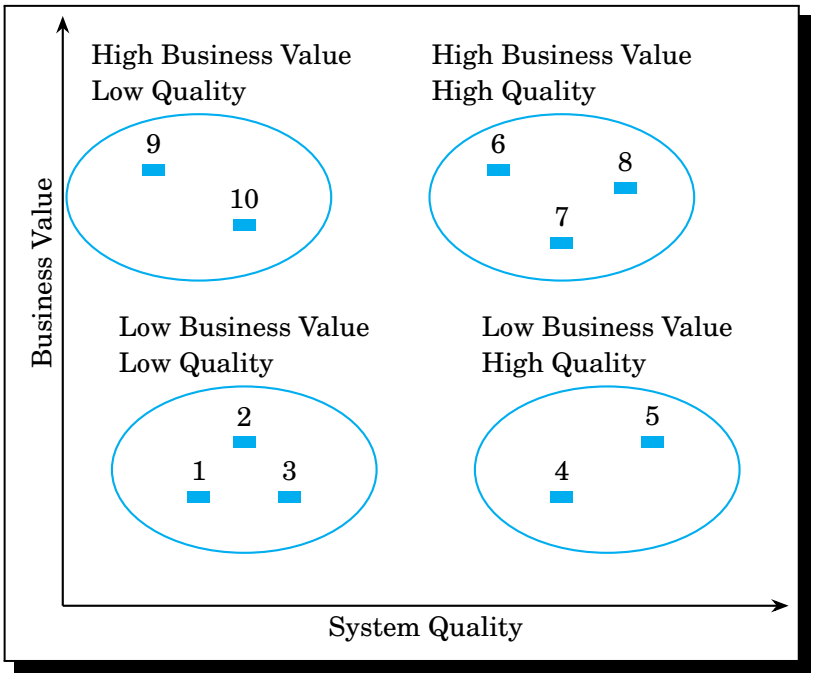

Figure 1. Legacy software system examples from the business and technical perspectives

3. Findings

3.1 Extent of Legacy Software Systems in Organizations

Survey results show that most organizations rely heavily on *legacy software systems* (LSSs). As shown in Figure 2, a large portion of these systems are more than five to ten years old or more, reflecting long-term dependence on outdated platforms. This reliance underscores insufficient modernization and highlights systemic challenges that organizations continue to face.

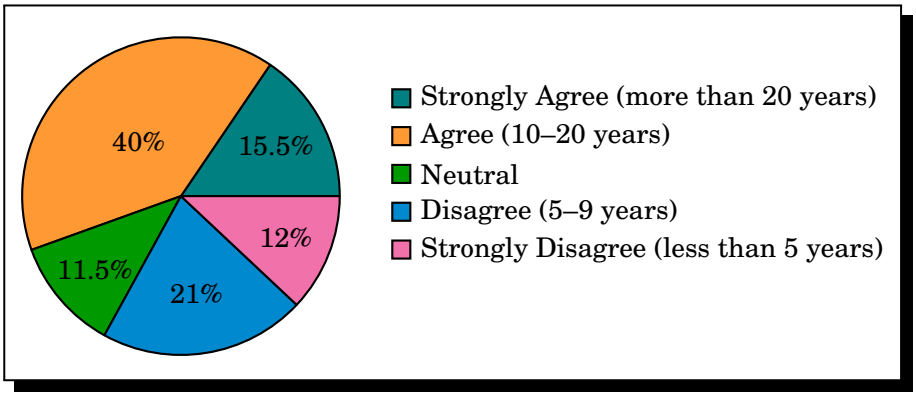

Figure 2. Average age of Legacy Software Systems (LSSs)

Building on these findings, the results indicate that more than half of the surveyed organizations have the majority of their software classified as legacy software systems, revealing a high level of dependency on legacy software systems across multiple sectors. In contrast, organizations in which most software systems are classified as modern do not exceed 17%, underscoring the widespread prevalence of legacy software systems. This distribution confirms

that the outcomes of this research are highly relevant to a broad range of organizations, particularly those experiencing the negative impacts associated with legacy software systems.

As illustrated in Figure 2, approximately 76% of organizations report that the average age of their legacy software systems exceeds five years. More critically, a substantial proportion of organizations operate legacy software systems with average ages extending to ten or even twenty years, indicating prolonged reliance on aging platforms. Furthermore, around 12% of organizations are at risk of transitioning their current systems into legacy status if timely updates or replacements are not undertaken. These results point to a significant challenge driven by weak long-term planning and insufficient continuity in system modernization efforts, emphasizing the urgency and importance of the present study.

