Webology (ISSN: 1735-188X)
Volume 18, Number 06, 2021

A Mathematical Model for Improving Model-Based Testing Approach in Agile Software Development

Sandhya Satyarthi[1], Dhirendra Pandey[2*]

[1]Department of IT, Babasaheb Bhimrao Ambedkar University, Lucknow

[2*]Department of IT, Babasaheb Bhimrao Ambedkar University, Lucknow

*Corresponding Author: Dhirendra Pandey

Received: 28.10.2021 Accepted: 29.11.2021 Published: 10.12.2021

Abstract
Continuous modifications and life cycle optimization are the cornerstones on which agile development is built. It offers an iterative technique to comprehend and enhance the software development process throughout the development process. The system's security is essential for maintaining and improving the system and its output. One of the main issues that both consumers and developers have is security. The risk can be reduced when efficiently promoting model-based testing in agile-based development by improving testing by using a proposed mathematical model. The security artifacts that help to characterize the tasks associated with the specific development with the contribution of the security to keep the system safe and reduce risk. Also, the understanding regarding the Model-based Testing in agile based development and discussion about the improved mathematical model for MBT in ASD is the main focus of this paper. Understanding security artifacts is crucial since it aids in the upkeep and upgrading of software. Developers may use them as a source of reference to assist address security-related issues and their utilization aids in secure Scrum-based agile software development process.

Keywords: Model-based Testing (MBT), Agile software Development (ASD), Agile Development (AD).

1. Introduction

Model-based testing may be very effective in identifying vulnerabilities in memory as well as possible incompatibilities that might result in a software collapse because computerized tests can run for a long time while submitting random data. The batch applications that use a single input, such as a file, and a single output, such as filling a database, are often the simplest to start using with MBT. This testing method is used and included in the testing methodologies. Today, a variety of business tools are being developed to support this kind of technique [1]. This form of software allows for run-time comparison of program behaviors to model-generated predictions. A system's behavior is mostly determined by the steps, order, circumstances, and input-output flow of a created process. When this is put into practice, we should be able to define the concept—whether it is important to the system or shareable—in a very specific way [2,3]. Model-based testing (MBT) is a software testing approach that uses models to represent the behavior and requirements of a system, and generates test cases from these models. MBT can be effectively integrated into agile software development processes to ensure software quality and accelerate the testing process. Software quality is growing more and more crucial, yet challenging to attain. The two primary strategies for resolving these issues are model-based testing (MBT) and agile development (AD) [4].

Software systems are frequently complicated and changed frequently in agile development. Teams can build abstract models or representations of the system, encapsulating its behavior, structure, and interactions, using model-based testing. These models are used as a foundation for creating test cases that may cover a variety of scenarios, ensuring thorough test coverage and lowering the chance of forgetting important system components. Early and continuous testing are stressed in agile techniques. By leveraging the models to generate test cases, model-based testing enables teams to begin testing early in the development cycle [5, 7]. Early defect detection enables rapid problem-solving, lowering rework, and raising overall program quality. Since it enables frequent and quick testing iterations, test automation is a key component of agile development [6,8]. Model-based testing creates test scripts straight from the models, laying a strong foundation for test automation. These scripts can be run automatically, allowing for speedier feedback cycles, regression testing, and effective resource management [9, 10].

Agile teams frequently deal with shifting requirements and developing software systems. Model-based testing helps agile development achieve the requisite adaptability. Models can be changed to reflect changes, and test cases can then be generated using the new models Teams may verify that testing efforts directly address the established requirements by connecting models to requirements and creating test cases from models, boosting transparency and verifiable.

The rest of this paper is organized as follows. In section 2, we discuss about the activities related to the motivation for our work and background studies that also help in finding the research gap. Section 3, describes the incorporation of model based-testing in agile based software development. Section 4, discuss about the various testing techniques that helps in rationalization for selecting MBT for ASD. The Section 5, discussion about mathematical model for improving model-based testing in agile software development. Section 6, defines observations and findings in the paper. Finally, the conclusion and the future direction of research in section 8.

