**RIGA TECHNICAL UNIVERSITY**

Faculty of Computer Science and Information Technology

Institute of Applied Computer Systems

**Pankaj Chettri**

Master Program “Business Informatics”

Student ID No 221AEM001

# OPTIMIZING TESTING PRACTICES IN AGILE DEVELOPMENT

## QUALIFICATION PAPER

RIGA 2024

# RIGA TECHNICAL UNIVERSITY

## FACULTY OF COMPUTER SCIENCE AND INFORMATION TECHNOLOGY

Institute of Applied Computer Systems

Business Informatics

**Work Performance and Assessment Sheet of the Qualification Paper**

Student Pankaj, Chettri _____

(signature, date)

# ABSTRACT

Agile testing, regression testing, automation testing tools, uncertainty in software development, optimization.

This literature analysis takes a systems approach to thoroughly investigate and connect major components driving Agile testing processes. The paper, which examines Agile testing approaches, uncertainty in software testing, and a comparison of manual and automated testing, emphasizes the necessity of a comprehensive evaluation of output quality. It investigates the adaptive techniques of hybrid testing and the integration of adaptive autonomous test cases, focusing on their systemic implications in the Agile software development lifecycle. Agile testing challenges, such as dynamic settings and acceptance-driven development, are addressed as systemic issues.

This paper offers useful insights on optimizing testing processes, recognizing the dual role of manual and automated testing within the complex fabric of Agile development.

The total number of pages in this paper is 29 with 6 figures, 1 table and 17 cited sources.

# TABLE OF CONTENTS

ABSTRACT....3
INTRODUCTION....5
1. AGILE DEVELOPMENT METHODOLOGY....7
2. DUAL ROLE OF MANUAL AND AUTOMATED TESTING IN AGILE....8
3. SOFTWARE TESTING STRATEGIES IN AGILE....9
3.1 Necessity for agile testing approaches....9
3.2 Agile testing....9
3.3 Proposed agile testing framework....12
3.4 Agile testing tools....16
4. DEALING UNCERTAINTY IN SOFTWARE TESTING....21
4.1 Background....21
4.2 Related Work....22
4.2.1 Uncertainty Preliminaries....22
4.2.2 Types of uncertainty....24
4.2.3 Test Execution....24
4.2.4 Modelling Uncertainty....25
4.3 Outcome....26
CONCLUSION....27
LIST OF REFERENCES....28

# INTRODUCTION

Software testing is the process of ensuring that the finished product satisfies the needs of the end user. (Anwar et al., 2019) Software testing is an essential component of Software Development Life Cycle (SDLC), as it is an ongoing activity that necessitates new approaches and inventive procedures to ensure software quality.

A quality product should be bug-free in multiple environments, meet specifications, and sustain delivery within acceptable timeframes. Software testing continually enhances quality throughout the development process and is required for all deliverables produced (Mohanty et al., 2016).

It is to be made clear that software testing and debugging are two independent stages in the software development process. Testing is done to ensure that software satisfies requirements. Debugging, on the other hand, entails identifying, isolating, and repairing code errors (Anwar et at., 2019).

The testing culture has evolved from a bug-centric approach to a more nuanced view of bugs, emphasizing the importance of identifying bugs in the same order that customers would encounter them (Elentukh, 1993). In addition, there is a paradigm shift toward agile software development models, which focus on frequent and rapid software development in response to end user dynamic needs and satisfying rapid product delivery. It does, however, limit the amount of time available for software testing, making it difficult for test managers to identify defects and issues in deliverables.

It also presents several impediments, such as short time frames for creating user-centric test environments, the need for dynamic test documentation, insufficient testing

coverage due to frequent changes in requirements and code, broken code after frequent builds, limited resources, and a lack of focused testing (Rehman et al., 2020).

(Latif Butt et al., 2017; Pathak et al., 2022) papers suggest the use of Agile testing (AST) it promotes agility in the software development due to its alignment with core principles. It ensures the delivery of working software in short iterations, facilitates continuous customer collaboration, and adapts to changing requirements. Agile testing identifies and addresses issues quickly by integrating testing early and continuously, fostering risk mitigation and supporting collaborative, cross-functional teams.

In the current context, the rapid adoption of agile methodologies has significantly emphasized automation, and many businesses are transitioning from manual to automated software testing. (Latif Butt et al., 2017) states that it may be justified by the nature of agile methodology and benefit in industrial competition, but it may not identify all issues. (Enoiu et al., 2019) research acknowledges the limitations of test design and automation, emphasizing that the decision between human and automated testing is dependent on the individual context and project objectives.

As a result, the purpose of this paper is to provide an effective and optimised agile testing practices and test case designs supported by real – world case studies using system approach and to aid in dealing with the uncertainty in software development that may be encountered during software development to foster software stability, quality, and effectiveness.

