JOURNAL OF COMPUTER SCIENCE IMPLICATIONS

A Conceptual Framework for Enhancing Software System Dependability

Dr. Silvia G. Conti
Department Of Computer Engineering, Politecnico Di Torino, Italy

Dr. Andrew T. Nguyen
Department Of Software Engineering, University Of Melbourne, Australia

V0LUME01 ISSUE01 (2022)
Published Date: 21 December 2022 // Page no.: - 8-15

ABSTRACT

Software reliability is a critical attribute of software quality, fundamentally influencing user satisfaction, operational efficiency, and overall system integrity. In an era of increasingly complex and interconnected software systems, particularly within critical infrastructures and Internet of Things (IoT) environments [1], ensuring and improving software dependability has become paramount. This article proposes a conceptual framework designed to systematically enhance software system reliability throughout the software development lifecycle. Drawing upon established software engineering principles, quality models, and contemporary approaches to defect detection and reliability prediction, the proposed model integrates proactive measures, continuous monitoring, and feedback mechanisms. We synthesize insights from various reliability assessment models, quality standards, and empirical studies on factors affecting software reliability. The framework emphasizes a holistic perspective, considering not only technical aspects but also environmental and organizational influences on reliability. This conceptual model aims to provide a structured approach for practitioners and researchers to identify, mitigate, and manage reliability risks, ultimately leading to more robust and trustworthy software systems.

Keywords: - Software dependability, conceptual framework, fault tolerance, system reliability, software quality, error handling, system resilience, dependable computing, software engineering, risk mitigation.

1. INTRODUCTION

In the contemporary digital landscape, software systems underpin nearly every facet of modern society, from critical infrastructure and financial transactions to healthcare and personal communication. The pervasive nature of software necessitates a relentless focus on its quality attributes, among which **reliability** stands out as a paramount concern [44, 45]. Software reliability is defined by the IEEE Standard Glossary of Software Engineering Terminology [20, 47] as the probability that software will operate without failure for a specified period of time in a specified environment. It is a dynamic property, evolving throughout the software lifecycle, and directly impacts the trustworthiness and utility of a system [13]. Failures in software systems can lead to significant financial losses, reputational damage, safety hazards, and even loss of life, as highlighted by reports on the cost of poor software quality [48].

The complexity of modern software systems, characterized by distributed architectures, concurrent processes, and intricate interactions with hardware and other software components, exacerbates the challenge of achieving high reliability. This is particularly true for emerging paradigms like the Internet of Things (IoT), where diverse devices and networks must operate seamlessly and reliably [1]. Furthermore, the rapid pace of software development, coupled with pressure to deliver new functionalities, often leads to overlooked quality issues, contributing to an increase in software defects and vulnerabilities [8, 9]. The historical evolution of software quality models, from early frameworks like McCall's factors [34] to the widely adopted ISO/IEC 9126 [26] and its successor, ISO/IEC 25010 (SQuaRE) [25, 19], underscores the continuous effort to define, measure, and improve software quality. These models provide taxonomies of quality characteristics, with reliability consistently being a primary focus.

Traditional approaches to ensuring reliability have often centered on testing and debugging, which are crucial but inherently reactive [40]. While various software reliability growth models (SRGMs) have been proposed to predict reliability based on observed failure data during testing [12, 32, 39], their predictive power can be limited by the availability and quality of such data, especially in early development phases [46, 51]. More comprehensive strategies are needed that integrate reliability considerations across the entire software development lifecycle, from requirements engineering and design to implementation, testing, and deployment.

This article proposes a conceptual model for improving software system reliability, addressing the multifaceted nature of this quality attribute. The model aims to provide a

structured approach for practitioners to integrate proactive reliability-enhancing activities into their development processes. By synthesizing insights from established quality standards, reliability models, and empirical studies, our framework emphasizes a holistic perspective that considers technical, environmental, and organizational factors [36, 42, 53, 54]. The ultimate goal is to foster the development of more dependable software systems that meet the stringent demands of contemporary applications and user expectations.

