

Empowering Self-Learning: Automated Assistance and Grading Mechanisms in Android Application Development

Yan Watequlis Syaifudin*

Department of Information Technology, State Polytechnic of Malang, Malang, Indonesia

Email: qulis@polinema.ac.id

ORCID iD: <https://orcid.org/0000-0001-6582-3495>

*Corresponding author

Nobuo Funabiki

Department of Electrical and Communication Engineering, Okayama University, Okayama, Japan

Email: funabiki@okayama-u.ac.jp

ORCID iD: <https://orcid.org/0000-0003-3234-3473>

Andi Baso Kaswar

Department of Informatics and Computer Education, State University of Makassar, Makassar, Indonesia

Email: andi.baso.kaswar@unm.ac.id

ORCID iD: <https://orcid.org/0000-0002-2847-3526>

Asep Sunandar

Department of Education Administration, State University of Malang, Malang, Indonesia

Email: asep.sunandar.fip@um.ac.id

ORCID iD: <https://orcid.org/0000-0003-3292-7655>

Suryani Dyah Astuti

Department of Biomedical Engineering, Surabaya, Indonesia

Email: suryani@unair.ac.id

ORCID iD: <https://orcid.org/0000-0003-3000-0792>

Triana Fatmawati

Department of Information Technology, State Polytechnic of Malang, Malang, Indonesia

Email: triana@polinema.ac.id

ORCID iD: <https://orcid.org/0009-0002-3308-0345>

Mustika Mentari

Department of Electrical and Communication Engineering, Okayama University, Okayama, Japan

Email: p2918093@s.okayama-u.ac.jp

ORCID iD: <https://orcid.org/0000-0001-5469-9724>

Alfiandi Aulia Rahmadani

Department of Information Technology, State Polytechnic of Malang, Malang, Indonesia

Email: 1941170055@student.polinema.ac.id

ORCID iD: <https://orcid.org/0009-0001-3659-5638>

Received: January 7, 2025; Revised: May 24, 2025; Accepted: October 26, 2025; Published: August 8, 2026

Abstract: The increasing dominance of Android devices has driven a surge in demand for skilled mobile programmers, prompting educational institutions to incorporate mobile programming courses into their curricula. This trend, combined with the rise of self-learning platforms, highlights the need for innovative educational technologies that enhance programming instruction. While automated assessment systems have improved the grading process in programming education, there remains a gap in mobile programming education, particularly due to the complexities of

assessing Android applications. To address this, an innovative framework for Android application development is proposed, leveraging automated grading and assistance mechanisms. The framework employs a Test-Driven Development (TDD) approach, providing structured guidance and immediate feedback through automated testing tools including JUnit and Robolectric. A study involving 125 students revealed high engagement and success in basic topics, though challenges persisted in more complex areas, indicating a need for ongoing refinement and additional instructional support to elevate the learning experience in mobile application development.

Index Terms: Android Applications, Self-Learning, Learning Assistance, Assignment Grading, Automated Testing.

1. Introduction

The popularity of smartphones, particularly those that use the Android operating system, has revolutionized the digital environment and markedly impacted the development of mobile applications, with Android devices representing approximately 72.04% of the global market share of mobile operating systems in the fourth quarter of 2024 [1]. The widespread adoption has resulted in a significant increase in the demand for proficient mobile programmers [2], leading many universities and professional institutions to incorporate mobile programming courses into their IT curriculum to provide students with vital development abilities. The increasing interest in mobile application development among professionals and hobbyists has generated a significant need for accessible self-learning platforms that support the acquisition and improvement of programming skills outside of conventional educational systems [3]. The convergence of Android's market supremacy, the increased demand for mobile developers, the integration of mobile programming into formal education, and the rise of self-directed learning underscore the imperative for innovative and user-friendly platforms that proficiently facilitate the education and skill enhancement of prospective mobile application developers [4].

Educational technology and automated systems have significantly transformed computer programming education, particularly through advancements in automated assessment, feedback generation, and learning analytics [5]. Automated assessment platforms have streamlined grading processes, allowing for more practical coding exercises, yet challenges such as the quality of feedback and potential biases from predetermined test cases remain. Techniques like program repair for automated feedback have demonstrated effectiveness in addressing common misconceptions [6, 7]. Learning analytics is becoming increasingly vital for understanding student behavior, informing instructional design, and creating more adaptive learning environments [8]. Automated grading systems are essential for efficient assessment in programming courses, delivering immediate feedback that aids learning, but often lack the depth needed for comprehensive feedback [9, 10]. Thus, there is a pressing need for an integrated approach that combines automated assessments with comprehensive assistance to foster deeper learning, enhance feedback quality, and improve educational outcomes.

Recent advancements in programming education have been driven by adaptive learning platforms, hands-on experiences, and online adaptability [11]. The integration of automated learning assistance and grading has further improved the effectiveness of programming education by providing personalized support and immediate feedback [12]. However, a critical gap remains in mobile programming education, as current platforms lack sophisticated automated features available in general programming environments that underscore the need for self-learning systems. Assessing mobile programming assignments, particularly for Android applications, is complex due to their interactive nature, requiring proficiency in programming languages such as Java or Kotlin, XML for layout, and Gradle for build management [13, 14]. Transforming an Android project into an executable application involves intricate steps that make manual grading labor-intensive and time-consuming. Teachers must also evaluate individual components and their integrations, complicating the assessment process. Educational frameworks such as MOOCs and LMS, while valuable, lack the essential automated assistance and classification features that enhance self-learning experiences [15, 16].

This paper introduces automated assistance and grading mechanisms within an innovative learning framework for Android application development. The framework emphasizes a computer-assisted learning approach that prioritizes hands-on programming practice, enhanced by the Test-Driven Development (TDD) methodology [17]. This model equips students with comprehensive learning materials that outline coding and application development processes, featuring structured guidance and test suites to verify code correctness. By integrating TDD, the framework allows automated code verification, providing students with immediate feedback through an iterative process that fosters skill improvement. Upon completing their assignments, students submit their projects, which are evaluated by an automated grading system that reflects results from the associated test suites, thus promoting independent learning and efficient performance assessment. Additionally, the framework outlines a robust automated testing strategy that includes unit testing, integration testing, and user interface testing, utilizing tools such as JUnit and Robolectric to validate code functionality and user experience. The code verification model ensures the robustness of the software through a structured process, where tests assess various components of the application, providing constructive feedback on failure.

The automated assistance mechanism utilizes JUnit-based testing, where tests are organized in structured classes with methods marked by the '@Test' annotation, allowing students to run various assertions to validate code correctness. If an assertion fails, informative feedback helps to identify discrepancies. Robolectric complements this by simulating an Android environment to test components and user interface behaviors using standard assertions. Together, these frameworks provide immediate feedback, enabling students to efficiently address issues and improve software reliability. The grading system employs a weighted average, which accounts for multiple test case outcomes, each assigned a weight based on its significance. This structured approach enhances student learning through tailored feedback while the validator program validates submitted Android projects, executes integrated test suites, and calculates overall grades from test results.

The evaluation involved 125 students from two mobile programming classes across two university programs, focusing on five learning topics: Basic Application, Advanced Widgets, Multiple Activities, Multimedia Resources, and Basic Data Storage. A total of 625 submissions were collected and validated in two weeks through a fully automated system, demonstrating seamless processing with only minimal discrepancies during assessment, thus highlighting the robustness and reliability of the validator program. Students received immediate and detailed feedback based on the execution of the test suite using *JUnit* and *Robolectric*, allowing iterative learning and significant improvements in foundational topics that are evidenced by high average grades (e.g. 99.86% for Basic Application). Student feedback was largely positive, with only a small percentage reporting technical issues such as IDE performance problems and version discrepancies. The system significantly reduced teacher workload by automating functional checks that traditionally required over 30 hours of manual effort, while also supporting targeted interventions based on common errors identified through test results. Despite its success, areas for improvement were identified, particularly in more complex topics like Multimedia Resources and Data Storage, where students requested clearer guidance and multilingual support. Overall, the framework proves effective in promoting self-directed learning and efficient assessment, though future work should include longitudinal studies to assess long-term concept retention and controlled comparisons with traditional grading methods.

The rest of this paper is organized to systematically present the design, implementation, and evaluation of the proposed automated assistance and grading framework. Section 2 provides a comprehensive review of related work in educational technology, automated grading systems, and challenges specific to mobile programming education. Section 3 details the design of the learning framework, including the integration of Test-Driven Development (TDD), automated testing strategies, and the code verification model. Section 4 outlines the mechanisms of automated assistance and grading, describing the validator program, test case design, and feedback generation process. Section 5 presents the evaluation results, including student performance metrics, feedback analysis, and technical outcomes. Section 6 discusses the findings, limitations, scalability, and future improvements of the framework. Finally, Section 7 concludes the paper by summarizing the contributions and emphasizing the potential impact of the framework on mobile programming education.

2. Contextual Review

This section presents an overview of how advancements in educational technology and automated systems have transformed programming education through improved assessment methods, feedback mechanisms, and learning analytics, while also addressing the existing gaps in mobile programming education and emphasizing the need for further development of automated learning tools and frameworks.

2.1. Educational Technology and Automated Systems in Programming Education

Educational technology and automated systems have significantly advanced computer programming education in recent years, particularly in automated assessment, feedback generation, and learning analytics. Automated assessment systems such as *Online Judge (OJ)* platforms have streamlined the grading process and allowed students to engage in more practical coding exercises [18]. However, challenges persist with respect to feedback quality and potential unfair grading due to the reliance on predefined test cases [19, 20]. Automated feedback generation techniques, such as program repair, have shown promise in providing constructive feedback and addressing common misconceptions [21]. Integration of Git log analysis into these systems allows tracking student progress over time, offering insights into individual learning trajectories [22].

Learning analytics has emerged as a crucial tool for understanding student behavior and performance in programming courses. By analyzing data from automated assessment tools and learning management systems, educators can gain valuable insights to inform instructional design and tailor teaching methods to student needs [23, 24]. The combination of learning analytics with automated assessment systems has the potential to create more responsive and adaptive learning environments, ultimately improving educational outcomes in programming education [7]. However, concerns remain regarding the implementation and effectiveness of these automated systems, including potential biases in assessment and the need for human judgment to evaluate complex programming tasks [25, 26].

While the integration of educational technology and automated systems in computer programming education offers numerous opportunities, it also presents challenges that require ongoing research and refinement. The effectiveness of

these tools depends on thoughtful design and continuous improvement to ensure alignment with pedagogical best practices. Future research should focus on optimizing automated systems to deliver *automated assistance* and support personalized learning experiences [6], thereby maximizing their impact on programming education. As these technologies continue to evolve, their potential to enhance the efficiency of assessment, provide valuable feedback, and support adaptive learning experiences in programming education remains significant.

2.2. Grading in Programming Courses

Automated grading systems have become a cornerstone of programming education, offering efficient and timely evaluations of student submissions by compiling code, executing it against predefined test cases, and comparing outputs to expected results [20]. These systems provide immediate feedback, crucial for learning in programming contexts, and reduce grading time [27]. Additionally, cloud-based tools for tracking coding activities have emerged as a method to ensure fair evaluations and detect potential academic dishonesty [28, 29]. The effectiveness of these systems is influenced by the type of feedback they provide, with research indicating that feedback generation tools vary significantly in their adaptability and the types of feedback offered [27, 30]. This variation is particularly important in programming education, where understanding underlying concepts is crucial for student success.