The findings also reveal that while 46% of organizations have invested considerable effort in replacing legacy software systems over the past five years —representing a positive step toward modernization— many others have made limited progress. In these organizations, a large portion of operational software remains within the legacy software system framework, reinforcing the conclusion that legacy software systems continue to dominate organizational IT environments.

Interestingly, the survey indicates that organizations generally possess a high level of awareness regarding the risks and challenges associated with legacy software systems. This awareness is a critical prerequisite for effective risk management and informed decision-making concerning system maintenance or replacement. However, despite this awareness, many organizations continue to struggle with legacy software systems, suggesting the presence of substantial barriers— such as cost, complexity, or organizational constraints —that hinder modernization efforts. These barriers are examined in later sections of this paper.

Finally, most organizations report conducting regular maintenance activities —monthly, quarterly, or annually —on their legacy software systems. While such practices reflect proactive attempts to mitigate risks associated with aging technologies and maintain operational continuity, they often fall short of addressing root causes. Continued reliance on frequent maintenance without decisive modernization can lead to escalating costs, particularly in environments dependent on expensive platforms such as mainframe systems, where updates and upgrades incur significant financial burdens.

3.2 Types of Legacy Software Systems and Associated Challenges

Organizations operate diverse categories of LSSs including in-house applications, COBOL-based systems, outdated ERP platforms, discontinued commercial software, and unsupported mainframes. This diversity indicates that legacy software systems are not limited to mainframes or COBOL environments; rather, many systems were considered modern at the time of deployment but gradually became legacy due to insufficient updates, upgrades, or replacement strategies.

Consistent with this diversity, most respondents agreed that their organizations face significant challenges related to legacy software systems, as shown in Figure 3. These challenges span multiple dimensions, including security risks, degraded performance, limited integration with modern software systems, lack of vendor support and updates, limited interoperability, and high maintenance and upgrade costs. In addition, organizations report challenges related to

outdated user interfaces, scarcity of skilled personnel, lack of documentation, limited scalability and flexibility, and reliance on obsolete programming languages such as COBOL, Pascal, and Fortran. Furthermore, legacy software systems are associated with inefficient workflows, system instability, difficulties in understanding legacy codebases, and risks of data corruption or loss. Such findings reinforce the negative implications of retaining legacy software systems without adequate modernization efforts.

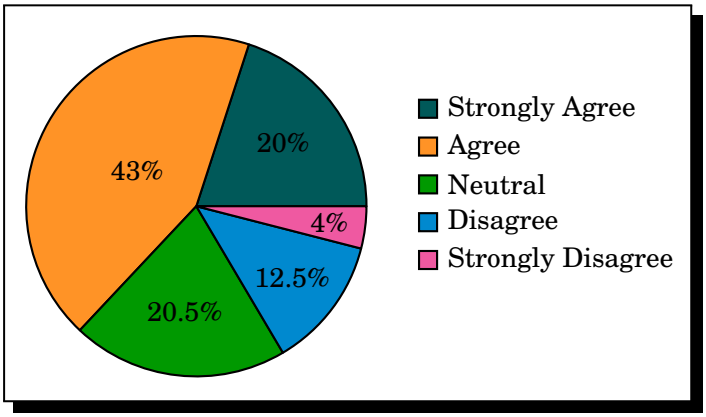

Figure 3. Challenges of Legacy Software Systems (LSSs)

Within the context of this research, the objective is to provide organizations with clear indicators that reveal the depth and breadth of their legacy software system landscape. The survey results further suggest that modern software systems frequently depend on integration with existing legacy software systems, highlighting the continued operational relevance of legacy environments.

3.3 Impact of Legacy Software Systems on the IT Business Environment

Legacy software systems impose significant operational constraints. Many organizations experience workflow delays, reduced flexibility in business rules, system downtime, and increased maintenance cost. Skill shortages further intensify risks as fewer professionals are trained to manage outdated technologies. These overarching challenges are further examined through the empirical findings presented below, which indicate that nearly half of the surveyed organizations confirm that legacy software systems have a negative impact on daily business workflows. These systems contribute to operational disruptions, process slowdowns, and reduced organizational agility, thereby constraining productivity, efficiency, and the ability to innovate or adapt to changing business demands. In addition, the continued reliance on legacy software systems limits organizations' capacity to scale their operations or sustain long-term business growth.