2. METHODS

Developing a comprehensive conceptual model for improving software system reliability necessitates a systematic approach that integrates theoretical foundations, established industry practices, and empirical insights from existing literature. Our methodology for constructing this framework involved a multi-stage process of systematic literature review, synthesis of key concepts, and architectural design of the model's components. This approach is informed by best practices in systematic literature reviews within software engineering [11, 52].

2.1 Foundational Concepts and Definitions

To establish a clear understanding of software reliability, we first revisited fundamental definitions and quality models. The IEEE defines software reliability as the probability of failure-free operation for a specified time in a specified environment [20, 47]. This definition underscores the probabilistic nature of reliability and its dependence on operational context. Key software quality models, such as McCall's Factors [34], Boehm's Model [10], ISO/IEC 9126 [26], and ISO/IEC 25010 (SQuaRE) [25], were analyzed to identify how reliability is characterized and integrated within broader quality frameworks [5, 19, 37, 41]. These models uniformly recognize reliability as a critical external quality attribute, often encompassing sub-characteristics like maturity, fault tolerance, and recoverability [23, 25].

2.2 Identification of Key Factors Affecting Reliability

A crucial step in model development was to identify the multifaceted factors that influence software reliability. This involved reviewing empirical studies and surveys that explore these influences. Factors identified include:

- **Software Product Attributes:** These relate to the inherent characteristics of the software itself, such as complexity, size, architecture [18], and the number and type of defects [8, 53]. Architectural design choices, for instance, significantly impact a system's fault tolerance and recovery mechanisms [16].

- **Development Process Characteristics:** The rigor and quality of the development process play a significant role. This includes aspects like coding standards [35], testing methodologies [7, 27, 30, 40], verification and validation activities [7], and quality assurance processes [6, 21].

- **Operational Environment:** External factors, such as hardware reliability [50], operating system stability, network conditions, and even human interaction patterns, can impact perceived software reliability [36, 42, 54].

- **Organizational Factors:** Management practices, team expertise, communication, and resource allocation also indirectly influence reliability by affecting the development process and product quality.

Understanding these factors is essential for designing a model that addresses reliability comprehensively rather than focusing on isolated aspects.

2.3 Review of Existing Reliability Models and Improvement Approaches

We conducted an extensive review of existing software reliability models and improvement approaches. This included:

- **Software Reliability Growth Models (SRGMs):** These models, such as the Goel-Okumoto model [39], use historical failure data during testing to predict future reliability [12, 28, 33]. While useful, their limitations in early lifecycle phases and sensitivity to data quality were noted [46]. Recent advancements address dependent failures and uncertain environments [32].

- **Architecture-Based Reliability Analysis:** Approaches that analyze reliability at the architectural level to predict system dependability early in the design phase [18].

- **Quality Models and Standards:** Examination of ISO 9001 [24], IEEE standards for software quality assurance [21], and metrics methodology [22]. These standards provide a framework for quality management and process improvement that can directly impact reliability.

- **Defect Detection and Prevention:** Review of methods for detecting vulnerabilities [9], automatic

test case generation [27], and the broader role of testing [14, 30].

- **Computational Intelligence Approaches:** Exploration of how machine learning and other AI techniques can be applied for software quality prediction, including reliability [2, 3].

This review provided a rich repository of techniques and perspectives from which to derive components for our conceptual model.

2.4 Conceptual Model Design

Based on the synthesis of the aforementioned concepts and reviews, we proceeded to design the conceptual model. The design process involved:

1. **Identifying Core Components:** Determining the main building blocks necessary for a holistic approach to reliability improvement.
2. **Defining Relationships:** Establishing how these components interact and influence each other across the software development lifecycle.
3. **Proposing Key Activities:** Outlining the practical actions and considerations within each component.
4. **Emphasizing Lifecycle Integration:** Ensuring that reliability is not treated as an afterthought but as an integral part of every phase.

The resulting model is intended to be a high-level framework that can be adapted to various software development contexts, from embedded systems in automated mining [17] to critical infrastructure software [29]. It moves beyond merely measuring reliability to actively guiding its enhancement.

3. Results (Conceptual Model Components)

The synthesis of literature and analysis of software reliability factors culminated in the proposed conceptual framework for enhancing software system dependability. This section outlines the key components of this model, illustrating how they interact to provide a systematic approach to reliability improvement throughout the software development lifecycle. Since this is a conceptual model, the "results" presented here are the model's architecture and its constituent elements, rather than empirical data from an experiment.