# 1. AGILE DEVELOPMENT METHODOLOGY

The Agile Model is a software development methodology that consists of a succession of stages, each of which focuses on a distinct component of the project. In the first phase, stakeholders discuss business potential, deadlines, and resource estimates to assess the system's viability. The requirements are then designed and illustrated using UFD or UML diagrams in the second step. The third phase entails creating/iterating on the product, which changes and improves with each cycle. In the fourth phase, the testing team evaluates the system's performance and reports any defects or anomalies. The last stage is deployment, during which the product is provided to end consumers.

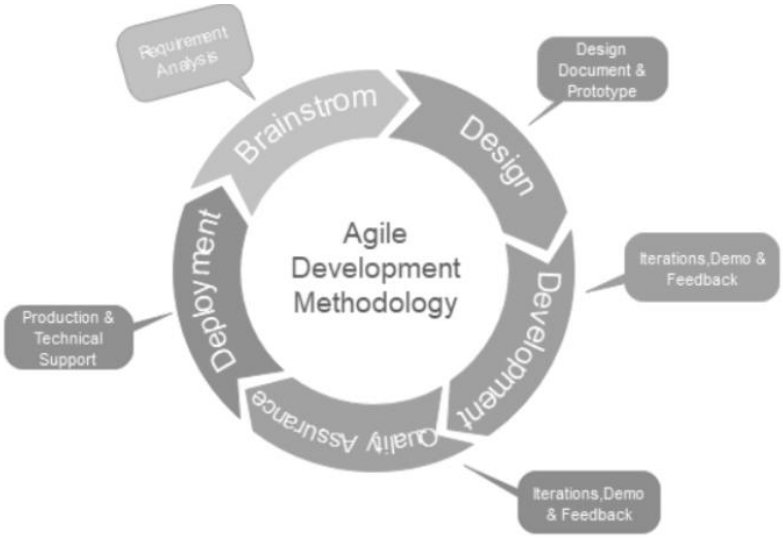


**Fig. 1: Agile development methodology** (Pathak et al., 2022)

Figure 1 shows the various phases in agile software development. Agile technique entails a thorough process of gathering requirements, developing the system, and executing it in iterative cycles known as "sprints." Engaging stakeholders to understand their requirements and expectations, as well as developing user stories, feature requests, and prioritized backlogs, are all part of this process.

# 2. DUAL ROLE OF MANUAL AND AUTOMATED TESTING IN AGILE

The function of manual and automated testing is now being debated in the software development business. Due to the obvious increasing use of Agile approaches, many people believe that manual testing is no longer necessary or reliable for ensuring product quality. It, however, is vital particularly for activities requiring critical thinking and less repetitive operations with frequently changing functionalities.

(Enoiu & et al., 2019; Halani et al., 2021) acknowledges the limitations of test design and automation, emphasizing that the decision between human and automated testing is dependent on the individual context and project objectives. (Klammer et al., 2017) The major goal is to reduce the time and effort required to run tests and freeing up testers for acceptance and regression testing.

(Ates et al., 2023) Automating all conceivable test cases on an application is not feasible. Manual and automated testing are intertwined and complement one another.

# 3. SOFTWARE TESTING STRATEGIES IN AGILE

## 3.1 Necessity for agile testing approaches

The traditional way , which lacks adequate protocols, is ineffective and may fail to meet expectations unless a structured and dynamic strategy that emphasis on adaptation, integration, feedback and productivity promoting stability, agility, and quality is introduced at the right stage of the Agile software development process (Anwar et al., 2019; Ates et al., 2023; Collins et al., 2012; Enoiu et al., 2019; Latif Butt et al., 2017; Mohanty et al., 2016; Pathak et al., 2022; Rehman et al., 2020; Sneha et al., 2017).

Agile approaches take an iterative, quick approach to software development, necessitating early software testing to avoid miscommunication, complexity, and code faults. However, testers must be proactive, beginning duties early with developers rather than waiting for system delivery at the conclusion of the project (Collins et al., 2012).

## 3.2 Agile testing

Agile testing provides high-quality software by removing faults, matching user expectations, and providing communication benefits across platforms (Latif Butt et al., 2017; Pathak et al., 2022; Rehman et al., 2020).

Agile development in software testing, however, imposes several issues namely insufficient test coverage, broken code after frequent builds, early defect identification, insufficient API testing, and a lack of focused testing due to limited resources (Latif Butt et al., 2017; Pathak et al., 2022; Rehman et al., 2020) and the possibility of

uncertainty in software testing (Mohanty et al., 2016), which the current agile testing framework struggles resulting in compromised resources and deliverables.

(Gurcan et al., 2022) study examined 14,684 software testing difficulties between 1980 and 2019, indicating 42 field-related themes centred on specification, detection, creation, assessment, and prediction. It also demonstrated a tendency toward prediction-based testing activities, with a stronger trend in the categories of security vulnerability, open source, and mobile application.