Interestingly, approximately one-third of respondents expressed neutral views regarding the negative workflow impact of legacy software systems. This neutrality may reflect the existence of isolated technical environments in which organizations have become closely accustomed to their legacy software systems, potentially underestimating their adverse effects on workflows. It also suggests a limited awareness of the advantages offered by modern software systems when compared to the legacy environments with which these organizations routinely interact.

The challenges are further compounded by persistent skill shortages. Organizations report increasing difficulty in recruiting and retaining professionals with expertise in legacy software systems, as experienced specialists transition to modern technology environments, retire, or leave the workforce. This growing skills gap intensifies the risks associated with maintaining and evolving legacy software systems, particularly as contemporary software development competencies increasingly dominate the labor market.

Moreover, legacy software systems frequently lack sufficient flexibility in implementing or modifying business rules, such as supporting new classifications or categories. Approximately half of the surveyed organizations have also experienced major disruptions or system downtime resulting from legacy software system failures, posing a significant threat to the continuity and stability of IT-supported business operations.

Figure 4 reveals that more than half of the respondents agreed or strongly agreed that users frequently request certain processes to be executed specifically through legacy software systems, indicating that key business processes remain tightly coupled to these environments. This observation demonstrates that legacy software systems continue to satisfy critical user and organizational needs. At the same time, it reinforces the difficulty of introducing change or modernization within such environments. The continued fulfillment of recurring business requests confirms the enduring operational value and necessity of legacy software systems within many organizational contexts, despite their well-documented limitations.

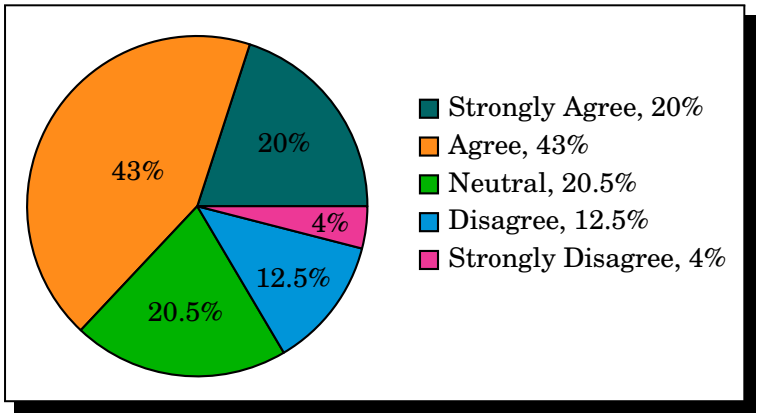

Figure 4. Frequency of user requests for LSS processes

3.4 Influence of Legacy Software Systems on Modern Software Systems

Modern software systems often depend on LSSs for core services and data. This creates tight architectural coupling that complicates modernization efforts. The empirical findings indicate that more than half of the surveyed organizations report strong integration and interdependence between legacy software systems and modern software systems. In many cases, contemporary applications —such as mobile applications— continue to rely on data and services provided by older systems. This dependency illustrates the depth of legacy software system penetration within organizational IT environments and highlights the extent to which legacy software systems influence the functionality and performance of modern software systems. Moreover, the findings suggest that modern software systems often do not operate effectively without integration with existing legacy software systems.

Conversely, organizations that have successfully updated, upgraded, or replaced their legacy software systems report fewer integration-related problems and operational obstacles compared to organizations that rely on hybrid environments combining legacy and modern systems or exclusively on legacy software systems. This contrast serves both as a warning for organizations that continue to maintain legacy software systems and as evidence of the benefits realized by those that have transitioned away from them.

According to Figure 5, many participants reported that integrating legacy and new systems is not straightforward, due to multiple technical and operational constraints. These constraints include compatibility issues between heterogeneous systems, differences in data formats, limited vendor support for legacy platforms, and the complexity of data migration processes that require careful planning and execution. Additional challenges involve security concerns arising from known vulnerabilities in legacy software systems, performance limitations that degrade overall system responsiveness, and the difficulty of implementing changes without disrupting existing functionality. Testing and debugging modifications are often time-consuming due to system complexity, and understanding outdated codebases and programming languages further exacerbates integration difficulties.

