East Journal of Computer Science
ISSN: 3079-9406

Volume 1, Issue 2
Publisher: East Publication & Technology
DOI: https://doi.org/10.63496/ejcs.Vol1.Iss2.79

Optimizing Software Quality: Integrating Test Case Prioritization, Defect Prediction, and Resource Allocation Strategies

Abdallh Mostafa Mohamed Mohamed Menshawy [1], Mohammad Nasar*[2]

[1]Student, Computing and Informatics Department, Mazoon College, Muscat, Oman, nasar31786@gmail.com.
[2]Computing and Informatics Department, Mazoon College, Muscat, Oman, 2219564@mazcol.edu.om.
*Corresponding Author.

Received: 29/03/2025, **Revised**: 30/03/2025, **Accepted**: 07/04/2028, **Published**: 08/04/2025

***Abstract*:**

Software quality assurance is essential for reliable and effective systems, especially in critical fields like healthcare and autonomous vehicles. Yet, limited resources, slow fault detection, and the growing complexity of software designs—such as modular and distributed setups—create tough challenges. This paper explores progress in four key areas: requirement-based test case prioritization, software defect prediction, reliability checks for component-based systems, and resource allocation strategies. We reviewed 35 studies from 1992 to 2021, comparing older methods with newer ones using machine learning and deep learning. Our work shows that smart prioritization catches 30% more faults early, defect prediction models hit precision scores of 0.88–0.92, and resource allocation cuts testing effort by 25% without losing coverage. Still, issues like scaling up, real-world testing, and linking these methods together need more work. This study points out these gaps and suggests a combined approach to bring prioritization, prediction, and allocation into one system, aiming to improve software quality for today's demanding applications.

***Keywords*: Software Testing, Test Case Prioritization, Defect Prediction, Reliability, Resource Allocation, Machine Learning, Deep Learning.**

1. Introduction

Software has become the backbone of critical industries like healthcare, aerospace, and transportation, where even small glitches can lead to big losses or safety risks [1]. Old-school testing methods, built on manual steps and basic benchmarks, can't keep up with today's software—huge, modular, and constantly changing [2]. With systems now juggling millions of lines of code and regular updates, developers face a tough job ensuring everything works right [3]. To tackle this, new ideas have popped up: test case prioritization to spot faults fast [4], defect prediction with machine learning (ML) and deep learning (DL) to catch problems early [5], reliability checks for component-based systems (CBS) to confirm they hold up [6], and resource allocation to make testing smarter under tight limits [7].

Test case prioritization picks out the most important tests first, a must for quick-turnaround setups like agile or continuous integration [8]. Defect prediction uses clever tools to find weak spots before they break, cutting the need for endless testing [9], [10]. Reliability work makes sure CBS—where parts work together—stays solid despite tricky interactions [11]. Resource allocation, often guided by techniques like genetic algorithms, spreads effort where it counts most [12]. All these pieces aim to keep software trustworthy in the real world [13].

This paper pulls together 35 studies from 1992 to 2021, from folks like Dahiya and Solanki [4], [8], Yadav and Kishan [6], [11], Nasar and Johri [7], [12], and Karunanithi [14]. We set out to: (1) check what's working now in these areas [15], (2) weigh old ways against new tricks [16], and (3) map out where research should head next to tie these together [17]. By connecting these dots, we hope to push software quality forward, matching the needs of a world that's more digital and linked than ever [18].

Copyright: ©2025 The author(s). Published by East Publication and Technology. This is an open-access article distributed under the terms of the Creative Commons Attribution 4.0 International License. which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

2. Related Work

This review covers 35 studies from 1992 to 2021, exploring test case prioritization, reliability prediction, defect prediction, and resource allocation. Early efforts introduced foundational models [14], while recent works leverage ML and DL [5], [10]. Dahiya and Solanki tackle prioritization and testing issues [4], [8], Yadav and Kishan focus on CBS reliability [6], [11], Nasar and Johri address resource allocation [7], [12], and defect prediction spans statistical to neural network methods [13], [19].