3.1 Integrated Quality Planning and Requirements Engineering

The foundation of software reliability is laid during the initial phases of development. The model emphasizes:

- **Reliability-Focused Requirements Elicitation:** Explicitly defining non-functional requirements related to reliability, including mean time between failures (MTBF), availability, and fault tolerance [10, 25]. This ensures that reliability is a primary consideration from the outset.
- **Early Risk Identification:** Proactively identifying potential reliability risks through architectural analysis [18] and threat modeling. This includes considering interactions with hardware [50] and external environmental factors [42].
- **Integration with Quality Standards:** Aligning initial planning with established software quality standards like ISO 9001 [24], IEEE 730-2014 for Software Quality Assurance Processes [21], and ISO/IEC 25010 [25]. These standards provide a robust framework for quality management [6].

3.2 Robust Design and Architecture

Reliability is significantly influenced by architectural choices. The model proposes:

- **Resilient Design Patterns:** Employing architectural patterns that promote fault tolerance, such as redundancy, error detection, and recovery mechanisms [16]. This includes designing for graceful degradation and self-healing capabilities.
- **Modularity and Decoupling:** Promoting modular design to limit the propagation of errors and simplify maintenance, which indirectly contributes to reliability [35].
- **Security by Design:** Incorporating security considerations from the architectural phase to reduce vulnerabilities, as security flaws can often manifest as reliability issues [4, 9].

3.3 Proactive Implementation and Code Quality Management

The implementation phase is where design decisions translate into executable code, and quality directly impacts reliability. This component includes:

- **Adherence to Coding Standards:** Enforcing strict coding standards and best practices to minimize the introduction of defects [35].

- **Static Code Analysis:** Utilizing automated tools for static analysis to identify potential bugs, vulnerabilities, and code smells early in the development cycle, before execution [9].

- **Peer Reviews and Inspections:** Conducting thorough code reviews to detect errors that might be missed by automated tools.

- **Defect Classification and Tracking:** Systematically classifying and tracking software defects to understand their root causes and prevent recurrence [8].

3.4 Comprehensive Verification and Validation (V&V)

Testing and V&V activities are crucial for uncovering defects and assessing reliability. The model emphasizes:

- **Multi-Stage Testing Strategy:** Implementing a comprehensive testing strategy that includes unit testing, integration testing, system testing, and acceptance testing [7, 40].

- **Reliability Testing:** Specifically designing tests to measure reliability metrics, such as failure rates and mean time to failure (MTTF) [33]. This involves subjecting the software to stress, load, and long-duration execution under realistic operational profiles.

- **Automated Test Case Generation:** Leveraging techniques for automatic test case generation to improve test coverage and efficiency in error detection [27, 30].

- **White-Box Testing:** Employing white-box testing techniques to scrutinize internal code paths and logic, which can reveal deeper reliability issues [14].

- **Software Reliability Growth Modeling (SRGM):** Applying SRGMs during the testing phase to track defect discovery and predict when a target reliability level might be achieved [12, 33, 39]. This provides a quantitative measure of progress and helps in making release decisions.

3.5 Continuous Monitoring and Feedback Loop

Reliability is not a static state but requires continuous attention throughout the operational life of the software. The model proposes:

- **Real-time Performance Monitoring:** Implementing tools to monitor system performance, errors, and failures in real-time in the production environment.

- **User Feedback Integration:** Establishing channels for collecting and analyzing user feedback on perceived reliability and encountered issues.

- **Incident Management and Post-Mortem Analysis:** Systematically addressing incidents, conducting root cause analysis for failures, and feeding lessons learned back into the development process to prevent future occurrences.

- **Predictive Maintenance/Reliability:** Utilizing computational intelligence and machine learning techniques for software quality prediction [2, 3], enabling proactive identification of potential reliability degradation before failures occur [49]. This includes analyzing environmental factors that can influence reliability [36, 42, 54].

3.6 Iterative Improvement and Knowledge Management

The model inherently supports an iterative improvement cycle driven by data and lessons learned.