Collectively, these challenges underscore the technical and operational complexity of integrating legacy and modern software systems. As a result, organizations are often required to invest additional time, financial resources, and long-term maintenance efforts. These findings highlight the importance of carefully evaluating integration strategies and associated risks when legacy software systems remain a core component of the organizational IT landscape.

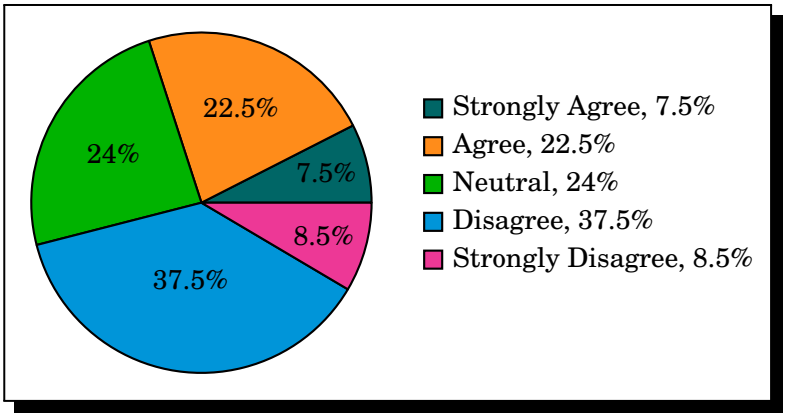

Figure 5. Ease of integration between LSSs and MSSs

3.5 Migration Plans and Modernization Strategies

Organizations increasingly acknowledge the need to modernize legacy software systems. Accordingly, the survey results indicate that most organizations recognize that their legacy software systems are no longer adequate and require replacement or modernization, as shown in Figure 6.

This acknowledgment reflects a growing awareness of the negative impact of legacy software systems on organizational performance and the necessity of modernization to meet evolving business demands. Consequently, many organizations report having active plans or ongoing efforts to replace or modernize their legacy software systems, which represents a positive shift toward addressing long-standing technical and operational limitations.

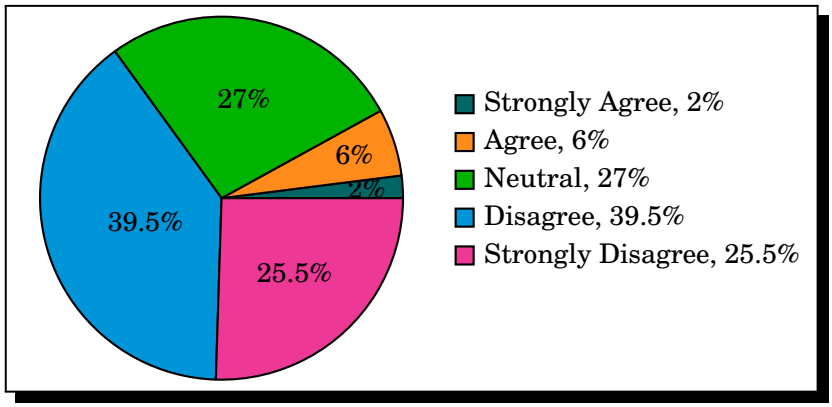

Figure 6. LSS capability without the need for replacement

To support effective decision-making, organizations prioritize legacy software systems targeted for replacement or modernization based on several criteria, including the criticality of business processes, user feedback and complaints, and the age of the software system. Based on these priorities, organizations adopt a range of migration and modernization approaches. These approaches include replacing legacy software systems with new solutions and migrating to cloud-based platforms such as *Infrastructure as a Service* (IaaS), *Platform as a Service* (PaaS), and *Software as a Service* (SaaS). In addition, organizations integrate legacy software systems with modern software systems through APIs or other integration mechanisms, while also implementing regular upgrades and patches to extend system usability. Furthermore, some organizations continue maintaining and enhancing existing systems, outsource maintenance and support activities, or build custom connectors and interfaces to improve interoperability. Finally, organizations increasingly leverage containerization or virtualization technologies to enable the coexistence of legacy software systems alongside modern platforms. When selecting among these strategies, organizations consider multiple factors, including cost, security, performance, ease of integration with existing systems, user experience, and vendor or brand reputation. These considerations play a critical role in ensuring that modernization decisions align with organizational objectives and constraints.

