International Journal of Engineering & Extended Technologies Research (IJEETR)

|ISSN: 2322-0163| www.ijeetr.com | A Bimonthly, Peer Reviewed, Scholarly Indexed Journal |

| Volume 7, Issue 6, November – December 2025 |

DOI:10.15662/IJEETR.2025.0706038

Improving Software Reliability Through Automated Testing Frameworks in Enterprise Systems

Mahesh Kumar Damarched[1], Suseela Pandity[2]

Enterprise Programmer Analyst, Louisville, Kentucky, USA[1]

PhD in IT with AI focus, University of Cumberlands, Louisville, Kentucky, USA[2]

Corresponding Author: Mahesh Kumar Damarched[1]*

ABSTRACT: Enterprise systems are at the heart of daily operations, and this active presence makes reliability a necessity for optimal efficiency. The paper examines the use of an automated testing structure to increase reliability by transforming checks into routines that can be repeated with the same rules and results that can be reviewed and acted upon in a timely manner. It describes how automation can be integrated with the current delivery, starting with a commit by the developer and continuing to build and test cycles and release readiness, with the feedback loops being very short to identify a mistake before a developer forgets about it. The discussion describes unit, functional, and regression testing in simple terms before demonstrating how the layers can be put together to secure logic, workflow, and long-term support across updates. It also describes the way tool decisions are determined, using the traditional combination of a test-writing framework and a browser-based runner as a real-world example of how the front-end automation is implemented. The case scenario presented in the paper explains how things are going to be different when automation becomes the norm. The concluding sections summarize the important quality steps based on which teams check the progress and the test suite health when systems increase in size.

KEYWORDS: Software reliability, automated testing, unit testing, functional testing, test automation frameworks, continuous integration, enterprise systems, test runners, test design, cross-browser testing, Jasmine, Karma, regression testing, quality metric.

I. INTRODUCTION

Software reliability refers to the ability of a system to continue to give the correct output and maintain the same stable behavior as real people continue to use it under different conditions, which Afrihyia et al. (2022) define as the core of user confidence in online platforms[2]. This concept is extremely serious in enterprise systems, as those systems interconnect payroll, supply chains, customer records, analytics, and external partners into one stream of data. In the event of failure of such systems, the damage spreads fast. The error in the payroll can freeze payment, a customer relationship management (CRM) malfunction can disrupt sales, and a data breach can expose confidential data. Blair (2021) gives examples of the weak points of automatic data processing (ADP), Salesforce, and MongoDB[7]. Cybersecurity researchers also demonstrate that these failures can be used by attackers to access sensitive assets, as enterprise systems expose high attack surfaces via application programming interfaces (APIs) and cloud connections, as Xiong et al. (2022) state when they talk about enterprise threat models[48]. Despite the risks, automated test generation for cloud applications continue to improve through diverse AI techniques application [31]. Uptime is not the only matter of reliability. It also ensures data integrity, regulatory compliance, and the trust of the people, which Judijanto et al. (2023) relate directly to the business survival in the digital economy[26].

Conventional testing practices are inflexible to achieve such a high degree of reliability since such practices are based on human input that is implemented towards the end of the delivery cycle, which Gurcan et al. (2022) and Graham and Paulson (2025) demonstrate has prevailed in testing over the decades[19,21]. It is also essential to acknowledge that efficient implementation helps attain optimal results which requires incorporating policy compliance, technological advancements, and sustainability practices[36]. Implementation should also constitute robust cybersecurity strategies[11]. In the current business, code is frequently published in distributed development organizations in microservice-based systems, which Cerny et al. (2020) deemed as the source of increasing complexity[9]. This speed and scale cannot be maintained with the help of manual testing. By then, a human tester has already cleared one workflow, as it has already been changed by a new code, which is discussed by Talakola (2022) when he talks about how late-stage testing overlooks integration failures[45]. You can have faster releases and updates, so errors can reach production before one realizes what happened, as demonstrated by Neelapu (2023) and Chen et al. (2020)[10,37]. The author shows this when

|ISSN: 2322-0163| www.ijeetr.com | A Bimonthly, Peer Reviewed, Scholarly Indexed Journal |

| Volume 7, Issue 6, November – December 2025 |

DOI:10.15662/IJEETR.2025.0706038

they explain API-driven systems, where minor modifications cause services reliant on other ones to fail[37]. This fact leads to the necessity of automated verification that is provided with each change of code that is being introduced. It acts as the sole feasible means of ensuring consistency in contemporary pipelines.