2. Literature review

Agile development strategies include concentrating on quick turnaround, being flexible, participating in daily stand-up meetings, utilizing a lightweight approach for requirements, and concretizing them through interaction with the product manager, who is also known as the product owner. The improvisational testing has proven to be quite effective and adaptable: It is simpler to adapt to requirements changes and maintain small, independent test modules. Early testing is therefore feasible. The tester may also respond instantly to dynamic information, such as suspicious signals. It is simpler to examine the test log and find oversights than it is to perform a big, fully automated test.
In 2007, Mika katara et. al., addresses the issue of model-based test generating tools' requirements not matching the artefacts created in agile software development projects. The approach was domain-specific and depends on the availability of subject-matter experts to create the test models. The testers connect use cases, which are transformed into sequences of so-called action words that, at a high level of abstraction, correspond to user actions, to test generation systems. [11]
In 2014, Sandeep Sivanandan et. al., publish a paper, that examines the construction of a framework for behavior-driven test automation using MBT and how it could be utilized effectively in Agile Development. The combination of Graphwalker, a model-based graphical user interface test generator, with behavior-driven development framework and Robot Framework is an experiment in the automation framework.[12]
In 2015, Robert V. Binder et.al., performed a study to find out how users of MBT feel about its effectiveness and efficiency and it was accessible to everyone who had examined or utilized an MBT strategy. Beginners are frequently perplexed by the more technical problems that seek to justify a standard MBT classification scheme and various MBT methodologies. Users might understand both the broad diversity and specific approaches with the use of a common classification system.[13]

In 2018, Dalton N. Jorge et. al., suggests adopting CLARET, a notation that enables use case specifications to be created using plain language, as the primary artefact for both RE and MBT practises. Additionally, provide preliminary research on the usage of CLARET to produce RE documents and on their integration into a system testing procedure based on MBT. Results indicate that using CLARET, we may efficiently and cheaply document use cases.[14]
In 2021, Athanasios Karapantelakis et. al., built and provided a collection of models that, given some starting criteria like personnel availability and skill level, as well as historical data utilization, can estimate the costs of adopting MBT. These models were constructed using knowledge gained from earlier MBT practice. We run a number of simulations on fictitious MBT adoption and use scenarios, which may be realistically applied to various teams considering adopting MBT, to show the practical value of our models.[15]
In 2022, Robbert Jongeling et. al., released a research that discussed about frequent, lightweight consistency tests across views and between heterogeneous models can help developers achieve multi-view consistency in the framework of agile model- based development. The checks are lightweight because they are simple to make, amend, use, and maintain, they identify discrepancies but do not try to fix them automatically, and they are straightforward to produce, use, and supervise.[16]
The various studies are done to efficiently working of the model based testing tools in the iterative development [17]. By following the theories and models we will examine the mathematical model for improving MBT in ASD that helps in early defect and risk reduction.