(Mohanty et al., 2016) emphasizes to create more effective testing frameworks, consider ways that use input variations to discover uncertainties. Testing is crucial for any software development, with poor test concept, the software testing might waste up to 50 to 70% of overall revenue causing project delays or cancellations (Ates et al, 2023).

As a result, a tailored testing approach in agile software development is a necessity to properly propel towards expected deliverables.

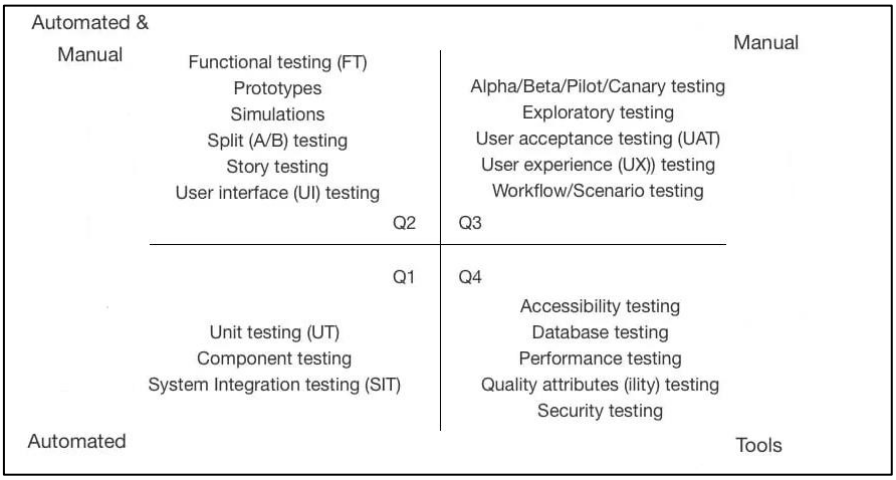


**Fig. 2: Agile testing quadrants** (Pathak et al., 2022)

Figure 2 showing testing quadrants adopted from (Brian Marick, 2003) serve as a compass for tailoring testing methodology depending on the needs.

Quadrant 1 is concerned with providing technological assistance to the team using unit and component tests. Quadrant 2 focuses on business support for the team, employing functional tests, examples, narrative tests, prototypes, and simulations. Exploratory testing, scenarios, usability testing, user acceptability testing, and alpha and beta testing are among the methodologies introduced in Quadrant 3 for analyzing the product from a commercial standpoint. Quadrant 4 examines the product from a technical standpoint, employing performance testing, load testing, security testing, and different "ility" testing types.

These quadrants direct the testing strategy and suggest if procedures are most suited for manual, automated, or a combination of the two. They offer a diverse toolbox for solving various testing requirements inside an Agile framework.

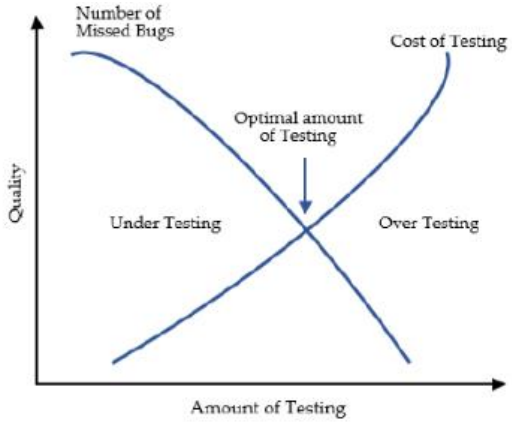


**Fig. 3: Optimal testing curve** (Anwar et al., 2019)

Figure 3 underlines the optimal approach for software testing strategy and effective resource utilization while preserving technical qualities with the testing methodologies.

## 3.3 Proposed agile testing framework

Following reviews of the research papers and case studies, the testing framework is modified to the greatest extent possible to support the agility, stability, and dependability of agile software development.