The report is concerned with the topic of the enhancement of software reliability in enterprise systems through automated testing frameworks by making it a central part of delivery. Testing helps create integrity and immutability of data [32]. Reliability ensure optimal performance of the system, as described by Nittala (2025)[38]. It provides the explanation of why automation is needed in the development pipeline and why it minimizes the risk of failure, relying on the works of Jyoti et al. (2024) and Afrihyia et al. (2022)[2,27]. It then contrasts unit, functional, and regression testing as defect defense levels, based on the evidence of Gudi (2023) and Umar (2023)[20,46]. The discussion shifts to Jasmine and Karma tools used in web applications of enterprises associated with cross-browser reliability. The literature provides a real enterprise case that demonstrates how these tools are applied to practice. The paper then compiles quality metrics which demonstrate the presence or absence of improvement of reliability through automation. It is essential to always have safe systems as projected by Marijan and Gotlieb (2020)[34]. The focus remains on the automated testing system and its reliability effect, and not on more general DevOps or security management issues, which enables the analysis to remain within the ground of quantifiable software behavior.

II. ROLE OF AUTOMATED TESTING

Automated testing is the act of running software scripts and tools to check an application and compare the real results with the behavior of the code. It is an establishment that Umar (2023) and Izzat and Saleem (2023) describes as the fundamental process through which a test framework confirms that a program makes sense[24,46]. Its importance is regarded as critical in industrial software development. Evidence from Fulcini et al. (2024) indicates that software testing activities can account for as much as 50% of total software development costs in many projects[17]. It is an element that constitute the substantial resources required to ensure software quality and reliability. According to Drofa (2025), high quality software development requires deploying agile development[15]. This process contributes to the reliability of enterprise systems since the identical tests are run the same way as always, eliminating the variability that human testing brings. To achieve this, it is integral to create a composable enterprise architecture (CEA), as garnered from Rusum and Anasuri (2023)[41]. Afrihyia et al. (2022) note this progression in their cross-platform study of automated test execution[2].

With automated tests, a team can run them overnight, after each code change, or before every release without having to wait to be taken through screens, which Srinivas et al. (2024) say is one method of ensuring feedback does not stagnate even when no developers are working[43]. Automation should be run with Dash's (2025) projection of them being consuming high amounts of energy[12]. Running them require consistency. Such consistency is significant as many enterprise applications are constantly modified with new features, security patches, and regulatory updates, and it has been demonstrated by Neelapu (2023) that APIs and services will not work when they are not constantly checked[37]. Automated tests serve as a safety mechanism as they verify business rules, user flows, and data exchanges every time the code is touched to ensure small mistakes do not lead to system-wide failures. System failures can be stubborn as Gudi (2023) reports in Java enterprise projects[20]. That is why it is necessary to efficiently plan as argued by Alaskari et al. (2021)[3], and explained by Bouzenia & Pradel (2025)[8] in their article addressing Execution Agent.

Automated testing in a modern delivery pipeline is an intermediate between code creation and deployment and serves as a filter that removes defects before users even see them, as described in the message-based framework of regression[14]. According to Afrihyia et al. (2022), when a developer writes code, it is compiled by the build system, which is then followed by fast unit tests that ensure logic, followed by more thorough functional and regression tests that ensure workflows and integrations, referred to as a layered testing flow[2]. This pattern enables the errors to be detected early, and the developer recalls what has changed. Srinivas and Goel (2025) list this incorporation as one of the significant factors contributing to the reduction of the time spent on repairing the automated testing[44]. This process results in the formation of feedback loops. The unsuccessful test will send the message back to the developer, who corrects the problem and returns the code, which, in such a solution, is associated with an easier release of web applications[45]. In the long run, the loops can inform teams about what aspects of the system are vulnerable and, thus, make design choices and reduce repetition errors, as Cerny et al. (2020) report in their article on code structures at enterprise level[9].

Risks are also involved with automation, as research makes apparent. Flakiness of tests can happen when the test is based on volatile data or resources or time constraints. It is a development that Jonson and Tornqvist (2025) show

|ISSN: 2322-0163| www.ijeetr.com | A Bimonthly, Peer Reviewed, Scholarly Indexed Journal |

| Volume 7, Issue 6, November – December 2025 |

DOI:10.15662/IJEETR.2025.0706038

through their experiments on how resource limits lead to random failures[25]. Marabesi et al. (2024) found that poorly written tests may pass even under the circumstances where the software is faulty and falsely give a feeling of safety, which relate to poor test design and ambiguous claims in test-driven development (TDD) settings[33]. Another issue is maintenance as systems keep changing, their tests should be updated, which is one of the expenses teams have to consider[16]. Teams alleviate these issues by maintaining a stable test code, writing specific checks that test a single idea at a time, and reviewing test code with the same diligence as production code. It is one of the fundamental principles of sound automation design. To the extent that these practices are undertaken, automated testing which can contribute to self-healing automation is a factor that propagates reliability as it offers quick, credible indications that drive each change that is made on an enterprise system[5].