The survey further reveals that organizations employ various strategies to minimize disruption during legacy software system replacement or modernization. These strategies include gradual or phased migration while keeping legacy software systems operational, extensive analysis and planning prior to implementation, providing training and upskilling opportunities for employees, offering user support during transition periods, engaging users to gather requirements and feedback, maintaining systematic documentation and knowledge transfer, and adopting automated testing tools to ensure compatibility between legacy and modern software systems. Such practices are essential for reducing operational risk and maintaining business continuity throughout the transition process.

In parallel, organizations place strong emphasis on ensuring data integrity and security during migration. Common measures include implementing data backup and recovery solutions, rigorously testing and validating data migrations, continuously auditing and monitoring data during transition phases, and enforcing access controls and user permissions.

Organizations also report using several indicators to assess the effectiveness of legacy software system replacement or modernization initiatives. These indicators include improvements in system performance and efficiency, achievement of business goals, increased user satisfaction and productivity, and reductions in maintenance effort and cost. An incremental modernization approach —where systems are replaced, upgraded, or updated in phases— is frequently viewed as an effective means of managing risk and controlling costs.

According to the survey findings, organizations anticipate multiple benefits from replacing legacy software systems, including improved integration capabilities, enhanced scalability and flexibility, stronger security features, better compatibility with modern software environments, improved user interfaces, higher performance and efficiency, expanded functionality, and lower maintenance costs. The emphasis on integration capabilities, in particular, reflects the persistent challenges organizations face when integrating legacy software systems with other applications.

Despite these anticipated benefits, many organizations continue to delay replacement or modernization initiatives due to factors such as high cost, technical complexity and risk, perceived adequacy of existing systems in meeting current business needs, integration challenges, limited executive commitment, shortages of skilled personnel, resistance from end users and stakeholders, potential disruption to daily operations, data migration difficulties, and uncertainty regarding the tangible benefits of new software systems.

4. Discussion

The findings of this study reveal a complex and deeply entrenched relationship between organizations and their legacy software systems (LSSs). Although technological advancements have introduced modern and more efficient alternatives, LSSs continue to persist across public and private sectors. This persistence is consistent with observations in the literature, which highlight cost, operational risk, and organizational resistance as central factors inhibiting modernization efforts. A dominant pattern that emerges from the results is the trade-off between stability and innovation. LSSs often provide reliability and continuity, which are critical for maintaining daily operations. However, this stability is accompanied by significant drawbacks, including limited scalability, security vulnerabilities, and lack of integration capabilities. These limitations directly impact organizations' ability to pursue digital transformation initiatives, confirming the long-term risks of maintaining outdated infrastructures.

The strong architectural coupling between legacy software systems and modern software systems highlights another critical issue. Many modern applications rely on LSSs for core data and processes, making replacement or modernization risky and complex. This phenomenon aligns with the work of Langer, who argues that integration challenges often deepen over time as organizations build additional systems on top of legacy platforms.

Furthermore, the human factor plays a significant role. As identified in prior research and reflected in the survey results, organizations face a shrinking pool of professionals trained to maintain outdated technologies such as COBOL, Pascal, and mainframe environments. This skills gap increases operational vulnerability and elevates long-term maintenance costs.

The modernization strategies identified in the findings include cloud migration, reengineering, incremental refactoring, virtualization, and API-based integration. However,

these strategies are often implemented cautiously and incrementally. While safer, incremental approaches may prolong legacy dependence unless supported by structured evaluation models. The need for systematic assessment frameworks emerges as an essential implication of this study. The *Goal–Question–Metric* (GQM) paradigm, proposed for future work, offers a promising path for quantifying dependency depth, technical debt, and modernization readiness. Applying GQM aligns with recommendations in several empirical studies and could provide organizations with much-needed clarity for making strategic decisions.

Overall, legacy software systems remain operationally indispensable; however, they represent significant barriers to growth, efficiency, and digital transformation. Sustainable modernization requires not only technical upgrades but also organizational commitment, workforce development, and structured evaluation methods to guide long-term transition.

5. Conclusion and Future Work

This study provides a comprehensive empirical assessment of legacy software systems (LSSs) and their continuing influence on organizational operations, decision-making, and modernization readiness. The findings confirm that LSSs remain deeply embedded across sectors, forming the backbone of mission-critical workflows while simultaneously introducing significant risks related to security, scalability, maintainability, and integration. This dual role—being both indispensable and obstructive—reflects a long-term technological dependency that many organizations struggle to overcome.