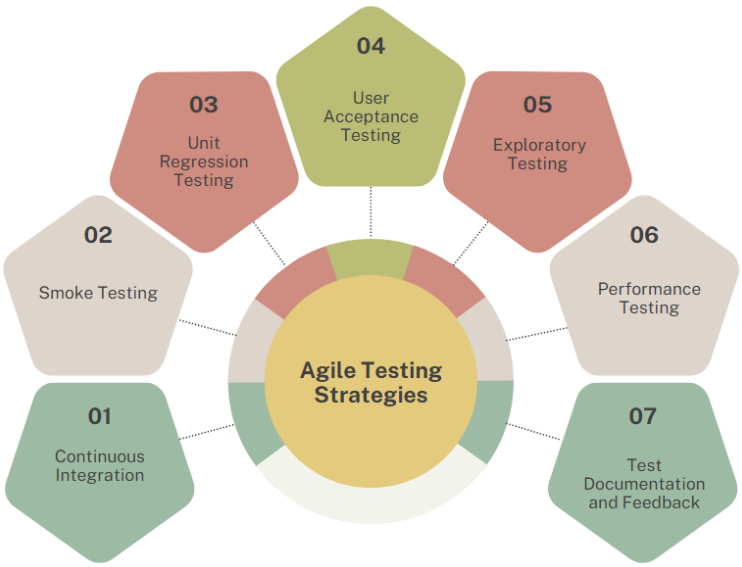


**Fig. 4: Proposed agile testing framework**

Figure 4 presents a tailored and dynamic testing framework which combines several real-world testing approaches with predictive solutions for modelling uncertainties while adhering to the system approach is crucial in agile software development to ensure optimum resource utilization and value deliverables.

1. Continuous Integration

   Code updates are streamlined using the continuous integration (CI) process, which unifies them into a single repository. By automating the validation procedure, it guarantees early testing to find problems. This procedure is essential to the software delivery pipeline because it encourages quick feedback,

makes problem solving easier, and maintains code stability throughout the project.

(Collins et al., 2012) emphasizes the importance of a continuous integration environment aiding in code compilation, unit testing, server deployment, and quick communication between developers, resulting in faster execution of acceptance tests and manual exploratory tests.

2. Smoke Testing

To make sure that application functionality is maintained after code modifications, smoke tests are crucial in the early phases of testing. They detect significant problems early, enabling prompt fixation prior to intensive testing. Software functionality and dependability are guaranteed throughout the development lifecycle by this proactive approach.

It is critical in agile development for measuring build health and conserving resources. It is preferred to be done manually in initial builds and transition is emphasized to automation after the created version is stable and most functionality stays unchanged, ensuring efficient and effective development (Latif Butt et al., 2017).

3. Unit & Regression Testing

To guarantee the quality and stability of the codebase, unit tests and regression tests are crucial components of the Continuous Integration (CI) process. Regression tests guard against unintentional side effects and functional interruptions, whereas unit tests validate specific code components. For prompt feedback and code stability maintenance, these tests are incorporated into the continuous integration pipeline.

Early defect detection is an important approach used by developers to save time during testing. This approach, which is frequently used at the module or component level (Latif Butt et al., 2017). It comes under white box testing techniques if done by the developers, although it is also termed as component testing, if used as part of UI testing. It provides flexibility to be incorporated with automated build process or CI system to ensure they are performed automatically to validate the unit's behaviour and run automatically when code is uploaded to version control, delivering rapid feedback on the health of the codebase.

Regression testing is an important technique in Agile development and optimization since it ensures code stability and uncovers faults during rapid changes. It promotes continuous integration and makes deployment easier. However, due to resource limits, it can be difficult to implement and may result in costly scheduling and poor test case prioritization (Enoiu et al., 2019). If automated, it saves time, improves test coverage, and encourages collaboration between development and testing teams. It contributes to code stability, eliminates business risks, and gradually verifies user stories.

4. User Acceptance Testing

It is an essential stage in which end users test a system to make sure it meets their needs and expectations. It incorporates previous testing stages and offers a functional assessment of the program from the perspective of the user, guaranteeing that it satisfies both technical requirements and real-world requirements.

It is an important step in software development that ensures the system can handle real-world scenarios and meet the requirements (Anwar et al., 2019). It aids in the identification of functional gaps, the mitigation of post-implementation difficulties, and the enhancement of user satisfaction.

During agile development encounters obstacles such as changing needs, time limits, and uncertainty, so some level of automation is needed without sacrificing resources (Mohanty et al., 2016).

5. Exploratory Testing

It puts the AUT through its tests by testing or investigating capabilities for which no documentation, such as test cases and scenarios, is available. It assists in identifying potential situations, documenting them, and using them for testing in the following sprint. It is frequently positioned carefully to offer a thorough investigation that makes use of prior discoveries, after organized testing.

It complements scripted testing approaches by providing extensive coverage, and it is beneficial in Agile development settings by allowing testers to explore through different features and functionalities (Dahiya et al., 2019; Sneha et al., 2017).

6. Performance Testing

It aids in making sure an application is scalable and reliable is performance testing. It is a crucial component of software development that assesses a software application's speed, responsiveness, stability, and scalability under various scenarios. Its goal is to guarantee that the application meets or exceeds performance objectives while also offering a great user experience and

optimizing for scalability and dependability (Anwar et al., 2019; Halani et al., 2021; Sneha et al., 2017).

The end objective is to identify and isolate bottlenecks such that the application meets or surpasses performance goals. It also successfully complements automation of aspects like load creation, outcomes analysis, and scripting capabilities.

