Shift-Left Testing: Integrating Testing Early in the Development Cycle

Dheer Maheshwari, Jaimin Gulale
22bec033, 22bec044
22bec033@nirmauni.ac.in, 22bec044@nirmauni.ac.in

Abstract—**In response to the growing complexity and pace of software development, shift-left testing has emerged as an innovative strategy to improve quality and efficiency. Traditionally, software testing is viewed as a terminal phase in the development lifecycle, often resulting in costly late-stage defect corrections. Shift-left testing, however, promotes integrating testing from the earliest stages of development, allowing for timely defect identification and mitigation. This paper explores the shift-left testing paradigm, examining its benefits, challenges, implementation strategies, and case studies to illustrate its transformative impact on modern software development.**

I. Introduction

In today's fast-paced world, the demand for software that's both fast and reliable is higher than ever. Users expect apps and systems to work seamlessly, and any glitches or delays can quickly lead to frustration. For developers, this means that the way they test software is more important than ever.

In traditional software development, testing is often done toward the end of the process, a practice known as "shift-right" testing. Imagine it like this: you're building a car, but you only check if everything works once the whole car is assembled. If something's wrong—maybe the engine doesn't start or the brakes don't work—you'd need to take things apart and fix them. This approach can be really costly, not to mention time-consuming.

That's where the "shift-left" testing approach comes in. The idea here is to move testing to the "left" on the development timeline, meaning it happens as early as possible. Instead of waiting until the end, developers test each part of the software as they build it. This way, they can catch and fix issues right away, before they turn into bigger problems down the road. It's like testing each component of the car—like the engine or the brakes—before it's fully assembled, so you know everything works as you go.

Shift-left testing fits perfectly with modern development practices, especially in agile and DevOps environments. These approaches focus on building software in smaller, iterative steps and constantly delivering updates. Shift-left testing means that testing is embedded in every stage of this process, from planning and designing to coding and integrating new features.

In this paper, we'll dive into the details of shift-left testing—its guiding principles, the benefits it brings, and the challenges teams might face. We'll also cover some best practices and real-world examples to show how this approach can make software development faster, more efficient, and ultimately, lead to better quality products that meet user expectations.

II. Related Work

Testing has traditionally occurred at the conclusion of the development process, but this practice is changing. The shift-left approach in software testing has been a well-established strategy for the early detection and rectification of flaws in the development process. This approach involves integrating testing practices into the early stages of the software development lifecycle (SDLC). Continuous testing, which integrates test automation into the software delivery pipeline, has also evolved, enabling early identification of business risks associated with a software release candidate. Numerous software companies now incorporate continuous testing, which allows for real-time feedback and issue resolution, thereby improving product quality.

Shift-Left Testing not only impacts developers and testers but also influences the collaboration between teams across the SDLC, improving the communication and feedback loops. Several studies and surveys highlight that while Shift-Left Testing is beneficial, it poses significant challenges such as ambiguous requirements, lack of skilled testers, and high automation costs.

III. The Shift-Left Testing Paradigm

The term "shift-left testing" represents a big change in how we think about quality and testing in software development. Instead of treating testing as something we do only at the end of a project, shift-left testing encourages us to make it a core part of the whole development process from the very beginning.

To understand why this matters, imagine building a house. In the traditional approach, it's like building the whole house first and then, once it's fully constructed, checking to see if everything is safe and works properly. If you find out after the house is built that there's a problem with the foundation, it can be extremely expensive and complicated to fix—maybe even requiring you to tear down parts of the house to make repairs.

The shift-left approach says, "Let's test the foundation, plumbing, and electrical systems as we go, so we can catch any problems early on." By testing throughout the process, you're much more likely to catch small issues before they turn into big, expensive problems.

Barry Boehm, a software engineering expert, developed a famous ”cost curve” that shows why catching issues early is so important. The curve illustrates that the cost to fix a defect goes up significantly as you move further along in development. Fixing a problem during the planning or design phase is far cheaper than trying to fix it right before launch—or worse, after it’s been released to users.

Shift-left testing helps teams avoid these late-stage, high-cost fixes by making testing a continuous part of the workflow. It requires a mindset shift, viewing quality assurance as everyone’s responsibility throughout the lifecycle, rather than something that just the testing team does at the end. This approach ultimately leads to better quality software, faster development cycles, and fewer last-minute surprises.

