International Journal of Management Science and Information Technology
IJMSIT

E-ISSN: 2774-5694 P-ISSN: 2776-7388 OPEN ACCESS

Volume 6 (1), January-June 2026, 348-358
DOI: https://doi.org/10.35870/ijmsit.v6i1.6610

Shift-Left Testing in the Software Testing Life Cycle: Evidence from the Ruang Murid Application on Time Efficiency and Software Quality

Agung Enrico [1*], Ana Kurniawati [2]
[1*,2] Department of Technology and Engineering, Universitas Gunadarma, West Java Province, Indonesia

Email: Agung.enrico@gmail.com [1*], ana@staff.gunadarma.ac.id [2]

Article history:
Received February 16, 2026
Revised April 11, 2026
Accepted April 15, 2026

Abstract

This study investigates the implementation of Shift-Left Testing within the Software Testing Life Cycle (STLC) and its implications for testing time efficiency and software quality in the Ruang Murid application. A quantitative case study design was applied using testing records, observation, documentation, and questionnaire data collected from the development team, including Software Quality Assurance staff, developers, and the project manager. Data were analysed using SPSS through validity and reliability testing, multiple linear regression, t-tests, F-tests, and coefficient of determination analysis. The results show that after the adoption of Shift-Left Testing, the number of detected bugs increased by 54.39%, indicating broader and earlier defect discovery during the development cycle rather than a decline in product quality. Bug rate rose from 1.9 to 2.9 bugs per KLOC, while defect rate increased from 4.2% to 7.1%, both of which remained within acceptable quality thresholds. In addition, defect leakage was recorded at 10.37%, test coverage and test execution rate each reached 100%, and the test case pass rate was 50.51%. These findings suggest that the earlier involvement of the SQA team improved defect visibility, expanded testing coverage, and strengthened quality control across development stages, although further improvements are still needed to increase pass stability and overall testing confidence.

Keywords:
Shift-Left Testing; Software Testing; Bug Detection; Development Cost; Development Time.

1. INTRODUCTION

The accelerated advancement of digital learning technologies has fundamentally reshaped the contemporary educational environment, driving the extensive adoption of large-scale learning management systems and interactive educational platforms. Within this evolving ecosystem, the Ruang Murid application emerges as a strategic digital learning solution that integrates instructional content delivery, real-time collaborative functionalities, and flexible accessibility for students, educators, and educational institutions. As digital education platforms increasingly function as mission-critical infrastructures, the assurance of system reliability, operational performance, and application quality has become an essential prerequisite for sustaining effective digital learning ecosystems (Aljawarneh, 2020). In this regard, rigorous software testing mechanisms play a pivotal role in safeguarding system stability, reducing operational interruptions, and improving overall user experience and satisfaction.

Despite the increasing importance of quality assurance in educational software development, many organizations continue to rely on traditional testing approaches in which testing activities are conducted predominantly at the later stages of the Software Testing Life Cycle (STLC). Such late-stage testing practices often lead to delayed detection of defects, increased rework costs, and extended-release cycles, thereby reducing overall development efficiency (Aripiyanto et al., 2021; Amarta & Anugrah, 2021). Empirical evidence suggests that when defects are identified only during the final testing phases, the cost of correction becomes significantly higher compared with early detection during the requirement or design stages (IBM,

2023). Consequently, the need for more proactive and preventive testing strategies has become increasingly urgent, particularly for complex digital platforms such as educational applications that involve multi-module integrations, real-time communication features, and cross-device compatibility requirements.

To address these challenges, the shift-left testing paradigm has emerged as a modern quality assurance strategy that emphasizes the early integration of testing activities throughout the software development lifecycle. Rather than positioning testing as a final validation step, shift-left testing promotes continuous testing beginning from requirement analysis and system design phases, enabling earlier detection of design inconsistencies, requirement ambiguities, and functional defects (Kavuri, 2024). By enabling early feedback loops between developers, testers, and stakeholders, this approach contributes to improved collaboration, faster iteration cycles, and reduced technical debt accumulation (Singh & Suri, 2020). Industry reports further indicate that organizations implementing early-stage testing strategies achieve improved defect detection rates, reduced defect leakage, and accelerated time-to-market performance (Capgemini, 2021).

In the context of digital education applications such as Ruang Murid, the relevance of shift-left testing becomes even more pronounced. Educational platforms typically operate under high user expectations regarding system availability, usability, content reliability, and uninterrupted access. Any functional failure, performance degradation, or integration error can directly affect the learning experience of students and educators. Early involvement of Software Quality Assurance (SQA) teams during the requirement and design phases allows test scenarios to be aligned with pedagogical workflows, user interaction patterns, and system integration requirements, thereby enhancing both functional and experiential quality outcomes (Ministry of Testing, 2023; Awati, 2023). Furthermore, the integration of testing activities into early development phases encourages a collaborative quality culture, where quality responsibility is shared among developers, testers, and product stakeholders rather than being confined to post-development testing processes (Devopedia, 2022).

