

Establishing a Scalable Test Automation and Traceability Framework for Enterprise Software Development Platforms

Brinda Pradeep

Submitted:02/06/2023

Revised:17/07/2023

Accepted:29/07/2023

Abstract: Software platforms used by enterprises require effective quality assurance practices to be taken to ensure that they remain reliable and compliant throughout development life cycle. The design and application of scalable verification and validation framework is presented in this paper. The requirements management, test planning, automated testing, defect tracking, and traceability are linked in the framework as one co-ordinated system. It interconnects requirements, development artifacts, test assets and quality records to allow teams to know about all the validation coverage. In the paper, an automated integration testing capability and a behavior driven testing architecture, which enhance the maintainability, reusability and speed of execution of tests are also introduced. The architecture was developed and experimented using geographically dispersed teams of engineers. Findings indicate that the quality of releases, compliance of processes and productivity of validation activities have improved. The paper defines the structure of the architecture, the traceability model, the test technique, and the results of the research were experienced in four release cycles. These results confirm the point that automation, traceability and quality governance are effective when implemented as a set of initiatives rather than considered as separate processes.

Keywords: *Enterprise software, Quality assurance, Continuous integration, Defect tracking, Requirements traceability, Behavior driven development, Test automation.*

1. Introduction

Software teams are under a continuous pressure to deliver the features at a higher rate maintaining the low rate of defects. The prevailing way of attaining this balance is continuous integration which is only successful when the test automation practices are mature (Wang et al., 2022). Lack of discipline can result in generating masses of test code that can be hard to maintain and trace requirements when automation is applied. This breaks the linkage between requirements and test evidence, posing actual risk to enterprise platforms, where change goes across many teams and many systems.

Traceability addresses some of this by linking requirements to the artifacts that validate requirements. This is not optional but is an obligatory practice in the safety critical fields. The MOOSE nuclear architecture-based nuclear simulation environment is an example that when implemented with much rigor automation and in code-documentation, it is possible to provide very

high nuclear quality assurance standards without significant overhead to daily development activity (Slaughter et al., 2021). Motor and aerospace teams are no exception, as defects are more likely to be missed when the specifications of the test cases of quality are poor (Juhnke et al., 2020).

To address these pressures, this paper offers a scalable framework to bring requirements management, automated testing, defect tracking and traceability under a single governance framework. It was constructed on the basis of an enterprise platform created by a distributed team spread around the world. Section 2 presents related work. The architecture of the framework is described in section 3. In section 4, the traceability model is presented. The automated testing and behavior driven architecture are covered in section 5. Section 6 describes the setup of the experiment. Result and discussion are reported in section 7. The paper is concluded in section 8.

2. Related Work

Studies on traceability have delivered a number of automation strategies during the last ten years. Escalona et al recommend a low-cost approach

Senior Software Engineer

Rockwell Automation

towards automation of requirements traceability framework by a model driven approach, due to the cost which discourages adoption due to manual traceability initiatives (Escalona et al., 2022). White and Krinke introduce TCTracer as a tool, which connects test artifacts to method-level and class-level code artifacts achieving a mean precision of more than 85 percent (White & Krinke, 2022). Großer et al. go beyond requirements traceability to the relational levels of individual requirement and explore relationships between entire requirement documents, revealing that document level traceability reveals inconsistency within large projects involving numerous contractors (Großer et al., 2022). Krause et al. investigate the advantages provided by traceability that exists prior to the finalization of a requirements specification, and discover significant role dependence on practitioner roles and project context (Krause et al., 2022).

Wang et al. perform a systematic review of the literature on the requirements traceability techniques published from 2006 to 2016 (114 papers). The review concludes that the most research is conducted on trustworthy and automated traceability challenges whereas the scalable and lightweight traceability techniques are under researched (B. Wang et al., 2018). This gap is also specific to enterprise systems, where the set of requirements, tests and defects can be in the thousands.