A. Why Shift-Left Testing Matters

By moving testing earlier in the lifecycle, teams can identify defects and quality issues before they escalate, thereby reducing development costs and mitigating potential delays. This approach emphasizes proactive quality measures, such as:

B. Why Shift-Left Testing Matters

When teams move testing earlier in the development lifecycle, they gain the advantage of identifying defects and quality issues before these problems have a chance to escalate. Addressing defects earlier saves time and reduces costs because it’s much easier and cheaper to fix issues at the start of development than after most of the work has been completed. Shift-left testing helps avoid potential delays and improves the overall efficiency of the project by emphasizing proactive quality measures. Key benefits of this approach include:

- **Early Requirements Verification**: Shift-left testing encourages teams to validate requirements right from the beginning. This means that the requirements are carefully reviewed and tested for feasibility and completeness as soon as they are gathered. By doing this, the team can avoid misunderstandings or incorrect assumptions about what the software is supposed to do. Catching any issues in the requirements phase prevents problems from becoming embedded in the design and code, reducing the likelihood of major redesigns later.
- **Testing Embedded in Design**: Shift-left testing integrates testing into the design phase, so teams don’t wait until the software is fully built to start checking for issues. Testing the design helps ensure that the architecture of the software—its structure and components—meets quality standards from the start. By examining things like scalability, performance, and security at the design stage, teams can address structural issues before coding begins. This proactive testing makes the development process smoother and more efficient, as developers have a more reliable foundation to build upon.
- **Agile Compatibility**: Shift-left testing is a natural fit for agile development. Agile’s iterative process involves developing software in small increments, where each iteration adds new features or improves existing ones. Shift-left testing aligns with this process by requiring quality checks with each iteration, ensuring that new code integrates well with existing functionality. Regular, incremental testing helps prevent defects from accumulating, so the software remains stable throughout development. This is especially important in fast-paced projects, where agile’s frequent releases demand reliable and consistent quality.

The shift-left approach is often represented visually as a timeline or diagram that shows testing spread across all stages of the development lifecycle—from requirements gathering and design through development, integration, and deployment. This approach contrasts with the traditional ”shift-right” model, where testing happens mostly at the end. By distributing testing activities across the entire development timeline, shift-left testing ensures a more balanced and thorough quality assurance process.

The shift-left approach is often visually represented as a diagram with testing spread across all stages of the development lifecycle, from requirements gathering and design to development, integration, and deployment.

IV. Benefits of Shift-Left Testing

Shift-left testing offers several important benefits that enhance both the quality and efficiency of software development. By integrating testing earlier in the development lifecycle, teams can catch and resolve issues sooner, improve software quality, and align the product more closely with user needs. Key benefits of shift-left testing include:

- **Early Defect Detection**: Identifying defects early in the development process is one of the biggest advantages of shift-left testing. By finding and fixing issues at the requirements or design phase, teams can prevent costly defects from reaching later stages, where they become more challenging and expensive to resolve. Early defect detection also reduces the likelihood of critical issues affecting the final product, leading to a smoother, more predictable release process.
- **Improved Software Quality**: Shift-left testing enhances overall software quality by embedding testing throughout the development process. This ensures that each component and feature undergoes rigorous quality checks at every stage. Continuous quality verification helps teams maintain high standards, leading to a more reliable and stable product. When issues are addressed as they arise, the software’s architecture, codebase, and functionality remain more cohesive and resilient.
- **Continuous Testing Integration**: In a shift-left approach, testing is no longer a separate phase at the end of development; instead, it becomes an integral part of each stage. This continuous testing integration is especially valuable in agile and DevOps environments, where iterative development and continuous delivery require ongoing testing to ensure the quality of each release. Continuous testing not only accelerates the feedback loop but also

enables developers to detect and address issues in real-time, which streamlines development and reduces rework.