3. Model-Based Testing in Agile Development

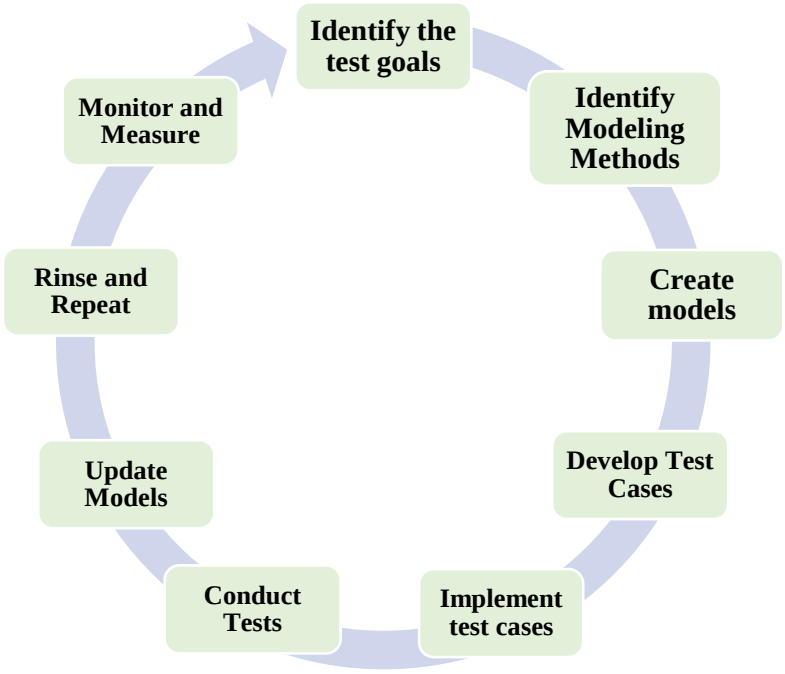

Figure.1: Using model-based testing in agile software development.

Model-based testing (MBT) is a software testing methodology that creates test cases based on models that capture the behavior and requirements of a system. To guarantee software quality and quicken the testing process, MBT can be efficiently included into agile software development methods.[18] Here is a methodology for using model-based testing in a setting that emphasizes agile software development:

1. Identify the test goals: Define the testing goals for your agile project, taking into account the priority, scope, and quality objectives. These goals will direct the MBT process and aid in your decision-making over what to model and what to test.

2. Identify Modeling Methods: Select modeling strategies that are compatible with your software development process's agile concepts and practices. State-transition diagrams, data flow diagrams, use case diagrams, and activity diagrams are typical MBT modeling tools.

3. Create models: Using the defined objectives as a guide, create models that depict the system under test's (SUT) anticipated behaviour. These models should include relationships, dependencies, and functional and non-functional requirements among system components. Include all pertinent parties in the design of the model, including as developers, testers, and business analysts, to ensure thorough coverage.

4. Develop Test Cases: Utilize the models to create test cases that are automatically generated and cover a variety of scenarios and combinations of inputs and outputs. These test cases ought to be based on the model and exhibit the SUT's anticipated behavior. Automating this procedure with test case creation tools or frameworks will guarantee consistency and effectiveness.

5. Implement test cases: Use your agile development environment to implement the generated test cases. Working closely with developers to match the test cases with the changing SUT throughout each sprint or iteration may entail writing code for automated tests, creating test data, and setting up test environments.

6. Conduct Tests: Execute the produced test cases against the SUT to check for errors and validate the SUT's behavior. Keep track of the test findings, examine them, and work with the development team to address any problems you find. Rapid feedback and ongoing improvement are made possible by this iterative approach, which aids in the early detection and correction of faults.

7. Update Models: Update the models to reflect changes in requirements or system behavior as the SUT evolves during agile development. As a result, the models and test cases continue to be applicable and useful in directing the testing process.

8. Rinse and Repeat: Carry out the MBT process repeatedly throughout the agile development lifecycle, iteratively updating the models, creating test cases, carrying out and executing tests, and working together with the development team to guarantee the quality of the software.

9. Monitor and Measure: Using relevant metrics, such as code coverage, defect detection rate, and test execution time, monitor and gauge the success of the MBT process. Make any necessary modifications to the MBT framework after analyzing the findings to find potential improvement areas. As shown in figure. 1, a methodology shown how to integrate model-based testing into an environment based on agile software development, allowing for early defect identification, effective test coverage, and increased overall product quality.

4. Comparative study of the testing

The selection of testing technique in agile based development is decisive due to iterative and continuous development. Model-based testing (MBT) is a testing technique that uses models to guide the design, generation, and execution of test cases. It differs from other testing techniques in its approach and benefits[19,22]. The comparison of MBT with a few other common testing techniques in table given below:

The benefits of model-based testing include increased test coverage, early defect detection, improved efficiency, and reduced reliance on human expertise. However, MBT requires investment in modeling tools, expertise in model creation, and maintenance of the models as the system evolves.