While automated grading systems offer numerous advantages, concerns have been raised about the depth and quality of feedback they provide. Some systems may fail to provide comprehensive feedback that addresses common misconceptions among students, leading researchers to advocate hybrid approaches that combine automated assessments with human oversight [31, 32]. The implications of mandatory assignments on student performance have also been examined, with findings suggesting that while such assignments can enhance learning outcomes, the resources spent on grading could potentially be better utilized for providing formative feedback [31]. This highlights the need for a balanced approach that prioritizes both assessment and meaningful feedback to foster deeper learning. As grading mechanisms in programming courses become increasingly reliant on automated systems, technological advancements should continue to explore the integration of grading automation and personalized feedback to enhance student learning outcomes. The effectiveness of these systems depends on the quality of feedback provided and the integration of human oversight to address the complexities of programming education.

2.3. Comparison with Existing Platforms

The landscape of mobile programming education is evolving alongside the growth of Massive Open Online Courses (MOOCs)[33] and innovative digital platforms [34]. Although platforms dedicated to mobile programming education remain limited, several existing MOOCs have adapted to encompass the needs of learners in this space. Both traditional and online educational platforms have demonstrated gaps in automated assistance and assignment grading, which are crucial for enhancing the learning experience.

Traditional MOOCs, such as *Coursera* and *edX*, primarily focus on a broad array of topics, often providing in-depth courses in programming languages. However, they have not fully integrated advanced automated assistance or grading systems tailored to individual learning paths. Research highlights that while MOOCs can effectively widen participation in education, their current grading mechanisms often lack the nuance needed for programming tasks, which may involve complex logic and coding challenges that automated systems struggle to accurately evaluate [35, 36]. Furthermore, solutions like "*Codeboard*," which allow programming directly in browsers, have been identified as promising yet underutilized tools that can enhance learners' engagement through interactive practices [37]. These platforms require more robust frameworks to provide feedback that is critical in programming education, which traditionally benefits from immediate and iterative feedback cycles.

Moreover, the exploration of platforms dedicated to mobile programming reveals a significant gap in providing comprehensive support for learners. A study by Zancanaro et al. indicates that the evaluation of existing MOOC platforms often overlooks essential features necessary for effective programming instruction [38]. For example, many courses that claim to teach fundamental programming skills do not adequately address the demand for mobile-specific content [39]. This is corroborated by the investigation of platforms such as Sololearn, which was selected primarily for its focused mobile programming offerings but still falls short when compared against more established educational resources [39]. In summary, while the integration of mobile programming education within the continuing evolution of MOOCs is progressing, significant gaps remain in the areas of automated assistance and assignment evaluation. The landscape requires innovative solutions that not only enhance platform interactivity but also ensure effective evaluation methodologies to benefit diverse learner needs in programming education.

2.4. Research Significance

The transformation of programming education has been significantly influenced by recent advancements in learning platforms that introduce adaptive learning[9], hands-on experiences [40], and online adaptability [41], leading to improved problem-solving skills and motivation. The integration of automated learning assistance and grading has further increased the effectiveness and efficiency of programming education by providing personalized assistance support and immediate feedback. Despite these improvements and the rise of mobile devices, a critical gap remains in mobile programming education, as existing platforms for mobile application development lack the sophisticated

automated features found in general programming environments. This gap underscores the urgent need for comprehensive self-learning systems in mobile programming that incorporate these advanced tools, addressing the demand for skilled mobile developers while maximizing the benefits of automated learning technologies.

Assessing mobile programming assignments, particularly for Android applications, presents significant complexity due to the inherently interactive nature of these applications, which incorporate numerous user interface (UI) and user experience (UX) components. An Android project requires proficiency in multiple programming languages, such as Java or Kotlin for programming logic, XML for layout designs, and Gradle for project build management [42]. The process of transforming an Android project into an executable application involves several intricate steps, starting with the Android Asset Packaging Tool (AAPT) which packages the compiled resources, followed by the Java compiler that compiles the source code into bytecode, and then the DEX compiler that converts the bytecode into Dalvik bytecode for execution on the Android runtime [43]. Finally, the APK builder packages everything into an Android Application Package (APK) file. This multistep compilation and packaging process requires considerable time, often taking teachers at least 20 minutes for each submission to manually build and verify the project [44]. Moreover, the teacher must check individual application components and evaluate their integration and interaction within the overall application, leading to a manual assessment process that is not only labor-intensive, but also time-consuming and challenging. Such complexities substantially hinder the efficiency of grading in mobile programming classes, making it difficult for educators to provide timely and effective feedback to students.

Despite the widespread adoption of mobile devices and the increasing demand for skilled mobile programmers, research on frameworks explicitly designed to support education in the development of mobile applications remains limited. Notable examples such as the *Android Programming Learning Assistance System (APLAS)* [45] and the *MIT App Inventor* [46] provide valuable resources and tools for learners, with APLAS focusing on assisting users in mastering the intricacies of Android development through targeted guidance and materials, while MIT App Inventor enables users, including those with minimal programming experience, to create mobile applications through a user-friendly visual interface. However, a significant gap exists in these learning frameworks regarding the integration of automated assistance and automatic classification features, which are crucial for improving students' self-learning experiences in mobile application development. The absence of such features hinders students' ability to study Android application development independently, as automated assistance could guide them in practicing writing mobile application code and providing feedback on their work. Furthermore, automated grading systems could offer scores for programming assignments, facilitating learning evaluation by teachers.

3. Design of Learning Framework

This section explains the design of a learning framework that incorporates a Test-Driven Development (TDD) approach, emphasizing hands-on practice, automated testing for code verification, and structured feedback mechanisms to enhance students' programming skills and performance assessment.

3.1. Learning Model

Defining a clear learning model is essential in the development of a self-learning framework as it establishes the pedagogical foundation, guides instructional design, and ensures structured, effective integration of automated tools to support student engagement, comprehension, and independent problem-solving in complex subjects like Android programming.

3.1.1. Learning Model Overview

The proposed learning model for Android programming is based on the computer-assisted learning approach that emphasizes hands-on programming practice as a key component of the educational framework [47]. This model is designed to provide students with comprehensive learning materials that include guidance on coding and application development. A central aspect of this framework is the adoption of the Test-Driven Development (TDD) methodology [17], which promotes experiential learning by allowing students to engage with the coding process through structured instructions. The learning materials are designed to include guidance that outlines step-by-step procedures for coding, in addition to test suites designed to verify the correctness of the student's written code, as shown in Figure 1.

3.1.2. Integration of Automatic Code Verification

The integration of TDD within this educational framework enhances the learning experience by incorporating automatic code verification, which provides instant feedback as students work through their assignments. As students write code in the default Integrated Development Environment (IDE) for Android development, they utilize the bundled test suite to assess their work [48], facilitating a proactive learning approach. Upon executing the verification process, students receive a detailed checklist that highlights successful outcomes as well as failed test cases. This *automated assistance* mechanism is crucial, as it enables students to identify specific areas that need improvement and offers guidance to assist them in refining their code. This iterative process fosters an environment in which students can continuously learn and improve their programming skills by tackling practical coding challenges.

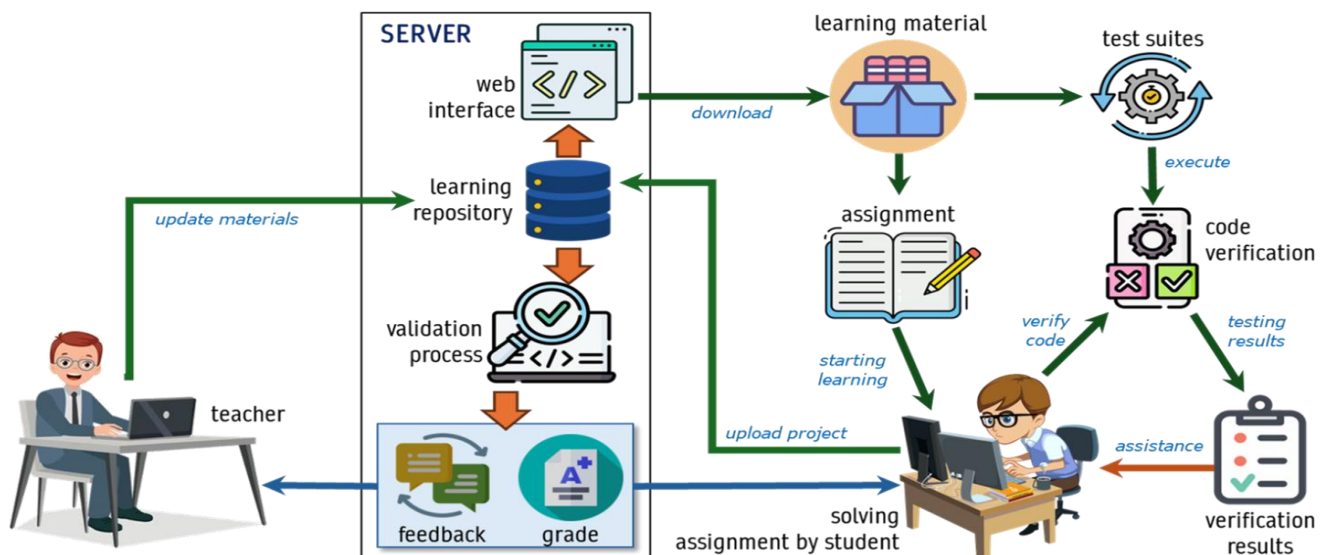


Fig. 1. The proposed learning model for Android programming.

Upon completion of their assignments, students submit their Android projects through a web application, which leverages an *automated grading* function to evaluate performance based on the results obtained from the test suite. This automated process not only promotes independent learning by encouraging students to take ownership of their educational progress but also provides educators with effective tools for assessing student performance. The results generated from this automated system provide constructive feedback for both students and teachers, facilitating targeted interventions as needed. Ultimately, the combination of hands-on programming practice, TDD methodology, and automated feedback systems cultivates an enriching learning environment that equips students with the essential skills necessary for successful mobile application development.

3.1.3. Alignment with Learning Objectives

In addition to promoting good coding practices, the TDD-based framework aligns with essential learning objectives such as writing modular and maintainable code, understanding the Android application lifecycle, and mastering UI component validation. These competencies are crucial for developing robust mobile applications and are especially beneficial in self-directed learning environments. The structured guidance and automated feedback embedded in the framework enhance student engagement and comprehension [49, 50], while also supporting independent problem-solving skills. Overall, the integration of TDD into the learning model not only improves code quality but also deepens students' understanding of software development principles, making it a valuable tool for effective Android programming education.

3.2. Automated Testing for Code Verification

The integration of Test-Driven Development (TDD) in Android application development requires the incorporation of automated testing as an essential element for code validation. Android applications, characterized by their interactive and user-centric design, require programmers to employ various languages and tools, including Java, Kotlin, XML, and Gradle. Due to the complexity of these applications, a strong and thorough testing framework is vital to accurately assess their functionality, appearance, and user experience. This framework generally includes three specific levels of testing: unit testing, which verifies individual code components; integration testing, which evaluates the interaction among various components; and user interface testing, which confirms the application's intuitiveness and accessibility for users.

In the context of automated testing for Android applications, JUnit and Robolectric emerge as two of the most trusted tools to facilitate effective code validation [51]. JUnit provides a structured framework designed specifically for unit testing, allowing developers to conduct precise evaluations of isolated segments of code. Conversely, Robolectric enhances the integration testing process by simulating real device environments, which is crucial to validate how components function in conjunction. This includes assessing integrated elements such as layouts, resources, activity lifecycles, and event listeners, as shown in Figure 2.