Outside the concept of traceability, automation systems are concerned with performance. Islam and Quadri outline a framework that is based on Selenium WebDriver and standardizes automation activities within an organization in a specific toolchain (Islam & Quadri, 2020). Garousi et al. cite a industry-academia project which developed a test-automation system around an aviation-simulation software, which in turn was implemented by multiple test teams in the manufacturing sector (Garousi et al., 2018). These studies indicate that automation structures can work well when they are coupled with organizational discipline coupled with standard tools, which was the guiding principle in designing automation structures presented in this paper.

3. Proposed Framework Architecture

The suggested framework divides the quality assurance activity into four layers that were based on each other. The initial layer takes care of requirements and contains a requirement with a unique identifier. The second layer deals with the test planning whereby the test cases are written and linked to one or more requirements. The third layer autonomously runs the tests and logs pass or fail results and metadata of the execution. The fourth layer monitors defects and traces every defect to the test case and requirement that revealed the defect. Figure 1 demonstrates this layered architecture and depicts where the data flows in and out of modules.

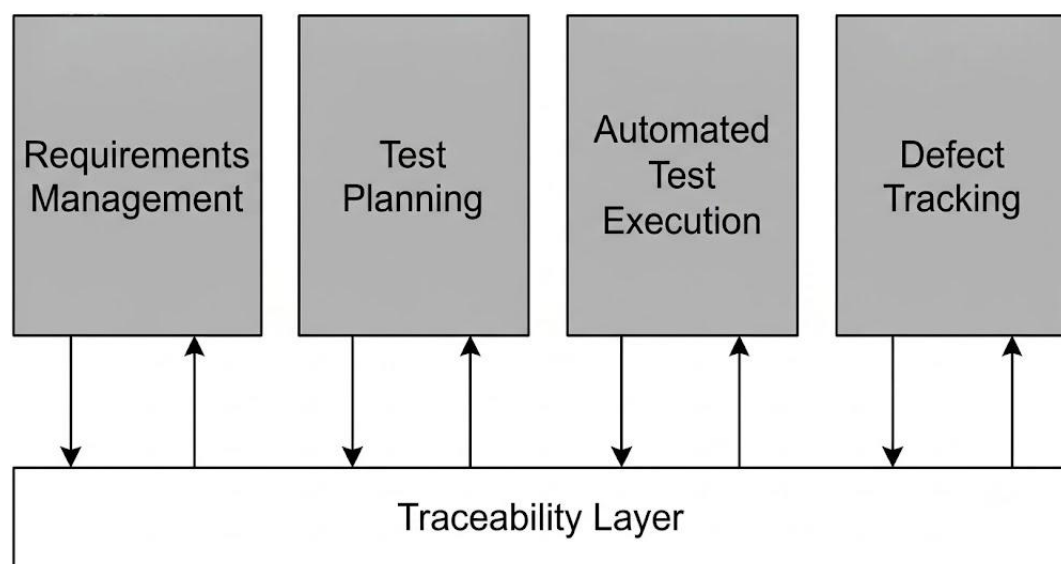


Figure 1: The proposed architecture in a layered form with the requirements and test planning, automated execution and defect tracking linked with a traceability layer.

Model driven methods are a good guide towards a linkup of natural language needs to structured artifacts. Wang et al. demonstrate that limited natural language specifications can be transformed into architecture models with automatically generated traceability links and eliminate ambiguity that in the past would halt automation (F. Wang et al., 2019). The requirements are formulated in a limited format such as a type field, a priority field, and an owner field, and according to this format the traceability engine will automatically generate links, not by manually tagging them.

The framework uses automation practices to determine the degree to which a team is utilizing automatization, which is determined by calculating a Test Automation Maturity Index. This index subsumes four scores of practices, automatization test coverage, test execution frequency, test detection and code review discipline. The scores of each practice are zero to one and each practice has a weight depending on the priority at the organization. The index is defined in equation 1.

$$TAMI = \frac{w_1P_1 + w_2P_2 + w_3P_3 + w_4P_4}{w_1 + w_2 + w_3 + w_4} \quad (1)$$

In the Equation 1, P_1 to P_4 are the four scores of the practice and w_1 to w_4 are the weights of the practice scores. An increased index means a greater discipline of automation, which earlier studies have linked with improved quality of products and reduced release time (Wang et al., 2022).