Previous studies on educational platforms have generally examined quality from the perspective of users rather than from the perspective of testing-process outcomes. Recent studies have focused on electronic learning service quality and learner satisfaction (Sumi & Kabir, 2021), e-learning software usability and student satisfaction (Alqurni, 2023), student acceptance of e-learning systems through system quality and task-technology fit (Alyoussef, 2023), and user satisfaction with e-learning systems through the DeLone and McLean model (Atukunda et al., 2024). These studies are valuable because they confirm that system quality, usability, support, and perceived usefulness are central to the success of educational platforms. However, they mainly evaluate perceived quality, adoption, or satisfaction outcomes, and provide limited evidence on how quality assurance interventions within the STLC affect measurable testing outcomes in real development settings.

Previous studies have also demonstrated the effectiveness of structured testing lifecycle implementation and automated testing strategies in improving software quality outcomes. For instance, automated black-box testing implementations have shown the ability to systematically detect functional inconsistencies while improving testing efficiency and repeatability (Zulianto et al., 2021). Similarly, the application of automation tools in mobile learning platforms has demonstrated improvements in testing accuracy and reliability, highlighting the importance of integrating testing methodologies within structured lifecycle frameworks (Herlinda et al., 2019). In addition, systematic documentation and structured testing processes are recognized as critical components of effective Software Quality Assurance, ensuring that software systems operate according to defined functional specifications while minimizing operational risks (Hilabi, 2018; Utami & Setyodewi, 2023). Nevertheless, broader software engineering research shows that quality improvement claims in agile and rapid software development need to be supported by clear and traceable metrics rather than general assertions alone (López et al., 2022; Natarajan & Pichai, 2024).

Accordingly, the research gap addressed in this study lies in the limited empirical evidence on how shift-left testing within the STLC of an educational platform affects operational testing indicators rather than only perceived user outcomes. In the case of Ruang Murid, this study evaluates impact using measurable indicators consisting of detected bugs before and after implementation, bug rate per KLOC, defect rate, defect leakage, test coverage, test execution rate, test case pass rate, and testing time efficiency. By using these indicators, the present study goes beyond prior educational-platform research that has largely emphasized system quality, usability, service quality, satisfaction, or acceptance, and instead provides evidence of how earlier SQA involvement changes testing performance and software quality control in an actual development environment.

Therefore, this study aims to investigate the impact of implementing shift-left testing within the Software Testing Life Cycle of the Ruang Murid application, focusing on its effects on defect detection effectiveness, testing efficiency, and software quality outcomes. By combining lifecycle-based testing evaluation with empirical performance indicators such as bug rate, defect leakage, test coverage, test execution rate, and test case pass rate, this research contributes to both theoretical and practical discussions on proactive quality assurance strategies in modern digital platform development. The findings are expected to provide strategic insights for software development organizations, educational technology providers, and quality assurance practitioners seeking to enhance development efficiency while maintaining high standards of software reliability and user experience (Ruliansyah et al., 2023; GeeksforGeeks, 2023).

2. RESEARCH METHOD

This study used a quantitative case-study design to examine the effect of implementing Shift-Left Testing within the Software Testing Life Cycle (STLC) on testing time efficiency and software quality outcomes in the Ruang Murid application, conducted in a real operational setting involving the Ruang Murid development team (Software Quality Assurance/SQA, developers, and a project manager).

The unit of analysis was the testing process executed across STLC phases, while the intervention of interest was the deliberate shift of testing activities "to the left," meaning earlier involvement of SQA starting from requirement analysis and test planning, rather than concentrating testing work only at the end of development. The research logic consisted of two complementary analytic tracks. The primary analytic track used a before-after design based on defect and testing records to assess the impact of Shift-Left Testing on testing efficiency and software quality outcomes. The secondary analytic track used a UTAUT-based survey to examine how system stability, facilitating conditions, and user perceptions may support actual use behavior of the Ruang Murid platform. In this study, the UTAUT analysis was not treated as a direct measure of Shift-Left effectiveness, but as a contextual explanatory analysis linking software quality conditions to platform usage.

The research procedure followed a structured workflow beginning with problem identification and literature review, then mapping the existing STLC implementation in Ruang Murid to identify pain points such as delayed defect discovery and limited early SQA engagement, followed by designing and implementing an adjusted STLC model incorporating Shift-Left Testing practices, and ending with comparative analysis of outcomes after To make the procedural logic transparent for journal reporting, the workflow is summarized in figure 1.