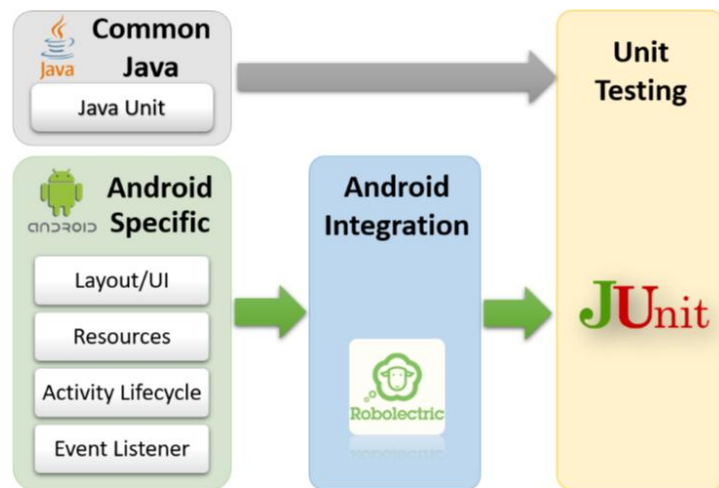


Fig. 2. Unit testing and integration testing for Android applications.

Within a self-learning model, students engaged in programming via Android Studio can effectively leverage the capabilities of both JUnit and Robolectric to gain immediate and constructive feedback on their coding initiatives. Automated testing processes can be run on a server, allowing the validation of submitted student code without the need for physical device demonstrations. This dynamic fosters an efficient learning environment, as students can iteratively refine their code based on the feedback generated from automated tests. Furthermore, the results obtained from these testing processes can seamlessly translate into assignment grades, streamlining the assessment framework, and supporting a structured approach to learning mobile programming. This integration of automated testing enhances the learning experience and prepares students for the complexities of real-world software development.

3.3. Code Verification Model

The code verification model is designed to ensure the robustness and functionality of software applications through a structured verification process containing:

1. **Initiation phase**, where the activity is constructed and necessary fields are assigned to establish a foundation for subsequent validations.
2. **Validation of components** is performed using a combination of integration and unit tests, specifically focusing on layout validation, resource validation, activity lifecycle validation, and event listener validation through integration testing. In parallel, unit tests are used to assess the functionality of individual Java classes in isolation.
3. If a test fails, an appropriate failure message is displayed, providing constructive feedback that assists students in identifying problems and improving their code.

Test cases play a central role in testing by specifying the inputs, expected results, and assertions. By implementing test cases, the code can be systematically verified for correctness and reliability. In both JUnit and Robolectric, test cases are established as methods marked with the '@Test' annotation and include assertions to verify if the expected behavior of a code unit aligns with the actual behavior. The outcomes of executing these test cases include the following:

1. **Pass**: This occurs when the software's actual behavior corresponds to the expected behavior, indicating that the functionality being tested is working correctly.
2. **Fail**: This occurs when the software's actual behavior differs from the expected behavior, pointing to an error that needs to be fixed.
3. **Error**: This happens when a test case cannot run due to an issue unrelated to the software being tested.

When a test results in a Fail or Error, it provides a message as feedback, which is used for automated support features.

In Android applications, ensuring the correctness and functionality of the application involves several validation processes, including:

- **UI Layout Validation:** Android UI is constructed using XML layouts containing various elements like Buttons and EditTexts, each identified by unique numbers and characterized by properties such as width, height, and background color. UI layout validation involves checking the completeness and sequence of layout elements, followed by verifying the correctness of each property's value.
- **Resource Validation:** Resource files separate static values from source code. Key resources to validate in the Unit Converter application include strings, colors, string arrays, fonts, images, and drawables. Each resource's availability and value correctness must be confirmed.
- **Java Class Validation:** Java classes consist of multiple methods, each with specific purposes. Their validity is verified through a method structure check (ensuring correct method names and parameters) and functionality check (ensuring all test methods pass).
- **Activity Lifecycle Validation:** The Activity Lifecycle outlines the states an Activity transitions through, from creation to destruction, including stages like *onCreate*, *onStart*, and *onDestroy*. Robolectric can test the proper progression through these lifecycle states.
- **Event Listener Validation:** Event listeners act as interfaces for UI elements, responding to user interactions. Validation involves generating events from UI elements (e.g., simulating button clicks) and comparing the resulting values with expected outcomes to ensure correct behavior.

4. Automated Assistance and Grading Mechanisms

This section outlines automated assistance and grading mechanisms designed to enhance the verification process for student coding assignments in Android programming by using structured testing frameworks, providing detailed feedback, and calculating grades based on the weighted importance of various test cases.

4.1. Code Verification Model

The automated assistance mechanism within testing frameworks is specifically designed to enhance the verification process by comparing actual results with expected results. In this context, tests are organized into specific groups with designated methods annotated for execution. When these tests are conducted, a variety of assertion methods are employed to assess whether the outputs align with predefined criteria. A test suite exemplifies this approach by structuring test files into organized classes, each containing one or more test methods marked with the '@Test' annotation, namely test cases, as shown in Figure 3. This framework provides a robust set of assertion methods, such as 'assertEquals()' and 'assertNotNull()', which allow for a direct comparison between actual results and expected results. If an assertion fails, informative messages are generated that provide context regarding the expected versus actual values, thereby assisting users in understanding the reasons for failures and guiding necessary corrections.

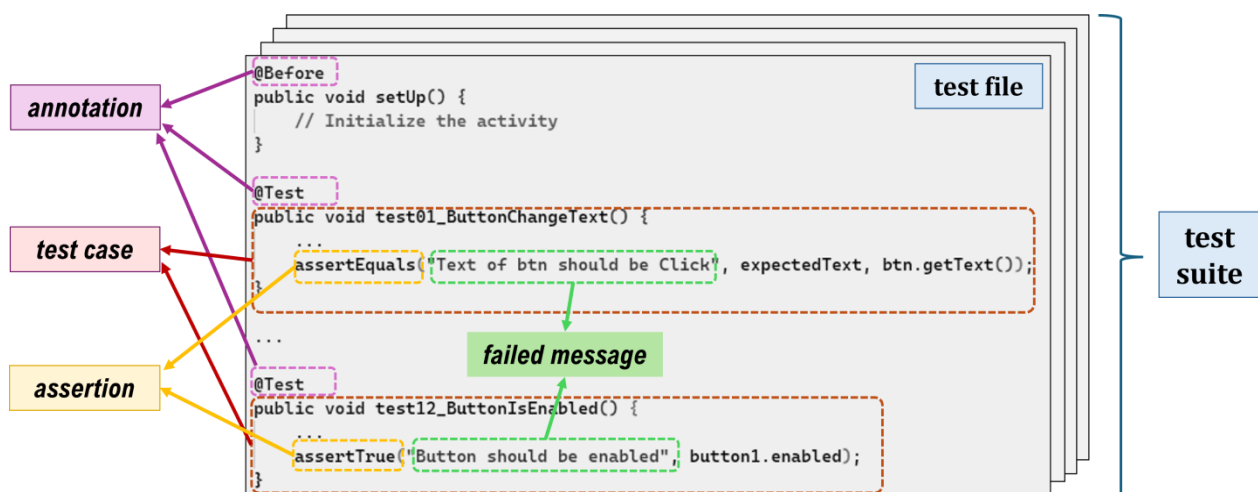


Fig. 3. A set of test files containing test cases.

Robolectric complements the JUnit framework by creating a simulated environment tailored to test Android-specific components and behaviors. Using standard JUnit assertions along with additional APIs, Robolectric enables interactions with the Android framework and allows for the simulation of various user scenarios. The tests constructed in Robolectric resemble those in JUnit, with a focus on verifying the behavior of application elements, such as checking that the content in a user interface component is correctly displayed. When assertions fail within Robolectric tests, clear messages are provided to inform users of the discrepancies encountered. Together, JUnit and Robolectric enhance the effectiveness of the testing process, improving the overall reliability and quality of software applications. By providing

immediate feedback and a structured learning process, these frameworks support users in identifying and resolving issues efficiently while fostering a more robust development environment.

Figure 4 displays example results from students' learning processes, illustrating the automatic feedback provided for code correction. In this scenario, a student creates XML code to specify the application's UI layout and then runs a test code related to UI component validation to confirm the code. An explanation of the feedback's findings is provided below:

- Incorrect code was found in the string resource configuration in the file 'string.xml'. The value 'APLAS SOCCER BOARD' in the 'app_name' resource should be 'Unit Conv'.
- The value of an ImageButton property, 'scaleType', is incorrectly coded in the file 'main.xml'. The student wrote 'FIT_CENTER' when it should have been 'FIT_END'.
- The text of a TextView with ID 'mainTitleTxt' still has an incorrect value in the file 'main.xml'. The student wrote 'APLAS [FOOTBALL MANAGER]' instead of 'CONVERTER'.

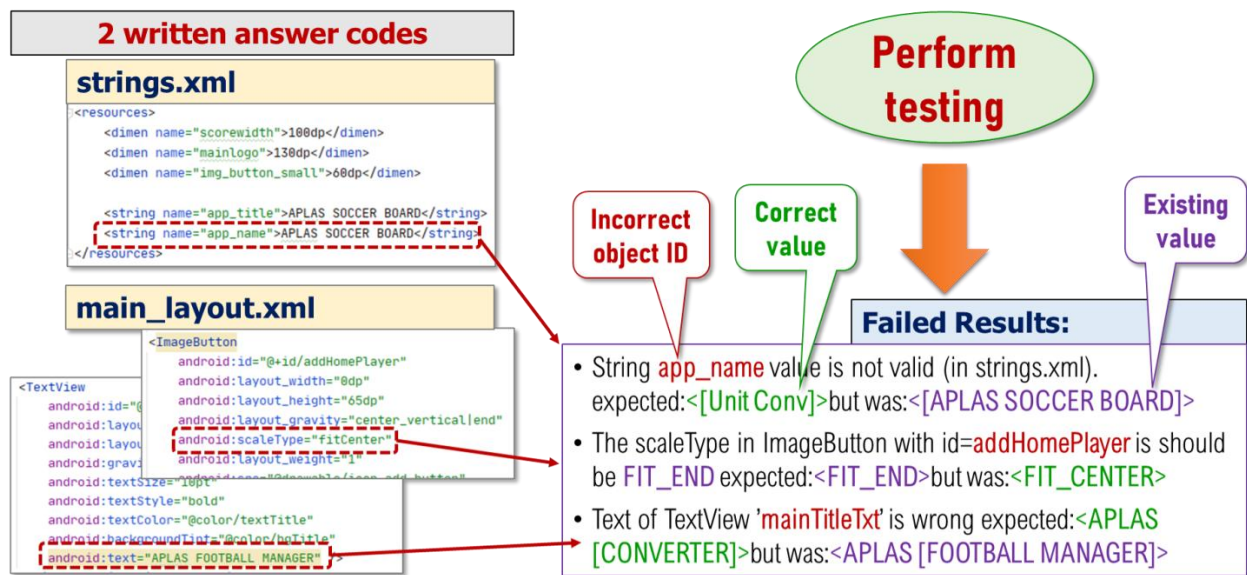


Fig. 4. Example of automated assistance on student code.

4.2. Assignment Grading

As a learning framework, it is important to accurately assess student performance through automated testing methodologies. Traditional grading approaches often rely on simple binary outcomes, such as pass or fail, without considering the complexity or significance of individual test cases. This lack of differentiation can lead to misleading evaluations of a student's capabilities and understanding. In programming contexts, assessments typically involve a variety of test cases that may differ significantly in their relevance and difficulty. The equation presented in Equation 1 is designed to calculate a weighted average grade based on the results of multiple test cases, where ($tc_result_idx_i$) indicates the outcome of the (i)-th test case, valued at 1 for pass and 0 for fail or error. A key feature of this equation is the incorporation of weights for each test case, which assigns different levels of importance to individual test cases based on their relevance to the overall learning objectives.

$$Grade = \frac{\sum_{i=1}^n (tc_result_idx_i \times tc_weight_i)}{\sum_{i=1}^n tc_weight_i} \times 100 \quad (1)$$

The equation is defined as follows:

1. **Numerator:** The numerator $\sum_{i=1}^n (tc_result_idx_i \times tc_weight_i)$ computes the total weighted score by summing the contributions of each test case. Each test case only contributes its weight to the total score if it passes; if it fails, its contribution is zero, thereby ensuring that the grade reflects only successful outcomes.
2. **Denominator:** The denominator $\sum_{i=1}^n tc_weight_i$ calculates the total weight of all test cases, acting as a normalization factor. This ensures that the final grade accurately reflects the relative importance of each test case in the context of the overall assessment.
3. **Percentage Conversion:** The product of the ratio is multiplied by 100 to express the score as a percentage, facilitating a clearer interpretation of the student's performance in alignment with traditional grading systems.