7. Test Documentation

Effective test documentation adds to testing process transparency, cooperation, and repeatability. It is a great resource for both present and future testing efforts, offering information about the software's quality and the testing strategy's efficacy. It also addresses any uncertainties that may arise during software development, potentially resulting in a considerable gap between stakeholders' expectations and actual outputs (Mohanty et al., 2016).

(Collins et al., 2012; Latif Butt et al., 2017; Rehman et al., 2020) stresses on automating the test documentation in order to transition from traditional to dynamic test documentation for agile based software development.

## 3.4 Agile testing tools

A whole toolkit thoughtfully included into the Agile development process. In an Agile environment that is collaborative and iterative, these technologies are essential for automating, controlling, and optimizing a variety of testing procedures that guarantee the delivery of high-quality software.

Continuous Integration (CI)

- Initial Build
  - Selenium : Test cases may be automatically run against the most recent code built using Selenium to automate integration testing. This can assist in finding flaws early in the development process and fixing them.
- Automated Testing
  - Selenium : After the initial build, regression testing may be automated with Selenium to make sure that newly introduced code doesn't create any flaws.
- Acceptance Testing (ATDD)
  - By automating the process of developing acceptance tests based on user stories, ATDD may be utilized to guide the development process. After that, these tests may be included in the continuous integration pipeline and run alongside regression and unit tests.

Smoke Testing

- Manual Testing
  - Before deploying the program, manual testing may be done for smoke testing to rapidly verify that its most important features are operating as intended.
- Automated Testing
  - Selenium : Smoke test cases may be automatically executed using automated testing technologies like Selenium, which speeds up feedback and lowers the possibility of flaws getting into production.

Unit Testing and Regression Testing

- Unit Testing
  - Junit : To make sure that individual code units are operating properly, Junit may be used for unit testing.
- API Testing
  - Tools for API testing, such as SoapUI, Postman, and JMeter: These tools may be used to confirm that the software's APIs comply with the criteria.
- Regression Testing
  - Robot framework + Selenium: Regression testing may be automated with Robot framework to make sure that new code doesn't affect the functionality that already exists.
  - Automated testing tools: Regression testing may be automated with technologies like Selenium to make sure that new code doesn't create any flaws.

User Acceptance Testing (UAT)

- FitNesse: Fitnesse is a tool for writing and administering acceptance tests in standard English. This may facilitate the involvement of stakeholders in the testing procedure.
- Visual Examination: Ghost Inspector: This program simulates real-world use scenarios by recording and replaying genuine user interactions. It also makes sure that the software is user-friendly.

Performance Testing

- Percy: It may be used to spot visual regressions and make sure the program maintains its visual style across various devices and browsers.
- Performance Testing: Gatling, Apache JMeter, and LoadNinja, may be used to simulate a high volume of users interacting with the program to pinpoint any bottlenecks in performance.

Test Management

- Issue Tracker
  - Mantis Bug Tracker: During load testing, defects may be tracked using Mantis Bug Tracker.
- Monitoring of Deployment
  - Selenium: This tool may be used to find performance bottlenecks in software by simulating many people interacting with it.
  - JMeter: JMeter can assist in locating performance bottlenecks by simulating many users interacting with the app.
- Testing for Deployment
  - Dynatrace: Test documentation may be automatically generated from code using Dynatrace. This can automate the documentation process, which can help testers save time and effort.
- Monitoring Deployment
  - Jira: Test cases and results may be tracked using Jira. This helps guarantee that there is enough documentation of the testing procedure.

The ratings in table 1, which range from 1 to 5 and consider strengths and considerations, may change depending on the requirements and preferences.

| Tool | Category | Strengths | Considerations | Rating (Out of 5) |
|---|---|---|---|---|
| Selenium | CI, Automated Testing, Smoke | Widely used to automate integration and regression tests. | Beginners will need to work through a learning curve. | 4.5 |
| Jenkins | CI | Good support for continuous integration and build automation. | Configuration might be difficult for intricate build processes. | 4.3 |
| JUnit | Unit Testing, Regression | A robust framework for Java unit testing. | Primarily suitable for Java projects. | 4.0 |
| Postman | API Testing | An easy-to-use API testing interface. | There is little capability for automating complicated scenarios. | 4.2 |
| Robot Framework | Regression | Supports keyword-driven testing for simple test case generation. | Additional libraries may be required for some features. | 4.1 |
| FitNesse | UAT | Allows you to create acceptance tests in simple English. | Some users may find the first setup problematic. | 3.8 |
| Ghost Inspector | Visual Testing | Visual testing involves recording and replaying real-world user interactions. | There is little support for complicated scenarios and dynamic material. | 4.0 |
| Percy | Visual Testing, Performance | Identifies visual regressions and maintains consistency. | There may be expenses connected with significant usage. | 4.3 |
| Gatling | Performance Load Testing | Scala-based tool for high-scale performance testing. | The learning curve is steeper than with some other load testing tools. | 4.2 |
| LoadNinja | Performance Load Testing | A cloud-based load testing tool that allows for simple scripting. | Limited customization possibilities. | 4.1 |
| Mantis Bug Tracker | Bug Tracking | Simple and effortless bug tracking. | When compared to other bug tracking systems, it may be lacking in sophisticated functionality. | 3.7 |
| Dynatrace | Deployment Testing | Automatically creates test documentation from code. | Configuring advanced features may require a higher learning curve. | 4.0 |
| Jira | Deployment Tracking | Seamless problem tracking and project management. | Configuration might be difficult for small teams. | 4.4 |