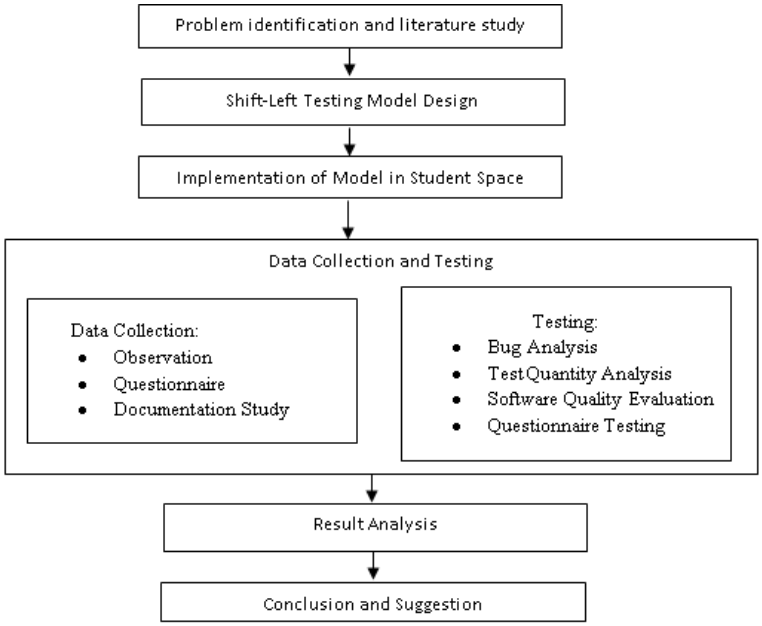

Figure 1. Research Workflow Applied in This Study

Operationally, the study treated STLC as the backbone process model, comprising requirement analysis, test planning, test case development, test environment setup, test execution, and test cycle closure (Vadaddi et al, 2023). In the requirement analysis phase, testers examined requirement artifacts (including brief-style documentation when formal PRD structures were incomplete), identified testable requirements from a QA perspective, classified expected testing types (functional, performance, security where relevant to planning), prioritized risks, and assessed test-environment needs and automation feasibility. In test planning, the team translated requirements into a test strategy that specified scope, test approach (manual and/or tool-supported), test schedule, resource needs, environments, and procedural steps, forming a test plan that later drove test case design and execution time estimates (Venkitaraman, 2024). Test case development then produced and reviewed test cases and optional scripts, created supporting test data, and refined scenarios to align with requirements and user flows. Environment setup ensured readiness of hardware/software and test data; where environments were provisioned by development, SQA emphasized smoke testing as a gate before broader execution. During test execution, test cases were run, results were recorded, defects were reported for correction, and retesting was performed after fixes. Finally, test cycle closure consolidated completion evidence, summarized test metrics, and documented defect patterns and lessons learned to improve subsequent cycles (Singh &Suri, 2020). The "shift-left" element of the intervention was implemented by strengthening SQA participation earlier in the lifecycle (requirement analysis and planning) to reduce

ambiguity, improve test readiness, and prevent defects from accumulating until the final execution phase, consistent with the thesis rationale and workflow.

Data were collected using four complementary methods: direct observation, questionnaire-based survey, documentation study, and defect/testing record analysis. Observation captured how testing work was actually performed across STLC phases, how cross-functional coordination occurred, and where bottlenecks emerged before and after the Shift-Left approach was integrated. The documentation study compiled testing artifacts such as test plans, test cases, bug reports, and prior evaluation outputs, enabling traceability and supporting metric computation. The survey was administered online via Google Forms to obtain structured quantitative data about perceptions and context of Ruang Murid usage and enabling conditions, using a five-point Likert scale from 1 (Strongly Disagree) to 5 (Strongly Agree). The Likert scale coding used for the instrument is presented table 1.

Table 1. Likert Scale Coding Used In The Survey

Response option	Score
Strongly Agree	5
Agree	4
Somewhat Agree / Neutral	3
Disagree	2
Strongly Disagree	1

The questionnaire items were organized using a UTAUT-informed variable structure adapted to the Ruang Murid context, capturing perceived performance expectancy, social influence, facilitating conditions, behavioral intention, and use behavior, aligned with the operational definitions already articulated. In the present study, the UTAUT-based questionnaire was retained as a secondary analytical component to complement the primary QA-metrics analysis. Its purpose was not to test the technical effectiveness of Shift-Left Testing directly, but to explain why software stability, availability, and supporting conditions matter for actual use behavior in an educational platform. This secondary analysis was considered relevant because improvements in defect control, release stability, and testing effectiveness may influence users' experience of system reliability and, in turn, their continued use of the application. The operational mapping of constructs to the Ruang Murid context is summarized as follows table 2.

Table 2. UTAUT-Oriented Constructs and Their Contextual Meaning In This Study

Construct	Contextual meaning in Ruang Murid
Perceived Performance Expectancy	Users perceive the app helps them understand learning content faster and better
Social Influence	Teachers require it, peers use it, parents encourage use
Facilitating Conditions	Device availability, stable internet, and school support
Behavioral Intention	Strength of intention to use Ruang Murid
Use Behavior	Frequency and intensity of actual use

Sampling employed a non-probability purposive strategy, targeting respondents relevant to the research context. Because the population size was large or not precisely known, minimum sample estimation followed a Lemeshow-based approach using a 95% confidence level ($z = 1.96$), maximum proportion assumption $p = 0.5$, and sampling error $d = 0.10$, which yielded an estimated minimum of approximately 96.4 respondents; therefore, a sample size of 100 responses was collected to exceed the minimum threshold while acknowledging practical constraints

The survey distribution period spanned approximately two months (November-December 2025), disseminated through social channels (e.g., WhatsApp and Instagram). Respondent inclusion criteria included both genders, age ranges from under 20 to above 50, domicile in Jabodetabek, and education levels spanning primary school through doctoral level as stated in the thesis, reflecting the application's broad user and stakeholder reach. To quantify the impact of Shift-Left Testing on time efficiency and quality, the analysis emphasized a set of operational metrics computed through comparison of conditions before and after implementation.