4. Requirements Traceability Model

The main component of the framework is the traceability layer. In each requirement, test case and defect record, there is a link identifier, which is indexed by the traceability engine. It has a structure based on the concept of layered traceability suggested by Großer et al., in which documents are considered as views on single requirements and there are relations within and among the views (Großer et al., 2022). The framework takes this concept a step further to test assets and records of defects such that a single query can be used to help trace a defect right back to the requirement that first started it.

To understand the extent to which the traceability layer worked in all the enterprise platform, the framework kept coverage data of four categories of requirements throughout the evaluation period. This data has been tabulated under Table 1.

Table 1. Requirements traceability coverage captured on four requirements categories in the twelve-week assessment on enterprise platform.

Requirement Type	Total Requirements	Linked Requirements	Coverage Percentage
Functional	420	398	94.8
Non-Functional	180	152	84.4
Safety Related	95	91	95.8
Interface	140	119	85.0

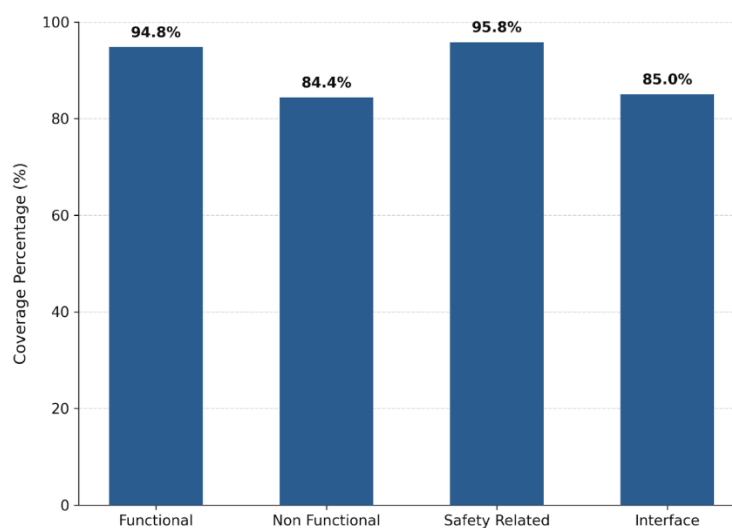


Figure 2: Bar chart that compares coverage of traceability percentage of each type of requirement functional, non-functional, safety related and interface requirement.

Table 1 supports pre specification traceability results of Krause et al., who observe that the benefits of traceability are different between the different requirements and the activities completed by the practitioners (Krause et al., 2022). Safety related requirements contributed most to coverage in this evaluation due to how they were treated as mandatory by the platform team to release them, which was also common in the review by Wang and colleagues with regards to trustworthy traceability techniques (B. Wang et al., 2018).

A simple ratio is used to determine the coverage as illustrated in Equation 2.

$$TCR = \frac{N_{linked}}{N_{total}} \times 100 \quad (2)$$

The N_{linked} of Equation 2 is the number of requirements with an active trace link and N_{total} is the total amount of requirements in that category. This ratio provided a continuity with the quality team tracking metric over releases and sprints.

5. Automated Testing Architecture

The framework supports behavior driven development and uses it to elucidate test cases in well-organized natural language expressive that can be read in both technical and non-technical stakeholders. Zampetti et al. also examined open-

source adoption of behavior driven frameworks and discovered that an average of 27 percent of the surveyed projects actually adopted them, the adoption being focused on Ruby projects (Zampetti et al., 2020). Their research also revealed that most teams end up writing tests during or after the coding process actually, as opposed to strictly adhering to behavior driven development because the discipline involved is perceived to be potentially work intensive. The risk in this framework was managed by making scenario writing an intrinsic part of the test planning layer discussed in Section 3 such that scenario writing was not an extra activity.

