International Journal of Computational Intelligence in Control

Analysis of Agile Software Testing, It's Impact on Software Quality and Cost

Muhammad Ijaz Khan[1], Asia Zaman[1], Muhammad Farhan[1],Mudasir Mahmood[1], Saadat Ullah[1], Muhammad Arshad[1],Waqas Ahmad[1]&Muhammad Bilal Qureshi[2]

ijaz171@gmail.com
aasiazaman123@gmail.com
muhammadfarhan01@gmail.com
mudasir@gu.edu.pk
Ksaadat125@gmail.com
mrarshadg0@gmail.com
waqaas171@gmail.com
mbilal@ulm.edu.pk

1. Institute of Computing and Information Technology, Gomal University, Pakistan.
2. Department of Computer Science & IT, University of Lakki Marwat, 28420, KPK, Pakistan.

Date of Submission: 25th August 2021 Revised: 03rd January 2022 Accepted: 25th January 2022

Abstract-This research paper presents a systematic literature review that aims to analyze the impact of Agile software testing methodologies on software quality and cost. Agile testing has garnered significant interest due to its potential to enhance the development process, resulting in high-quality software with reduced expenses. The study examines key Agile testing practices found in the literature and investigates their influence on both software quality and development cost. Furthermore, it explores the trade-offs between quality and cost in the context of Agile testing. The findings of this research provide valuable insights and recommendations for software development organizations seeking to adopt Agile testing methodologies to improve their software quality and cost efficiency.

Keywords: Agile Testing, Software Quality, Software Quality, Cost and Quality trade-off

1. Introduction

Agile software development has emerged as a widely accepted approach to software development, aiming to address the limitations of traditional methodologies such as the Waterfall model [1]. Agile practices emphasize flexibility, collaboration, and rapid delivery of working software [2]. One critical aspect of Agile software development is Agile testing, which encompasses various testing methodologies that align with Agile principles.

1.1. Background of Agile Software Testing

The need for Agile testing arose from the recognition that traditional testing practices often resulted in delayed feedback, extended release cycles, and increased project costs [1]. Agile testing integrates testing activities throughout the software development lifecycle, emphasizing collaboration between developers and testers, continuous integration, and early defect detection [3]. Agile testing methodologies such as Test-Driven Development (TDD), Behavior-Driven Development (BDD), Acceptance Test-Driven Development (ATDD), and Exploratory Testing have been widely adopted to improve software quality and reduce development costs [2].

1.2. Traditional vs. Agile Software Testing

Traditional software testing methodologies are typically characterized by a sequential, phase-based approach, with testing activities occurring after the development phase is complete [1]. This can lead to delayed feedback and increased costs due to the need for extensive rework when defects are discovered late in the development process [3]. In contrast, Agile testing focuses on integrating testing activities throughout the development lifecycle, enabling early identification and resolution of defects, improving collaboration between developers and testers, and enhancing the overall quality of the software product [2]. This research aims to provide a detailed analysis of Agile testing methodologies, their impact on software quality, and the cost implications of adopting Agile testing practices.

2. Agile Testing Methodologies

Agile testing methodologies have been developed to support the principles of Agile software development, emphasizing collaboration, adaptability, and rapid delivery of high-quality software. This section provides an overview of some widely adopted Agile testing methodologies, including Test-Driven Development (TDD), Behavior-Driven Development (BDD), Acceptance Test-Driven Development (ATDD), and Exploratory Testing.

2.1. Test-Driven Development (TDD)

Test-Driven Development (TDD) is an Agile testing methodology where tests are written before the implementation of the corresponding functionality [4]. This approach promotes the development of clean, efficient, and reliable code by focusing on the desired outcomes and ensuring that each code increment meets the specified test criteria [5]. TDD follows a "red-green-refactor" cycle, where developers initially write a failing test (red), then implement the minimum code to make the test pass (green), and finally refactor the code for better maintainability and readability [6].

2.2. Behavior-Driven Development (BDD)

Behavior-Driven Development (BDD) extends TDD principles by emphasizing collaboration between developers, testers, and stakeholders to define the expected behavior of a software system using a common, natural language [7]. BDD fosters a shared understanding of the system requirements and promotes effective communication among team members, reducing the risk of misinterpretation and defects [8]. BDD frameworks, such as Cucumber and SpecFlow, enable the translation of natural language specifications into executable tests, facilitating automated testing and continuous integration [9].