- **Security-First Development**: Shift-left testing encourages teams to consider security from the beginning, making security testing an essential part of the development lifecycle. By integrating security assessments early, teams can identify and mitigate vulnerabilities before they become serious risks. This proactive approach helps safeguard user data, protect system integrity, and reduce the need for costly security patches later on. A security-first mindset aligns with modern compliance standards and builds user trust in the final product.
- **Earlier User Feedback**: Incorporating testing early in the development process allows teams to gather user feedback sooner, often through prototypes, demos, or beta releases. Early feedback from users helps ensure the software aligns with real-world needs and expectations, allowing for adjustments before significant development resources are committed. This iterative feedback loop not only improves user satisfaction but also minimizes the risk of launching a product that does not meet user requirements.

Overall, shift-left testing promotes a proactive approach to quality assurance, resulting in more efficient, cost-effective, and user-centered development processes. By detecting and addressing issues early, teams can reduce delays, lower costs, and enhance the final product's quality and security.

A. Early Defect Identification

Early defect identification is one of the most significant advantages of shift-left testing. By conducting tests as soon as development begins, defects are detected closer to their source, making them faster and less costly to resolve. This approach prevents defects from being embedded deeper into the codebase or progressing undetected to later stages, where they would become more challenging and expensive to address. Early identification not only reduces the overall development cost but also minimizes disruptions to project timelines, resulting in a more efficient workflow.

In traditional testing, defects often accumulate unnoticed until the final stages of development. This accumulation can lead to a "testing bottleneck," where a large number of defects must be identified and fixed within a limited timeframe. Resolving these issues late in the cycle can delay project completion and may even result in quality compromises. Early defect identification helps avoid this bottleneck, allowing teams to maintain steady progress and quality assurance throughout the development process.

Additionally, early testing and defect identification promote a culture of proactive quality assurance. Team members, including developers, testers, and designers, become more quality-conscious from the outset, making fewer mistakes as they build on a foundation of continuous feedback. This mindset shift fosters collaboration, as teams work together to maintain high standards and prevent issues before they escalate. As a result, the final product is not only more robust but also more closely aligned with user expectations and requirements.

B. Improved Software Quality

Shift-left testing promotes software quality as a core priority throughout the development lifecycle, rather than as a final, last-minute concern. By embedding quality checks at each stage of the process, from requirements gathering to deployment, shift-left testing ensures that the product evolves with a continuous focus on meeting user requirements and maintaining high standards. This proactive approach reduces the risk of major issues surfacing toward the end of the development cycle, when they are harder and costlier to fix.

One of the main ways shift-left testing improves software quality is by fostering a collaborative workflow between developers, testers, and other team members. Rather than working in isolation, these roles are aligned from the beginning, jointly responsible for ensuring the software meets quality criteria at each phase. This collaborative workflow encourages a shared commitment to quality, where both developers and testers can provide input and feedback, identify potential issues early, and adjust plans accordingly. As a result, the final product more accurately reflects the expectations and needs of users, reducing the chances of post-release defects.

Moreover, establishing quality checkpoints at each stage allows teams to detect issues related to code, design, performance, and security before they escalate. Each phase acts as a gate, ensuring that only high-quality code progresses to the next step. These checkpoints provide opportunities for early corrections, saving time and resources in the long run. This step-by-step focus on quality creates a stronger, more resilient product, ultimately leading to increased customer satisfaction and trust in the software.

C. Continuous Testing

Continuous Testing (CT) is a core component of shift-left testing and plays a significant role in maintaining software quality throughout the development process. CT involves running automated tests at multiple points within the Continuous Integration and Continuous Deployment (CI/CD) pipeline, allowing teams to receive immediate feedback on each code change. This approach ensures that testing isn't confined to the end of the development cycle, but rather becomes a continuous part of every step, allowing for real-time monitoring of code quality.

By integrating CT with shift-left testing, teams can detect issues early in the development process. Since automated tests are triggered whenever code is modified or new code is added, any defects or inconsistencies are identified immediately. This real-time feedback loop enables quick adjustments and corrections, preventing defects from accumulating and reducing the risk of issues slipping into later stages of development. As a result, CT enhances the reliability of each build, making the development process more stable and predictable.

Another key advantage of continuous testing is risk reduction. By frequently checking for security vulnerabilities, performance bottlenecks, and functional errors, teams can address potential risks as they emerge, rather than after they have compounded. This proactive approach ensures that quality standards are maintained consistently, minimizing the likelihood of critical failures in production. The stability provided by CT also allows teams to confidently move toward rapid deployment cycles, supporting agile and DevOps goals of delivering high-quality software faster.