4.3. Validator Program

The web-based platform requires the validator program to automatically evaluate student-submitted Android projects by executing predefined test suites, generating detailed feedback, and calculating weighted grades, ensuring consistent, efficient, and educational assessment that supports both learning and teaching processes.

4.3.1. Functionalities

The web-based platform, constructed using the Laravel framework for the server-side, serves as an interactive interface for both students and teachers, fostering an environment conducive to collaborative learning and effective teaching. Through this platform, students have access to learning materials. Upon completing their assignments, students can submit their developed Android applications directly to the server via the web application. The Java-developed *validator program* then validates these submissions and generates comprehensive feedback along with grades based on predefined criteria. It runs as a background process on *Ubuntu Linux* operating system and takes a structured approach to validate submitted Android projects. Figure 5 explains the process in the validator program, which begins with the program acquiring the submitted Android project files, followed by the integration of relevant test suites, which involves extracting the project and incorporating the associated test cases designed for validation. Once the test suites are integrated, the program executes them via the command prompt by using Gradle as the build automation tool, utilizing JUnit and Robolectric to conduct comprehensive testing. To execute integration testing, the validator program executes a command such as `./gradlew test --tests org.aplas.BasicApp.Test1` to run the defined test cases in the test file `Test1.java`.

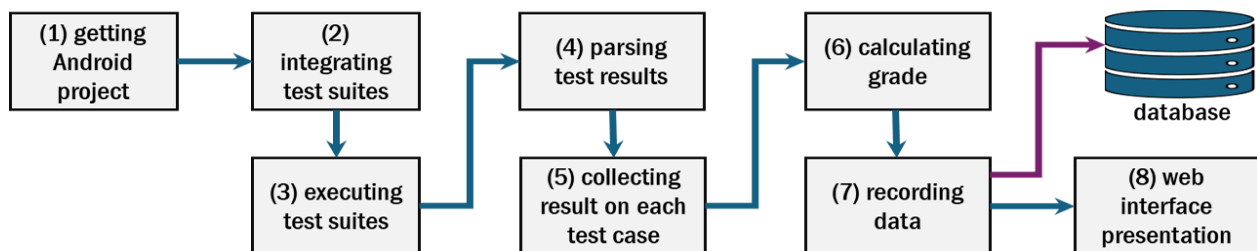


Fig. 5. Workflow of the validator program.

4.3.2. Technical Process

Subsequently, the validator program parses the test results to capture each test case result, including passed or failed results along with feedback messages, as shown in Figure 6. Data regarding the status of each test case result is systematically collected and stored in a MySQL database, ensuring accurate tracking of all testing outcomes. The program then consolidates these validation results to calculate an overall grade for each submission of the student. Each test case is associated with a specific level category outlined in Table 1, which determines its weight based on the complexity and relevance of the test. Using Equation 1, the validator program calculates the student's grade by factoring in the weighted contributions of each test case. Finally, the results are presented through a web interface, granting both students and teachers access to the validation results and grades, thereby enhancing the educational feedback loop.

Table 1. Weights for five levels of test cases.

Code	Level	Description	Weight
L1	Basic	<i>Setting Up the Development Environment and Starting Project:</i> Install Android Studio, configure the Android SDK, create a new Android project, and explore the project structure, including Gradle files and manifest configuration.	10
L2	Elementary	<i>Designing UI and Resources with XML:</i> Learn to design user interfaces using XML layouts, implement various UI components (TextViews, Buttons, ImageViews), and manage resources using colors, strings, and drawable files for a consistent design.	15
L3	Intermediate	<i>Navigation, Activities, and Events:</i> Understand the Activity lifecycle, implement navigation using Intents, create multiple Activities, and manage user interactions by handling events through OnClickListener and other event listeners.	0.7886
L4	Proficient	<i>Libraries, Basic Data Storage, and Dynamic UI/UX Elements:</i> Explore and integrate third party libraries, utilize SQLite databases and SharedPreferences, and create dynamic UI elements such as RecyclerView or CardView.	0.1422
L5	Advanced	<i>API Integration, Third-Party Libraries, and Complex Widget:</i> Integrate APIs for data retrieval using libraries, handle JSON parsing, and manage complex UI components and widgets, including custom views, animations, and multi-part forms.	0.1822

4.3.3. Handling Incomplete Submissions

The validator program handles incorrect or incomplete submissions constructively. When students submit Android projects that fail one or more test cases, the validator program provides specific feedback indicating which parts of the code did not meet expectations. For example, if a required UI component was missing or an event listener was incorrectly implemented, students receive targeted messages explaining the issue and suggesting corrective actions. In cases where submissions were incomplete (e.g., missing files or unimplemented methods), the system returned descriptive error messages rather than simply marking the submission as failed, guiding students toward completion before reattempting validation. This approach encourages iterative learning and ensures that students understand their mistakes rather than merely receiving a binary pass/fail result.

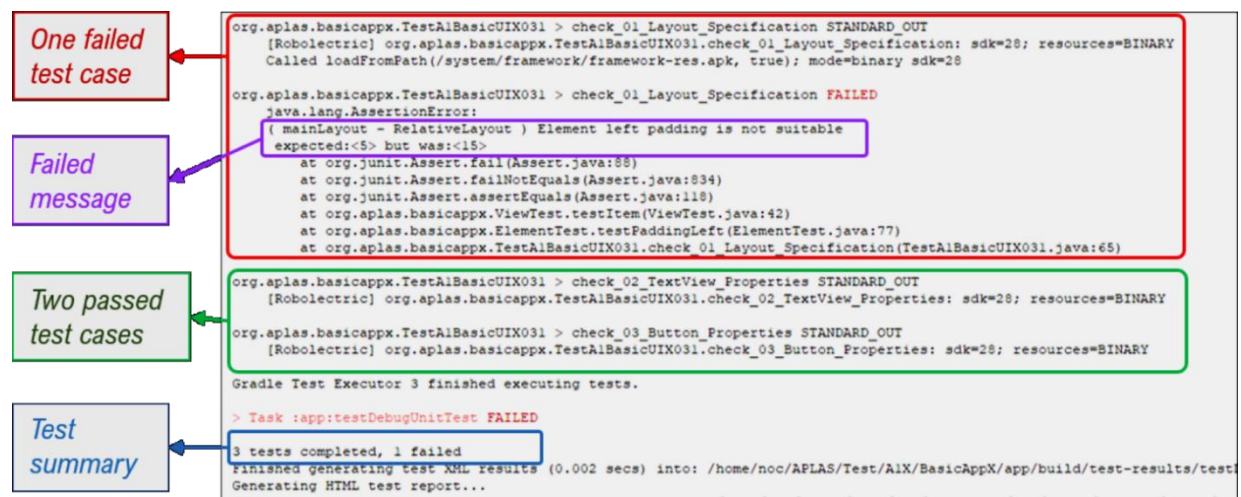


Fig. 6. Gathering passed and failed test cases with message from test results.

4.4. Design and Validity of Test Cases

The test cases integrated into the automated grading system were systematically designed to assess students' mastery of key Android programming concepts. Each test case was developed using a structured approach grounded in educational best practices and Android development standards.

4.4.1. Design Methodology

Each test case in the automated grading system corresponds to a specific component of Android application development, ensuring comprehensive evaluation across key areas such as *UI Layout Validation* (checking correct XML structure, element IDs, and property values), *Resource Validation* (verifying proper use of strings, colors, and drawables), *Java Class Logic* (validating method structures, object instantiations, and functional correctness), *Activity Lifecycle Management* (testing transitions through 'onCreate()', 'onStart()', 'onPause()', and 'onDestroy()'), and *Event Handling* (confirming that user interactions produce expected outcomes). These test cases were implemented using JUnit for unit-level checks and Robolectric for integration-level behavior simulation, providing a robust framework for validating both individual components and their interactions within the application[45].

4.4.2. Validity and Alignment with Learning Objectives

Each test case was carefully aligned with the learning outcomes of the five topics to ensure validity and instructional relevance. For example, in *Basic Application*, test cases assessed setup procedures, UI creation, and basic event handling, while in *Multimedia Resources*, tests focused on audio/video playback, animation triggers, and media lifecycle management. This alignment ensured that assessments accurately reflected instructional goals and provided students with meaningful, targeted feedback.

4.4.3. Expert Review and Refinement

All test cases were reviewed by two experienced Android developers and two university teachers who specialize in the education of mobile programming. Their feedback was used to refine test descriptions, adjust complexity levels, and eliminate ambiguities. Additionally, during the implementation phase, student performance data and feedback were analyzed to identify areas where test cases needed clarification or enhancement.

4.4.4. Comprehensive Coverage across Topics

Table 2 shows the distribution of test cases across topics and complexity levels (L1–L5), demonstrating progressive skill development from foundational to advanced levels. This tiered structure ensures that students build competence incrementally while being exposed to realistic development challenges.

5. Evaluation on Implementation Results

This section evaluates the implementation of the learning framework involving 125 students and analyzes their performance and feedback, demonstrating the effectiveness of automated assistance in enhancing self-learning and identifying areas for improvement in the complexity of topics covered.

5.1. Evaluation Setup

This study evaluates the effectiveness of the proposed learning framework through real-world implementation in mobile programming education settings across two institutions.

5.1.1. Participant Demography

The implementation was conducted in two mobile programming classes at two universities in Indonesia. It involved a total of 125 second-year undergraduate students, comprising 70 students from the *Informatics program* and 55 from the *Information Systems program*, representing two distinct academic pathways within undergraduate computer science education.

All students had successfully completed foundational programming courses, including fundamental algorithms, object-oriented programming, and web development, which provided them with basic programming skills necessary to engage with mobile application development. These prerequisite courses ensured that students possessed a common baseline of knowledge before beginning the mobile programming curriculum.

5.1.2. Implemented Learning Topics

The curriculum encompassed five key learning topics derived from previous research: Basic Application, Advanced Widgets, Multiple Activities, Multimedia Resources, and Basic Data Storage. Each topic was accompanied by related test suites, comprising test cases of varying complexity levels as detailed in Table 2. Students utilized their personal computers, which met the minimum hardware requirements to run Android Studio, indicating varied access to computing resources, while ensuring a standardized software execution environment. Two weeks were allocated for students to complete assignments spanning all five topics. Upon finishing each topic's assignment, students were required to submit their developed Android projects through a provided online web application. Throughout this evaluation process, comprehensive data were collected, including solving activity metrics, student feedback on their self-learning experiences, and assignment grades.

Table 2. Technical details of five learning topics for evaluation.