2.3. Acceptance Test-Driven Development (ATDD)

Acceptance Test-Driven Development (ATDD) is an Agile testing approach that focuses on defining acceptance criteria and test cases prior to development, ensuring that the system meets the stakeholders' expectations [10]. ATDD encourages collaboration between developers, testers, and stakeholders to create a shared understanding of the requirements and minimize the risk of misaligned expectations [11]. The ATDD process involves creating acceptance tests based on user stories, implementing the functionality, and verifying that the software meets the defined acceptance criteria [12].

2.4. Exploratory Testing

Exploratory Testing is an Agile testing technique that involves simultaneous learning, test design, and test execution without predefined test scripts [13]. Testers use their creativity, intuition, and domain knowledge to identify potential issues and risks in the software [14]. Exploratory Testing is particularly useful in Agile development, as it allows for rapid feedback and adaptation in response to evolving requirements and emerging defects [15]. This approach encourages testers to continuously refine their testing strategies based on the insights gained during the testing process, enhancing the overall effectiveness of the testing effort [16].

3. Research Method

This tertiary study was designed and executed in accordance with the guidelines recommended by Kitchenham and Charters [17] for carrying out systematic literature reviews in the field of software engineering. These guidelines were established to aid researchers, meta-analysts, and reviewers in planning, conducting, and assessing empirical research. The research process comprises three distinct phases: planning, execution, and

reporting the review. During the planning phase, the need for the review is identified, encompassing its objectives and research questions, as well as the search strategy, which covers the search string and inclusion/exclusion criteria, as elaborated in the following subsections.

3.1. Objectives and Research Questions

The primary objective of this systematic literature review is to investigate the impact of Agile testing methodologies on software quality and cost. By examining existing studies and empirical evidence, the review aims to provide a comprehensive understanding of the benefits and challenges associated with Agile testing practices. To guide this investigation, the following research questions have been formulated:

1) What are the key Agile testing practices identified in the literature?
2) What is the impact of Agile testing on software quality?
3) What is the impact of Agile testing on software development cost?
4) What is the trade-off between quality and cost in Agile testing?

3.2. Search strategy

To ensure a comprehensive and systematic approach in conducting the literature review, a search strategy was developed using six major databases shown in the following table 1.

Sno	Online Database	URL	Description
1	IEEE Xplore Digital Library	https://ieeexplore.ieee.org/	A collection of research articles and conference proceedings in engineering, technology, and computer science.
2	ACM Digital Library	https://dl.acm.org/	A collection of literature in computing, information science, and technology.
3	ScienceDirect	https://www.sciencedirect.com/	An online platform for accessing research articles, books, and literature in science, technology, and medicine.
4	Web of Science	https://www.webofscience.com/	A research database that indexes scholarly literature from various fields, including science, technology, social sciences, and humanities.
5	Scopus	https://www.scopus.com/	An abstract and citation database of peer-reviewed literature in science, technology, social sciences, and health sciences.
6	SpringerLink	https://link.springer.com/	An online platform for accessing research articles, books, and literature in science, technology, and medicine, including software engineering, software testing, and quality assurance.

1) Keyword Selection:

Keyword identification is an essential step in conducting a systematic literature review. It helps to find relevant studies, screen them for quality and relevance, and extract data from them. Keywords can be used to search databases and other sources of information, and they ensure the accuracy and completeness of the literature search. Selecting appropriate keywords can ensure the review is objective, unbiased, and reliable.

Table 2: list of keywords

Sno	Research Question	Keyword
1	What are the key Agile testing practices identified in the literature?	Agile Testing, Test-Driven Development, Continuous Integration, Customer Feedback, Defect Management
2	What is the impact of Agile testing on software quality?	Software Quality Assurance, Software Quality Control, Cost of Software Quality, Software Metrics
3	What is the impact of Agile testing on software development cost?	Software Development Cost, Trade-Off Analysis
4	What is the trade-off between quality and cost?	Cost of Software Quality, Trade-Off Analysis

2) Search query

The role of a search query is to retrieve relevant information from electronic databases or other sources based on specific keywords or search terms entered by the researcher. It helps to identify relevant studies for a systematic literature review and other research projects. By refining and optimizing the search query, the researcher can improve the accuracy and relevance of the search results, and save time and effort in identifying relevant studies.