III. UNIT VS. FUNCTIONAL VS. REGRESSION TESTING

Unit

Unit testing is concerned with the smallest components of a software system, namely, a single function, a single class, or a service method that depicts as the primary defense against logic errors [1,35,42,46]. Gudi (2023) posit that they are used to test the output of a given piece of code when provided with controlled input to determine that it produces the correct output, which helps a developer identify errors in their code before they propagate to the other system components in the context of Java enterprise applications[20]. Due to the speed with which they are executed and the lack of necessitating complete system configurations, unit tests can be run in large quantities (thousands) in minutes, as explained by Srinivas and Goel (2025)[44]. This critical component makes it well-suited to continuous integration pipelines. Unit tests can be used to show that a developer has altered the expected behavior when changing a pricing rule, a tax calculation, or a validation function. According to Neelapu (2023) this trait attributes to early defect discovery in service-based systems[37]. Mocking and stubbing can make the unit under test isolated by replacing external services or databases with simple stubs, which makes the test concentrate on the logic under test only, as elaborated by as Datla and Thodupunuri (2021) when addressing formal approaches toward Java web applications[13].

A unit in enterprise systems can tend to have more than one line of code since business rules, validation logic, and service methods can contain much of the application[49]. A unit test may ensure that a loan approval rule does not violate invalid data or that a claims processor uses the appropriate policy, which Afrihyia et al. (2022) explain is, according to them, the testing of the core business logic of cross-platform applications[2]. The test data in such situations should be small, predictable, and simple to reset in such a manner that test results do not change between executions, thereby minimizing the number of random failures as demonstrated by Jonson and Tornqvist (2025)[25]. Khankhoje (2023) postulated that good unit tests are readable, fast, and independent[28]. These are characteristics that assist teams in having confidence in their automation. With one rule or calculation per test, the developers can immediately know what failed and why, which helps to make quick and safe changes.

Functional

Functional testing changes the agenda to external system behavior. This behavior can be viewed as an assessment of whether features function from the system or user perspective. These tests are a replica of a real process, like logging into a system, filling out a form, or making an order. It is a propagation that Talakola (2022) demonstrates through the address of how web applications demonstrate that processes have not been altered[45]. In enterprise systems, end-to-end business processes are verified by functional tests such as the interactions between front-end interfaces, APIs, and databases, which Afrihyia et al. (2022) attribute to stable cross-platform performance[2]. Teamwork is able to perform functional tests via user interfaces or via APIs, which offer two avenues of checking on the same line of business[14]. User interface (UI) tests demonstrate the experience of end users, whereas API tests ensure that services are acting as they need to, even without the browser, which is helpful in isolating faults in case of failures.

Regression

Regression testing secures an old behavior when the code is modified, but it is defined as a way to defend against the reappearance of the old bugs [42,46]. Enterprise systems require robust regression suites since they can be modified with regular patches, configuration, and third-party updates, which, according to Cerny et al. (2020) expose them to the threat of unintended side effects[9]. It could be a regression test to ensure that a calculation in a payroll remains possible following a tax update or a reporting feature remains able to generate the identical totals following a database upgrade, a factor linked by Blair (2021) to business continuity[7]. Regression tests are chosen depending on risk, frequency of use, and history of past failures, which Afrihyia et al. (2022) state is one method of concentrating efforts on where failures would be most detrimental to the organization[2]. Neelapu (2023) explains that this has the benefit of ensuring that

|ISSN: 2322-0163| www.ijeetr.com | A Bimonthly, Peer Reviewed, Scholarly Indexed Journal |

| Volume 7, Issue 6, November – December 2025 |

DOI:10.15662/IJEETR.2025.0706038

breakage is identified by the teams before getting to the customers[37]. It is an establishment that associates the advantages with higher reliability of the services [6].

Comparing and Connecting

The functions of each type of test are different. Unit tests are fast and directly indicate faulty logic. Gudi (2023) show that it is an attribute that is demonstrated to be the best in daily development[20]. Full workflows and slow functional tests are run to provide an assurance that actual users can accomplish tasks, as depicted by Talakola (2022)[45]. Regression tests encompass a wide safety net that prevents failure to repeat in the past, which Gurcan et al. (2022) pinpointed as a crucial aspect of long-lived systems[21]. Collectively, these layers create a balanced security in which unit tests secure code specifics, functional tests secure business processes, and regression tests secure system stability during updates. A pragmatic combination consists of numerous units check speedy tests, fewer functional tests based on workflows, and a precise regression set assembled on high-risk regions. According to Afrihyia et al. (2022) such incorporations offer an efficient testing design in enterprise settings[2]. It is important to incorporate human–AI collaboration for optimal testing results[22].