Binamungu et al. surveyed practitioners on factors that constitute a behavior driven test suite to be high-quality and offered four fundamental quality principles such as clarity, business relevance, low duplication, and small scenario size (Binamungu et al., 2020). The framework used effects of these principles by examining the quality of scenarios prior to automation which resulted in less rework in subsequent sprints.

The data on execution of scenarios has been gathered over four sprints to assess whether execution was better in terms of reliability and speed after this integration. The results have been shown in Table 2 and the trend of the pass rate has been shown in Figure 3.

Table 2. Scenario execution outcomes were identified in four development sprints on the enterprise platform which were behavior driven.

Sprint	Scenarios Executed	Pass Rate Percentage	Average Execution Time Minutes
Sprint 1	210	78.1	42
Sprint 2	245	83.7	38
Sprint 3	268	88.4	33
Sprint 4	290	92.1	29

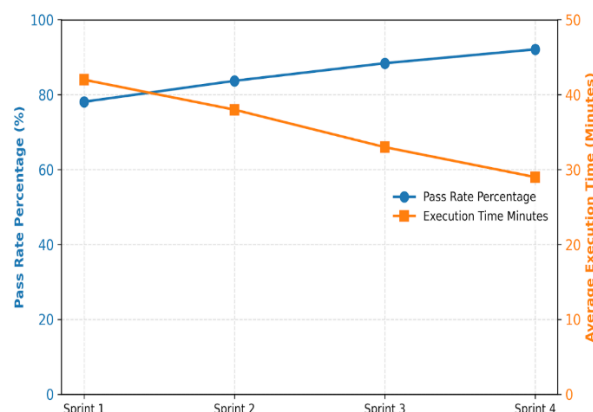


Figure 3: Line chart illustrating the increase in scenario pass rate, and the decrease in average execution time over four development sprints.

Table 2 has the same trend of improvement compared to the overall conclusion of Zampetti et al. that the quality of scenarios and code are likely to progress together when a team has a disciplined workflow (Zampetti et al., 2020). It also indicates the framework suggestion by Islam and Quadri who posit that standardized tooling in an organization enhances scalability and reliability of automated execution (Islam & Quadri, 2020).

6. Experimental Setup

The structure has been implemented on an enterprise software system designed by three engineering departments that were in different geographies. A total of four consecutive releases were evaluated with the first 2 being a baseline stage before complete adoption of the framework and the second two releases being adopted framework. This arrangement enabled a comparison of the results of defect detection.

The use of test case prioritization was also involved in the experiment. In their study, Flemström et al.

discovered that the pre-ordering of test cases, such that like phases could be reused, led to a verified decrease in manual test effort, averaging 12 percentage points off four industrial undertakings (Flemström et al., 2018). The framework used a rule of prioritization in the automation planning phase by prioritizing test cases similarly in terms of shared similar steps then handing over these cases to the automated execution layer as in Section 3.

Defect detection efficiency is a ratio of the defects that are identified during testing and not at the time when the product has been published. This is defined as equation 3.

$$DDE = \frac{D_{test}}{D_{test} + D_{post}} \times 100 \quad (3)$$

D_{test} is the number of defects found in a test and D_{post} is the number of defects found post-release (Equation 3). Table 3 records this measure to the four releases under consideration and Figure 4 visualizes this trend in the form of a chart.

Table 3. The efficiency of defect detection in terms of the number of releases that were made in four releases in compiling a comparison between baseline releases before adoption of the framework to release after adoption.