- **Metrics-Driven Improvement:** Using software quality metrics, particularly those related to reliability [22, 43, 51], to identify areas for improvement and track the effectiveness of interventions.

- **Knowledge Base Development:** Creating a centralized knowledge base of common failure modes, solutions, and best practices to facilitate organizational learning and prevent recurring reliability issues.

- **Regular Audits and Reviews:** Conducting periodic audits of development processes and system performance against established reliability goals and standards.

These components collectively form a holistic framework, recognizing that improving software reliability is an ongoing process that permeates all stages of the software lifecycle and requires collaboration across all stakeholders.

4. DISCUSSION

The proposed conceptual framework for enhancing software system dependability offers a structured and comprehensive approach to addressing one of the most critical aspects of software quality. By integrating elements from various software engineering disciplines and drawing upon an extensive body of research, this model extends

beyond traditional, often reactive, reliability testing to encompass proactive measures throughout the entire software development lifecycle. This holistic view aligns with the evolving understanding of software quality, where attributes like reliability, usability, and maintainability are increasingly interdependent [41].

A key strength of the model is its emphasis on *early intervention*. By advocating for reliability considerations during requirements engineering and architectural design, the framework aims to prevent defects that are typically more costly to fix in later stages. This resonates with the principles of "building quality in" rather than "testing quality in." The inclusion of explicit reliability requirements and architectural patterns for resilience directly tackles the inherent challenges of complex systems and systems-of-systems [16], moving beyond simple defect counting to considering the system's ability to withstand and recover from faults.

The detailed focus on *proactive implementation and code quality management* underlines the fact that reliability is not solely a design artifact but is deeply embedded in the code itself. Adherence to coding standards, rigorous static analysis, and peer reviews are fundamental practices that minimize latent defects, which, if left unaddressed, can lead to runtime failures [35]. The systematic classification of defects [8] is crucial for learning from past mistakes and continuously refining development processes.

The *comprehensive verification and validation* component stresses the importance of a multi-faceted testing strategy, including dedicated reliability testing. While software reliability growth models (SRGMs) have historical significance [12, 33], the model acknowledges their limitations and advocates for their judicious application alongside other V&V activities. The integration of advanced techniques like automated test case generation [27, 30] and white-box testing [14] further enhances the model's practical utility, ensuring a more thorough exploration of the software's behavior under various conditions.

Perhaps one of the most forward-looking aspects of the framework is its emphasis on *continuous monitoring and a feedback loop*. In today's dynamic operational environments, reliability cannot be guaranteed solely at release time. Real-time performance monitoring, coupled with robust incident management and root cause analysis, enables organizations to swiftly respond to failures and leverage operational data for continuous improvement. The integration of computational intelligence for predictive reliability [2, 3, 49] represents a significant shift from reactive problem-solving to proactive risk mitigation, allowing for the anticipation and prevention of failures before they impact users. This addresses the influence of dynamic environmental factors on reliability [36, 42, 54].

The iterative nature of the model, driven by metrics and knowledge management, creates a virtuous cycle of improvement. By systematically tracking reliability metrics [22, 43, 51] and formalizing lessons learned, organizations can build institutional knowledge that continually enhances their ability to produce dependable software. This approach supports a culture of quality that is essential for long-term software success.

Limitations of the Conceptual Model:

As a conceptual framework, this model does not prescribe specific tools, techniques, or quantitative measures for every component. Its implementation would require tailoring to specific organizational contexts, project sizes, and domain requirements. The effectiveness of the model relies heavily on organizational commitment, resource allocation, and a mature software engineering culture. Furthermore, the interplay between software reliability and other non-functional requirements (e.g., performance, security, usability) is complex, and while the model touches upon some of these interactions (e.g., security contributing to reliability [9]), a deeper exploration of these interdependencies could further refine the framework.

Future Work:

Future research could focus on validating this conceptual model through empirical studies in diverse industrial settings. This would involve:

- Developing specific metrics and benchmarks for each component of the framework.
- Conducting case studies on projects that adopt elements of this framework to evaluate its impact on actual software reliability.
- Exploring the practical challenges and success factors in implementing such a comprehensive model within different organizational structures and development methodologies (e.g., Agile, DevOps).
- Investigating the specific contributions of various computational intelligence techniques in predicting and enhancing reliability within the proposed framework.
- Further dissecting the influence of human factors and organizational culture on the successful adoption and execution of reliability improvement processes.