Ultimately, the choice of testing technique depends on the specific project, resources available, time constraints, system complexity, and the desired level of automation and coverage.

S.no.	Testing	Purpose	Challenges
1	Manual Testing	• Manual testing relies on human testers to execute test cases manually. • Test cases are designed based on human knowledge, experience, and understanding of the system.	• Manual testing can be time-consuming, labor-intensive, and prone to human error. • It is suitable for small-scale projects or scenarios where automation is not feasible or cost-effective.
2	Automated Testing	• Automated testing uses tools and scripts to automate the execution of test cases. • Test cases are designed and implemented as scripts that simulate user interactions with the system.	• Automated testing can be faster, more reliable, and repeatable compared to manual testing. • It is suitable for projects with frequent releases, large-scale systems, and regression testing.
3	Exploratory Testing	• Exploratory testing involves simultaneous learning, test design, and test execution. • Testers explore the system dynamically, often without predefined test cases.	• Exploratory testing relies heavily on tester intuition, creativity, and domain knowledge. • It is suitable for scenarios where requirements are ambiguous, usability is a concern, or ad-hoc testing is required.
4	Model-Based Testing	• MBT utilizes models to guide test case design, generation, and execution. • Models can be graphical, mathematical, or textual representations of system behavior. • Test cases are automatically derived from the models, ensuring comprehensive coverage. • MBT provides systematic and structured test design, reducing human bias and improving efficiency.	• It is suitable for complex systems, requirements-driven testing, and generating large sets of test cases. • Early defect detection, improved efficiency, and reduced reliance on human expertise. However, MBT requires investment in modeling tools, expertise in model creation, and maintenance of the models as the system evolves.

Table 1: Comparison of MBT with other Testing Techniques

5. A mathematical model for improving model-based testing in agile software development

Testing plays a vital role in development, that impact the overall productivity of the process of development

Let:

T = Total number of test cases
N = Number of test cases generated from models
E = Number of test cases generated from exploratory testing
M = Number of defects found by model-based testing
D = Number of defects found by exploratory testing
R = Rate of defect detection by model-based testing ($0 <= R <= 1$)
S = Rate of defect detection by exploratory testing ($0 <= S <= 1$)
P = Proportion of defects found by model-based testing ($0 <= P <= 1$)

The mathematical model for improving model-based testing in agile software development can be represented as follows:

The total no. of test cases are calculated by adding the no. of test cases generated from models and No. of test cases generated from exploratory testing.
As, T = N + E
The number of defects found by model-based testing (M) is calculated as the rate of defect detection by model-based testing (R) multiplied by the proportion of defects found by model-based testing (P), multiplied by the number of test cases generated from models (N).
M = R * P * N

Similarly, the number of defects found by exploratory testing (D) is calculated as the rate of defect detection by exploratory testing (S) multiplied by the complement of the proportion of defects found by model-based testing (1 - P), multiplied by the number of test cases generated from models (N).
D = S * (1 - P) * N
This model allows agile software development teams to assess the effectiveness of their model-based testing approach by quantifying the number of defects found by model-based testing and exploratory testing, and their respective rates of defect detection. By adjusting the rate of defect detection by model-based testing (R), the proportion of defects found by model-based testing (P), and the rate of defect detection by exploratory testing (S), teams can optimize their testing strategy to improve the overall quality of their software product.

6. Observations and findings

The model is formed by in cooperation of exploratory and model based testing by the essential adjustment in the no. of test cases generated by model and the defects found by model based testing. By quantifying the number of defects detected by model-based testing and qualitative testing and their respective instances of defect detection, this model enables agile software development teams to evaluate the efficacy of their model-based testing approach [20]. Model-based testing (MBT) is a method for testing software that creates test cases automatically using models. Due to a number of difficulties, MBT implementation in an agile software development environment might be difficult. In the context of agile software development, MBT frequently causes the following problems:
1. Iterations in Agile Development: It may be challenging to construct and maintain accurate models for MBT because needs could change often while the system is being developed. It can be difficult to make sure that the models used for MBT are timely updated to reflect the most recent software updates.