2.1 Test Case Prioritization and Testing Challenges

Prioritization optimizes testing by targeting critical faults first [8]. Dahiya and Solanki explore requirement-based methods, linking test cases to key functions for early fault detection [4]. Their review of regression testing sorts techniques into coverage-based, fault-history-based, and hybrid types [8]. Coverage-based approaches cover extensive code but demand heavy computation [20], while fault-history methods use past data efficiently, though they weaken with code changes [21]. They note challenges like incomplete requirements and test environment mismatches [1], proposing risk-based testing to prioritize severe faults, vital for safety-critical software [1], [22].

2.2 Software Reliability in Component-Based Systems

CBS reliability hinges on component interactions [6]. Yadav and Kishan define it as the likelihood of failure-free operation, using models like McCall's to assess quality [23]. Traditional models falter with dynamic dependencies [11], prompting neural network solutions with high accuracy [24]. These require significant data and resources [25]. Karunanithi et al.'s early connectionist models [14] evolved into advanced frameworks like evolutionary networks [26], improving CBS reliability prediction [27].

2.3 Software Defect Prediction

Defect prediction has shifted from manual checks to ML and DL [5]. Al Qasem and Akour use DL for precise fault detection [5], while Qiao et al. analyze code metrics with neural networks, outperforming older ML [10]. Ensemble methods [13] and LASSO-SVM [19] handle data imbalance [28], building on early SVM and ARIMA models [29], [30]. These tools excel but need large datasets [16].

2.4 Resource Allocation in Software Testing

Resource allocation balances effort and quality [7]. Nasar and Johri use genetic algorithms to cut testing effort while maintaining coverage [12], effective in distributed systems [31]. Dynamic strategies adapt to project needs [17], and early forecasting models enhance scalability [32], though complexity limits small-team use [18].

While all four approaches aim to improve software quality, they target different stages of testing. Test case prioritization organizes tests for efficiency, defect prediction identifies vulnerable code areas, reliability models assess system stability, and resource allocation optimizes testing efforts. Traditional methods work sufficiently for small projects, but machine learning techniques prove more accurate for complex systems—though they require more data and computing power. Together, these approaches could be combined into a unified framework for stronger software quality assurance.

3. Methodology

This study employs a systematic literature review (SLR) to evaluate 35 peer-reviewed articles published between 1992 and 2021, drawn from reputable databases including IEEE Xplore, SpringerLink, Scopus, and Elsevier ScienceDirect [1], [2]. The purpose is to analyze progress in four critical areas of software quality assurance: requirement-based test case prioritization, reliability prediction for component-based systems (CBS), software defect prediction, and resource allocation strategies. The review follows a structured SLR framework based on established software engineering research practices [3], ensuring a thorough and impartial assessment.

Study selection relied on precise criteria, limiting inclusion to peer-reviewed journal articles and conference papers offering substantial empirical or theoretical insights into test case prioritization [4], [8], CBS reliability [6]–[11], defect prediction [5], [13]–[15], and resource allocation [7], [12], [16]–[22]. Non-peer-reviewed works, non-

English publications, or papers lacking relevance were excluded [23]. The search process began with keywords such as “test case prioritization,” “software reliability,” “defect prediction,” and “resource allocation,” combined with terms like “machine learning” to encompass both conventional and advanced methods [24]. This yielded 150 articles, which were refined through a two-step process: title and abstract screening reduced the set to 75, followed by full-text analysis to select the final 35, spanning contexts from open-source to industrial applications [25], [26].

Data were extracted focusing on three key metrics: fault detection rate (prioritization), prediction accuracy (reliability and defect prediction), and resource efficiency (allocation) [27]. Both quantitative results, such as accuracy percentages, and qualitative observations, like scalability challenges, were documented [28]. The analysis compared traditional approaches—e.g., coverage-based prioritization [8]—with modern techniques, such as deep learning [3] and genetic algorithms [2], using numerical data and thematic evaluation [29]. Findings are organized in Table 1 (study domains) and Table 2 (method comparisons) [30]. Validation involved cross-referencing with existing reviews [31] and verifying primary sources [32], providing a reliable foundation for the subsequent findings.

4. Analysis and Findings

Our analysis compares traditional and intelligent methods across the four domains, detailed in Tables III and IV.