In essence, continuous testing contributes to a smoother, more efficient development cycle. It empowers teams to work iteratively, with quality validation seamlessly embedded in each cycle. The end result is a more stable product, greater efficiency, and the ability to innovate quickly without compromising on quality.

D. Security-First Development

Security-first development is a key benefit of shift-left testing, where security practices are embedded at every stage of the development lifecycle. Unlike traditional approaches where security checks are often delayed until the final stages, shift-left testing incorporates security considerations from the beginning. This proactive approach to security helps development teams identify and address potential vulnerabilities early, long before they can become deeply embedded in the system and difficult to fix.

By focusing on security at the outset, teams can design and build software with potential threats in mind. This involves running security tests during early phases, such as code review, requirements gathering, and design, to ensure that secure coding practices and risk mitigation measures are integrated throughout the process. When teams catch security issues early, they reduce the likelihood of critical vulnerabilities, which could lead to costly fixes and potential data breaches if left undetected until later stages.

This security-first approach aligns closely with ”DevSecOps” — a practice where security is treated as a continuous, integrated part of the DevOps workflow. In DevSecOps, security checks, vulnerability assessments, and compliance testing are automated and conducted alongside other quality assurance measures. This ensures that security does not create a bottleneck but instead becomes a seamless part of the development process. Additionally, integrating security checks early supports compliance with industry standards and regulations, helping organizations avoid costly penalties and reputational risks.

Overall, adopting a security-first approach within shift-left testing promotes more resilient software. By addressing security proactively, development teams can focus on building robust applications that protect user data and maintain trust. This approach not only strengthens the software but also reduces risk management costs, as security flaws are significantly less expensive to fix early in the lifecycle.

E. User Feedback

Shift-left testing also supports early user feedback integration, allowing teams to validate features with real users or stakeholders before finalization. Early feedback ensures that the software aligns more closely with user expectations, reducing the risk of misalignment.

TABLE I
SUMMARY OF SHIFT-LEFT TESTING BENEFITS

Benefit	**Description**
Early Defect Identification	Timely resolution of defects, minimizing costs.
Improved Software Quality	Collaborative quality checkpoints, ensuring software meets user needs.
Continuous Testing	Stability ensured through consistent testing during development.
Security-First Development	Early security integration reduces risks and enhances resilience.
User Feedback	Early user involvement ensures better alignment with requirements.

V. IMPLEMENTING SHIFT-LEFT TESTING

Successful shift-left testing involves strategic integration of testing throughout the development process. Several best practices and methodologies facilitate this integration.

A. Automated Testing

Automated testing plays a critical role in shift-left testing. Integrated into the CI/CD pipeline, automated tests can be executed frequently and consistently, ensuring rapid defect detection. Automated tests can include unit, integration, and functional tests, which run as part of each code change, reducing the risk of introducing defects.

B. High-Level Test-Driven Development

Test-Driven Development (TDD) at a higher level ensures that tests are created before development, guiding code toward meeting quality criteria from the outset. This TDD approach, often employed within agile frameworks, allows teams to maintain quality at every stage of development and enables continuous alignment with project requirements.

C. Collaboration between Developers and Testers

Shift-left testing requires close collaboration between developers and testers from the start. Establishing communication channels and shared goals between these roles ensures that testing is seamlessly incorporated into the development process. Collaborative planning sessions help synchronize testing efforts with development progress, avoiding quality assurance bottlenecks.

D. Shift-Left Testing in DevOps

In DevOps, where the development and operations processes are tightly integrated, shift-left testing brings an additional level of coordination. DevOps leverages continuous integration and delivery (CI/CD), enabling testing at each deployment stage. Aligning testing with DevOps practices allows for rapid feedback and streamlined release cycles, fostering a cohesive, efficient development workflow.

E. Fuzzy Logic-Based Shift-Left Testing

Fuzzy logic-based testing incorporates probabilistic reasoning to accommodate uncertainties inherent in early-stage testing. For example, fuzzy logic can be used to manage varying risk levels across test cases, allowing testers to prioritize test scenarios based on likelihood and potential impact.