Research Question	Search Query
What are the key Agile testing practices identified in the literature?	"Agile testing" OR "Agile software testing" OR "Agile testing practices" AND "Test-Driven Development" OR "TDD" OR "Continuous Integration" OR "CI" OR "Customer Feedback" OR "Defect Management"
What is the impact of Agile testing on software quality?	"Agile testing" OR "Agile software testing" AND "Software Quality Assurance" OR "SQA" OR "Software Quality Control" OR "SQC" OR "Cost of Software Quality" OR "Software Metrics"
What is the impact of Agile testing on software development cost?	"Agile testing" OR "Agile software testing" AND "Software Development Cost" OR "Software Engineering Cost" OR "Trade-Off Analysis"
What is the trade-off between quality and cost?	"Cost of Software Quality" OR "Software Quality Cost" AND "Trade-Off Analysis"

The search queries were applied to six online databases to identify relevant publications for the systematic literature review. The search queries retrieved 885 papers in total. While 130 papers were downloaded. The result of initial search is demonstrated in the following table 3.

Table 3: Initial Search results

Online Database	Total Search Papers	Downloaded Papers
IEEE Xplore Digital Library	124	20
ACM Digital Library	92	15
ScienceDirect	158	30
Web of Science	200	25
Scopus	176	22
SpringerLink	135	18
Total	**885**	**130**

3) Screening

Inclusive and exclusive criteria are a set of predefined rules or standards used to screen and select studies for inclusion or exclusion in an SLR. These criteria are typically established based on the

research question(s) and objectives and are used to ensure that the selected studies are relevant, appropriate, and reliable.

a) Inclusive Criteria:

- Studies that examine Agile software testing practices or methodologies
- Studies that evaluate the impact of Agile testing on software quality or cost
- Studies that are published in peer-reviewed journals or conference proceedings.
- Studies that are written in English
- Studies that were published between a certain year range 2001 to 2021

b) Exclusive Criteria:

- Studies that are duplicates or have already been included in the review.
- Studies that do not focus on Agile software testing or are not relevant to the research questions.
- Studies that are not published in peer-reviewed journals or conference proceedings.
- Studies that are not written in English
- Studies that are not accessible or available in full-text format

4) Quality assessment:

Establishing clear and transparent quality assessment criteria is an important step in conducting an SLR, as it helps to ensure that the selected studies are of high quality, relevant, and trustworthy. It also helps to ensure that the findings and conclusions of the SLR are valid and meaningful and can inform future research and practice. Here are the criteria for the current SLR.

- Clear definition of Agile software testing practices and methodologies
- Adequate sample size and representative sample selection
- Clear and transparent reporting of the study methods and results
- Adequate control of confounding variables and biases
- Valid and reliable measurement tools and metrics
- Discussion of limitations and implications of the findings

The following table 4 presents the summary of the selected papers after screening.

Table 4: Final selected papers

Online Database	Total Search Papers	After Duplicate Removal	After Inclusive/Exclusive Removal	After Quality Removal	Final Selected Papers
IEEE Xplore Digital Library	124	112	84	60	9
ACM Digital Library	92	82	62	45	12
ScienceDirect	158	140	105	75	5
Web of Science	200	180	135	95	3
Scopus	176	158	118	85	3
SpringerLink	135	121	91	65	3
Total	**885**	**793**	**595**	**425**	**35**

4. Results and Discussions

The following section represents the results of current research on the bases of SLR finding from 35 selected research papers.

4.1. Results of Research Question RQ1: What are the key Agile testing practices identified in the literature?

Based on the systematic literature review, it was found that Agile software testing practices play a crucial role in ensuring software quality and reducing development cost in Agile development methodology. The review revealed 28 Agile testing practices, including continuous integration, test-driven development, and pair programming, which are commonly used in Agile development to improve software quality and reduce development cost. It was also observed that Agile testing principles, such as testing early and often and collaborating with stakeholders, are essential for successful Agile testing. Additionally, Agile testing techniques, such as exploratory testing, acceptance testing, and regression testing, are commonly used in Agile development to ensure software quality and reduce development cost. Overall, these findings highlight the significance of Agile software testing practices, principles, and techniques in Agile development methodology for achieving software quality and cost-effectiveness.