**Table 1: Summary of agile testing tools**

# 4. DEALING UNCERTAINTY IN SOFTWARE TESTING

## 4.1 Background

(Mohanty et al., 2016) examines software testing difficulties and trends, focusing on uncertainty, and underlines the significance of evaluating deliverable quality and using test cases as triplets. It distinguishes between risk and uncertainty, employs complicated oracles in unpredictable contexts, and argues for incorporating uncertainty measures throughout the software development lifecycle, embedding solutions in models such as the V model and agile approaches.

(Gurcan et al., 2022) comprehending software testing papers from 1980 to 2019 found 42 themes, a trend of prediction-based testing arose, with an emphasis on security vulnerabilities, open source, and mobile applications. This could help improve Agile development testing techniques.

The use of predictive testing techniques is essential for enhancing testing protocols in agile settings. By doing tests that have the highest chance of failing, they save resources and enable optimised development of high-calibre software and feedback. By employing probabilistic models to identify trustworthy tests, predictive testing also tackles non-determinism of test outcomes, a prevalent problem in automated testing.

Additionally, these systems adjust to the dynamic, often-changing code environment of agile software development. Test selection criteria can be modified based on the most recent data by utilizing machine learning techniques.

## 4.2 Related Work

(Kornecki et al., 2013a) digs into the use of Bayesian Belief Networks (BBN) and their limits. Bidirectional Bayesian Networks (BBNs) are widely utilized in knowledge representation and reasoning under uncertainty. It also provides rough sets theory as an alternative way to assess data with likelihoods rather than precise probabilities, solving some of the computing issues and restrictions associated with BBNs.

(Watthayu & Peng, 2004a) examines Bayesian networks, focusing on how they are structured as directed acyclic graphs and how conditional probability tables are used to represent uncertainty. Inference in BNs with a general DAG structure is NP-hard, it also emphasizes the advancement of effective algorithms for calculating posterior probabilities, including Junction tree methods and belief propagation, as well as statistical sampling strategies for large BNs.

These studies add to our understanding of Bayesian Belief Networks (BBNs) and offer useful strategies for handling uncertainty across a range of areas and their subtle uses and constraints.

### 4.2.1 Uncertainty Preliminaries

The assessment of deliverable quality in software development is inextricably linked to successful testing methods. This necessitates the introduction of a basic concept: the test case, which is represented as a triplet [I, S, O], where I is the input data, S is the system state, and O is the desired output or result (Mohanty et al., 2016).

(Mohanty et al., 2016) A test case claims that for all inputs in I, the function f(i) produces a matching output in O.

$$\text{Test Case (I, S, O)}: \forall\, i \in I, \exists f(i) = r, \text{ where } r \in O.$$

To assess the quality of the product, a test suite is used, which includes several test cases. When predicted outputs are known, the system state is called certain if the actual output matches the expected one.

$$\text{Certainty (I, S, O)}: \exists i \in I \,\&\, o \in O, \text{ such that } f(i) = o.$$

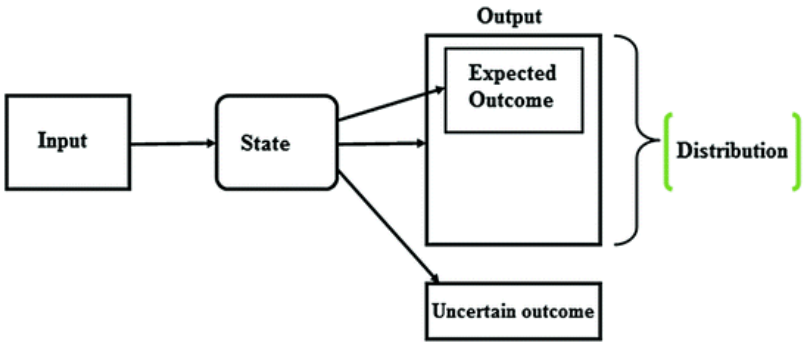


**Fig. 5: System under test** (Mohanty et al., 2016)

In contrast, risk is defined as an uncertain result with a normal distribution, indicating the likelihood of the outcome.

$$\text{Risk (I, S, O)}: [\exists i \in I, \text{ such that } f(i) = r, \text{ where } r \in O].$$

Uncertainty differs from risk, highlighting circumstances when neither the result nor the distribution is known.

$$\text{Uncertainty (I, S, O)}: [\exists i \in I, \text{ such that } f(i) = p, \text{ where } p \notin O]$$

The amount of information regarding unwanted occurrences distinguishes risk from uncertainty. (Troya et al., 2021) emphasize this distinction, formalizing uncertainty by assuming an identical link between reality (set A) and the developed model (set B). Any discrepancy in results between these sets suggests problems with the application or system development, emphasizing the need of exploring uncertainty in software development to produce high-quality software.