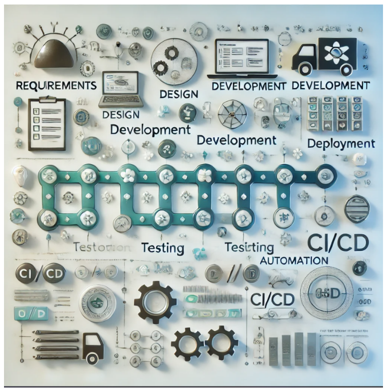


Fig. 1. Key Steps in Shift-Left Testing Implementation

VI. Challenges and Considerations

Despite its benefits, shift-left testing poses several challenges that require careful consideration.

A. Cultural Shift

Implementing shift-left testing often requires a shift in organizational culture. Testing early in the development process demands collaboration and open communication, which may be unfamiliar to teams accustomed to sequential, phase-based development.

B. Skill Set Alignment

Shift-left testing relies on cross-functional skills, as developers and testers need to work together seamlessly. Training and skill development are crucial, as both developers and testers must be adept in collaborative quality assurance practices, test automation, and CI/CD processes.

C. Tool Integration

Seamless tool integration is critical for successful shift-left testing. The selected tools must integrate smoothly with the existing development pipeline and facilitate automated testing, continuous integration, and reporting. Achieving this integration can be complex and may require adapting the infrastructure.

D. Balancing Testing Efforts

Determining the appropriate level of testing coverage at each stage requires balance. Overemphasis on early testing can lead to inefficiencies, while insufficient early testing increases the risk of late-stage issues. It is essential to define clear testing priorities and scope for each stage of development.

E. Continuous Improvement

Shift-left testing is an evolving process. Regular evaluations are necessary to adapt testing practices to changing project needs and optimize test coverage over time.

TABLE II
Challenges in Shift-Left Testing Implementation

Challenge	Description
Cultural Shift	Fostering collaborative quality practices across teams.
Skill Set Alignment	Ensuring team skills align with shift-left testing needs.
Tool Integration	Harmonizing testing tools with development infrastructure.
Balancing Testing Efforts	Avoiding inefficiencies by prioritizing testing scope.
Continuous Improvement	Regularly refining testing practices to maintain effectiveness.

VII. Results

A. Key Challenges

While shifting to the left has numerous advantages, many organizations face significant challenges in its implementation. The survey results indicate that over half (56%) of the participants consider in-sprint testing to be highly challenging. Key challenges identified include:

- Ambiguous requirements or user stories that contribute to coding errors and undermine testing efficacy.
- Crafting and sustaining meaningful test cases that align with user expectations.
- Maintaining test scripts and handling the associated costs of automation.
- Difficulty in obtaining qualified engineers to design continuous testing models.
- Non-functional testing, especially performance testing, remains a challenge.

B. Model-Based Shift-Left Testing (MBSLT)

Model-Based Testing (MBT) plays a crucial role in the Shift-Left approach. MBT generates test cases based on system models such as flowcharts, Unified Modeling Language (UML) diagrams, and process flows. These models guide the testing process and ensure measurable results. Increasingly, MBT tools incorporate script engines that automatically generate test scripts, streamlining the testing process.

C. Predicting Crucial Outcomes

Shift-Left Testing proactively manages risks by defining clear objectives early in the development process. This involves early unit testing, static code analysis, code reviews, and integrating security and performance testing early in the lifecycle. Early user feedback is also crucial for predicting user satisfaction and making necessary adjustments before the final release.

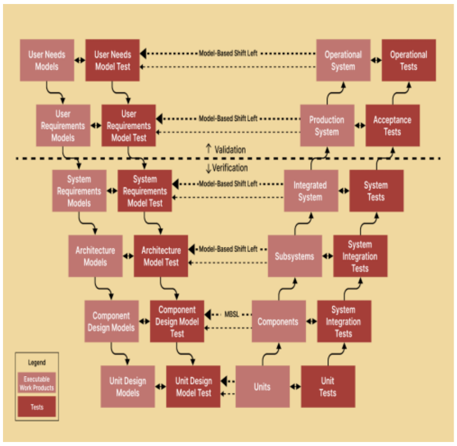


Fig. 2. Model-Based Shift-Left Testing Framework

VIII. Case Studies

Several organizations have successfully implemented shift-left testing principles, achieving significant improvements in development speed, quality, and cost efficiency.