IV. TOOLS

The testing tools influence the level of automation-assisted reliability as they regulate the manner in which tests are authored, performed, and documented. This observation is defined by Antonenko and Vostrikov (2024) as the working backbone of automated quality assurance[4]. Test code is organized into a testing framework that specifies the manner in which assertions are formatted and the manner in which results are assessed, and a test runner determines when and where such tests execute (Garcia et al., 2020, p. 2). This is a depiction that is observed in the modern JavaScript testing ecosystems, such as the Kieker development that has taken place since 2006[23]. The selection of tools is essential in enterprise settings as a system will need to test numerous browsers, devices, and build servers, which are associated with the necessity of stable cross-platform execution. Once they are incorporated into continuous integration pipelines in a clean fashion, tools enable every single code change to be inspected without human latency. Inspection is an integral component as it results in a reduced number of defects being missed. The tools that are poorly integrated, in turn, leave blind spots, bugs can go through without being noticed, creating a potential vulnerability in automation assertions[16].

Jasmine

Jasmine is a structured, clear test case in JavaScript applications. Garcia et al. (2020) and Nawagamuwa (2023) list Jasmine's structure as one of the reasons why it is pervasive in enterprise web systems[35]. It has readable syntax to define test suites, expectations, and spies, enabling teams to express how a component or service is supposed to behave in plain code, as William and Virginia (2022) do in Angular-based enterprise platforms[47]. Jasmine is a good fit with business logic testing since it enables developers to test the performance of functions in various scenarios. It is a characteristic that Khankhoje (2022) associates with a more robust test design[28]. Jasmine provides teams with easy maintenance and a reliable structure, which helps to save time on the interpretation of the result, as seen in automation frameworks.

Karma

Karma is the implementation layer that executes Jasmine tests through real browsers and reports the results. In their explanation, Garcia et al. (2020, p. 4) outline Karma as the key to the reliability of the front-end. It opens Chrome or Firefox and other browsers, loads the application code, executes the test suites, and sends pass or fail signals back to the build server. All this process links to cross-browser confidence. Karma can be used in enterprising pipelines with automatic execution of tests, which Srinivas et al. (2024) associate with quick feedback[43]. Karma, when combined with Jasmine, is a typical configuration that allows tests to be written in a friendly fashion and run in the real world, which enables stable release to many user devices, as is the case with large Angular applications [29,47].

Case Example

A practical example of the ideas explained in the above-mentioned section appears in Afrihyia et al. (2022), where a cross-platform enterprise application was tested across web and mobile environments[2]. The tests were conducted using automated frameworks such as Selenium and browser-based runners. The system had several modules, which processed user accounts, data, and reporting, which produced a lot of defects when updates were done[2]. The manual testing failed to keep pace with the pace of change and hence failed to be detected early enough in the release cycle, which delayed the deployments and increased the risk of production problems.

|ISSN: 2322-0163| www.ijeetr.com | A Bimonthly, Peer Reviewed, Scholarly Indexed Journal |

| Volume 7, Issue 6, November – December 2025 |

DOI:10.15662/IJEETR.2025.0706038

To solve this, the group put forward a layered testing approach comprising unit assessments of fundamental business logic, functional assessments of key operations, and a regression package that was executed on every development. The team describes all this in their methodology[2]. Automation tools based on browsers were deployed to confirm the behavior of the application in different platforms, whereas the test scripts confirmed that the user functions, e.g., log-in, data entry, and reports generation, worked as expected. In web systems of the enterprise, the structure is similar, with Jasmine and Karma being the expected behavior and the part of execution by a multiplicity of browsers, respectively though the paper deals with such tools as Selenium and Appium (Garcia et al., 2020, p. 3).

The outcomes of this strategy revealed that flaws were identified earlier, and the workflows remained steady throughout the updates, and the releases were made easier. This is a development that Afrihyia et al. (2022) relate to enhanced software reliability in their research findings[2]. The number of repeat bugs that occurred after deployment also reduced, and teams increased their confidence that the existing features would not be destroyed because of changes. This finding reinforces the hypothesis that front-end instruments such as Jasmine and Karma are deployed in a rigorous testing plan; these are part of the enterprise safety network that ensures that the behavior of software can be predictable over time as systems evolve. Kohvakka (2020) reiterate the need for programmers in organizations not skipping any test-driven development process, as this may adversely affect performance in the long run [30].