### 4.2.2 Types of uncertainty

(Mohanty et al., 2016) describes that there are two main kinds of uncertainty in software development. The first kind is centered on the guarantee that the description given correctly represents the program specification; this serves as the foundation for requirement validation. Time and expense overruns can occur when code is generated without the necessary requirements. The duration of valid specifications is connected to the second category.

Inadequate comprehension of the issue area and disregard for software quality and user requirements can lead to uncertainty. Contrasting artifact quality during software development can lead to ambiguity, and the end user wants a high-quality output that satisfies their requirements (Mohanty et al., 2016; Troya et al., 2021).

### 4.2.3 Test Execution

(Mohanty et al., 2016) emphasizes how crucial it is to use sophisticated oracles while attempting to successfully manage and govern uncertainty. It recognizes the difficulty in precisely predicting test case results in the face of uncertainty and cautions against too relaxing the test oracle distribution as this might make problem identification more difficult.

Regression testing places emphasis on the accuracy of test oracles obtained from formal requirements. The input set for testing is influenced by the introduction of two system types: sequential and reactive (Kornecki et al., 2013a; Mohanty et al., 2016).

One of the main challenges in testing is error detection, which calls for efficient mistake tracing via networked software artifacts.

### 4.2.4 Modelling Uncertainty

The probability connection between random variables is represented by a Bayesian network. Each edge in a DAG contains a probability matrix that illustrates the cause-and-effect connection. This implies that the likelihood of one variable is affected by the value of another (cause) (effect) as shown in figure 6.

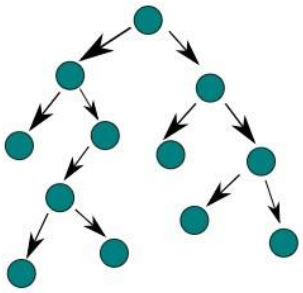

**Fig. 6: Bayesian Network**

Based on given constraints, multi-objective functions optimize incompatible test objectives. The objective is to choose a chosen set of test cases to arrive at a workable response. A multiple-purpose function with fuzzy logic and weight assignment is used to create test cases (Kornecki et al., 2013b; Mohanty et al., 2016).

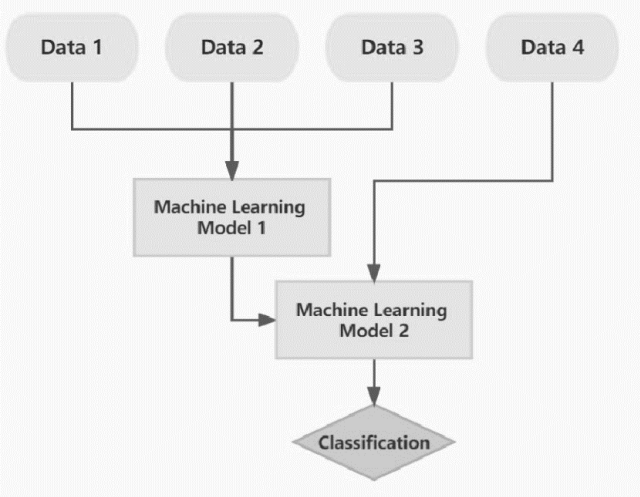


**Fig. 7: Fuzzy Logic**

Figure 7 demonstrates the general fuzzy logic structure. The fuzzy feature weighting method considers a variety of factors, including test case repository size, frequency of parameters in each module, weight of fitness value, and total amount of parameters.

## 4.3 Outcome

Metrics that are suggested at every level, from requirements to testing, assess completeness and clarity while providing information about possible uncertainties. Propagation measurements monitor changes in uncertainty during development. It is recommended to discover uncertainty using efficient testing frameworks, such as dynamic test oracles and automated scripts.(Mohanty et al., 2016).

(Kornecki et al., 2013b; Mohanty et al., 2016; Watthayu & Peng, 2004b) emphasizes the need for general principles and case studies, and calls for the development of machine learning frameworks for managing uncertainty. It also emphasizes the flexibility of adaptive closed-loop systems, which exhibit continuous monitoring and modification.