To quantify the impact of Shift-Left Testing on time efficiency and software quality, this study compared testing outcomes before and after implementation using operational QA metrics. In this study, a 'bug' refers to an issue identified during testing in the staging environment, whereas a 'defect' refers to a fault that escaped pre-release testing and was identified in the production environment. Bug findings before and after the intervention were compared descriptively to examine whether earlier testing integration increased defect visibility during development. Bug rate was calculated as bugs per KLOC (Kilo Lines of Code) to normalize defect occurrence by codebase size. To maintain comparability across semesters, the KLOC basis was held constant using the same estimated code-size baseline for the observed release stream in both semesters. This approach was intended to reduce distortion from codebase-size differences and to support a fairer comparison of defect density across periods.

Release complexity was not experimentally controlled as a separate variable. However, to reduce interpretive bias, the study considered several contextual comparators across the two semesters, including the number of executed test scenarios, the use of the same KLOC baseline, and the distinction between staging bugs and production defects. Therefore, differences between the pre- and post-implementation periods were interpreted cautiously, with attention to the possibility that variations in release scope, regression depth, or feature complexity may also influence bug discovery volume and defect rate. The engineering interpretation thresholds used in the article to contextualize bug density are summarized (table 3).

Table 3. Engineering Target Levels For Bug Rate (Per KLOC)

Level	Bug rate (per KLOC)	Notes
Excellent	0–1	High maturity team
Good	1–5	Typical stabilized delivery
Acceptable	5–10	Early release or complex legacy
Need Attention	>10	High defect density requiring immediate improvement

Defect rate was also computed from a testing perspective and interpreted using target levels aligned with test maturity framing as documented in the thesis, supporting an additional lens on product stability and test effectivenes.

Table 4. Testing Target Levels For Defect Rate

Level	Defect rate	Notes
Excellent	<2%	High coverage and efficient testing
Good	2%–5%	Normal for many organizations
Acceptable	5%–10%	Actively testing complex features
Need Attention	>10%	Potential gaps in test design or high product instability

Beyond defect metrics, the study assessed testing process depth and coverage using changes in the number of test cases (growth in test case volume), test case coverage (executed vs planned), test execution rate, and test case pass rate. Defect leakage was used to estimate the proportion of defects that escaped pre-release testing and were found only after release, serving as a direct indicator of test effectiveness and risk exposure, consistent with the thesis explanation. A qualitative “testing confidence level” was also interpreted through score thresholds (high >80, medium 60–80, low <60) to summarize perceived readiness and residual risk.

Survey data were analyzed using standard quantitative procedures as described in the thesis: validity testing used product-moment correlation by comparing each item’s r-value to the critical r-table threshold, with items considered valid when r-calculated exceeded r-table; reliability was tested using Cronbach’s Alpha, with alpha values greater than 0.60 interpreted as acceptable internal consistency. Hypothesis testing used t-tests to evaluate partial effects of independent variables on dependent outcomes and an F-test (ANOVA) to assess simultaneous effects within regression modeling, using a 5% significance level ($\alpha = 0.05$) for decision criteria. Model explanatory power was evaluated using the coefficient of determination (R^2) to indicate how much variance in the dependent variable(s) was explained by the predictors, consistent with the statistical framework. Collectively, these methods were intended to provide a rigorous and replicable assessment of Shift-Left Testing through two complementary perspectives: a primary before-after evaluation of defect outcomes and testing efficiency, and a secondary contextual analysis of user-related conditions through the UTAUT framework.

3. RESULTS AND DISCUSSION

3.1. Shift-Left Testing Implementation in Ruang Murid

The empirical results indicate that the Shift-Left Testing approach in the Ruang Murid development workflow was operationalized by moving quality activities upstream, particularly by involving SQA earlier in requirement clarification, risk identification, and the early drafting of test scenarios. In the planning stage, the testing team conducted internal requirement analysis for both the student and teacher applications, emphasizing functional specifications and early testability checks. This early engagement reduced ambiguity in acceptance criteria and allowed the team to prioritize critical scenarios (for example, student login and core learning flows) as “release-gating” test cases during test planning. In the design stage, the team produced more detailed and execution-ready test cases, including explicit steps, input parameters, and expected outcomes, supported by realistic test data assumptions (e.g., concurrent student activity and high-volume interactions).