No	Learning Topic	Description	#test cases	L1	L2	L3	L4	L5
1	Basic Activity	Provides an introduction to Android application development, emphasizing initial configuration, user interface design, and event handling through the practical assignment of creating the <i>UnitConverter</i> application.	117	6	61	44	6	0
2	Advanced Widgets	Focuses on the incorporation of rich widgets to develop interactive features within Android applications, illustrated by the assignment to create a game application called <i>ColorGame</i> .	79	4	27	36	9	3
3	Multiple Activities	Guides students on managing multiple activities within an application using Intents and Fragments, demonstrated through the assignment of developing the <i>SoccerGame</i> application.	113	8	28	52	17	8
4	Multimedia Resources	Focuses on the integration of various multimedia elements, including video, audio, and animations, through the assignment to develop the <i>AnimalTour</i> application, enhancing user interactivity.	135	6	24	40	41	24
5	Basic Data Storage	Guides students on implementing persistent data storage techniques using local storage and shared preferences, as exemplified by the assignment to create the <i>MyShop</i> application.	161	10	27	44	58	22

Each learning topic is accompanied by a package of learning materials that facilitates self-learning. This package includes learning guidance in the form of PDF files that provide students with clear and organized information on specific topics relevant to the topic. Supplemental files related to application requirements, such as images, icons, and fonts, are provided to support students in their assignments. Furthermore, a test suite is included, consisting of five Java-based test files that represent the defined test cases for various grading levels, as detailed in Table 1. Students are instructed to use the learning guidance to study the material and write their code, incorporating the supplemental files

where necessary. After completing their assignments, students can utilize the test suite to verify their code, receiving automated assistance for correcting errors until they successfully pass all defined test cases.

5.2. Students' Progress Results

The 125 students completed and submitted their assignments in five different learning topics, with submission times reflecting a diverse range of levels of engagement. The fastest completion occurred in five days, while the longest submissions were submitted on the final day of the two-week assignment period. Table 3 presents the validation results, highlighting important metrics for each learning topic, including the total number of submissions, the fastest completion times, the average completion times, and the longest completion times. Each of the five topics recorded 125 submissions, showing full-class participation. The Advanced Widgets topic had the fastest submission time of 30 minutes, while Basic Activity and Multiple Activities had the quickest average times of 64.72 and 71.55 minutes, respectively. In contrast, both Multimedia Resources and Multiple Activities demonstrated the highest maximum completion time of 180 minutes, indicating that these topics were more complex and potentially challenging for students. Overall, the data suggest that while students performed well in the basic topics, the more advanced topics required additional time and effort, highlighting the varying complexities inherent in the curriculum.

Table 3. Students' progress in solving assignments on five learning topics.

Learning Topic	Number of Submissions	Fastest	Average	Longest
Basic Activity	125	35 minutes	64.72 minutes	140 minutes
Advanced Widgets	125	30 minutes	69.19 minutes	160 minutes
Multiple Activities	125	35 minutes	71.55 minutes	180 minutes
Multimedia Resources	125	40 minutes	70.36 minutes	180 minutes
Basic Data Storage	125	30 minutes	64.48 minutes	150 minutes

5.3. Feedback on Automated Assistance Function for Self-learning

Feedback collected from 94 students, totaling 181 responses, offers insightful reflections on their experiences with the learning framework designed for mobile application development. Among the responses, 136 (approximately 75.14%) were categorized as positive, highlighting the effectiveness of automated assistance in enhancing the learning experience. Students indicated that this support was particularly beneficial in addressing challenges encountered during assignments, deepening their understanding of Android programming, and facilitating the verification of code correctness. They also found validation and grading results to be informative, expressing enthusiasm for studying mobile programming, especially appreciating the Multimedia Resources topic as the most engaging. Meanwhile, 38 responses (approximately 20.99%) provided suggestive feedback that focused on avenues for improvement, such as the need for multilingual instructions, better-automated information for common mistakes, and the inclusion of multimedia resources in learning materials. Only seven responses (approximately 3.87%) were classified as negative, mainly related to technical issues, including performance problems during code compilation and version discrepancies between Android Studio and Gradle.

This feedback strongly demonstrates the effectiveness and crucial role of the automated assistance function within the learning framework, particularly in supporting self-learning in mobile application development, which presents varying levels of complexity. As presented in Table 4, the overwhelmingly positive feedback is further reinforced by the fact that all 125 students successfully completed assignments across the five learning topics, demonstrating the framework's capability to facilitate mastery of both fundamental and complex programming concepts. The use of automated assistance helped students navigate the challenges of more complex topics and bolstered their confidence and engagement in the learning process. The integration of automated feedback mechanisms within the framework proves essential in cultivating independent learning, illustrating that such support significantly enhances students' ability to tackle diverse programming tasks effectively. In general, the results underscore the importance of automated assistance in modern education, particularly in equipping students with the necessary skills for success in the rapidly evolving field of mobile application development.

Table 4. Summary of students' feedback in solving assignments.

Positive Feedback	Suggestive Feedback	Constructive Feedback
▪ Helpfulness of automated assistance; ▪ Challenges faced in solving assignments; ▪ Support for studying Android programming; ▪ Ease of checking code correctness; ▪ Informative validation and grading results; ▪ Simplicity in solving assignments; ▪ Most interesting topic: Multimedia Resources; ▪ Increased enthusiasm for mobile programming.	▪ Need for multilingual support in learning instructions; ▪ Automated assistance to warn about common mistakes; ▪ Request for multimedia resources in materials; ▪ Instruction on using versions and system updates; ▪ Suggestions for a cloud-based environment.	▪ Technical issues with PC performance during compilation; ▪ Incompatibility between versions of Android Studio and Gradle.

5.4. Students Grading Results

In the evaluation, a total of 625 submissions were collected from all students, each representing completed Android projects resulting from assignments across five distinct learning topics. The projects were submitted to an online server that facilitated the validation process, ensuring that each submission was subject to a thorough evaluation by the validator program. Impressively, the validation results indicated a seamless process with only minor discrepancies encountered during the assessment, highlighting the robustness of the evaluation framework. This seamless validation process reflects the efficiency and reliability of the integrated educational framework, which plays a critical role in providing precise feedback about students' understanding and application of programming skills.

The performance results for the five learning topics are detailed in Table 5, which highlights the notable achievements of the students. The average grades were high, with Basic Application (99.86) and Advanced Widgets (99.67) demonstrating exceptional comprehension of core programming principles. Despite this success, scores on more complex topics like Multimedia Resources (92.13) and Basic Data Storage (91.86) revealed areas of difficulty, as reflected in their lower minimum scores (78.22 and 79.16, respectively). Additionally, the lack of perfect scores in Multimedia Resources points to the intrinsic challenges of the topic, marking it as an area that may require enhanced instructional focus.

Table 5. Students' grading results on five learning topics.

Learning Topic	Basic Application	Advanced Widgets	Multiple Activities	Multimedia Resources	Basic Data Storage
Student Number	125	125	125	125	125
Grading Average	99.86	99.67	98.16	92.13	91.86
Grading Median Score	100	100	100	100	100
Maximum Grade	100	100	100	100	100
Minimum Grade	96.34	95.53	80.52	78.22	79.16
Students with Score 100	117 (93.6%)	113(90.4%)	97(77.6%)	0(0%)	34(27.2%)
Students with Score 95- 99.99	8(6.4%)	9(7.2%)	12(9.6%)	46(36.8%)	7(5.6%)
Students with Score 90-94.99	0(0%)	3(2.40%)	9(7.20%)	43(34.4%)	43(34.4%)
Students with Score 80-89.99	0(0%)	0(0%)	7(5.6%)	30(24%)	34(27.2%)
Students with Score less than 80	0(0%)	0(0%)	0(0%)	6(4.8%)	7(5.6%)

Applying statistical methods to the data reveals critical insights into student performance variations across learning topics. The high percentages of perfect scores in Basic Application (93.6%) and Advanced Widgets (90.4%) demonstrate the effectiveness of the educational framework in facilitating comprehension of fundamental topics. However, a notable decrease in perfect scores in Multiple Activities (77.6%) and the absence of such scores in Multimedia Resources highlight areas that posed significant learning challenges. The increasing standard deviation from Basic Application (0.59) to Basic Data Storage (7.01) further underscores the variance in student performance, with greater disparities observed in more advanced topics. These findings support the hypothesis that immediate, structured feedback enhances learning outcomes, particularly in foundational topics.

The evaluation results affirm the effectiveness of the self-learning framework augmented by automated assistance functions. The high average grades and smooth validation process underscore the framework's capacity to support independent learning of students in the development of mobile applications. Figure 7 illustrates an example of a student's result, showing a score of 96.56 in Basic Application, displaying detailed validation results and highlighting both successful and failed components. This capability for immediate feedback supports students in understanding and correcting errors, facilitating mastery of programming skills.

1	Project Configuration Pengumpulan pada 24 Jul 2023 16.18	✓ passed ✓ passed ✓ passed ✓ passed ✓ passed ✓ passed ✓ passed
3	Main layout, textview, button Pengumpulan pada 24 Jul 2023 16.19	✓ passed ✗ failed ✓ passed
Score 96.56 Enrolled : 24 July 2023 16.17 Total Estimate : 22 m		

Fig. 7. Verification results on a student's web interface with final score.

Although foundational topics demonstrated successful outcomes, challenges in more complex subjects underscore the need for ongoing improvements in the educational framework. By refining instructional strategies and strengthening resources, educators can further optimize student engagement and achievement, ensuring that learners are well-equipped to tackle the intricacies of mobile programming.

5.5. Technical Issues

During the implementation phase, students reported several technical issues, particularly related to development environment setup and compatibility. The most frequently mentioned problems included performance delays during code compilation, version mismatches between Android Studio and Gradle, and environment-specific errors affecting test execution on different operating systems. To address these concerns, we implemented a multistep resolution process:

- Technical problems were gathered through structured feedback forms and categorized based on severity and frequency.
- Common issues were traced back to differences in local development environments, outdated dependency versions, or missing configuration files.
- Standardized build.gradle templates were provided to ensure consistency, and a troubleshooting guide was added to the learning materials. Additionally, teacher-led support sessions were held for students experiencing persistent issues.
- Minor adjustments were made to the validator program to improve its tolerance for slight variations in file structure and build tools, reducing false negatives due to environmental inconsistencies.

6. Discussion on Evaluation Results

This section discusses the framework's scalability, effectiveness in improving student performance and engagement through automated assistance and grading, limitations related to sample size and generalizability, accuracy of the automated grading system, challenges such as the need for multilingual support, and recommendations for future improvements.

6.1. System Overview and Scalability

The proposed framework demonstrates scalability and adaptability through its modular design and structured curriculum. Implemented with 125 students across two university programs, the system successfully validated 625 Android project submissions with only minimal discrepancies, indicating robustness in handling moderate-scale class sizes. Built on a Laravel-based web interface and a Java-developed *validator program* running on *Ubuntu Linux*, the backend infrastructure supports parallel execution of test cases, minimizing bottlenecks during peak submission times. This architecture enables horizontal scaling via cloud deployment or containerization tools like Docker[52], [53], making it suitable for larger deployments such as MOOCs or institutional-wide implementations. Additionally, the curriculum follows a progressive structure aligned with formal departmental syllabi, ranging from foundational topics like *Basic Application* and *Advanced Widgets* to complex areas such as *Multimedia Resources* and *Basic Data Storage*, which incorporate advanced programming concepts including database integration, service interaction, and asynchronous task handling. The modular structure allows for easy extension with additional modules and ensures compatibility with various instructional models[54].

6.2. Curriculum Structure and Flexibility

The framework's curriculum is designed to follow a modular structure that ensures flexibility and compatibility with various instructional models. It progresses from basic to advanced topics, including database integration, service interaction, and asynchronous task handling as key concepts in modern mobile application development. Teachers can customize the system by defining new practical assignments and corresponding test cases, ensuring adaptability across different course structures and teaching objectives. The system is currently deployed online and has been used by students from two universities, providing a solid foundation for future expansion. However, successful large-scale deployment will require enhancements in infrastructure and performance tuning to support increased traffic and usage.

6.3. Impact on Student Performance and Engagement

The proposed framework has shown measurable improvements in student performance and engagement, with high average grades (especially in foundational topics) and positive student feedback indicating enhanced comprehension and problem-solving skills. The success of 625 validated submissions, including 93.6% perfect scores in the *Basic Application* topic, supports the effectiveness of timely, targeted automated feedback in promoting learning outcomes. While no direct comparison was made with traditional grading systems due to practical constraints, future studies should conduct controlled experiments for quantitative validation. Although the framework emphasizes automated feedback to support self-directed learning, it also enables teachers to provide finalized evaluations and personalized insights post-submission through a web-based interface, including annotations, written comments, or mini-review

sessions. Teachers can also deliver group-level interventions based on recurring errors identified through test results, creating a hybrid model that combines immediate automated responses with meaningful teacher interaction[55, 56], thereby enhancing the overall learning experience in mobile application development.