Quality Metrics

Evidence that automated testing enhances reliability is required by teams, not the belief in the tools. It is an incorporation that Srinivas et al. (2024) refer to as the motivation for including quality measurements in the current enterprise testing practice[43]. Defect escape rate is one of such measures, but it includes the number of bugs that pass-through testing and move to production, with Afrihyia et al. (2022) correlating a smaller escape rate with a more efficient automation[2]. Test pass rate indicates the percentage of tests that are successful in a particular run, which is associated with the immediate health of a build. Mean time to detect refers to how quickly a failure is found after it is introduced, while mean time to fix reflects how long it takes to correct it. These are indicators of how well feedback loops work. Change failure rate quantifies the prevalence of a deployment leading to a problem, which directly relates to the service reliability in API-based systems. Both of these metrics represent the other side of reliability, between the number of mistakes that sail through and the speed with which the teams recover.

In most organizations, code coverage is a topic of attention, but it has to be approached carefully. Marabesi et al. (2024) mention this when talking about how the tests can reach the code but fail to assess its behavior[33]. Instead of determining whether the correct conditions are being tested, coverage gauges the proportion of the code base that is executed during tests. It is a concept that Afrihyia et al. (2022) report in their research study as often mistakenly understood[2]. Defects may be concealed by high coverage with weak assertions, and critical paths may be safeguarded by low coverage with strong checks, associating with test design quality. Branch or path coverage is also one of the studies that are considered when studying complex[13]. The thing about reliability is that the tests should cover the cases of significant decisions and errors, rather than run through all the lines. It is a propagation that can be achieved with the increased development of machine learning (ML) and artificial intelligence that is in place, as garnered from Riccio et al. (2020)[40].

Jonson and Tornqvist (2025) demonstrate that measures prove to be beneficial when the teams use them in their study of flaky tests[25]. An increase in the number of failures may be indicative of unstable data or overprovisioned environments, which groups correct by stabilizing test environments or isolating resources. The mishaps in a single field are indicative of incomplete coverage, and thus the teams add or revise tests as described in the improvement cycle by Afrihyia et al. (2022)[2]. Trend reviews and dashboards enable teams to observe trends across time. Srinivas et al. (2024) associates this review propagation with the gradual increase in test reliability[43]. Teams can use metrics as feedback instead of punishments to create test suites that become more robust with each release, which helps to predict the behavior of enterprise software. Generally, Rahman and Jyoti (2022) depicts that with the introduction of software, their reliability is necessary as this will benefit human-computer interaction (HCI) which continue to enhance enterprise systems[39]. As AI advance, it is essential for enterprises to also adopt AI-powered testing tools to promote their testing mechanisms, as projected by Garousi et al. (2024)[18].

V. CONCLUSION

With enterprise software, reliability can be enhanced by considering testing as a routine work that occurs with every change rather than occurring at the end. That routine is made feasible through a stratified approach. Unit tests safeguard the tiny regulations that motivate the business conduct, functional tests safeguard those streams that individuals rely on,

|ISSN: 2322-0163| www.ijeetr.com | A Bimonthly, Peer Reviewed, Scholarly Indexed Journal |

| Volume 7, Issue 6, November – December 2025 |

DOI:10.15662/IJEETR.2025.0706038

and regression protects yesterday's features that were functioning to be ruined by the updates of today. A structured test framework with a runner to run tests in real browsers provides the team with a predictable method to write checks, run repeatedly, and read the output without making guesses. In cases where automation is incorporated in the delivery process, the distance between an error and its detection is reduced. That change is crucial as it minimizes repetitive defects, restricts release, and facilitates predictability of system behavior in the long term. The report also demonstrates the importance of metrics: they transform the question into, Are we getting more reliable? to a form that a team can respond to through observing trends, addressing the areas in which it is falling behind, and establishing the test suite as an equally high-quality standard as the product.

The way this approach will stand will be determined by future concerns. With the evolution of enterprise platforms to place more emphasis on microservices, third-party APIs, and fast front-end response, test suites may become heavy systems themselves and may slow down delivery unless handled with discipline. Flaky tests are a long-term menace since they are training teams to disregard flakes, which is a silent form of reliability loss. Also, the issue of test data and environments is that modern systems frequently rely on a shared service, real-time events, and large datasets, which are difficult to simulate in a testing environment safely. The trend of automation will continue as AI-aided tools will produce tests and even conduct them, which will accelerate coverage, but create new risks in the case of teams accepting outcomes without analysis. The only way out is gradualism: make tests small and transparent, test them in lieu of production code, follow trends rather than individual outcomes, and invest in stable environments. When teams do so, automation remains a safety net, and not an additional uncertainty.