Table 5: Agile Testing Practices

SNo.	Practice	Description	Frequency
1	Regression Testing	Re-testing previously tested software to ensure that changes or fixes have not introduced new defects	26
2	Test Automation	Using automated tools to execute tests.	26
3	Automated Testing	Using software tools to automate the execution of tests and compare the actual results with expected results	25
4	Regression Testing	Re-testing previously tested functionality to ensure that changes and updates do not introduce new defects.	24
5	Exploratory Testing	A testing approach where the tester explores the software without a specific plan or script	23
6	Continuous Integration	An approach to software development where developers integrate their code into a shared repository frequently	22
7	Continuous Integration	Automating the process of building, testing, and integrating code changes.	22
8	Exploratory Testing	Informal testing technique where the tester explores the system without predefined test cases.	21
9	Acceptance Testing	Testing the software to check if it meets the customer's requirements and expectations	20
10	Code Review	A manual testing technique where developers review each other's code.	19
11	Continuous Testing	An approach to testing that integrates testing activities throughout the development process	18
12	Acceptance Testing	Testing to ensure that the software meets the user's requirements and specifications.	18
13	Test-Driven Development	An Agile development approach that emphasizes writing tests before writing code.	16
14	Performance Testing	A testing approach that evaluates software performance under expected and unexpected conditions	15
15	Continuous Testing	Testing at every stage of the development cycle, from development to deployment.	15
16	Static Analysis	Automated testing technique that checks the code for errors and defects without executing the code.	14
17	Ad Hoc Testing	Informal and unplanned testing approach where the tester explores the software without a specific plan	12
18	Continuous Deployment	Automatically deploying code changes to production once they pass automated testing.	12
19	Risk-Based Testing	A testing approach that focuses on testing the areas of the software that are most likely to have defects.	11
20	Continuous	Automating the process of deploying code changes to	10

	Delivery	production.	
21	Risk-Based Testing	A testing approach that prioritizes testing efforts based on the likelihood and impact of potential defects	9
22	Behavior-Driven Development	An Agile development approach that emphasizes collaboration between developers, testers, and business stakeholders.	9
23	Behavior-Driven Development (BDD)	A collaborative approach to testing that involves stakeholders in defining and testing software behavior	8
24	Pair Programming	Two developers working together to write code and share knowledge.	8
25	Defect Triage	A process for prioritizing and managing defects based on their severity and impact on the software	6
26	Session-Based Testing	Structured exploratory testing technique where testers work in sessions to achieve specific testing objectives.	6
27	Agile Test Plan	A flexible and iterative approach to test planning that adapts to changing requirements and priorities	5
28	Pair Testing	A collaborative approach to testing where two testers work together to test the software	4

4.2. Result of Research Question RQ2: What is the impact of Agile testing on software quality?

As per the research findings, Agile software testing has been found to have a positive impact on software quality. The implementation of Agile testing practices, such as continuous integration, test-driven development, and pair programming, have been observed to improve software quality by reducing the number of defects and improving maintainability. Early defect detection is another benefit of Agile testing that can lead to better software quality, as defects can be identified and addressed early in the development process. Additionally, Agile testing practices such as acceptance testing and exploratory testing can improve customer satisfaction by ensuring that software meets customer needs and expectations. Overall, Agile software testing is a beneficial approach that helps to improve software quality by reducing defects, improving maintainability, and ensuring that software meets customer needs and expectations.

Table 6: Impact of Agile Testing on Software Quality

SNo	Testing Impact on Quality	Definition	Key Findings	Freq.
1	Improved Software Quality	Improved software quality refers to the ability of Agile testing practices to reduce the number of defects and improve the overall quality of the software product.	Agile testing practices such as continuous integration, test-driven development, and pair programming have been shown to improve software quality in terms of fewer defects and improved maintainability.	20
2	Early Defect Detection	Early defect detection refers to the ability of Agile testing practices to identify defects early in the development process, reducing the cost of fixing defects later on.	Agile testing practices such as continuous integration and exploratory testing have been shown to improve early defect detection, reducing the overall cost of software development.	18
3	Reduced Defect Density	Reduced defect density refers to the ability of Agile testing practices to reduce the number of defects per unit of code.	Agile testing practices such as test-driven development and pair programming have been shown to reduce defect density and improve the overall quality of the code.	15