Release	Defects Found in Testing	Defects Found Post Release	Detection Efficiency Percentage
Release 1 Before	156	64	70.9
Release 2 Before	172	58	74.8
Release 3 After	210	26	89.0
Release 4 After	234	19	92.5

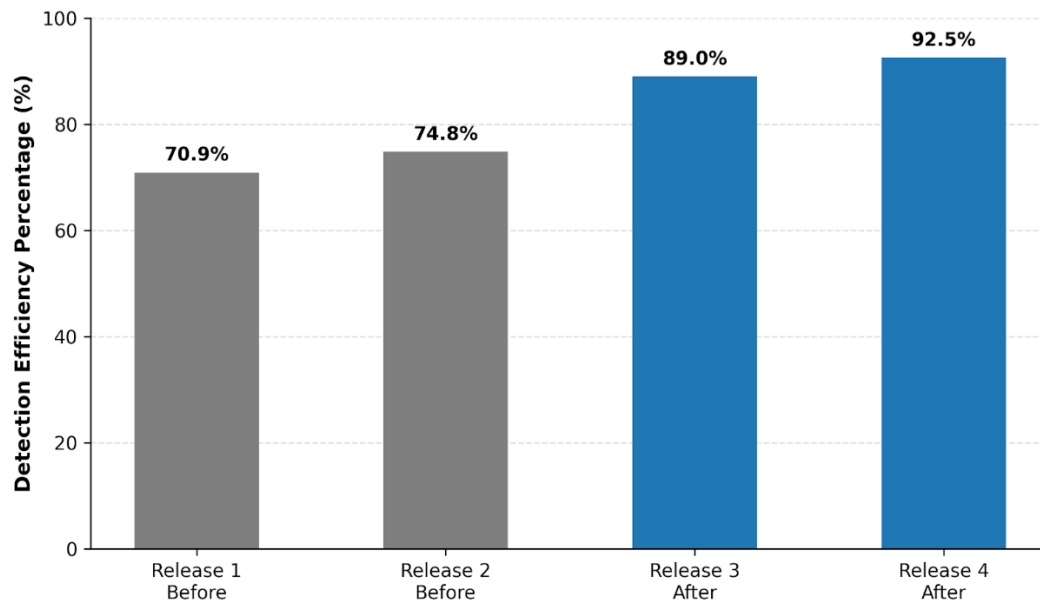


Figure 4: Comparison between defect detection efficiency under two baseline releases and two releases after the adoption of the framework in the form of bar chart.

The efficiency improvement as presented in Table 3 is corroborative to the larger argument by Wang et al. that the greater the maturity of the automation of tests, the greater the quality of the products in the real project environment (Wang et al., 2022). The gain also implies the benefit of traceability as explained by Escalona et al. as the defects could be traced faster back to the requirement that caused them in case trace connections were made automatically (Escalona et al., 2022).

7. Results and Discussion

In addition to the quantitative measures, the evaluation process also captured qualitative data (feedbacks) of sixteen engineers in the three distributed teams. This feedback assisted the quality team in knowing the areas that had to be changed more in the framework. The summarized table 4 represents key themes based on structured interviews, which were carried out at the culmination of the evaluation process.

Table 4. Qualitative themes were based on structured interviews with sixteen engineers of three distributed work teams that were conducted after the evaluation period.

Theme	Representative Observation	Frequency Category
Documentation clarity	There was a lack of clarity in requirement descriptions in initial sprints	High
Tool integration	There were smoother requirements-to-defect tracking tool transition	Medium
Communication across teams	Distributed teams emphasized better visibility of test status by common dashboards	High
Training needs	Emerging team members needed to be further guided on the understanding of traceability links	Medium

The documentation clarity theme in Table 4 corresponds to the challenges with the taxonomy proposed by Juhnke et al. who found the issues with content in the input artifact to be the primary goal hindering the good quality of test case specification

in a large network of automotive suppliers (Juhnke et al., 2020). This implies that traceability tooling cannot be the initiative that resolves quality issues that start with the ambiguous formulation of requirements. The theme of communication is also

consistent with the argument that Wang et al. make when stating that coordination issues are one of the under researched in the field of traceability, although this issue is of high importance to distributed teams (B. Wang et al., 2018).

Also, release cycle time was better during the period of the evaluation. The average base release times were 22 and 20 days (bases values) for the two base releases and the values were found to be 14 and 12 days respectively after the adoption of the framework. This comparison is given in figure 5.