6.4. Limitations and Generalizability

It is important to note that the current study involved 125 students from two mobile programming classes across two university programs. While this sample size allows for an in-depth analysis of student engagement, learning outcomes, and feedback related to the automated assistance system, it is relatively modest and may limit the generalizability of the findings to more diverse student populations. The study was primarily focused on evaluating the functionality and pedagogical effectiveness of the proposed framework within a specific educational context. Future work should aim to replicate and extend these findings by implementing the framework across multiple institutions and student demographics to validate its broader applicability and effectiveness.

6.5. Automated Assistance and Grading Mechanisms

The automated assistance function, which includes automated testing and immediate feedback mechanisms, is a critical component of the framework's success. This feature has significantly enhanced students' understanding of Android application development by allowing them to quickly identify and address errors in their code. The ability to iterate on assignments without significant delays has fostered improved confidence and programming competence. Moreover, the automated grading process has revolutionized the feedback system for both students and teachers. Students benefit from timely evaluations that enable informed adjustments to their work, while teachers are relieved of the burden of manual grading, allowing them to focus more on pedagogical strategies and student engagement. The seamless validation of submissions through the *validator program* has dramatically reduced assessment time compared to traditional manual processes, creating a more responsive and efficient educational environment[42, 57].

6.6. Accuracy and Reliability of Automated Grading

The accuracy of the automated grading system was evaluated by comparing test suite results with manual assessments conducted by teachers. Out of 625 total submissions, only five cases were flagged for potential inaccuracies—three due to overly strict test conditions and two due to unexpected edge cases not covered by the initial test suites. Upon review, these instances were resolved by refining the corresponding test cases to better align with the implementation styles and learning objectives of the students. As a result, the overall rate of false positives and false negatives was minimal, reinforcing the system's reliability. However, we acknowledge the importance of further improving test case robustness and expanding coverage to include a wider variety of coding approaches.

6.7. Challenges and Areas for Improvement

Among the feedback provided by students, the need for multilingual support was frequently mentioned, particularly from non-native English speakers who encountered challenges understanding technical terms and instructions. To address this concern, we are planning a structured implementation of multilingual support through a two-phase approach. In the first phase, the localized versions of the user interface and core learning materials into Bahasa Indonesian and Japanese will be introduced, leveraging resource bundles for efficient language switching. In the second phase, automatic translation services[58], such as Google Translate API or Microsoft Azure Cognitive Services, will be used, integrated with manual quality checks to ensure accuracy and pedagogical effectiveness[59].

Integrating Test-Driven Development (TDD) into the learning framework has proven to be a key factor in enhancing student learning outcomes. Students who followed the TDD workflow consistently demonstrated higher accuracy in code implementation[60], particularly in foundational topics where understanding structure and logic was critical. For instance, 93.6% of students achieved perfect scores in the Basic Application topic, indicating strong mastery of core concepts through TDD-guided practice. TDD also supports higher-order learning goals such as problem-solving, debugging, and writing clean, testable code—skills highly valued in real-world software development. Student feedback further reinforced that the step-by-step testing process helped them stay focused, understand expectations, and build confidence in their coding abilities. These findings suggest that TDD not only improves technical proficiency but also reinforces metacognitive strategies such as reflection and error correction[61].

6.8. Cost-Benefit Analysis

The implementation of the proposed automated assistance and grading framework in mobile programming education presents numerous advantages over traditional manual assessment methods. Primarily, the framework offers significant time and effort savings, which enhances both student autonomy and learning outcomes. In conventional teaching methodologies, the preparation and execution of projects such as Android applications are labor intensive, requiring students and teachers to invest substantial time configuring environments, managing dependencies, and performing manual iterative tests. In contrast, the automated system minimizes these challenges through locally executed test suites, streamlining the verification process and reducing workload. This transition from manual to automated assessments also alleviates the burden on teachers, who traditionally are tasked with conducting detailed

manual functional checks across numerous student submissions. With the automation of these checks, teachers can focus more on teaching and less on repetitive verification, thereby improving the overall quality of instruction.

Furthermore, the automated system supports independent learning by integrating Test-Driven Development (TDD) principles and providing immediate feedback. These features are crucial for fostering self-directed learning and critical problem-solving skills, which are key components in software development. Students benefit from rapid feedback, which mitigates the delays typical in traditional systems that can hinder learning momentum[62]. The automated grading system provides students with immediate insights into their performance, allowing them to adjust their learning strategies in real-time. From an instructional perspective, teachers gain access to comprehensive analytics, enabling the identification of common errors and areas for improvement across classes, thus enhancing instructional focus. Although developing test suites requires initial efforts, the long-term benefits include improved scalability and effectiveness, particularly suited for expansive educational settings such as MOOCs[63]. Ultimately, while automated tools dramatically increase efficiency and learning effectiveness, they are best complemented by qualitative, human-provided feedback to address creative and nuanced aspects of learning. This supports a balanced, hybrid educational model.

6.9. Future Work and Recommendations

While the current implementation demonstrates measurable improvements in student performance and engagement in the short term, further investigation into the long-term retention of programming concepts taught through the framework is needed. Due to logistical constraints, our evaluation focused on immediate outcomes such as assignment completion, test scores, and student feedback during the course. However, the iterative nature of the Test-Driven Development (TDD) methodology, in which students repeatedly refine their code based on targeted feedback, suggests a positive impact on conceptual understanding and long-term learning. To validate this hypothesis, future studies should include follow-up assessments conducted weeks or months after course completion, or track student performance in advanced programming courses where prior knowledge is applied. Such longitudinal analysis will provide critical insights into the framework's effectiveness in fostering long-lasting skill acquisition and mastery of Android programming fundamentals. Additionally, future enhancements are planned to include automated environment detection, dynamic dependency resolution, and real-time error suggestions within the IDE, all aimed at minimizing technical barriers and improving the overall learning experience. Furthermore, the framework should continue to evolve to meet emerging educational needs by incorporating multilingual instruction and clearer guidance on common coding pitfalls. Automating various aspects of the learning process significantly reduces the complexities associated with these courses, ultimately equipping students with the knowledge and tools necessary for success in the rapidly evolving field of mobile development.

7. Conclusion

This study demonstrated the effectiveness of an innovative learning framework for Android programming that incorporates automated assistance and grading mechanisms. It integrates a Test-Driven Development (TDD) approach, which has proven effective in enhancing students' programming skills and their engagement in mobile application development. Automated testing strategies utilizing tools such as JUnit and Robolectric enable effective validation of code functionality and UI behavior, ensuring that students develop robust applications while promoting independent learning and efficient assessment. By emphasizing hands-on practice, automated testing, and structured feedback, the framework addresses the challenges students face in comprehending complex programming concepts. The successful implementation of this system across various learning topics facilitated full participation and high average grades among the 125 students. Moreover, student feedback highlighted the significance of the automated assistance feature, which facilitated a deeper understanding of the subject matter by allowing learners to identify and rectify code errors efficiently.

The evaluation of the proposed automated assistance and grading framework involved 125 students from two mobile programming classes across two universities, focusing on five learning topics: Basic Application, Advanced Widgets, Multiple Activities, Multimedia Resources, and Basic Data Storage. A total of 625 submissions were collected and validated through a fully automated system, with only minor discrepancies encountered during assessment, demonstrating the robustness and reliability of the validator program. Students received immediate feedback based on test suite execution using JUnit and Robolectric, enabling iterative learning and significant improvements in foundational topics, as evidenced by high average grades such as 99.86 for Basic Application. Student feedback was largely positive, with only a small percentage reporting technical issues related to IDE performance and version compatibility. The system also significantly reduced teacher workload by automating functional checks that traditionally required more than 30 hours of manual effort, while allowing targeted interventions based on recurring errors identified through test results.

Despite overall success, areas for improvement were identified, particularly in more complex topics such as Multimedia Resources and Basic Data Storage, where lower minimum scores indicated greater difficulty for students. Some learners requested clearer instructional guidance, multilingual support, and better explanations for common coding mistakes. Technical inconsistencies due to local development environments were addressed through

standardized build templates and validator adjustments to reduce false negatives and false positives. Future work includes longitudinal studies to assess long-term concept retention, refinement of test case design to accommodate diverse coding approaches, and comparative analyses with traditional grading systems. Expanding the framework using cloud-based or containerized solutions is also recommended to enhance scalability and accessibility in diverse educational settings.

All the Declarations and Statements

Author Contributions Statement

Yan Watequlis Syaifudin – Conceptualization; Methodology; Software development (learning framework, validator system); Investigation; Writing on original draft; Project administration; Corresponding author responsibilities.

Nobuo Funabiki – Methodology refinement; Formal analysis; Supervision; Validation of system design and experimental framework; Writing for review & editing.

Andi Baso Kaswar – Educational design; Curriculum alignment; Investigation (learning implementation); Data interpretation; Writing for review & editing.

Asep Sunandar – Pedagogical framework design; Learning model validation; Analysis of educational outcomes; Writing for review & editing.

Suryani Dyah Astuti – Data analysis; Evaluation methodology; Statistical interpretation of student performance; Validation of results; Writing for review & editing.

Triana Fatmawati – Software testing; Implementation support; Data collection; System validation; Writing for review & editing.

Mustika Mentari – Technical implementation support (JUnit/Robolectric testing); Software validation; Experimental setup assistance; Writing for review & editing.

Alfiandi Aulia Rahmadani – Data collection; System operation; Validation of student submissions; Visualization and documentation support.

All authors have read and agreed to the published version of the manuscript.

Conflict of Interest Statement

The authors declare no conflicts of interest.

Funding Declaration

This research was supported by the grant “Vocational Product Research” from Ministry of Education, Culture, Research, and Technology, Republic of Indonesia.

Data Availability Statement

None.

Ethical Declarations

The study involved human participants (undergraduate students enrolled in mobile programming courses). All procedures were conducted in accordance with institutional and educational research ethical standards. Participation was part of regular coursework, and all students were informed about the use of their anonymized data for research purposes. Informed consent was obtained from participants prior to data collection.

Acknowledgments

Authors would like to express our heartfelt gratitude to the Ministry of Education, Culture, Research, and Technology, Republic of Indonesia, for their generous research grant through the Vocational Education Directorate. This funding has been instrumental in supporting our research endeavors. We extend our sincere appreciation to the Laboratory of Applied Informatics Laboratory, Department of Information Technology, State Polytechnic of Malang, Indonesia, and the Distributed System Design Laboratory at the Department of Electrical and Communication Engineering, Okayama University, Japan, for their collaborative efforts and invaluable contributions. Additionally, we thank the Department of Informatics and Computers, State University of Makassar, for their support throughout this research project.

Declaration of Generative AI in Scholarly Writing

During the preparation of this manuscript, the authors used generative AI tools to improve the clarity, grammar, and readability of the text. The use of these tools was limited to language enhancement only and did not involve the generation of scientific content, analysis, or conclusions. All outputs were carefully reviewed, edited, and validated by the authors to ensure accuracy and integrity.

The authors confirm that they take full responsibility for the content of this manuscript. Generative AI tools are not listed as authors or co-authors.

Abbreviations

The following abbreviations are used in this manuscript:

AAPT - Android Asset Packaging Tool
API - Application Programming Interface
APK - Android Application Package
IDE - Integrated Development Environment
IT - Information Technology
JUnit - Java Unit Testing Framework
LMS - Learning Management System
MOOC - Massive Open Online Course
OJ - Online Judge
SDK - Software Development Kit
TDD - Test-Driven Development
UI - User Interface
UX - User Experience

Appendix

None.