The implementation practice described in this study also leveraged operational data from Jira tickets to accelerate test design and feedback. Test parameters (such as input constraints, timing boundaries, and user-profile conditions) were extracted from story tickets and aligned with automated scripts where feasible,

enabling earlier validation at unit and API-integration levels through mocking and controlled data. When automation ran inside the CI/CD pipeline, outcomes were synchronized back to Jira tickets, enabling rapid feedback loops so developers could fix defects while feature context remained fresh. This created a shorter defect-feedback cycle and reduced the "end-of-sprint bug pile-up" risk, which is one of the practical advantages typically expected from a shift-left strategy.

To clarify how work roles changed under this approach, Figure 1 summarizes the shift-left way of working: engineers performed self-testing at local environments guided by merge request checklists and acceptance criteria, while SQA focused on higher-level validation such as end-to-end testing, exploratory testing, regression, sanity testing, and UAT preparation. Under this model, SQA also contributed earlier by refining acceptance criteria at the story level and supporting test-case documentation (manual and automated) in the team's test management workflow.

Figure 2. Shift-Left Way of Working and Feedback Loop

Table 5. Comparison Of Traditional SQA vs Shift-Left SQA Operating Model

Aspect	Traditional SQA Pattern	Shift-Left SQA Pattern
Primary focus	Defect detection late in the cycle	Defect prevention from early phases
SQA involvement	Mostly during testing phase	Active across SDLC phases
Defect discovery timing	Late; higher cost of rework	Early; lower rework cost
Collaboration	Reactive, limited	Proactive, tightly coupled
Cost of quality	Higher due to late fixes	Lower due to early prevention
Release predictability	Risk of last-minute delays	Faster and more predictable
Product quality logic	Improve by finding defects	Improve by preventing defects and continuous improvement

3.2. Quantitative Measurement Results from Testing Records

Based on the documented testing records throughout 2025, the development team executed a total of 297 test case scenarios across the year. When disaggregated by quarter, the total test cases were 52 in Q1, 90 in Q2, 105 in Q3, and 50 in Q4. In parallel, the team recorded 147 bugs in the staging environment across 2025, distributed as 24 (Q1), 33 (Q2), 59 (Q3), and 29 (Q4). Post-release defects in the production environment totaled 17 across the year, with 6 found in semester 1 and 11 in semester 2. This staging-versus-production split is important because it distinguishes defects captured during development from those that escaped into live usage, which directly reflects test effectiveness and release risk management.

Table 6. Summary Of Test Cases, Bugs, And Production Defects (2025)

Metric	Total	Breakdown
Test cases	297	Q1: 52, Q2: 90, Q3: 105, Q4: 50
Bugs (staging)	147	Q1: 24, Q2: 33, Q3: 59, Q4: 29
Defects (production)	17	Semester 1: 6, Semester 2: 11

A key finding concerns the "before vs after" comparison of bug discovery volume. In this study, semester 1 (pre-shift-left) recorded 57 bugs from 142 scenarios (Q1 + Q2), while semester 2 (post shift-left) recorded 88 bugs from 155 scenarios (Q3 + Q4). Using the study's comparative percentage formula, the measured change was 54.39%, meaning the number of detected bugs increased after shift-left adoption. Importantly, this increase should not automatically be interpreted as poorer software quality. In the present case, the increase coincided with a rise in executed scenarios from 142 to 155, earlier prioritization of

release-gating scenarios such as login and core learning flows, and broader validation activities spanning unit/API-integration checks, end-to-end testing, exploratory testing, regression, sanity testing, and UAT preparation. These changes suggest that the post-implementation period involved wider and deeper testing, making higher bug discovery more consistent with increased defect visibility than with a simple deterioration in product quality. Under a prevention-oriented logic, higher detection in earlier stages can reduce late-stage and production escape, but it also depends on the maturity of automation, requirement stability, and environment reliability.

Table 7. Defect Leakage by Semester

Semester 1 (Before): Staging bugs 57	Production defects 6	Leakage 9.52%
Semester 2 (After): Staging bugs 88	Production defects 11	Leakage 11.11%

An additional period-based view was calculated to clarify defect leakage before and after the adoption of Shift-Left Testing. In semester 1, defect leakage was 9.52% based on 6 production defects out of a total of 63 identified issues (57 staging bugs + 6 production defects). In semester 2, defect leakage increased slightly to 11.11% based on 11 production defects out of 99 identified issues (88 staging bugs + 11 production defects). This period-based comparison indicates that, although more defects were identified earlier during development after shift-left adoption, post-release escape was not yet reduced and remained within the acceptable range. Therefore, the post-implementation effect is better interpreted as an improvement in early defect visibility rather than as a uniform improvement in all release-quality indicators.

Table 8. Bug Findings Before vs After Shift-Left

Period	Test scenarios	Bugs found
Semester 1 (Before)	142	57
Semester 2 (After)	155	88
Change (After vs Before)		+54.39%

To evaluate defect density more fairly, the study computed Bug Rate using KLOC normalization. With an estimated and comparable code-size basis across semesters, Bug Rate increased from 1.9 bugs/KLOC in semester 1 to 2.9 bugs/KLOC in semester 2. According to the study's benchmark categories, both values remain within the "Good" range. This implies that even though more bugs were detected post shift-left, the defect density did not cross into a problematic band; instead, it suggests that the development and testing process likely became more sensitive in surfacing defects while maintaining overall defect density within an acceptable maturity range.