REFERENCES

1. Abdulwareth, A. J., & Al-Shargabi, A. A. (2021). Toward a multi-criteria framework for selecting software testing tools. *IEEE Access*, *9*, 158872-158891. doi: 10.1109/ACCESS.2021.3128071.
2. Afrihyia, E., Umana, A. U., Appoh, M., Frempong, D., Akinboboye, O., Okoli, I., ... & Omolayo, O. (2022). Enhancing software reliability through automated testing strategies and frameworks in cross-platform digital application environments. *Journal of Frontiers in Multidisciplinary Research*, *3*(2), 517-531. https://doi.org/10.54660/.JFMR.2022.3.1.517-531
3. Alaskari, O., Pinedo-Cuenca, R., & Ahmad, M. M. (2021). Framework for implementation of enterprise resource planning (ERP) systems in small and medium enterprises (SMEs): A case study. *Procedia Manufacturing*, *55*, 424-430. https://doi.org/10.1016/j.promfg.2021.10.058
4. Antonenko, A. V., Vostrikov, S. O., Burachynskyi, A. Y., Tverdokhlib, A. O., Balvak, A. A., & Slobodian, O. A. (2024). Features of automated testing using frameworks. *Таврійський науковий вісник. Серія: Технічні науки*, (4), 3-14. https://dspace.ksaeu.kherson.ua/bitstream/handle/123456789/10507/%D0%A2%D0%9D%D0%92_%D0%A2%D0%9D_4_2024.pdf?sequence=1&isAllowed=y#page=3
5. Bari, M. S., Sarkar, A., & Islam, S. M. (2024). AI-augmented self-healing automation frameworks: Revolutionizing QA testing with adaptive and resilient automation. *AIJMR-Advanced International Journal of Multidisciplinary Research*, *2*(6). https://doi.org/10.62127/aijmr.2024.v02i06.1118
6. Berihun, N. G., Dongmo, C., & Van der Poll, J. A. (2023). The applicability of automated testing frameworks for mobile application testing: A systematic literature review. *Computers*, *12*(5), 97. https://doi.org/10.3390/computers12050097
7. Blair, R. (2021). Enterprise systems and threats. https://www.iiisci.org/JOURNAL/PDV/sci/pdfs/ZA435QP21.pdf
8. Bouzenia, I., & Pradel, M. (2025). You name it, i run it: An llm agent to execute tests of arbitrary projects. *Proceedings of the ACM on Software Engineering*, *2*(ISSTA), 1054-1076. https://doi.org/10.5281/zenodo.15202434
9. Cerny, T., Svacina, J., Das, D., Bushong, V., Bures, M., Tisnovsky, P., ... & Huang, J. (2020). On code analysis opportunities and challenges for enterprise systems and microservices. *IEEE access*, *8*, 159449-159470. https://ieeexplore.ieee.org/abstract/document/9179733
10. Chen, T. Y., Cheung, S. C., & Yiu, S. M. (2020). Metamorphic testing: a new approach for generating next test cases. *arXiv preprint arXiv:2002.12543*.
11. Dalal, A. (2024). Implementing Robust Cybersecurity Strategies for Safeguarding Critical Infrastructure and Enterprise Networks. *International Journal of Management, Technology And Engineering*. https://www.academia.edu/download/124416737/PK_46_Implementing_Robust_Cybersecurity_Strategies_for_Safeguarding_Critical_Infrastructure_and_Enterprise_Networks.pdf
12. Dash, S. (2025). Green AI: Enhancing sustainability and energy efficiency in AI-integrated enterprise systems. *IEEE Access*, *13*, 21216-21228. https://ieeexplore.ieee.org/abstract/document/10849555

|ISSN: 2322-0163| www.ijeetr.com | A Bimonthly, Peer Reviewed, Scholarly Indexed Journal |