References

- [1] Statista, "Market share of mobile operating systems worldwide 2009-2024, by quarter," Statista, Dec. 2024.
- [2] A. Kaur and K. Kaur, Systematic literature review of mobile application development and testing effort estimation, *Journal of King Saud University - Computer and Information Sciences*, vol. 34, no. 2, 2022, doi: 10.1016/j.jksuci.2018.11.002.
- [3] M. Tezer and M. Glyaz, The Use of Mobile Learning Technologies for an Online Mathematics Course: Student Opinions in The Pandemic Process, *Int. J. Interact. Mob. Technol.*, vol. 16, no. 23, 2022, doi: 10.3991/ijim.v16i23.36209.
- [4] B. P. Etemi, H. Uzunboylu, S. Latifi, and U. Abdigapbarova, The Effect of the Flipped Learning Approach on Engineering Students' Technology Acceptance and Self-Directed Learning Perception, *Sustain.*, vol. 16, no. 2, 2024, doi: 10.3390/su16020774.
- [5] D. Ifenthaler, J. Heil, and S. Greiff, Toward a categorisation of indicators for assessment analytics, *Learn. Lett.*, vol. 1, p. 3, 2023, doi: 10.59453/CCTB2003.
- [6] S. M. Arifi, R. Ben Abbou, and A. Zahi, Assisted learning of C programming through automated program repair and feedback generation, *Indones. J. Electr. Eng. Comput. Sci.*, vol. 20, no. 1, pp. 454–464, 2020, doi: 10.11591/ijeecs.v20.i1.pp454-464.
- [7] J. C. Paiva, Á. Figueira, and J. P. Leal, Bibliometric Analysis of Automated Assessment in Programming Education: A Deeper Insight into Feedback, *Electronics*, vol. 12, no. 10, 2023, doi: 10.3390/electronics12102254.
- [8] T. Xu, L. Chen, Y. Pei, T. Zhang, M. Pan, and C. A. Furia, Restore: Retrospective Fault Localization Enhancing Automated Program Repair, *IEEE Trans. Softw. Eng.*, vol. 48, no. 1, 2022, doi: 10.1109/TSE.2020.2987862.
- [9] T. Wang, D. B. Santoso, K. Wang, and X. Su, Automatic Grading for Complex Multifile Programs, *Complexity*, vol. 2020, 2020, doi: 10.1155/2020/3279053.
- [10] M. Messer, N. C. C. Brown, M. Klling, and M. Shi, Automated Grading and Feedback Tools for Programming Education: A Systematic Review, *ACM Trans. Comput. Educ.*, vol. 24, no. 1, 2024, doi: 10.1145/3636515.
- [11] M. Kaouni, F. Lakrami, and O. Labouidya, The Design of An Adaptive E-learning Model Based on Artificial Intelligence for Enhancing Online Teaching, *Int. J. Emerg. Technol. Learn.*, vol. 18, no. 6, 2023, doi: 10.3991/ijet.v18i06.35839.
- [12] J. H. Yu, X. Z. Chang, W. Liu, and Z. Huan, An online integrated programming platform to acquire students' behavior data for immediate feedback teaching, *Comput. Appl. Eng. Educ.*, vol. 31, no. 3, 2023, doi: 10.1002/cae.22596.
- [13] Y. W. Syaifudin, S. Rohani, N. Funabiki, and P. Y. Saputra, Blending Android Programming Learning Assistance System into Online Android Programming Course, in 2021 9th International Conference on Information and Communications Technology (ICOIACT), 2021, pp. 26–33, doi: 10.1109/ICIET51873.2021.9419650.
- [14] H. Nakamura, A. Sonoyama, T. Kamiyama, M. Oguchi, and S. Yamaguchi, Performance Study of Kotlin and Java Programs with Bytecode Analysis, *J. Inf. Process.*, vol. 32, pp. 380–395, 2024, doi: 10.2197/ipsjip.32.380.
- [15] D. Gamage, I. Perera, and S. Fernando, MOOCs lack interactivity and collaborativeness: Evaluating MOOC platforms, *Int. J. Eng. Pedagog.*, vol. 10, no. 2, 2020, doi: 10.3991/ijep.v10i2.11886.
- [16] D. Wea, E. S. Pudjiarti, and R. K. Sarang, Assessing the Quality of Learning Materials in a Learning Management System (LMS): Its Impact on Learning Outcomes, *Interciencia*, 2023, doi: 10.59671/gc23o.
- [17] M. Marabesi, A. Garca-Holgado, and F. J. Garca-Pealvo, Exploring the Connection between the TDD Practice and Test Smells—A Systematic Literature Review, *Computers*, vol. 13, no. 3, 2024, doi: 10.3390/computers13030079.
- [18] M. Wang, W. Han, and W. Chen, MetaOJ: A massive distributed online judge system, *Tsinghua Sci. Technol.*, vol. 26, no. 4, pp. 548–557, 2021, doi: 10.26599/TST.2020.9010016.

- [19] A. Gordillo, Effect of an Instructor-Centered Tool for Automatic Assessment of Programming Assignments on Students' Perceptions and Performance, *Sustainability*, vol. 11, no. 20, 2019, doi: 10.3390/su11205568.
- [20] J. Thangaraj, M. Ward, and F. O'Riordan, Use of Assessment and Feedback Systems for Introductory Computer Programming Modules of Higher Education: A Comparative Study, 2022, doi: 10.4995/HEAd22.2022.14660.
- [21] U. Z. Ahmed, Z. Fan, J. Yi, O. I. Al-Bataineh, and A. Roychoudhury, "Verifix: Verified Repair of Programming Assignments," *ACM Trans. Softw. Eng. Methodol.*, vol. 31, no. 4, 2022, doi: 10.1145/3510418.
- [22] B.-A. Nguyen, T.-V. H. Thi, and H.-M. Chen, "ANALYZING GIT LOG IN A CODE-QUALITY AWARE AUTOMATED PROGRAMMING ASSESSMENT SYSTEM: A CASE STUDY," *Tra Vinh Univ. J. Sci.*, vol. 13, no. 3, 2024, [Online]. Available: <https://journal.tvu.edu.vn/index.php/journal/article/view/48>.
- [23] S. López-Pernas, M. Saqr, and O. Viberg, "Putting It All Together: Combining Learning Analytics Methods and Data Sources to Understand Students' Approaches to Learning Programming," *Sustainability*, vol. 13, no. 9, 2021, doi: 10.3390/su13094825.
- [24] F. D. Pereira *et al.*, Using learning analytics in the Amazonas: understanding students' behaviour in introductory programming, *Br. J. Educ. Technol.*, vol. 51, no. 4, pp. 955–972, 2020, doi: 10.1111/bjet.12953.
- [25] R. Romli, S. Sarker, M. Omar, and M. Mahmud, Automated Test Cases and Test Data Generation for Dynamic Structural Testing in Automatic Programming Assessment Using MC/DC, *Int. J. Adv. Sci. Eng. Inf. Technol.*, vol. 10, no. 1, pp. 120–127, 2020, doi: 10.18517/ijaseit.10.1.10166.
- [26] I. Mekterović, L. Brkić, B. Milašinović, and M. Baranović, Building a Comprehensive Automated Programming Assessment System, *IEEE Access*, vol. 8, pp. 81154–81172, 2020, doi: 10.1109/ACCESS.2020.2990980.
- [27] D. M. Le, "Model-based automatic grading of object-oriented programming assignments," *Comput. Appl. Eng. Educ.*, vol. 30, no. 2, pp. 435–457, 2022, doi: 10.1002/cae.22464.
- [28] Y. Kim, K. Lee, and H. Park, "Watcher: Cloud-Based Coding Activity Tracker for Fair Evaluation of Programming Assignments," *Sensors*, vol. 22, no. 19, 2022, doi: 10.3390/s22197284.
- [29] R. Maertens *et al.*, "Dolos: Language-agnostic plagiarism detection in source code," *J. Comput. Assist. Learn.*, vol. 38, no. 4, 2022, doi: 10.1111/jcal.12662.
- [30] C. Andujar, C. R. Vijulie, A. Vinacua, B. S. Santos, and G. Alford, Syntactic and Semantic Analysis for Extended Feedback on Computer Graphics Assignments, *IEEE Comput. Graph. Appl.*, vol. 40, no. 3, 2020, doi: 10.1109/MCG.2020.2981786.
- [31] V. Hellem and M. Loras, The effect of mandatory assignments on students learning outcome and performance in introductory programming courses, in 2020 IEEE Global Engineering Education Conference (EDUCON), 2020, pp. 704–712, doi: 10.1109/EDUCON45650.2020.9125198.
- [32] C. Y. Chou and Y. J. Chen, Virtual Teaching Assistant for Grading Programming Assignments: Non-dichotomous Pattern based Program Output Matching and Partial Grading Approach, in 2021 4th IEEE International Conference on Knowledge Innovation and Invention (ICKII), 2021, pp. 170–175, doi: 10.1109/ICKII51822.2021.9574713.
- [33] E. Bitakou, M. Ntaliani, K. Demestichas, and C. Costopoulou, Assessing Massive Open Online Courses for Developing Digital Competences among Higher Education Teachers, *Educ. Sci.*, vol. 13, no. 9, 2023, doi: 10.3390/educsci13090900.
- [34] Kanika, S. Chakraverty, and P. Chakraborty, Tools and Techniques for Teaching Computer Programming: A Review, *J. Educ. Technol. Syst.*, vol. 49, no. 2, 2020, doi: 10.1177/0047239520926971.
- [35] K. Jain and S. Gupta, A Pragmatic Analysis of Mobile and MOOCs Based Learning Methods, *J. Eng. Educ. Transform.*, vol. 35, no. 4, 2022, doi: 10.16920/jeet/2022/v35i4/22100.
- [36] A. F. Tatang and A. M. Algarni, MOOCMaven: Bridging M-Learning and Open Data for Enhanced Unified MOOC Exploration, *Int. J. Interact. Mob. Technol.*, vol. 17, no. 20, 2023, doi: 10.3991/ijim.v17i20.42445.
- [37] J. M. Gallego-Romero, C. Alario-Hoyos, I. Estévez-Ayres, and C. D. Kloos, Analyzing learners' engagement and behavior in MOOCs on programming with the Codeboard IDE, *Educ. Technol. Res. Dev.*, vol. 68, no. 5, 2020, doi: 10.1007/s11423-020-09773-6.
- [38] A. Zancanaro, C. S. Nunes, and M. J. C. de Souza Domingues, Evaluation of free platforms for delivery of massive open online courses (MOOCs), *Turkish Online J. Distance Educ.*, vol. 18, no. 1, 2017, doi: 10.17718/tojde.285814.
- [39] T. A. Vakaliuk, O. V. Chyzhmotria, O. H. Chyzhmotria, S. O. Didkivska, and V. V. Kotsedailo, The use of massive open online courses in teaching the fundamentals of programming to software engineers, *Educ. Technol. Q.*, vol. 2023, no. 1, 2023, doi: 10.55056/etq.37.
- [40] K. von Hausswolff, Practical thinking while learning to program—novices' experiences and hands-on encounters, *Comput. Sci. Educ.*, vol. 32, no. 1, 2022, doi: 10.1080/08993408.2021.1953295.
- [41] C. W. Tsai *et al.*, The effects of online peer-facilitated learning and distributed pair programming on students' learning, *Comput. Educ.*, vol. 203, 2023, doi: 10.1016/j.compedu.2023.104849.
- [42] M. Novak and D. Kermek, Assessment Automation of Complex Student Programming Assignments, *Educ. Sci.*, vol. 14, no. 1, 2024, doi: 10.3390/educsci14010054.
- [43] Y. Zhao *et al.*, Compile-time code virtualization for android applications, *Comput. Secur.*, vol. 94, 2020, doi: 10.1016/j.cose.2020.101821.
- [44] S. Buyrukoglu, F. Batmaz, and R. Lock, Improving marking efficiency for longer programming solutions based on a semi-automated assessment approach, *Comput. Appl. Eng. Educ.*, vol. 27, no. 3, 2019, doi: 10.1002/cae.22094.
- [45] Y. W. Syaifudin, N. Funabiki, M. Kuribayashi, and W. C. Kao, A proposal of Android Programming Learning Assistant System with implementation of basic application learning, *Int. J. Web Inf. Syst.*, vol. 16, no. 1, 2020, doi: 10.1108/IJWIS-08-2019-0038.
- [46] J. Gao, C. Su, E. Miller, K. Lu, and M. Yu, Rapid Mobile App Development for Generative AI Agents on MIT App Inventor, *J. Adv. Inf. Sci. Technol.*, vol. 2, no. 3, pp. 1–8, 2024, doi: 10.5281/ZENODO.10899798.
- [47] E. B. Hanci-Azizoglu and E. Vural, Computer-Assisted Autonomous Linguistic Learning: A Novel Approach to Second Language Expertise, *SiSal J.*, vol. 14, no. 3, 2023, doi: 10.37237/140306.
- [48] B. S. Clegg, P. McMinn, and G. Fraser, Diagnosability, Adequacy & Size: How Test Suites Impact Autograding, in Proceedings of the 55th Hawaii International Conference on System Sciences (HICSS), 2022, doi: 10.24251/hicss.2022.107.