Table 9. Bug Rate (Defect Density) Results

Period	Bugs	Estimated size (KLOC basis)	Bug rate
Semester 1 (Before)	57	30	1.9
Semester 2 (After)	88	30	2.9

From a testing-effectiveness lens, Defect Rate (defects relative to executed test cases) increased from 4.2% (6 defects out of 142 executed cases) in semester 1 to 7.1% (11 defects out of 155 executed cases) in semester 2. The study's interpretation classifies 4.2% as "Good" and 7.1% as "Acceptable." This shift indicates that the second semester carried higher defect exposure during execution, which can occur when release content becomes more complex, when regression scope expands, or when the team intentionally intensifies negative testing and edge-case validation. The result reinforces that shift-left adoption must be interpreted together with development complexity and test strategy changes, rather than solely as a simple "lower defects is always better" metric.

This interpretation is in line with recent empirical findings in software testing research. Wang et al. (2022) reported that higher test automation maturity is positively associated with better product quality and shorter release cycles, indicating that stronger testing capability does not necessarily appear first as fewer detected defects during development. Bauer et al. (2024) also found that richer support for GUI-based regression testing can improve testing efficiency, while Tran et al. (2025) emphasized that fault detection, reliability, and coverage are among the most important quality attributes of test cases and test suites. In a similar data-driven perspective, Madeyski and Stradowski (2025) showed that testing-process data can be used to predict defect-induced test failures and support earlier corrective feedback. Therefore, the increase in detected bugs in the present study is more plausibly interpreted as a sign of broader and earlier defect exposure than as direct evidence of declining software quality.

3.3. Software Quality Evaluation Indicators

The quality evaluation metrics provide a more nuanced picture of readiness and risk. Defect leakage was computed by comparing post-release defects against the total defects detected before and after release. Using the study's data (147 staging bugs and 17 production defects), the leakage rate was 10.37%. Under the

classification used in the study, this places leakage in the "Acceptable" band (10–30%), indicating a small but non-trivial amount of defect escape into production. Practically, these findings indicate that pre-release testing captured a substantial proportion of issues, but post-release escape remained present in both semesters and even increased slightly after implementation. Accordingly, the results suggest that Shift-Left Testing strengthened early defect visibility, yet additional attention is still required for high-risk areas, regression depth, and production-like environment parity, particularly on flows that are most used by students and teachers.

Table 10. Key Quality Indicators After Shift-Left Adoption

Indicator	Result	Interpretation
Defect leakage	10.37%	Acceptable; review high-risk areas
Test coverage	100%	All planned scenarios covered
Test execution rate	100%	All planned scenarios executed
Test case pass rate	50.51%	Low; indicates many failures during execution
Testing confidence	45.27%	Low; overall release confidence needs improvement

Two metrics show strong process discipline: test coverage and test execution rate were both reported at 100%, meaning planned scenarios were fully covered and executed. However, this strength is offset by a low-test case pass rate of 50.51%, implying that about half of executed cases did not pass cleanly during the evaluation window. This pattern is consistent with a context where the team runs comprehensive testing but faces instability from code changes, environment inconsistency, incomplete fixes, or overly complex test design that is sensitive to small variations. The testing confidence level, calculated at 45.27%, aligns with this interpretation: even with full coverage and execution, low pass rate and measurable leakage reduce overall confidence for production readiness. The practical implication is that Shift-Left Testing appears to have improved early defect visibility and testing process discipline, as reflected in full coverage and execution and in broader scenario exposure. However, the low pass rate, low testing confidence, and slightly higher semester-based leakage indicate that release stability was not yet uniformly improved. Thus, the findings support a qualified rather than blanket interpretation of quality improvement.

3.4. Questionnaire Results and Statistical Discussion

The survey instrument showed strong measurement quality. Validity testing using a pilot sample (n = 30) indicated that all items across Performance Expectancy, Social Influence, Facilitating Conditions, Behavioral Intention, and Use Behavior exceeded the r-table threshold (0.374), meaning the indicators reliably represented their intended constructs. Reliability testing further confirmed internal consistency, with Cronbach's Alpha values above 0.60 for all constructs (Performance Expectancy: 0.918; Social Influence: 0.858; Facilitating Conditions: 0.863; Behavioral Intention: 0.789; Use Behavior: 0.851). These results support the credibility of subsequent hypothesis testing because the constructs are both valid and reliable under standard criteria.