4.1 Test Case Prioritization

Risk-based prioritization boosts fault detection by 30% over random methods [1], with coverage-based techniques achieving 85% code coverage but needing more resources [8].

4.2 Reliability Prediction

Neural network models for CBS hit 85–90% accuracy, surpassing traditional metrics [9], building on early efforts [14].

4.3 Defect Prediction

DL models achieve 0.88–0.92 precision [10], with LASSO-SVM improving recall by 15–20% on skewed data [19].

4.4 Resource Allocation

Genetic algorithms reduce effort by 25% [12], with dynamic methods adding 10–15% efficiency [17].

Table 1: Study Categories

Domain	References	Focus
Prioritization	[1], [4], [8]	Fault detection order
Reliability	[6]–[11], [14]	CBS reliability
Defect Prediction	[5], [13]–[15]	Fault prediction
Resource Allocation	[7], [12], [16]–[22]	Effort optimization

Table 2: Method Comparison

Method Type	Example	Advantage	Drawback
Traditional	Coverage [8]	Broad coverage	High resource use
Intelligent	DL [10], GA [12]	High accuracy	Data needs

Table 3: Key Metrics

Domain	Metric	Value	Studies
Prioritization	Fault Detection Gain	30%	[1], [4]
Reliability	Accuracy	85–90%	[9], [14]
Defect Prediction	Precision	0.88–0.92	[10], [19]
Resource Allocation	Effort Reduction	25%	[12], [17]

Table 4: Insights

Domain	Strength	Challenge
Prioritization	Early fault catch	Scalability
Reliability	High precision	Data reliance
Defect Prediction	Predictive power	Dataset quality
Resource Allocation	Efficiency	Complexity

5. Discussion

Our findings reveal that modern methods significantly improve software quality, though they bring challenges that demand practical solutions [1], [2]. Risk-based test case prioritization excels at detecting faults quickly, a clear advantage for fast-paced environments like agile and DevOps [4]. By targeting high-risk areas first, it proves invaluable for systems where errors could lead to serious consequences [8]. However, scaling it to large projects remains difficult—coverage-based techniques require substantial computational resources [5]. Developing adaptable prioritization rules that adjust to project size could maintain efficiency without excessive demands [12]. Simpler heuristic approaches might also ease the burden for complex systems, offering a workable compromise [18].

Reliability prediction for component-based systems (CBS) benefits greatly from neural networks, which deliver impressive accuracy [9]. These models adeptly manage intricate component interactions, surpassing traditional statistical methods [11]. Yet, their dependence on extensive data and advanced hardware limits their reach, particularly for smaller teams [14]. Incorporating real-time data from operational software could reduce this reliance, sharpening predictions with less upfront investment [25]. Blending these sophisticated tools with basic statistical techniques might broaden access, drawing on proven earlier concepts [21].

Deep learning elevates defect prediction, achieving precision far beyond standard machine learning [3]. Techniques like LASSO-SVM and under-sampling address data imbalances effectively, crucial for safety-critical software [19], [22]. The drawback lies in securing sufficient high-quality data, especially for new or proprietary projects [15]. Generating synthetic data or adapting models from existing datasets could bridge this gap [23]. Adding domain-specific knowledge about common defects might further refine accuracy with minimal extra effort [26].

Resource allocation thrives with genetic algorithms, optimizing testing across diverse systems [2]. They perform exceptionally well in distributed and modular setups, and dynamic adjustments enhance their flexibility as needs shift [7], [16]. However, their complexity can deter smaller teams lacking expertise [20]. Streamlined tools or guides could make them more accessible [24]. Integrating allocation with prioritization and defect prediction could sharpen focus on critical areas, maximizing testing efficiency [17].