5. CONCLUSION

Software reliability remains a cornerstone of software quality, critical for the success and trustworthiness of modern systems. This article has proposed a conceptual framework designed to systematically enhance software system dependability by integrating proactive measures across the entire software development lifecycle. The model moves beyond reactive testing to advocate for early reliability planning, robust architectural design, diligent code quality management, comprehensive verification and validation, continuous operational monitoring, and an iterative improvement feedback loop.

By synthesizing insights from decades of software engineering research, quality standards, and contemporary advancements in defect detection and reliability prediction, this framework offers a holistic perspective. It underscores that achieving high software reliability is a continuous journey that requires a concerted effort from all stakeholders, integrating technical rigor with robust process management and a culture of quality. While further empirical validation is warranted, this conceptual model provides a valuable roadmap for practitioners and researchers striving to build and maintain more dependable software systems in an increasingly complex technological landscape.

REFERENCES

[1] Abdallah, M., Jaber, T., Alabwaini, N., & Abd Alnabi, A. (2019). A proposed quality model for the Internet of Things systems. In *2019 IEEE Jordan International Joint Conference on Electrical Engineering and Information Technology*, 23-27.

[2] Alaswad, F., & Poovammal, E. (2022). Software quality prediction using machine learning. *Materials Today: Proceedings*, 62, 4714-4720. https://doi.org/10.1016/j.matpr.2022.03.165.

[3] Albeanu, G., Madsen, H., Popenţiu-Vlădicescu, F. (2020). Computational Intelligence Approaches for Software Quality Improvement. *Reliability and Statistical Computing: Modeling, Methods and Applications*, 305-317.

[4] Alguliyev, R. M. & Mahmudov R. Sh. (2019). Sensitive personal data in the national mentality context and its security provision. *Problems of Information Society*, №2, 117–128.

[5] Al-Qutaish, R. E. (2010). Quality models in software engineering literature: an analytical and comparative study. *Journal of American Science*, 6(3), 166-175.

[6] Bayramova, T.A. (2020). Analysis of software engineering standards. *Problems of Information Society*, 11(1), 83–95.

[7] Bayramova, T.A. & Abbasova N.P. (2016). Verification and validation of software / «Questions of application of mathematics and new information technologies» *III republican scientific conference, Sumgayit, December 15*, 197-198.

[8] Bayramova, T.A. (2022). Classification of software defects. *Proceedings of the III international scientific conference on information systems and technologies: achievements and perspectives, Sumgayit*, 256-258.

[9] Bayramova, T.A. (2022). Analysis of Modern Methods for Detecting Vulnerabilities in Software for Industrial Information Systems // *Cybersecurity for Critical Infrastructure Protection via Reflection of Industrial Control Systems, NATO Science for Peace and Security Series D: Information and Communication Security*. - Amsterdam, 160-162.

[10] Boehm, B. W., Brown, J. R., Kaspar, H., Lipow, M., McLeod, G., Merritt, M. (1978). *Characteristics of Software Quality*. North Holland Publishing, Amsterdam, The Netherlands, 45-68, 169.

[11] Brereton, P., Kitchenham, B. A., Budgen, D., Turner, M., Khalil, M. (2007). Lessons from applying the systematic literature review process within the software engineering domain. *Journal of systems and software*, 80(4), 571-583. https://doi.org/10.1016/j.jss.2006.07.009.

[12] Cristescu, M. P., Stoica, E. A., Ciovică, L. V. (2015). The comparison of software reliability assessment models. *Procedia Economics and Finance*, 27, 669-675. https://doi.org/10.1016/S2212-5671(15)01047-3.

[13] Cusick, J. J. (2019). The first 50 years of software reliability engineering: A history of SRE with first person accounts. *arXiv preprint arXiv:1902.06140*.

[14] Daniels, D., Myers, R., & Hilton, A. (2003). White box software development. In *Current Issues in Safety-Critical Systems: Proceedings of the Eleventh Safety-critical Systems Symposium, Bristol, UK, 4–6 February 2003*, 119-136. London: Springer London. https://doi.org/10.1007/978-1-4471-0653-1_7.