# CONCLUSION

This research paper investigates the changing landscape of Agile testing, regression testing, and automation technologies, highlighting the role of testing in enhancing software quality across the Agile Software Development Life Cycle (SDLC). The study investigates the difficulties caused by dynamic settings, acceptance-driven development, and time restrictions in Agile contexts. It emphasizes the mutually beneficial interaction between human and automated testing and offers insights into adaptable strategies for success in the Agile software development lifecycle.

The study also sheds light about uncertainty and possible countermeasures in software development. The study also uses a variety of testing methods to give practical insights into enhancing testing procedures.

This literature review is useful for testing specialists, software developers, and project managers as it guides attempts to improve testing techniques in the ever-changing Agile development environment.

# LIST OF REFERENCES


Anwar, N., & Kar, S. (2019). *Review Paper on Various Software Testing Techniques & Strategies*.

Ates, Z., & Sencan, Y. E. (2023). A Case Study to Increase Quality of Industrial Edge Software Product's Dynamic Data Testing. *Proceedings - 2023 International Conference on Software and System Engineering, ICoSSE 2023*, 25–29. https://doi.org/10.1109/ICoSSE58936.2023.00013

Collins, E. F., & Lucena Jr., V. F. (2012). *Software Test Automation Practices in Agile Development Environment: An Industry Experience Report* (E. F. Collins & V. F. de Lucena Jr., Eds.). IEEE.

Dahiya, R., & Ali, S. (2019). *Importance of Manual and Automation Testing*. https://doi.org/10.5121/csit.2019.91719

Elentukh, A. (1993). *IEEE Computer Society, Test Technology Technical Committee.* (A. Elentukh, Ed.). The Conference.

Enoiu, E., & Mirgita Frasheri. (2019). Test Agents : The next generation of test cases. *Proceedings - 2019 IEEE 12th International Conference on Software Testing, Verification and Validation Workshops, ICSTW 2019*, 305–308. https://doi.org/10.1109/ICSTW.2019.00070

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://doi.org/10.1109/ACCESS.2022.3211949

Halani, K. R., Kavita, & Saxena, R. (2021). Critical Analysis of Manual Versus Automation Testing. *2021 International Conference on Computational Performance Evaluation, ComPE 2021*, 132–135. https://doi.org/10.1109/ComPE53109.2021.9752388

Klammer, C., & Ramler, R. (2017). A Journey from Manual Testing to Automated Test Generation in an Industry Project. *Proceedings - 2017 IEEE International Conference on Software Quality, Reliability and Security Companion, QRS-C 2017*, 591–592. https://doi.org/10.1109/QRS-C.2017.108

Kornecki, A. J., Wierzchon, S. T., & Zalewski, J. (2013a). REASONING UNDER UNCERTAINTY WITH BAYESIAN BELIEF NETWORKS ENHANCED

WITH ROUGH SETS. In *Janusz Zalewski / Computing* (Vol. 12). http://faculty.erau.edu/korn/http://www.ipipan.waw.pl/~stw/http://www.fgcu.edu/zalewski/

Latif Butt, F., Bhatti, S. N., Sarwar, S., Jadi, A. M., & Saboor, A. (2017). Optimized Order of Software Testing Techniques in Agile Process-A Systematic Approach. In *IJACSA) International Journal of Advanced Computer Science and Applications* (Vol. 8, Issue 1). www.ijacsa.thesai.org

Mohanty, H., Mohanty, J. R., & Arunkumar Balakrishnan. (2016). Trends in software testing. In S. A. Moiz (Ed.), *Trends in Software Testing*. Springer Singapore. https://doi.org/10.1007/978-981-10-1415-4

Pathak, K., Ninoria, S., & Bharadwaj, S. (2022). Scope of Agile Approach for Software Testing Process. *Proceedings of the 2022 11th International Conference on System Modeling and Advancement in Research Trends, SMART 2022*, 1079–1083. https://doi.org/10.1109/SMART55829.2022.10047649

Rehman, A. U., Nawaz, A., Ali, M. T., & Abbas, M. (2020, December 16). A Comparative Study of Agile Methods, Testing Challenges, Solutions Tool Support. *2020 14th International Conference on Open Source Systems and Technologies, ICOSST 2020 - Proceedings*. https://doi.org/10.1109/ICOSST51357.2020.9332965

Sneha, K., & Gowda M, M. (2017). *Research on Software Testing Techniques and Software Automation Testing Tools*. IEEE.

Troya, J., Moreno, N., Bertoa, M. F., & Vallecillo, A. (2021). Uncertainty representation in software models: a survey. *Software and Systems Modeling*, *20*(4), 1183–1213. https://doi.org/10.1007/s10270-020-00842-1

Watthayu, W., & Peng, Y. (2004a). *A BAYESIAN NETWORK BASED FRAMEWORK FOR MULTI-CRITERIA DECISION MAKING*.