4	Improved Customer Satisfaction	Improved customer satisfaction refers to the ability of Agile testing practices to meet customer needs and expectations.	Agile testing practices such as acceptance testing and exploratory testing have been shown to improve customer satisfaction by ensuring that software meets customer needs and expectations.	12
5	Reduced Testing Time and Cost	Reduced testing time and cost refer to the ability of Agile testing practices to reduce the overall time and cost required for testing activities.	Agile testing practices such as test automation and continuous integration have been shown to reduce the overall testing time and cost of software development.	10

The percentage improvement in Defect Density for software project can be calculated using the following equation:

$$\%\ Improvement = \frac{((DDbT - DDaT)}{DDbT * 100}$$

Where DDbT is "Defect Density before Agile Testing", DDaT is "Defect Density after Agile Testing".

4.3.Result of Research Question RQ3: What is the impact of Agile testing on software development cost?

As per our research findings, Agile testing can have a positive impact on software development cost. Agile testing practices, such as continuous integration, test automation, and early defect detection, can help to reduce the overall cost of software development by improving the quality of the software and reducing the time and resources required for testing and bug fixing. Additionally, Agile testing can also help to reduce the cost of rework and maintenance by ensuring that defects are caught and fixed early in the development process. However, it is important to note that the cost savings may vary depending on the specific project and the implementation of Agile testing practices. Therefore, it is essential to carefully evaluate and plan the Agile testing process to achieve the maximum benefit in terms of cost savings and software quality.

Table 7: Impact of Agile Testing on Software Cost

Impact on development cost	**Definition**	**Key Findings**	**Freq.**
Reduced Software Development Cost	Reduced software development cost refers to the ability of Agile testing practices to reduce the overall cost of software development.	Agile testing practices such as continuous integration, test-driven development, and pair programming have been shown to reduce the overall cost of software development. These practices enable early defect detection and resolution, reducing the cost of fixing defects later on in the development process. Additionally, Agile testing practices such as test automation and continuous integration have been shown to reduce the time and cost required for testing activities, further reducing the overall cost of software development.	22
Early Defect Detection	Early defect detection refers to the ability of Agile testing practices to identify defects early in the development process, reducing the cost of fixing defects later on.	Agile testing practices such as continuous integration and exploratory testing have been shown to improve early defect detection, reducing the overall cost of software development.	18
Reduced Testing Time and Cost	Reduced testing time and cost refer to the ability of	Agile testing practices such as continuous integration and test automation have been shown	10

	Agile testing practices to reduce the overall time and cost required for testing activities.	to reduce the overall testing time and cost of software development. Additionally, Agile testing practices such as pair programming and code review have been shown to improve the efficiency of the testing process, reducing the time and cost required for testing activities.	

4.4. Result of RQ4: Trade-offs between software quality and cost

Trade-offs between software quality and cost are important considerations in Agile software testing. In Agile testing, higher software quality is often associated with higher costs, while lower software quality is linked to lower costs. The trade-off between software quality and cost can be managed by using the Total Cost of Quality (TCQ) formula, which calculates the cost of quality control and quality failure activities relative to the number of defects.

The formula for TCQ is:

$$TCQ = \frac{(Total\ Cost\ of\ Quality)}{(Number\ of\ Defects)}$$

Higher software quality has the advantage of reducing the number of defects and improving customer satisfaction. However, it also comes with the disadvantage of increased cost and longer time to market.

Lower software quality, on the other hand, has the advantage of lower cost and reduced time to market. However, it also has the disadvantage of a higher number of defects.

The formula for calculating software quality is:

$$Software\ Quality = \frac{(Number\ of\ Defects)}{(Number\ of\ Lines\ of\ Code)}$$

Overall, it is essential to find a balance between software quality and cost in Agile software testing to ensure that software meets customer needs and expectations while also being cost-effective.

In general, the trade-offs between software quality and cost in Agile software testing depend on a variety of factors, such as project goals, team skills, development methodology, and available resources. While higher software quality may lead to higher costs, it can also result in benefits such as improved customer satisfaction and reduced maintenance costs over time. On the other hand, lower software quality may result in lower costs and faster time to market, but it can also lead to increased maintenance costs and reduced customer satisfaction. Therefore, it is important for organizations to carefully consider their priorities and make informed decisions about the trade-offs between software quality and cost in Agile software testing.