Survey responses show that LSS persistence is driven by high migration costs, operational risks, technical debt accumulated over decades, and the deep architectural coupling between LSSs and modern software systems (MSSs). Organizations continue to rely on these systems because they encode critical business logic and have historically offered stability and reliability. However, this stability comes at the cost of flexibility and innovation. Integration difficulties, limited scalability, outdated programming languages, and diminishing pools of skilled personnel significantly hinder strategic growth and digital transformation.

The study further highlights that modernization strategies—such as cloud migration, partial reengineering, incremental refactoring, virtualization, and API-based integration—are increasingly adopted but rarely implemented fully. Organizations tend to approach modernization cautiously; balancing innovation needs with operational continuity. This incremental approach, while safer, can also prolong dependency if not guided by structured decision-making frameworks.

Future work will focus on applying the *Goal–Question–Metric* (GQM) paradigm to formalize the evaluation of LSS dependency depth and breadth. Developing quantifiable metrics will enable organizations to better assess modernization readiness, track technical debt, reduce uncertainty in migration decisions, and prioritize interventions based on measurable indicators. Additional research should also explore the socio-technical barriers associated with modernization—such as resistance to change, staff capabilities, organizational culture, and long-standing process dependencies—since these elements strongly influence modernization success.

In summary, this study contributes meaningful insights to the field of software evolution by combining empirical evidence with established theoretical perspectives. It offers actionable guidance for organizations aiming to reduce long-term reliance on legacy software systems while maintaining operational stability. Ultimately, sustainable modernization will require an integrated approach that addresses technical, organizational, and human factors in tandem.

Competing Interests

The authors declare that they have no competing interests.

Authors' Contributions

All the authors contributed significantly in writing this article. The authors read and approved the final manuscript.

References

[1] M. Ali, S. Hussain, M. Ashraf and K. Paracha, Addressing software related issues on legacy systems — A review, *International Journal of Scientific and Technology Research* **9**(3) (2020), 3738 – 3742.

[2] W. Aljedaibi and S. Khamis, Towards measuring the project management process during large scale software system implementation phase, *The ISC International Journal of Information Security* **11**(3) (2019), 161 – 172, DOI: 10.22042/isecure.2019.11.0.21.

[3] H. K. A. Bakar, R. Razali and D. I. Jambari, Implementation phases in modernisation of legacy systems, in: *6th International Conference on Research and Innovation in Information Systems* (ICRIIS, Johor Bahru, Malaysia), pp. 1 – 6, (2019), DOI: 10.1109/ICRIIS48246.2019.9073628.

[4] K. Bennett, Legacy systems: Coping with success, *IEEE Software* **12**(1) (1995), 19 – 23, DOI: 10.1109/52.363157.

[5] J. Crotty and I. Horrocks, Managing legacy system costs: A case study of a meta-assessment model to identify solutions in a large financial services company, *Applied Computing and Informatics* **13**(2) (2017), 175 – 183, DOI: 10.1016/j.aci.2016.12.001.

[6] A. Dedeke, Improving legacy-system sustainability: A systematic approach, *IT Professional* **14**(1) (2012), 38 – 43, DOI: 10.1109/MITP.2012.10.

[7] M. F. Gholami, F. Daneshgar, G. Beydoun and F. Rabhi, Challenges in migrating legacy software systems to the cloud – an empirical study, *Information Systems* **67** (2017), 100 – 113, DOI: 10.1016/j.is.2017.03.008.

[8] M. H. Hasan, M. H. Osman, N. I. Admodisastro and M. S. Muhammad, Legacy systems to cloud migration: A review from the architectural perspective, *Journal of Systems and Software* **202** (2023), 111702, DOI: 10.1016/j.jss.2023.111702.

[9] S. M. Hussain, S. N. Bhatti and M. F. U. Rasool, Legacy system and ways of its evolution, in: *International Conference on Communication Technologies* (ComTech, Rawalpindi, Pakistan, 2017), pp. 56 – 59, (2017), DOI: 10.1109/COMTECH.2017.8065750.

[10] Z. Irani, R. M. Abril, V. Weerakkody, A. Omar and U. Sivarajah, The impact of legacy systems on digital transformation in European public administration: Lesson learned from a multi case analysis, *Government Information Quarterly* **39**(2) (2022), article 101613, DOI: 10.1016/j.giq.2022.101784.