2. Rapidly Altering Requirements: Agile development frequently entails alterations to requirements, which may have an effect on the MBT models. This would entail more work and might cause testing to be delayed. MBT may find it difficult to keep up with the rapid changes in agile development.

3. Continuous Integration and Deployment: Due to the necessity to represent the most recent software changes in the models used to generate test cases, this can provide difficulties for MBT [21]. It might be difficult and time-consuming to keep the MBT models up to date with the software's regular updates.

3. Limited Documentation: It can be difficult to develop and maintain correct models for MBT in an agile setting due to the lack of precise documentation, particularly for complex systems.

5. Team Collaboration: Agile development places a strong emphasis on close communication and cooperative teamwork. But not all team members may have the specialised knowledge and experience needed to create and maintain models for MBT. In an agile setting, it might be difficult to work effectively with domain experts, testers, and developers to design, validate, and update models for MBT.

7. Test Oracles: It can be difficult to ensure that the models used for MBT accurately represent the expected behaviour of the software and serve as reliable test oracles in an agile development environment when requirements and design might change quickly.

8. Tool Selection and Integration: MBT frequently needs specialised tools to develop test cases, create models, and manage models. It can be difficult to choose and integrate the proper MBT tools into an agile development environment because they must be compatible with agile practices and flow easily into the development workflow.

If model-based testing might be a useful technique for software testing, its use in an environment focused on agile software development may provide difficulties. It is possible to successfully use MBT in an agile setting by addressing problems with continuously changing requirements, continuous integration and deployment, limited documentation, team collaboration, test oracles, tool selection, and integration. In order to overcome these difficulties and make MBT a successful testing strategy in an agile-based software development process, proper planning, coordination, and teamwork are essential.

7. Conclusion and future scope

In conclusion, model-based testing in agile software development improves test coverage, allows for early defect identification, helps test automation, encourages cooperation, allows for change-adaptability, and makes it easier to trace and document processes. Agile teams can enhance software quality, feedback cycles, and stakeholder satisfaction by using models to guide testing efforts. This model lets agile software development teams figure out how well their model-based testing method works by figuring out how many bugs model-based testing and exploratory testing find and how fast they find them. The comparative study related to testing in this paper helpful in concluding the reasoning behind choosing exploratory and MBT from others testing techniques. The work in this area, still a challenging task to show how the working model is effectively improve the early defect detection and risk management in ASD. However, abstract models are required for the integration, which current MBT tools cannot effectively manage. In future we expand our research with more MBT Tools like Selenium, Appium, and Cypress for automating the execution of test cases generated from models.

References

[1]. Michiel Pieter and Willem Jacob van Osch, "Automated Model-based Testing of Hybrid Systems", pp. 2009-04, 2009.
[2]. B. Gopularam, "Mechanism for on demand Tag-Based software testing in virtualized environments", 4th International Conference on Advance Computing - IcoAC-2012, Dec 13–15, 2012.
[3]. “Applied Model Based Testing - Experiences & Examples", SimonEjsing, August 2007, [online] Available: http://appMBT.blogspot.in/.
[4]. R. V. Binder, B. Legeard and A. Kramer, "Model-based testing: Where does it stand", Queue, vol. 40, no. 1, pp. 40-40, Dec. 2014.
[5]. E. Bjarnason, M. Unterkalmsteiner, M. Borg and E. Engström, "A multi-case study of agile requirements engineering and the use of test cases as requirements", Information and Software Technology, vol. 77, pp. 61-79, 2016.
[6]. K. El-Far and J. A. Whittaker, "Model-based software testing" in Encyclopedia on Software Engineering, Wiley-Interscience, 2001.