| Volume 7, Issue 6, November – December 2025 |

DOI:10.15662/IJEETR.2025.0706038

13. Datla, L. S., & Thodupunuri, R. K. (2021). Applying formal software engineering methods to improve java-based web application quality. *International Journal of Artificial Intelligence, Data Science, and Machine Learning*, *2*(4), 18-26. https://doi.org/10.63282/3050-9262.IJAIDSML-V2I4P103
14. Demircioğlu, E. D., & Kalipsiz, O. (2022). API message-driven regression testing framework. *Electronics*, *11*(17), 2671. https://doi.org/10.3390/electronics11172671
15. Drofa, D. (2025). Optimization of software development processes through the use of full-stack technologies and automation. *Contemporary Issues in Artificial Intelligence*, *1*. https://doi.org/10.69635/ciai.2025.12
16. Fatima, S., Mansoor, B., Ovais, L., Sadruddin, S. A., & Hashmi, S. A. (2022). Automated testing with machine learning frameworks: A critical analysis. *Engineering Proceedings*, *20*(1), 12. https://doi.org/10.3390/engproc2022020012
17. Fulcini, T., Coppola, R., Ardito, L., & Torchiano, M. (2023). A review on tools, mechanics, benefits, and challenges of gamified software testing. *ACM Computing Surveys*, *55*(14s), 1-37. https://dl.acm.org/doi/full/10.1145/3582273
18. Garousi, V., Joy, N., Jafarov, Z., Keleş, A. B., Değirmenci, S., Özdemir, E., & Zarringhalami, R. (2024). AI-powered software testing tools: A systematic review and empirical assessment of their features and limitations. *arXiv preprint arXiv:2409.00411*. https://doi.org/10.48550/arXiv.2409.00411
19. Graham, O., & Paulson, M. (2025). How Artificial Intelligence Is Transforming Test Case Design and Test Data Generation in Software Testing. https://www.preprints.org/frontend/manuscript/4a901a1461108c964c94ed4e82c008cb/download_pub
20. Gudi, S. R. (2023). Enhancing reliability in java enterprise systems through comparative analysis of automated testing frameworks. *International Journal of Emerging Trends in Computer Science and Information Technology*, *4*(2), 151-160. https://doi.org/10.63282/3050-9246.IJETCSIT-V4I2P115
21. Gurcan, F., Dalveren, G. G. M., Cagiltay, N. E., Roman, D., & Soylu, A. (2022). Evolution of software testing strategies and trends: Semantic content analysis of software research corpus of the last 40 years. *IEEE Access*, *10*, 106093-106109. https://ieeexplore.ieee.org/abstract/document/9910177
22. Hasan, R. (2025). A Systematic Review Of Human-AI Collaboration In It Support Services: Enhancing User Experience And Workflow Automation. *American Journal of Interdisciplinary Studies*, *6*(3), 01-37. https://doi.org/10.63125/0fd1yb74
23. Hasselbring, W., & Van Hoorn, A. (2020). Kieker: A monitoring framework for software engineering research. *Software Impacts*, *5*, 100019. https://doi.org/10.1016/j.simpa.2020.100019
24. Izzat, S., & Saleem, N. N. (2023). Software testing techniques and tools: A review. *Journal of Education and Science*, *32*(2), 31-40. 10.33899/edusj.2023.137480.1305
25. Jonson, M., & Törnqvist, S. (2025). Analyzing Root Causes and Smells of Test Flakiness by Simulating Resource Usage: A study about how system resource limitations can induce flaky behavior. https://www.diva-portal.org/smash/record.jsf?pid=diva2%3A1955392&dswid=-9573
26. Judijanto, L., Hindarto, D., Wahjono, S. I., & Djunarto, A. (2023). Edge of enterprise architecture in addressing cyber security threats and business risks. *International Journal Software Engineering and Computer Science (IJSECS)*, *3*(3), 386-396. https://pdfs.semanticscholar.org/0f2c/c27de0917d4159bbf6ceaa1681309bb79b13.pdf
27. Jyoti, S. N., Islam, M. R., & Kudapa, S. P. (2024). The Role of Test Automation Frameworks In Enhancing Software Reliability: A Review Of Selenium, Python, And API Testing Tools. *International Journal of Business and Economics Insights*, *4*(4), 01-34. https://doi.org/10.63125/bvv8r252
28. Khankhoje, R. (2023). Revealing the foundations: The strategic influence of test design in automation. *International Journal of Computer Science & Information Technology (IJCSIT) Vol*, *15*. https://ssrn.com/abstract=4687814
29. Klementowski, C., Reid, T., & Arnold, R. (2020). Tactical applications JavaScript development tools recommendations. https://apps.dtic.mil/sti/html/trecms/AD1090464/
30. Kohvakka, S. (2020). Automation tools in software development and production. https://lutpub.lut.fi/bitstream/handle/10024/161628/Bachelors_thesis_Sami_Kohvakka.pdf?sequence=1
31. Kothamali, P. R. (2025). Ai-powered quality assurance: Revolutionizing automation frameworks for cloud applications. *Journal of Advanced Computing Systems*, *5*(3), 1-25. https://doi.org/10.69987/JACS.2025.50301
32. Lal, C., & Marijan, D. (2021). Blockchain testing: Challenges, techniques, and research directions. *arXiv preprint arXiv:2103.10074*. https://doi.org/10.48550/arXiv.2103.10074
33. Marabesi, M., García-Holgado, A., & García-Peñalvo, F. J. (2024). Exploring the connection between the TDD practice and test smells—A systematic literature review. *Computers*, *13*(3), 79. https://doi.org/10.3390/computers13030079
34. Marijan, D., & Gotlieb, A. (2020, April). Software testing for machine learning. In *Proceedings of the AAAI Conference on Artificial Intelligence* (Vol. 34, No. 09, pp. 13576-13582). https://doi.org/10.1609/aaai.v34i09.7084