[15] Febrero, F., Calero, C., & Moraga, M. Á. (2016). Software reliability modeling based on ISO/IEC SQuaRE. *Information and Software Technology*, 70, 18-29. https://doi.org/10.1016/j.infsof.2015.09.006.

[16] Ferreira, F. H. C., Nakagawa, E. Y., dos Santos, R. P. (2023). Towards an understanding of reliability of software-intensive systems-of-systems. *Information and Software Technology*, 158, 107186.

[17] Ghodrati, B., Hadi Hoseinie, S., Garmabaki, A. H. S. (2015). Reliability considerations in automated mining systems. *International journal of mining, reclamation and environment*, 29(5), 404-418.

[18] Gokhale, S. S. (2007). Architecture-based software reliability analysis: Overview and limitations. *IEEE Transactions on dependable and secure computing*, 4(1), 32-40.

[19] Gordieiev, O., Kharchenko, V., Fominykh, N., Sklyar, V. (2014). Evolution of software quality models in context of the standard ISO 25010. In *Proceedings of the Ninth International Conference on Dependability and Complex Systems DepCoS-RELCOMEX. June 30–July 4, 2014, Brunów, Poland*, pp. 223-232. Springer International Publishing. https://doi.org/10.1007/978-3-319-07013-1_21

[20] IEEE 610.12-1990 Standard Glossary of Software Engineering Terminology. https://ieeexplore.ieee.org/document/159342

[21] IEEE 730-2014 Standard for Software Quality Assurance Processes. https://ieeexplore.ieee.org/document/6835311

[22] IEEE 1061-1992 Standard for a Software Quality Metrics Methodology. https://ieeexplore.ieee.org/document/237006.

[23] In use qualities from ISO/IEC 25010. 3-4. https://www.irit.fr/recherches/ICS/projects/twintide/upload/435.pdf

[24] ISO 9001 and related standards. https://www.iso.org/iso-9001-quality-management.html

[25] ISO/IEC 25010:2011 Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models. https://www.iso.org/obp/ui/#iso:std:iso-iec:25010:ed-1:v1:en

[26] ISO/IEC 9126-1:2001 Software engineering — Product quality — Part 1: Quality model. https://www.standards.ru/document/3617603.aspx

[27] Jalilian, S., & Mahmudova, S. J. (2022). Automatic generation of test cases for error detection using the extended Imperialist Competitive Algorithm. *Problems of Information Society*, 46-54.

[28] Jatain, A., & Mehta, Y. (2014). Metrics and models for software reliability: A systematic review. In *2014 International Conference on Issues and Challenges in Intelligent Computing Techniques (ICICT)*, 210-214. IEEE. doi: 10.1109/ICICICT.2014.6781281.

[29] Jharko, E. (2021). Ensuring the software quality for critical infrastructure objects. *IFAC-PapersOnLine*, 54(13), 499-504. https://doi.org/10.1016/j.ifacol.2021.10.498.

[30] Kazimov, T. H., Bayramova, T. A., & Malikova, N. J. (2021). Research of intelligent methods of software testing. *System Research & Information Technologies*, 4, 42-52.

[31] Kumar, A., & Gupta, D. (2017). Paradigm shift from conventional software quality models to web based quality models. *International Journal of Hybrid Intelligent Systems*, 14(3), 167-179.

[32] Lee, D. H., Chang, I. H., & Pham, H. (2022). Software reliability growth model with dependent failures and uncertain operating environments. *Applied Sciences*, 12(23), 12383.

[33] Maevsky, D., Kharchenko, V., Kolisnyk, M., & Maevskaya, E. (2017, September). Software reliability models and assessment techniques review: Classification issues. In *2017 9th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications*, 2, 894-899. doi: 10.1109/IDAACS.2017.8095216.

[34] McCall, J., Paul, K., Richards and F. Walters (1977). Factors in software quality: concept and definitions of software quality / *Final Technical Report General Electric Company*, 1, 25-31, 168.

[35] McConnell, S. (2004). *Code complete*. Pearson Education. 952.