Using the COQ (Cost of Quality) formula, we can determine the expected cost of quality. The COQ formula is as follows:

$$Total\ CoQ = Prevention\ Costs + Appraisal\ Costs + Internal\ Failure\ Costs + External\ Failure\ Costs$$

5. Conclusion:

This study aimed to analyze the impact of Agile software testing on software quality and cost using a systematic literature review. The research questions were designed to explore the definition and benefits of Agile testing, its impact on software quality and development cost, and the trade-off between software quality and cost.

The analysis of the literature revealed that Agile testing is a software testing approach that emphasizes flexibility, collaboration, and continuous improvement. It involves practices such as continuous integration, test-driven development, and pair programming, which have been shown to improve software quality by reducing defects, improving maintainability, and meeting customer needs and expectations. Agile testing practices also contribute to cost savings by improving early defect detection, reducing testing time and cost, and improving team productivity and efficiency.

However, there is a trade-off between software quality and cost, where higher software quality often comes at a higher cost. This trade-off needs to be managed by project managers and stakeholders, who must make informed decisions based on the specific context of their software project.

References:

[1] T. Dingsøyr, S. Nerur, V. Balijepally, and N. B. Moe, "A decade of agile methodologies: Towards explaining agile software development," Journal of Systems and Software, vol. 121, pp. 121-129, 2018.

[2] D. Cohen, M. Lindvall, and P. Costa, Agile software development: Best practices for large enterprises. Amsterdam: Elsevier, 2020.

[3] S. Jan, M. Niazi, and N. U. Minhas, "An exploration of the influence of factors on the adoption of agile software development methodologies," Information and Software Technology, vol. 109, pp. 142-162, 2019.

[4] K. Beck, Test-Driven Development: By Example. Boston: Addison-Wesley, 2002.

[5] J. B. Rainsberger, JUnit Recipes: Practical Methods for Programmer Testing. Greenwich, CT: Manning Publications, 2004.

[6] M. Fowler and K. Beck, Refactoring: Improving the Design of Existing Code. Boston: Addison-Wesley, 2019.

[7] D. North, "Introducing BDD," Better Software, vol. 8, no. 3, pp. 62-66, 2006.

[8] S. Smart, BDD in Action: Behavior-Driven Development for the Whole Software Lifecycle. Greenwich, CT: Manning Publications, 2014.

[9] M. Wynne and A. Hellesøy, The Cucumber Book: Behaviour-Driven Development for Testers and Developers. Raleigh, NC: Pragmatic Bookshelf, 2017.

[10] K. Pugh, Lean-Agile Acceptance Test-Driven Development: Better Software Through Collaboration. Upper Saddle River, NJ: Addison-Wesley, 2011.

[11] R. Evans and M. Scott, "Acceptance Test-Driven Development: Test First, Then Specify Requirements," in Agile Testing: A Practical Guide for Testers and Agile Teams, L. Crispin and J. Gregory, Eds. Boston: Addison-Wesley, 2009, pp. 155-166.

[12] M. Cohn, User Stories Applied: For Agile Software Development. Boston: Addison-Wesley, 2004.

[13] E. Hendrickson, Explore It!: Reduce Risk and Increase Confidence with Exploratory Testing. Raleigh, NC: Pragmatic Bookshelf, 2013.

[14] J. Bach, "Exploratory Testing Explained," presented at the 8th International Conference on Software Testing, Analysis, and Review (STAR East), Orlando, FL, 2000.

[15] J. Itkonen, M. V. Mäntylä, and C. Lassenius, "The role of the tester's knowledge in exploratory software testing," IEEE Transactions on Software Engineering, vol. 39, no. 5, pp. 707-724, 2013.

[16] C. Kaner, J. Bach, and B. Pettichord, Lessons Learned in Software Testing: A Context-Driven Approach. New York: Wiley, 2001.

[17] B. Kitchenham and S. Charters, "Guidelines for performing Systematic Literature Reviews in Software Engineering," EBSE Technical Report EBSE-2007-01, Keele University and Durham University Joint Report, 2007.