The toughest challenge is combining these approaches into a cohesive system [31]. Imagine a framework where prioritization selects key tests, deep learning identifies flaws, genetic algorithms distribute effort, and reliability checks ensure stability [32]. This requires interoperable tools to replace today's fragmented efforts [33]. Real-world trials in industry settings—not just controlled experiments—will test its viability and highlight needed adjustments [29]. Successfully addressing these issues could elevate software quality assurance, preparing it for the complexities of modern systems [28]

6. Conclusion

This review of 35 studies from 1992 to 2021 demonstrates how innovative methods are reshaping software quality. Risk-based prioritization boosts fault detection by 30%, a vital edge for tight schedules where delays aren't an option. Deep learning drives defect prediction to precision levels of 0.88–0.92, empowering teams to address issues before they escalate. Reliability assessments for component-based systems achieve 85–90% accuracy, reinforcing confidence in today's modular designs. Genetic algorithms cut testing effort by 25%, delivering efficiency without compromising thoroughness—a lifeline for projects stretched thin on time or budget.

Progress aside, obstacles persist. Scaling prioritization and resource allocation for massive systems taxes available resources, often beyond what smaller teams can manage. Reliability and defect prediction falter without robust datasets, a hurdle for unique or early-stage projects. Most critically, these methods operate in isolation rather than as a unified front. A proposed solution integrates them: prioritization to target tests, prediction to pinpoint defects, allocation to optimize resources, and reliability to verify performance. Achieving this demands seamless tools, rigorous testing in actual industry conditions—not just labs—and simplified approaches to suit diverse teams. Overcoming these barriers could align software quality with the intense demands of modern technology, ensuring dependable systems in an ever-connected world.

References

[1] O. Dahiya, K. Solanki, and A. Dhankhar, "Risk-based testing: Identifying, assessing, mitigating & managing risks efficiently in software testing," Int. J. Adv. Res. Eng. Technol., vol. 11, pp. 192–203, 2020.

[2] P. Johri, Md. Nasar, and U. Chanda, "A genetic algorithm approach for optimal allocation of software testing effort," Int. J. Comput. Appl., vol. 68, no. 5, pp. 21–25, Apr. 2013.

[3] L. Qiao, X. Li, Q. Umer, and P. Guo, "Deep learning based software defect prediction," Neurocomputing, vol. 385, pp. 100–110, 2020.

[4] O. Dahiya and K. Solanki, "Prevailing standards in requirement-based test case prioritization: An overview," in ICT Analysis and Applications, 2021, pp. 467–474.

[5] O. Al Qasem and M. Akour, "Software fault prediction using deep learning algorithms," Int. J. Open Source Softw. Process., vol. 10, pp. 1–19, 2019.

[6] S. Yadav and B. Kishan, "Reliability of component-based systems—A review," Int. J. Adv. Trends Comput. Sci. Eng., vol. 8, pp. 293–299, 2019.

[7] Md. Nasar and P. Johri, "Testing resource allocation for modular software using genetic algorithm," Int. J. New Comput. Archit. Appl., vol. 5, no. 1, pp. 29–38, 2015.

[8] O. Dahiya and K. Solanki, "A systematic literature study of regression test case prioritization approaches," Int. J. Eng. Technol., vol. 7, pp. 2184–2191, 2018.

[9] S. Yadav and B. Kishan, "Component-based software system using computational intelligence technique for reliability prediction," Int. J. Adv. Trends Comput. Sci. Eng., vol. 9, pp. 3708–3721, 2020.

[10] S. Yadav and B. Kishan, "Analysis and assessment of existing software quality models to predict the reliability of component-based software," Int. J. Emerging Trends Eng. Res., vol. 8, pp. 2824–2840, 2020.

[11] S. Yadav and B. Kishan, "Assessments of computational intelligence techniques for predicting reliability of component based software parameter and design issues," Int. J. Adv. Res. Eng. Technol., vol. 11, pp. 565–584, 2020.

[12] Md. Nasar and P. Johri, "Testing resource allocation for fault detection process," in Proc. First Int. Conf. Smart Trends Inf. Technol. Comput. Commun. (SmartCom-2016), Springer CCIS, Aug. 2016.

[13] T. Wang, W. Li, H. Shi, and Z. Liu, "Software defect prediction based on classifiers ensemble," J. Inf. Comput. Sci., vol. 8, pp. 4241–4254, 2011.

[14] N. Karunanithi, D. Whitley, and Y. K. Malaiya, "Prediction of software reliability using connectionist models," IEEE Trans. Softw. Eng., vol. 18, p. 563, 1992.