Table 11. Hypothesis Testing Summary (t-test)

Analysis Partial	t-value	p-value	Result
Performance Expectancy → Use Behavior	4.427	0.000	Significant
Social Influence → Use Behavior	4.528	0.000	Significant
Facilitating Conditions → Use Behavior	3.698	0.000	Significant
Behavioral Intention → Use Behavior	2.794	0.006	Significant

The partial-effect tests (t-tests) showed that each independent construct significantly influenced Use Behavior. Performance Expectancy ($t = 4.427$, $p < 0.001$), Social Influence ($t = 4.528$, $p < 0.001$), Facilitating Conditions ($t = 3.698$, $p < 0.001$), and Behavioral Intention ($t = 2.794$, $p = 0.006$) were all statistically significant. This implies that users' perceptions of usefulness, social encouragement, enabling infrastructure/support, and intention to use jointly shape actual usage behavior of Ruang Murid. In adoption-oriented interpretation, this is consistent with the idea that digital learning platforms are not only driven by functional value but also by social reinforcement (teacher/peer/parent) and practical access conditions (devices, connectivity, institutional support).

Table 12. F-Test result

ANOVA[a]						
Model		Sum of Squares	df	Mean Square	F	Sig.
1	Regression	775,972	4	193,993	96,480	,000[b]
	Residual	191,018	95	2,011		
	Total	966,990	99			

a. Dependent Variable: Use Behavior

b. Predictors: (Constant), Behavioral Intention, Performance Expectancy, Facilitating Conditions, Social Influence

The simultaneous-effect test (F-test) confirmed model significance ($F = 96.480$, $p < 0.001$), meaning the predictors jointly explain meaningful variance in Use Behavior. The coefficient of determination further indicates a strong explanatory model, with $R^2 = 0.802$ and Adjusted $R^2 = 0.794$. Practically, the model explains about 80.2% of the variance in Use Behavior, suggesting that the selected adoption constructs capture the primary drivers of usage in this dataset, while roughly 19.8% remains attributable to other factors not modeled (e.g., content quality, perceived enjoyment, system reliability during peak load, or institutional policy differences).

Connecting these survey findings back to the shift-left testing results offers a practical insight: higher use behavior and strong reliance on facilitating conditions imply that stability and availability are critical. When users strongly depend on the platform and enabling conditions are salient, production defects and leakage even at "acceptable" levels can disproportionately harm trust and continued use. Therefore, the results collectively suggest that implementing shift-left testing is strategically aligned with adoption drivers: improving early defect discovery, stabilizing critical user flows, and reducing production escapes can support the high explanatory adoption pathway evidenced by the UTAUT-based model. At the same time, the relatively low pass rate and low testing confidence highlight the next improvement frontier: strengthening environment fidelity, test design quality, and regression automation so that increased early detection translates into higher pass stability and lower leakage over time.

4. CONCLUSION

The implementation of Shift-Left Testing in the development of the Ruang Murid application produced several important outcomes. After adoption, detected bugs increased by 54.39%, indicating earlier and broader defect identification during development rather than simply poorer software quality. Although the bug rate rose from 1.9 to 2.9 bugs/KLOC and the defect rate increased from 4.2% to 7.1%, both remained within acceptable thresholds. In addition, the number of test cases increased to 155 scenarios, while test coverage and test execution each reached 100%. These findings indicate that the most evident improvements were stronger early defect visibility, broader scenario execution, and more complete testing coverage. However, the low test case pass rate of 50.51%, the testing confidence level of 45.27%, and the continued presence of defect leakage show that build stability, fix quality, regression robustness, and production-like environment parity still require further improvement.

The secondary analysis also shows that technical and social factors significantly influence use behavior in the Ruang Murid application. Performance Expectancy, Social Influence, Facilitating Conditions, and Behavioral Intention were all found to significantly affect actual use, collectively explaining 80.2% of the variance. This suggests that system stability and supporting conditions remain important not only for software quality control but also for sustained user adoption. Therefore, in addition to maintaining earlier testing integration, future improvement efforts should focus on strengthening release stability and user support to enhance both software reliability and continued platform use.

REFERENCES

Aljawarneh, S. (2020). Software quality assurance for e-learning systems. International Journal of Emerging Technologies in Learning, 15(1), 4–16. https://doi.org/10.3991/ijet.v15i01.11325

Alqurni, J. (2023). Assessing the usability of e-learning software among university students: A study on student satisfaction and performance. International Journal of Information Technology and Web Engineering, 18(1), 1–26. doi:10.4018/IJITWE.329198.

Alyoussef, I. Y. (2023). Acceptance of e-learning in higher education: The role of task-technology fit with the information systems success model. Heliyon, 9(3), e13751. doi:10.1016/j.heliyon.2023.e13751.

Amarta, A. A. F., & Anugrah, I. G. (2021). Implementasi agile scrum dengan menggunakan Trello sebagai manajemen proyek di PT Andromedia. Jurnal Nasional Komputasi dan Teknologi Informasi, 4(6), 528–534. https://doi.org/10.32672/jnkti.v4i6.3702

Aripiyanto, A., Muhyiddin, A., & Huda, B. (2021). Web-based teacher performance assessment information system using simple additive weighting method. Buana Information Technology and Computer Science (BIT CS), 2(2), 48–54. https://doi.org/10.36805/bit-cs.v2i2.1871

Atukunda, P., Khabusi, S. P., & Othieno, J. (2024). Analysis of user satisfaction of e-learning systems in Uganda using DeLone and McLean model. Discover Education, 3, 194. doi:10.1007/s44217-024-00304-6.