|ISSN: 2322-0163| www.ijeetr.com | A Bimonthly, Peer Reviewed, Scholarly Indexed Journal |

| Volume 7, Issue 6, November – December 2025 |

DOI:10.15662/IJEETR.2025.0706038

35. Nawagamuwa, J. (2023). Infrastructure as code frameworks evaluation for serverless applications testing in AWS. *Tampere University*. https://trepo.tuni.fi/bitstream/handle/10024/149140/NawagamuwaJanaka.pdf?sequence=2
36. Ndaba, Z., Pogiso, K., Thango, B., & Mankge, F. (2024). A systematic review of success factors and failure reasons in enterprise systems for executive, managerial, and operational support. *Managerial, and Operational Support (October 19, 2024)*. https://dx.doi.org/10.2139/ssrn.4996122
37. Neelapu, M. (2023). Enhancement of software reliability using automatic API testing model. https://www.allmultidisciplinaryjournal.com/uploads/archives/20250408184534_MGE-2025-2-236.1.pdf
38. Nittala, E. P. (2025). AI-based autonomous code generation and optimization for enhancing software reliability in computer systems. *International Journal of AI, BigData, Computational and Management Studies*, *6*(3), 55-64. https://doi.org/10.63282/3050-9416.IJAIBDCMS-V6I3P107
39. Rahman, M. A., & Jyoti, S. N. (2022). A systematic literature review of user-centric design in digital business systems: Enhancing accessibility, adoption, and organizational impact. *Review of Applied Science and Technology*, *1*(04), 01-25. https://doi.org/10.63125/ndjkpm77
40. Riccio, V., Jahangirova, G., Stocco, A., Humbatova, N., Weiss, M., & Tonella, P. (2020). Testing machine learning based systems: a systematic mapping. *Empirical Software Engineering*, *25*(6), 5193-5254. https://doi.org/10.1007/s10664-020-09881-0
41. Rusum, G. P., & Anasuri, S. (2023). Composable enterprise architecture: A new paradigm for modular software design. *International Journal of Emerging Research in Engineering and Technology*, *4*(1), 99-111. https://doi.org/10.63282/3050-922X.IJERET-V4I1P111
42. Savolainen, T. (2024). Improving and Automating Design System Testing. https://aaltodoc.aalto.fi/items/6d33fdfd-c523-4d6e-b429-c9fe880dd9a4
43. Srinivas, N., Mandaloju, N., & Nadimpalli, S. V. (2024). Leveraging Automation in Software Quality Assurance: Enhancing Efficiency and Reducing Defects. *The Metascience*, *2*(4), 84-95. https://yuktabpublisher.com/index.php/TMS/article/view/208
44. Srinivas, S., & Goel, L. (2025). Designing and Implementing Robust Test Automation Frameworks using Cucumber BDD and Java. *arXiv preprint arXiv:2505.17168*. https://doi.org/10.48550/arXiv.2505.17168
45. Talakola, S. (2022). Exploring the effectiveness of end-to-end testing frameworks in modern web development. *International Journal of Emerging Research in Engineering and Technology*, *3*(3), 29-39. https://ijeret.org/index.php/ijeret/article/download/119/109
46. Umar, M. A. (2023). A study of software testing: categories, levels, techniques, and types. *Authorea Preprints*. https://doi.org/10.36227/techrxiv.12578714.v1
47. William, S., & Virginia, W. (2022). Optimizing angular applications for enterprise-scale performance and scalability. *International Journal of Trend in Scientific Research and Development*, *6*(7), 2340-2348. https://www.ijtsrd.com/other-scientific-research-area/other/52409/optimizing-angular-applications-for-enterprisescale-performance-and-scalability/william-shakespeare
48. Xiong, W., Legrand, E., Åberg, O., & Lagerström, R. (2022). Cyber security threat modeling based on the MITRE Enterprise ATT&CK Matrix. *Software and Systems Modeling*, *21*(1), 157-177. https://link.springer.com/article/10.1007/s10270-021-00898-7
49. Yu Chung Wang, W., Pauleen, D., & Taskin, N. (2022). Enterprise systems, emerging technologies, and the data-driven knowledge organization. *Knowledge Management Research & Practice*, *20*(1), 1-13. https://doi.org/10.1080/14778238.2022.2039571