- [49] A. Kozak, L. Blyznyuk, T. Knysh, O. Ivanashko, and K. Honchar, Online Learning As a Tool for the Education System in the Context of Digitalisation, *J. Curric. Teach.*, vol. 12, no. 2, 2023, doi: 10.5430/jct.v12n2p132.
- [50] F. G. K. Yilmaz, The effect of learning analytics assisted recommendations and guidance feedback on students' metacognitive awareness and academic achievements, *J. Comput. High. Educ.*, vol. 34, no. 2, 2022, doi: 10.1007/s12528-021-09304-z.
- [51] X. Zhou *et al.*, A Static Assignment Algorithm of Uniform Jobs to Workers in a User-PC Computing System Using Simultaneous Linear Equations, *Algorithms*, vol. 15, no. 10, 2022, doi: 10.3390/a15100369.
- [52] R. Vaño, I. Lacalle, P. Sowiński, R. S-Julián, and C. E. Palau, Cloud-Native Workload Orchestration at the Edge: A Deployment Review and Future Directions, *Sensors*, vol. 23, no. 4, 2023, doi: 10.3390/s23042215.
- [53] G. Volpe, A. M. Mangini, and M. P. Fanti, An Architecture Combining Blockchain, Docker and Cloud Storage for Improving Digital Processes in Cloud Manufacturing, *IEEE Access*, vol. 10, 2022, doi: 10.1109/ACCESS.2022.3194264.
- [54] E. Kähkönen and K. Hättö-Otto, From crossing chromosomes to crossing curricula—a biomimetic analogy for cross-disciplinary engineering curriculum planning, *Eur. J. Eng. Educ.*, vol. 47, no. 3, 2022, doi: 10.1080/03043797.2021.1953446.
- [55] D. P. Adorno *et al.*, An innovative tailored instructional design for computer programming courses in engineering, *U.Porto J. Eng.*, vol. 9, no. 3, 2023, doi: 10.24840/2183-6493_009-003_001898.
- [56] N. S. Raj and V. G. Renumol, A systematic literature review on adaptive content recommenders in personalized learning environments from 2015 to 2020, *J. Comput. Educ.*, vol. 9, no. 1, 2022, doi: 10.1007/s40692-021-00199-4.
- [57] Y. Lee and J. Cho, Quantifying the effects of programming education on students' knowledge representation and perception in computational thinking, *Int. J. Innov. Creat. Chang.*, vol. 9, no. 4, 2019.
- [58] J. Lin, G. Sun, G. Beydoun, and L. Li, Applying Machine Translation and Language Modelling Strategies for the Recommendation Task of Micro Learning Service, *Educ. Technol. Soc.*, vol. 25, no. 1, 2022.
- [59] W. T. Tseng, Y. T. Liu, Y. T. Hsu, and H. C. Chu, Revisiting the effectiveness of study abroad language programs: A multi-level meta-analysis, *Lang. Teach. Res.*, vol. 28, no. 1, 2024, doi: 10.1177/1362168820988423.
- [60] W. Ren and S. Barrett, Test-driven development, engagement in activity, and maintainability: An empirical study, *IET Softw.*, vol. 17, no. 4, 2023, doi: 10.1049/sfw2.12135.
- [61] Y. Zhang, X. Fu, L. Wei, and C. Wang, Improving Higher Education Course Cognitive Outcomes with a Test-Driven Design Strategy for Instruction, *Int. J. Learn. Teach.*, vol. 10, no. 1, 2024, doi: 10.18178/ijlt.10.1.64-69.
- [62] A. P. Cavalcanti *et al.*, Automatic feedback in online learning environments: A systematic literature review, *Comput. Educ. Artif. Intell.*, vol. 2, 2021, doi: 10.1016/j.caeai.2021.100027.
- [63] Y. Shao, J. Zhang, E. Costello, and M. Brown, Public perceptions towards MOOCs on social media: an alternative perspective to understand personal learning experiences of MOOCs, *Interact. Learn. Environ.*, vol. 31, no. 2, 2023, doi: 10.1080/10494820.2020.1799413.

Authors' Profiles



Yan Watequlis Syaifudin is a Full Professor and Head of Applied Informatics Laboratory in Department of Information Technology, State Polytechnic of Malang. He received his bachelor degree in Informatics from Bandung Institute of Technology, Indonesia, in 2003, the master degree in Information Technology from Sepuluh Nopember Institute of Technology, Surabaya, Indonesia, in 2011, and his Ph.D. degree in Information and Communication Systems from Okayama University, Okayama, Japan, in 2021, respectively. In 2005, he joined State Polytechnic of Malang, Indonesia, as a lecturer. His research interests include technology enhanced learning, intelligent systems, data engineering, blockchain, and distributed systems. He is the director of Academic Association of Creative Economy and is a member of IEEE, IAENG, and Consumer Technology Society.



Nobuo Funabiki, received the B.S. and Ph.D. degrees in mathematical engineering and information physics from the University of Tokyo, Japan, in 1984 and 1993, respectively. He received the M.S. degree in electrical engineering from Case Western Reserve University, USA, in 1991. From 1984 to 1994, he was with the System Engineering Division, Sumitomo Metal Industries, Ltd., Japan. In 1994, he joined the Department of Information and Computer Sciences at Osaka University, Japan, as an assistant professor, and became an associate professor in 1995. He stayed at University of Illinois, Urbana-Champaign, in 1998, and at University of California, Santa Barbara, in 2000-2001, as a visiting researcher. In 2001, he moved to the Department of Communication Network Engineering (currently, Department of Electrical and Communication Engineering) at Okayama University as a professor. His research interests include computer networks, optimization algorithms, educational technology, and Web technology. He is a member of IEEE, IEICE, and IPSJ.



Andi Baso Kaswar completed his undergraduate program in the Informatics and Computer Engineering Education Study Program at State University of Makassar, Indonesia from 2010 to 2014. Furthermore, the author completed his education in the Informatics Engineering postgraduate program at the Sepuluh Nopember Institute of Technology Surabaya, Indonesia from 2014 to 2016. Since 2016, the author has worked as a lecturer and is currently a permanent lecturer at State University of Makassar in the Computer Engineering study program. The area of expertise that the author has studied is Computer Vision and Digital Image Processing, and he has published several papers related to this field. Besides being active as a lecturer and researcher, the author has also written programming books and published them at national publishers. The author has also won several achievements. 2 Achievements in the last few years, namely winning second place in a national textbook writing competition and with his team representing Indonesia at the 2019 Asean ICT Awards in Laos.



Asep Sunandar is a Full Professor in Educational Management at Universitas Negeri Malang, Indonesia, specializing in school unit management and educational leadership. With a doctoral degree from Universitas Negeri Malang (2015), his research focuses include inclusive education, educational entrepreneurship, organizational culture in higher education, and bureaucratic reform in educational institutions. He has led numerous national research projects, published extensively in reputable international journals such as the European Journal of Contemporary Education, International Journal of Educational Management, and Open Education Studies, and authored several books in his field. Currently teaching graduate-level courses in educational policy and strategic management, he is also actively involved in academic governance, professional organizations such as the Southeast Asia Association for Special Education (SAASE), where he serves as President, and public service initiatives focused on community empowerment and inclusive education.



Suryani Dyah Astuti is a Full Professor in Biomedical Physics at Universitas Airlangga, Surabaya, Indonesia. With over two decades of experience in biomedical engineering and physics, her research focuses on medical imaging optimization, photodynamic therapy, biosensors, and applications of electromagnetic fields in healthcare. She has made significant contributions to the development and analysis of MRI techniques, laser-based medical treatments, and electronic nose technology for disease detection and food safety. Prof. Astuti has published over 50 articles in reputable international journals such as Lasers in Medical Science, Sensing and Bio-Sensing Research, Heliyon, and Veterinary Medicine International, and has presented her work at numerous national and international conferences. She has also served as a reviewer for various scientific journals and has been actively involved in graduate education and research supervision. Her interdisciplinary approach bridges physics, biology, and engineering, making her a leading figure in Indonesia's biomedical research community. Currently, she continues to lead research initiatives that integrate advanced instrumentation with clinical applications to improve diagnostic and therapeutic technologies.



Triana Fatmawati completed her bachelor's degree in Informatics Engineering from Bandung Institute of Technology in 2023. Subsequently, she continued her master's degree in Industrial Engineering and Management at the same university, graduating in 2008. Triana began her career as a lecturer in the Information Systems program at Polytechnic of STMI Jakarta under the Ministry of Industry from 2005 until 2022. In 2023, she joined State Polytechnic of Malang as a lecturer in the Information Technology Department, where she continues to contribute to the field. Triana's research interests include information systems, machine learning, and software engineering.



Mustika Mentari completed her bachelor's degree in computer science study program from Brawijaya University in 2011. Subsequently, she continued her master's degree in informatics engineering at Sepuluh Nopember Institute of Technology (ITS), graduating in 2014. Mentari has been active in her career as a lecturer in the Diploma 4 informatics engineering study program at the State Polytechnic of Malang since 2015. In 2023, she began her doctoral studies in the Graduate School of Environmental and Life Science at Okayama University. Mentari's research interests include Programming Learning Assistant Systems, Digital Image Processing, Intelligent Systems and Computer Vision.



Alfandi Aulia Rahmadani completed her Bachelor of Applied Electrical Engineering degree from the State Polytechnic of Malang, Indonesia, in 2023, followed by a Master of Applied Electrical Engineering degree from the same institution in 2024. In 2024, she joined the research team at the State Polytechnic of Malang. She is currently a Ph.D. candidate at the Graduate School of Natural Sciences and Technology at Okayama University, Japan. Her research interests include environmental monitoring technologies, data transmission, Internet of Things, web technologies, and sensor systems technologies.

How to cite this paper

Yan Watequlis Syaifudin, Nobuo Funabiki, Andi Baso Kaswar, Asep Sunandar, Suryani Dyah Astuti, Triana Fatmawati, Mustika Mentari, Alfiandi Aulia Rahmadani, "Empowering Self-Learning: Automated Assistance and Grading Mechanisms in Android Application Development", *International Journal of Modern Education and Computer Science(IJMECS)*, Vol.18, No.4, pp. 99-121, 2026. DOI:10.5815/ijmeecs.2026.04.07