[15] B. S. Deshpande, B. Kumar, and A. Kumar, "Assessment of software reliability by object oriented metrics using machine learning techniques," Int. J. Grid Distrib. Comput., vol. 14, pp. 01–10, 2021.

[16] Md. Nasar, P. Johri, and U. Chanda, "Dynamic effort allocation problem using genetic algorithm approach," Int. J. Mod. Educ. Comput. Sci., vol. 6, no. 6, pp. 46–52, 2014.

[17] Md. Nasar, P. Johri, and U. Chanda, "A differential evolution approach for software testing effort allocation," J. Ind. Intell. Inf., vol. 1, no. 2, pp. 111–115, 2013.

[18] P. Johri, Md. Nasar, and S. Das, "Open source software reliability growth models for distributed environment based on component-specific testing-efforts," in Proc. Second Int. Conf. Inf. Commun. Technol. Competitive Strategies (ICTCS-2016), ACM ICPS, Mar. 2016.

[19] K. Wang, L. Liu, C. Yuan, and Z. Wang, "Software defect prediction model based on LASSO–SVM," Neural Comput. Appl., pp. 8249–8259, 2021.

[20] S. Yadav and B. Kishan, "Assessment of software quality models to measure the effectiveness of software quality parameters for component based software (CBS)," J. Appl. Sci. Comput., vol. 6, pp. 2751–2756, 2019.

[21] Md. Nasar and P. Johri, "Testing and debugging resource allocation for fault detection and removal process," Int. J. New Comput. Archit. Appl., vol. 4, no. 4, pp. 193–200, 2014.

[22] K. N. Rao and C. S. Reddy, "A novel under sampling strategy for efficient software defect analysis of skewed distributed data," Evolving Syst., vol. 11, pp. 119–131, 2020.

[23] C. Shan, H. Zhu, C. Hu, J. Cui, and J. Xue, "Software defect prediction model based on improved LLE-SVM," in Proc. 4th Int. Conf. Comput. Sci. Netw. Technol. (ICCSNT), vol. 1, 2015, pp. 530–535.

[24] Md. Nasar, P. Johri, and U. Chanda, "Resource allocation policies for fault detection and removal process," Int. J. Mod. Educ. Comput. Sci., vol. 6, no. 11, pp. 52–57, 2014.

[25] S. L. Ho, M. Xie, and T. N. Goh, "A study of the connectionist models for software reliability prediction," Comput. Math. Appl., vol. 46, pp. 1037–1045, 2003.

[26] C. Jin and S. W. Jin, "Prediction approach of software fault-proneness based on hybrid artificial neural network and quantum particle swarm optimization," Appl. Soft Comput., vol. 35, pp. 717–725, 2015.

[27] P. F. Pai and W. C. Hong, "Software reliability forecasting by support vector machines with simulated annealing algorithms," J. Syst. Softw., vol. 79, pp. 747–755, 2006.

[28] J. H. Lo, "A study of applying ARIMA and SVM model to software reliability prediction," in Proc. Int. Conf. Uncertainty Reasoning Knowl. Eng., vol. 1, 2011, pp. 141–144.

[29] H. Li, M. Zeng, M. Lu, X. Hu, and Z. Li, "Adaboosting-based dynamic weighted combination of software reliability growth models," Qual. Rel. Eng. Int., vol. 28, pp. 67–84, 2012.

[30] L. Tian and A. Noore, "Evolutionary neural network modeling for software cumulative failure time prediction," Rel. Eng. Syst. Safety, vol. 87, pp. 45–51, 2005.

[31] Q. P. Hu, M. Xie, S. H. Ng, and G. Levitin, "Robust recurrent neural network modeling for software fault detection and correction prediction," Rel. Eng. Syst. Safety, vol. 92, pp. 332–340, 2007.

[32] P. Roy, G. S. Mahapatra, and K. N. Dey, "Neuro-genetic approach on logistic model based software reliability prediction," Expert Syst. Appl., vol. 42, pp. 4709–4718, 2015.

[33] E. O. Costa, G. A. de Souza, A. T. R. Pozo, and S. R. Vergilio, "Exploring genetic programming and boosting techniques to model software reliability," IEEE Trans. Rel., vol. 56, pp. 422–434, 2007.