[36] Mengmeng Z., Xuemei Z., Hoang P. (2015). A comparison analysis of environmental factors affecting software reliability, *Journal of Systems and Software*, 109, 150-160, https://doi.org/10.1016/j.jss.2015.04.083.

[37] Miguel, J. P., Mauricio, D., Rodríguez, G. (2014). A review of software quality models for the evaluation of software products. *arXiv preprint arXiv:1412.2977*.

[38] Musa, J. D., & Everett, W. W. (1990). Software-reliability engineering: Technology for the 1990s. *IEEE Software*, 7(6), 36-43.

[39] Nagar, P., & Thankachan, B. (2012). Application of Goel-Okumoto model in software reliability measurement. *Int. J. Comp. Appl. Special Issue ICNICT*, 5, 1-3.

[40] Nayyar, A. (2019). *Instant approach to software testing: Principles, applications, techniques, and practices*. BPB Publications, India, 2019, 99-101, 368.

[41] Ndukwe, I. G., Licorish, S. A., Tahir, A., MacDonell, S. G. (2023). How have views on software quality differed over time? Research and practice viewpoints. *Journal of Systems and Software*, 195, 111524.

[42] Ozcan, A., Çatal, Ç., Togay, C., Tekinerdogan, B., Donmez, E. (2020). Assessment of environmental factors affecting software reliability: A survey study. *Turkish Journal of Electrical Engineering and Computer Sciences*, 28(4), 1841-1858.

[43] Rashid, J., Mahmood, T., Nisar, M. W. (2019). A study on software metrics and its impact on software quality. *Technical Journal, University of Engineering and Technology (UET) Taxila, Pakistan*, 24(1), 1-14.

[44] Sahu, K., & Srivastava, R.K. (2019). Revisiting software reliability. *Data Management, Analytics and Innovation: Proceedings of ICDMAI 2018*, 1, 221-235.

[45] Sahu, K., & Srivastava, R. K. (2020). Needs and importance of reliability prediction: An industrial perspective. *Information Sciences Letters*, 9(1), 33-37. https://digitalcommons.aaru.edu.jo/isl/vol9/iss1/5

[46] Smidts, C., Stutzke, M., Stoddard, R. W. (1998). Software reliability modeling: an approach to early reliability prediction. *IEEE Transactions on Reliability*, 47(3), 268-278. doi: 10.1109/24.740500.

[47] Standard Glossary of Software Engineering Terminology, STD-729-1991, ANSI/IEEE, 1991

[48] The cost of poor software quality in the us: a 2022 report. (2023). https://www.it-cisq.org/the-cost-of-poor-quality-software-in-the-us-a-2022-report/

[49] Van Driel, W. D., Bikker, J. W., Tijink, M. (2021). Prediction of software reliability. *Microelectronics Reliability*, 119, 114074.

[50] Van Driel, W. D., Schuld, M., Wijgers, R., Van Kooten, W. E. J. (2014). Software reliability and its interaction with hardware reliability. In *2014 15th International Conference on Thermal, Mechanical and Mulit-Physics Simulation and Experiments in Microelectronics and Microsystems (EuroSimE) Ghent, Belgium*, 1-8. IEEE. doi: 10.1109/EuroSimE.2014.6813774

[51] Yadav, H. B., & Yadav, D. K. (2017). Early software reliability analysis using reliability relevant software metrics. *International Journal of System Assurance Engineering and Management*, 8, 2097-2108. https://doi.org/10.1007/s13198-014-0325-3

[52] Yang, L., Zhang, H., Shen, H., Huang, X., Zhou, X., Rong, G., Shao, D. (2021). Quality assessment in systematic literature reviews: A software engineering perspective. *Information and Software Technology*, 130, 106397. https://doi.org/10.1016/j.infsof.2020.106397

[53] Zhang, X., & Pham, H. (2000). An analysis of factors affecting software reliability. *Journal of Systems and Software*, 50(1), 43-56. https://doi.org/10.1016/S0164-1212(99)00075-8

[54] Zhang, X., Shin, M. Y., Pham, H. (2001). Exploratory analysis of environmental factors for enhancing the software reliability assessment. *Journal of Systems and Software*, 57(1), 73-78.