Awati, R. (2023, August 14). What is shift-left testing? TechTarget.

Bauer, A., Frattini, J., & Alégroth, E. (2024). Augmented testing to support manual GUI-based regression testing: An empirical study. Empirical Software Engineering, 29, 140. https://doi.org/10.1007/s10664-024-10522-z

Capgemini. (2021). World quality report 2021–2022. Capgemini, Sogeti, & Micro Focus. https://www.capgemini.com/research/world-quality-report-2021-22/

GeeksforGeeks. (2023). Shift left testing – Software testing.

Herlinda, H., Katarina, D., & Ambarsari, E. W. (2019). Automation testing tool dalam pengujian aplikasi belajar tajwid pada platform Android. STRING (Satuan Tulisan Riset dan Inovasi Teknologi), 4(2), 205. https://doi.org/10.30998/string.v4i2.5285

Hilabi, S. S. (2018). Analisis kualitas perangkat lunak terhadap sistem informasi STT Wastukancana Purwakarta. Jurnal Informatika, 1, 27–32.

IBM. (2023). What is shift-left testing? IBM.

Kavuri, S. (2024). Shift-Left and Shift-Right Testing Approaches: A Practical Roadmap for Continuous Quality in Agile and DevOps. Journal of Information Systems Engineering and Management, 9(4), 1-10. https://doi.org/10.52783/jisem.v9i4.127

López, L., Burgués, X., Martínez-Fernández, S., Vollmer, A. M., Behutiye, W., Karhapää, P., Franch, X., Rodríguez, P., & Oivo, M. (2022). Quality measurement in agile and rapid software development: A systematic mapping. Journal of Systems and Software, 186, 111187. doi:10.1016/j.jss.2021.111187.

Madeyski, L., & Stradowski, S. (2025). Predicting test failures induced by software defects: A lightweight alternative to software defect prediction and its industrial application. Journal of Systems and Software, 223, 112360. https://doi.org/10.1016/j.jss.2025.112360

Ministry of Testing. (2023). Shift-left testing.

Natarajan, T., & Pichai, S. (2024). Behaviour-driven development and metrics framework for enhanced agile practices in scrum teams. Information and Software Technology, 170, 107435. doi:10.1016/j.infsof.2024.107435.

Ruliansyah, R., Tukino, Huda, B., & Hananto, A. L. (2023). Application of software testing life cycle in automated testing of Dzikra platform. Computer Science Research and Its Development Journal, 15(1), 1–11. https://doi.org/10.22303/csrid-.15.1.2023.01-11

Singh, H., & Suri, B. (2020). Improving software testing efficiency using shift left testing approach. International Journal of Advanced Trends in Computer Science and Engineering, 9(2), 2140–2145. https://doi.org/10.30534/ijatcse/2020/47922020

Sumi, R. S., & Kabir, G. (2021). Satisfaction of e-learners with electronic learning service quality using the SERVQUAL model. Journal of Open Innovation: Technology, Market, and Complexity, 7(4), 227. doi:10.3390/joitmc7040227.

Tran, H. K. V., Ali, N. b., Unterkalmsteiner, M. (2025). Quality attributes of test cases and test suites-importance & challenges from practitioners' perspectives. Software Quality Journal, 33, 9. https://doi.org/10.1007/s11219-024-09698-w

Utami, & Setyodewi. (2023). Dokumentasi software testing untuk aplikasi DAFBIN berstandar IEEE 829-2008. Jurnal Restikom: Riset Teknik Informatika dan Komputer, 5(2), 107–117. https://doi.org/10.52005/restikom.v5i2.143

Vaddadi, S.A., Thatikonda, R., Padthe, A. et al. RETRACTED ARTICLE: Shift left testing paradigm process implementation for quality of software based on fuzzy. Soft Comput (2023). https://doi.org/10.1007/s00500-023-08741-5

Venkitaraman. (2024). Early bug detection through shift left testing. International Journal of Innovative Science and Research Technology, 9(11), 185–190. https://doi.org/10.38124/ijisrt/IJISRT24NOV177

Wang, Y., Mäntylä, M. V., Liu, Z., & Markkula, J. (2022). Test automation maturity improves product quality—Quantitative study of open source projects using continuous integration. Journal of Systems and Software, 188, 111259. https://doi.org/10.1016/j.jss.2022.111259

Zulianto, A., Purbasari, A., Suryani, N., Susanti, A. I., Rinawan, F. R., & Purnama, W. G. (2021). Pemanfaatan Katalon Studio untuk otomatisasi pengujian black-box pada aplikasi iPosyandu. Jurnal Edukasi dan Penelitian Informatika, 7(3), 370. https://doi.org/10.26418/jp.v7i3.46954