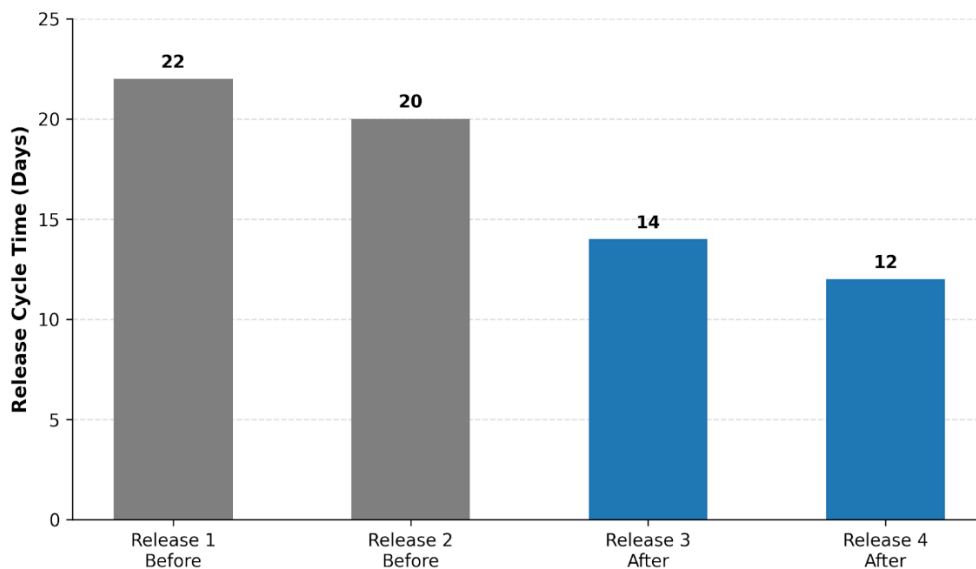


Figure 5: Comparison of average release cycle time in days of the releases at the baseline of releases and release after adoption of framework.

This decrease confirms the result reported by Wang et al. that in conditions of continuous integration, the maturity of test automation is higher with lower release cycles (Wang et al., 2022). It is also an indication of the generation strategy suggested by Mustafa et al., because the generated test cases based on structured requirements, the time taken to create the test cases manually which would have typically increased the release dates was minimized (Mustafa et al., 2021). This reduction was also achieved with the help of the prioritization technique of Flemström et al., which decreased the amount of automation work on the subsequent sprints, which were redundant with the first one (Flemström et al., 2018).

Some limitations remain. The assessment was only done on four releases and three teams which restricts the extent to which the results can be totally generalized to the enterprise platforms on a larger level. The quality of writing the requirements was also different among teams that impacted the coverage of traceability in the non-functional and interface categories as indicated in Table 1. In future assessments a scheme should expand the scheme over additional releases and broader project types and we should heed the suggestion of Wang et al. to

diversify traceability research to include a wider variety of projects (Wang et al., 2022).

8. Conclusion

In this paper, a scalable structure was discussed that links the requirement management, test planning, automated testing, defect tracking, and traceability as a single system to the enterprise software platforms. The framework used layered architecture, a limited requirement format and behavior driven test method to enhance coverage of the traces as well as the efficiency of the test execution. Four releases of experimental results were indicated to have high defect detection efficiency, reduced release cycle and increased coverage of traceability coverage post adoption of frame works. Distributed engineering teams used qualitative feedback to confirm that these gains were in addition to better communication and dashboard visibility but documentation clarity and training needs are areas which require improvement.