[11] R. Khadka, *Revisiting Legacy Software System Modernization*, Ph.D Thesis, Utrecht University, Utrecht, Netherlands (2016), URL: https://www.staff.science.uu.nl/~hage0101/downloads/ravikhadka16revisitinglegacysoftwaresystemmodernization-phd.pdf.

[12] R. Khadka, B. V. Batlajery, A. M. Saeidi, S. Jansen and J. Hage, How do professionals perceive legacy systems and software modernization?, in: *Proceedings of the 36th International Conference on Software Engineering* (ICSE'14, Hyderabad, India), Association for Computing Machinery, New York, NY, USA, pp. 36 – 47 (2014), DOI: 10.1145/2568225.2568318.

[13] S. Khamis and W. Aljedaibi, A framework for measuring critical success factors of large-scale software systems, *International Journal of Computer Engineering & Technology* **7**(6) (2016), 71 – 82.

[14] A. M. Langer, Legacy systems and integration, in: *Guide to Software Development: Designing and Managing the Life Cycle*, Springer, London, pp. 179 – 213, (2016), DOI: 10.1007/978-1-4471-6799-0_10.

[15] A. de Lucia, A. R. Andrea, E. Fasolino and E. Pompelle, A decisional framework for legacy system management, in: *Proceedings IEEE International Conference on Software Maintenance* (ICSM, Florence, Italy, 2001), pp. 642 – 651, (2001), DOI: 10.1109/ICSM.2001.972781.

[16] R. K. Mallidi, M. Sharma and J. Singh, Legacy digital transformation: TCO and ROI analysis, *International Journal of Electrical and Computer Engineering Systems* **12**(3) (2021), 163 – 170, DOI: 10.32985/ijeces.12.3.5.

[17] J. Marchant, C. Tjortjis and M. Turega, A metric of confidence in requirements gathered from legacy systems: Two industrial case studies, in: *Conference on Software Maintenance and Reengineering* (CSMR'06, Bari, Italy), pp. 7 – 12, (2006), DOI: 10.1109/CSMR.2006.5.

[18] F. Ponce, G. Márquez and H. Astudillo, Migrating from monolithic architecture to microservices: A rapid review, in: *2019 38th International Conference of the Chilean Computer Science Society* (SCCC) (Concepcion, Chile, 2019), pp. 1 – 7 (2019), DOI: 10.1109/SCCC49216.2019.8966423.

[19] M. Rahgozar and F. Oroumchian, A transformational approach for legacy systems' evolution, in: *WSEAS Transactions on Systems* **2**(2) (2003), 407.

[20] C. S. Ramos, K. M. Oliveira and N. Anquetil, Legacy software evaluation model for outsourced maintainer, *Eighth European Conference on Software Maintenance and Reengineering* (CSMR 2004, Tampere, Finland), pp. 48 – 57, (2004), DOI: 10.1109/CSMR.2004.1281405.

[21] J. Ransom, I. Sommerville and I. Warren, A method for assessing legacy systems for evolution, in: *Proceedings of the Second Euromicro Conference on Software Maintenance and Reengineering, 1998*, pp. 128 – 134 (1998), DOI: 10.1109/CSMR.1998.665778.

[22] H. M. Sneed, Integrating legacy software into a service-oriented architecture, in: *Conference on Software Maintenance and Reengineering* (CSMR'06, Bari, Italy), pp. 1 – 14, (2006), DOI: 10.1109/CSMR.2006.28.

[23] I. Sommerville, *Software Engineering*, 10th edition, Pearson, 810 pages (2016).

[24] M. Srinivas, G. Ramakrishna, K. R. Rao and E. S. Babu, Analysis of legacy system in software application development: A comparative survey, *International Journal of Electrical and Computer Engineering* **6**(1) (2016), 292 – 300, DOI: 10.11591/ijece.v6i1.pp292-297.

[25] I. Warren, *The Renaissance of Legacy Systems: Method Support for Software-System Evolution*, 1st edition, Springer, xiii + 182 pages (1999), DOI: 10.1007/978-1-4471-0817-7.

[26] N. Wirth, A brief history of software engineering, *IEEE Annals of the History of Computing* **30**(3) (2008), 32 – 39, DOI: 10.1109/MAHC.2008.33.