[7]. B. Hois, S. Sobernig and M. Strembeck, "Natural-language scenario descriptions for testing core language models of domain-specific languages", Model-Driven Engineering and Software Development (MODELSWARD) 2014 2nd International Conference on, pp. 356-367, 2014.
[8]. D. N. Jorge, W. L. Andrade, P. D. L. Machado and E. L. G. Alves, "Claret - central artifact for requirements engineering and model-based testing", Brazilian Conference on Software (CBSoft Tools), 2017.
[9]. Marques, F. Ramalho and W. L. Andrade, "Comparing model-based testing with traditional testing strategies: An empirical study", IEEE ICST Workshops/AMOST 2014, 2014
[10]. Wang, F. Pastore, A. Goknil, L. Briand and Z. Iqbal, "Automatic generation of system test cases from use case specifications", Proceedings of the 2015 International Symposium on Software Testing and Analysis ISSTA 2015, pp. 385-396, 2015.
[11]. Mika Katara and Antti Kervinen, “Making Model-Based Testing More Agile: A Use Case Driven Approach. In: Bin, E., Ziv, A., Ur, S. (eds) Hardware and Software, Verification and Testing. HVC 2006. Lecture Notes in Computer Science, vol 4383. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-70889-6_17, 2007.
[12]. S. Sivanandan and Yogeesha C. B, "Agile development cycle: Approach to design an effective Model Based Testing with Behaviour driven automation framework," 20th Annual International Conference on Advanced Computing and Communications (ADCOM), Bangalore, India, 2014, pp. 22-25, doi: 10.1109/ADCOM.2014.7103243.
[13]. Robert V. Binder, Bruno Legeard, and Anne KramerModel-based Testing:Where Does It Stand?, 10.1145/2716276.2723708
[14]. D. N. Jorge, P. D. L. Machado, E. L. G. Alves and W. L. Andrade, "Integrating Requirements Specification and Model-Based Testing in Agile Development," 2018 IEEE 26th International Requirements Engineering Conference (RE), Banff, AB, Canada, 2018, pp. 336-346, doi: 10.1109/RE.2018.00041.
[15]. Karapantelakis, A. (2021). Estimating Costs for Adopting and Using Model-Based Testing in Agile SCRUM Teams. 2021 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW). https://doi.org/10.1109/icstw52544.2021.00042 10.1109/icstw52544.2021.00042
[16]. Jongeling, R., Ciccozzi, F., Cicchetti, A., Carlson, J. (2019). Chapter 6 Lightweight Consistency Checking for Agile Model-Based Development in Practice. In: Bosch, J., Carlson, J., Holmström Olsson, H., Sandahl, K., Staron, M. (eds) Accelerating Digital Transformation. Springer, Cham. https://doi.org/10.1007/978-3-031-10873-0_8
[17]. Agarwal, P.: Continuous scrum: agile management of saas products. In: Proceedings of the 4th India Software Engineering Conference, pp. 51–60 (2011)
[18]. Albuquerque, C., Antonino, P., Nakagawa, E.: An investigation into agile methods in embedded systems development. In: Computational Science and Its Applications, Lecture Notes in Computer Science, vol. 7335, pp. 576–591. Springer (2012). URL
[19]. Basri, S., Dominic, D.D., Murugan, T., Almomani, M.A.: A proposed framework using exploratory testing to improve software quality in sme’s. In: International Conference of Reliable Information and Communication Technology, pp. 1113–1122. Springer (2018)
[20]. Berger, C., Eklund, U.: Expectations and challenges from scaling agile in mechatronics-driven companies–a comparative case study. In: International Conference on Agile Software Development, pp. 15–26. Springer (2015)
[21]. Höpfner G, Jacobs G, Zerwas T, et al (2021) Model-based design workflows for cyber-physical systems applied to an electric-mechanical coolant pump. In: IOP Conference Series: Materials Science and Engineering 1097, 1
[22]. C. Wang, F. Pastore, A. Goknil, L. Briand and Z. Iqbal, "Automatic generation of system test cases from use case specifications", Proceedings of the 2015 International Symposium on Software Testing and Analysis ISSTA 2015, pp. 385-396, 2015.