References

- [1] Binamungu, L. P., Embury, S. M., & Konstantinou, N. (2020). Characterising the quality of behaviour driven development specifications. *Lecture Notes in Business*

- Information Processing*, 87–102.
https://doi.org/10.1007/978-3-030-49392-9_6
- [2] Escalona, M., Koch, N., & Garcia-Borgoñon, L. (2022). Lean requirements traceability automation enabled by model-driven engineering. *PeerJ Computer Science*, 8, e817.
<https://doi.org/10.7717/peerj-cs.817>
- [3] Flemström, D., Potena, P., Sundmark, D., Afzal, W., & Bohlin, M. (2018). Similarity-based prioritization of test case automation. *Software Quality Journal*, 26(4), 1421–1449.
<https://doi.org/10.1007/s11219-017-9401-7>
- [4] Garousi, V., Tasli, S., Sertel, O., Tokgoz, M., Herkiloglu, K., Arkin, H. F. E., & Bilir, O. (2018). Automated Testing of simulation software in the Aviation industry: An Experience report. *IEEE Software*, 36(4), 63–75.
<https://doi.org/10.1109/ms.2018.227110307>
- [5] Großer, K., Riediger, V., & Jürjens, J. (2022). Requirements document relations. *Software & Systems Modeling*, 21(6), 1–37.
<https://doi.org/10.1007/s10270-021-00958-y>
- [6] Islam, M. N., & Quadri, S. M. K. (2020). Framework for Automation of Cloud-Application Testing using Selenium (FACTS). *Advances in Science Technology and Engineering Systems Journal*, 5(1), 226–232.
<https://doi.org/10.25046/aj050129>
- [7] Juhnke, K., Tichy, M., & Houdek, F. (2020). Challenges concerning test case specifications in automotive software testing: assessment of frequency and criticality. *Software Quality Journal*, 29(1), 39–100.
<https://doi.org/10.1007/s11219-020-09523-0>
- [8] Krause, J., Kaufmann, A., Riehle, D., & Jung, M. (2022). The Benefits of Pre-Requirements Specification Traceability. *The Benefits of Pre-Requirements Specification Traceability*, 166–177.
<https://doi.org/10.1109/re54965.2022.00021>
- [9] Mustafa, A., Wan-Kadir, W. M. N., Ibrahim, N., Shah, M. A., Younas, M., Khan, A., Zareei, M., & Alanazi, F. (2021). Automated Test Case Generation from Requirements: A Systematic Literature Review. *Computers, Materials & Continua/Computers, Materials & Continua (Print)*, 67(2), 1819–1833.
<https://doi.org/10.32604/cmc.2021.014391>
- [10] Slaughter, A. E., Permann, C. J., Miller, J. M., Alger, B. K., & Novascone, S. R. (2021). Continuous integration, In-Code documentation, and automation for nuclear quality assurance conformance. *Nuclear Technology*, 207(7), 923–930.
<https://doi.org/10.1080/00295450.2020.1826804>
- [11] Wang, B., Peng, R., Li, Y., Lai, H., & Wang, Z. (2018). Requirements traceability technologies and technology transfer decision support: A systematic review. *Journal of Systems and Software*, 146, 59–79.
<https://doi.org/10.1016/j.jss.2018.09.001>
- [12] Wang, F., Yang, Z., Huang, Z., Liu, C., Zhou, Y., Bodeveix, J., & Filali, M. (2019). An approach to generate the traceability between restricted natural language requirements and AADL models. *IEEE Transactions on Reliability*, 69(1), 154–173.
<https://doi.org/10.1109/tr.2019.2936072>
- [13] Wang, Y., Mäntylä, M. V., Liu, Z., & Markkula, J. (2022). Test automation maturity improves product quality—Quantitative study of open source projects using continuous integration. *Journal of Systems and Software*, 188, 111259.
<https://doi.org/10.1016/j.jss.2022.111259>
- [14] White, R., & Krinke, J. (2022). TCTracer: Establishing test-to-code traceability links using dynamic and static techniques. *Empirical Software Engineering*, 27(3).
<https://doi.org/10.1007/s10664-021-10079-1>
- [15] Zampetti, F., Di Sorbo, A., Visaggio, C. A., Canfora, G., & Di Penta, M. (2020). Demystifying the adoption of behavior-driven development in open source projects. *Information and Software Technology*, 123, 106311.
<https://doi.org/10.1016/j.infsof.2020.106311>