A. Shift-Left Testing in DevOps: Microsoft's Azure Development Team

In DevOps environments, continuous integration and continuous delivery (CI/CD) demand robust testing to ensure quality with each deployment. Microsoft's Azure development team implemented a shift-left testing strategy to handle the complex, large-scale nature of cloud infrastructure.

a) Example:: Microsoft's Azure development team incorporated shift-left testing by embedding automated tests at every phase of the CI/CD pipeline. They used tools such as Azure DevOps to automate tests that checked code quality, security, and functionality as soon as new code was pushed. The team emphasized early validation through unit tests, integration tests, and security scans, allowing them to detect issues earlier and avoid costly late-stage fixes.

b) Impact:: By catching issues early, Microsoft reduced the number of critical bugs reaching production, thereby improving the stability and performance of the Azure platform. This proactive testing approach decreased downtime and boosted customer satisfaction. The team also reported a significant reduction in the time spent on post-release bug fixes, allowing developers to focus more on feature development.

B. Shift-Left Testing in Agile Environments: Spotify's Feature Squads

Spotify, a major player in the music streaming industry, employs an agile development model with cross-functional "squads" responsible for different features. Shift-left testing became essential for maintaining high quality across Spotify's rapid development cycles.

a) Example:: Each Spotify squad integrated shift-left testing into its agile sprints by running tests on requirements and designs before coding began. For instance, in one project focused on optimizing Spotify's recommendation algorithm, the squad began by validating assumptions through test cases aligned with user stories. They also utilized Test-Driven Development (TDD), writing test cases alongside feature code, ensuring that all new functionality was covered by automated tests from day one.

b) Impact:: The early testing approach allowed squads to validate ideas with minimal waste and catch potential issues before significant development resources were invested. As a result, Spotify was able to deploy new features more quickly and with fewer post-release bugs, providing users with a more reliable and engaging experience. Shift-left testing was especially effective in addressing user experience (UX) issues early on, as user feedback was incorporated into each sprint.

C. Fuzzy Logic-Based Testing in Financial Services: PayPal's Risk Management System

Financial service providers face unique challenges regarding risk, security, and regulatory compliance. PayPal, a global payments company, adopted a fuzzy logic-based testing approach as part of its shift-left strategy to handle unpredictable data patterns in its fraud detection and risk management systems.

a) Example:: In PayPal's risk management system, fuzzy logic-based testing was used to simulate complex scenarios involving fraud detection. Fuzzy logic allowed PayPal to introduce probabilistic testing, where scenarios with different levels of risk were generated and tested dynamically. This testing occurred as early as the design phase, enabling developers to adjust algorithms and refine risk models before full-scale implementation.

b) Impact:: Early testing with fuzzy logic enabled PayPal to proactively detect and resolve potential vulnerabilities, reducing the risk of fraud slipping through the system undetected. Shift-left testing also improved PayPal's ability to meet regulatory compliance standards, as early security testing allowed them to identify and document controls before audits. This approach minimized the cost of fixing security issues late in the development process, reducing the likelihood of data breaches and ensuring user trust.

IX. Conclusion

Shift-left testing is more than just a change in timing—it's a shift in the whole approach to quality within software development. By embedding testing from the earliest phases, teams can catch issues at their roots, reducing both the cost and the complications that arise when defects are discovered late. Think of it as quality assurance becoming a constant presence throughout the process, rather than a final checkpoint. This change can mean fewer surprises, less time spent on rework, and smoother progress through development.

The benefits of shift-left testing are amplified when paired with practices that support it, like automation, test-driven development (TDD), and strong collaboration between developers and testers. Automation allows tests to run repeatedly and consistently with every code change, catching errors as soon as they appear. High-level TDD helps teams clarify requirements and build features with quality as a core focus.

Collaboration brings everyone together—developers, testers, and other stakeholders—working on the same goals, with aligned understanding and expectations.

In modern development environments, like agile and DevOps, where speed and adaptability are crucial, shift-left testing serves as a natural fit. It's designed to keep pace with frequent code updates and ensure that quality remains a priority throughout. Ultimately, shift-left testing isn't just a strategy but a cultural shift towards prioritizing quality from the start, leading to more reliable software and a smoother, more efficient development journey.

References

[1] Boehm, B. W. (1981). *Software Engineering Economics*. This foundational text introduces the concept of the cost curve in software development, emphasizing the importance of early defect detection and its economic implications.

[2] Crispin, L., & Gregory, J. (2009). *Agile Testing: A Practical Guide for Testers and Agile Teams*. This book discusses how testing fits into agile methodologies, including strategies for implementing shift-left testing.

[3] Meyer, B. (1997). *Object-Oriented Software Construction*. This work touches on the principles of software design and testing, providing insights into integrating testing throughout the development lifecycle.

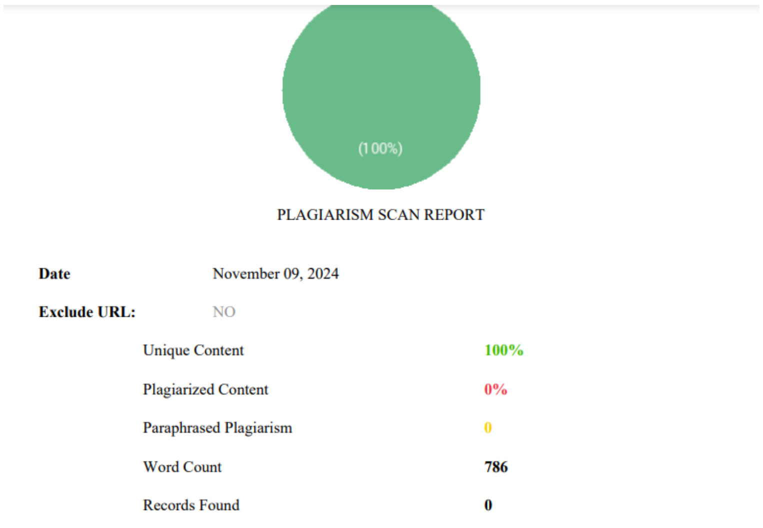
[4] Fowler, M. (2006). *Continuous Integration*. In this article, Fowler discusses the role of continuous integration in modern software development and how it aligns with shift-left testing practices.

[5] Kaner, C., Falk, J., & Nguyen, H. (1999). *Testing Computer Software*. This book provides a comprehensive overview of software testing practices and methodologies, including early testing strategies.

[6] Graham, D., & Van Veenendaal, E. (2010). *Foundations of Software Testing: ISTQB Certification*. This reference covers various testing techniques and methodologies that support the shift-left approach.

[7] Beck, K., & Andres, C. (2005). *Extreme Programming Explained: Embrace Change*. This book outlines principles of extreme programming that correlate with early testing and quality assurance practices.

X. Plagiarism Report

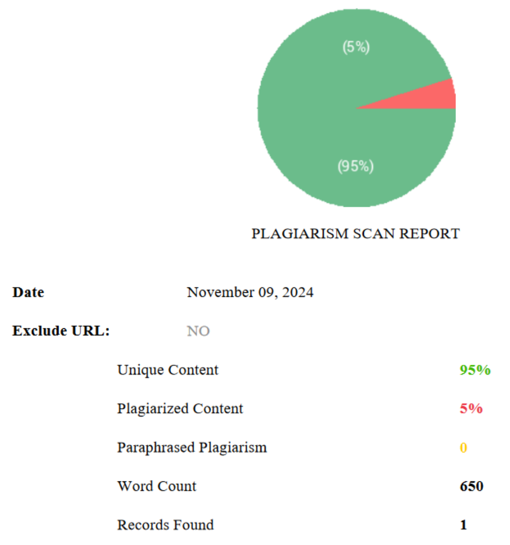


PLAGIARISM SCAN REPORT

Date	November 09, 2024
Exclude URL:	NO

Unique Content	**100%**
Plagiarized Content	**0%**
Paraphrased Plagiarism	**0**
Word Count	**786**
Records Found	**0**

Fig. 3. PLAGIARISM SCAN REPORT



PLAGIARISM SCAN REPORT

Date	November 09, 2024
Exclude URL:	NO

Unique Content	**95%**
Plagiarized Content	**5%**
Paraphrased Plagiarism	**0**
Word Count	**650**
Records Found	**1**

Fig. 4. PLAGIARISM SCAN REPORT