ISSN 2522-6398 Vol. 9, Issue 7: 51-62, DOI: 10.25236/FER.2026.090706

From Line-by-Line Checking to Reading the Data: How a Vocational Java Teacher's Grading Practice Changed over Two Semesters

Lei Peng[1,a], Kai Yang[2,b,*]

[1]*Library and Information Science Center, Chongqing Three Gorges Medical College, Chongqing, China*
[2]*School of Public Health and Management, Chongqing Three Gorges Medical College, Chongqing, China*
[a]*amon5728@163.com*, [b]*948244117@qq.com*
**Corresponding author*

Abstract: *In vocational programming education, teachers are supposed to act as technical coaches, yet most of their time gets swallowed by manually grading large batches of code assignments. To address this, I developed an automated code-quality analysis tool for Java projects and used it throughout the 2024 Spring semester. This paper tracks the same teacher—myself—across two consecutive semesters: the baseline semester (2023–2024-2) with fully manual grading, and the intervention semester (2024–2025-2) supported by the automated system. By analyzing teaching logs, system reports, stimulated-recall interviews, and student assignment samples, the study documents three practical shifts in the teacher's daily work. First, attention moved from inspecting individual lines of code to optimizing the teaching process as a whole. Second, diagnostic thinking moved from scattered impressions of single cases to recognizing class-wide patterns in aggregated data. Third, the nature of feedback moved from corrective annotations to developmental guidance. The core value of the tool lies in acting as a cognitive radar: it lifts the burden of repetitive code review and returns that time to the teacher for data-informed instructional decisions. This creates a systemic feedback loop from assessment back into teaching.*

Keywords: *Vocational programming education, Java language, Code analysis, Assignment assessment, Longitudinal study*

1. Introduction

Vocational education worldwide is pushing digital transformation and skills-based talent development [1]. Programming sits at the center of this shift. Students need timely, specific feedback to build complex practical competencies, and programming in particular depends on it [2]. Yet vocational programming classrooms face an old, stubborn problem: the gap between the deep, contextual guidance teachers want to give and the crushing workload of grading high-quality, complex assignments for large classes [3].

Automated assessment systems have been introduced to ease this load [4]. Most existing research, however, asks how these tools affect student outcomes—grades, skill mastery, or student satisfaction—while paying far less attention to how they reshape the teacher's own workflow, diagnostic habits, and professional identity over time [5]. This imbalance leaves a real gap: we do not know enough about how technology can empower teachers rather than merely replace them.

As programming instruction moves from syntax drills to project development, the focus of assessment naturally shifts from "does it run?" to "how well is it built?"—code quality, design patterns, and engineering habits [6]. Manual grading then becomes a punishing task: the teacher must download, unzip, open, run, and read dozens or even hundreds of project files one by one. Valuable professional energy gets trapped in low-level inspection, leaving little room for deep instructional diagnosis, personalized intervention, or formative feedback that supports long-term growth [7]. How automated tools can be woven into daily workflow—moving beyond simple efficiency gains to change how teachers actually see their role—remains underexplored.

To investigate this, the study adopts a longitudinal comparative case design. Instead of comparing different teachers at one point in time, it tracks the same teacher-researcher across two full semesters. The first semester serves as baseline, relying entirely on manual grading. The second semester serves as intervention, systematically introducing a self-developed automated tool focused on multidimensional

ISSN 2522-6398 Vol. 9, Issue 7: 51-62, DOI: 10.25236/FER.2026.090706

code-quality analysis. By holding the teacher constant, the design isolates the tool's effect in a near-natural experimental setting [8].

The study addresses three questions: (1) How does the implementation of an automated code-analysis system reshape the practical workflow and time allocation of a vocational programming instructor when assessing complex, project-based assignments? (2) In what ways do the aggregate, data-driven insights generated by the system influence the instructor's understanding of student learning patterns, and how do these insights inform subsequent instructional design? (3) What processes of situated professional learning are evidenced as the instructor moves from manual evaluator to technology-augmented diagnostic designer?

2. Literature and Theoretical Framework

2.1. The Challenge of High-Level Assessment in Vocational Programming

In competency-based vocational education, assessment drives skill development. Once students can write functionally correct code, the next task is to cultivate software developers who can solve complex problems. Assessment must then shift from "correctness" to "quality"—code structure, design, maintainability, and engineering practice [9]. This higher-level assessment involves complex judgments across multiple dimensions: modularity, coupling, use of object-oriented features, data-structure choices, naming conventions, and comments. It is highly contextual and cognitively demanding. In large classes, it creates an "unscalability dilemma": teachers have the expertise for deep diagnosis, but lack the time to apply it to every submission, so grading becomes superficial or feedback arrives too late.

2.2. The Teacher as Diagnostic Designer: Ideal Versus Reality

From a formative-assessment perspective, the point of grading is to promote learning [7]. Teachers should act as learning diagnosticians and design coaches: they analyze student work, identify misconceptions or gaps in technical reasoning, and provide feedback that fosters professional growth [10]. In programming, this means looking past the output to understand the student's thinking and design decisions. In practice, however, heavy grading loads reduce teachers to "function verifiers" and "syntax proofreaders." Cognitive resources are spent on basic inspection, making it difficult to spot cross-student patterns or conduct deep diagnosis. This gap between the ideal role and daily reality is the starting point of the study.

2.3. Technology as a Mediator of Practice Change: Activity Theory

To understand how a tool can reshape teaching practice, the study uses Cultural-Historical Activity Theory [11]. Human activity is viewed as a social system built from subject, object, tools, rules, community, and division of labor. A key insight is that changes in tools often introduce contradictions that drive the "expansive transformation" of the whole system.

In this study, the teacher's "programming assignment assessment activity" is the core system. During the baseline period, the subject is the teacher; the object is to judge each piece of code individually and assign a score; the main tools are the IDE and manual review. The central contradiction is the conflict between limited teacher time and the immense workload of grading. This conflict blocks the teacher from becoming the "learning diagnostician" described above.

The automated analysis system introduced here is treated as a new tool. Activity theory predicts that a new tool disrupts the old balance, triggers contradictions, and may push the system toward a more advanced form. The analysis therefore focuses on how the tool shifts the object from individual judgment toward analysis of group patterns; how it mediates the teacher's relationship with the ultimate goal of cultivating high-quality programming ability; and how it triggers chain reactions in other elements—rules shifting from personal experience to data evidence, division of labor shifting so that the system handles standardized measurement while the teacher focuses on interpretation and intervention, and community interaction shifting so that the teacher engages in dialogue with data reports.

Figure 1 presents a conceptual model comparing the core elements of the teacher's activity system in the baseline and intervention periods. The triangle vertices represent the six system components. Green arrows indicate the direction of transformation triggered by the tool. Bold, underlined annotations highlight key evolutions in the intervention period, especially the expansion of the object (from

ISSN 2522-6398 Vol. 9, Issue 7: 51-62, DOI: 10.25236/FER.2026.090706

individual judgment to group pattern analysis), the compounding of tools, and the restructuring of the division of labor.

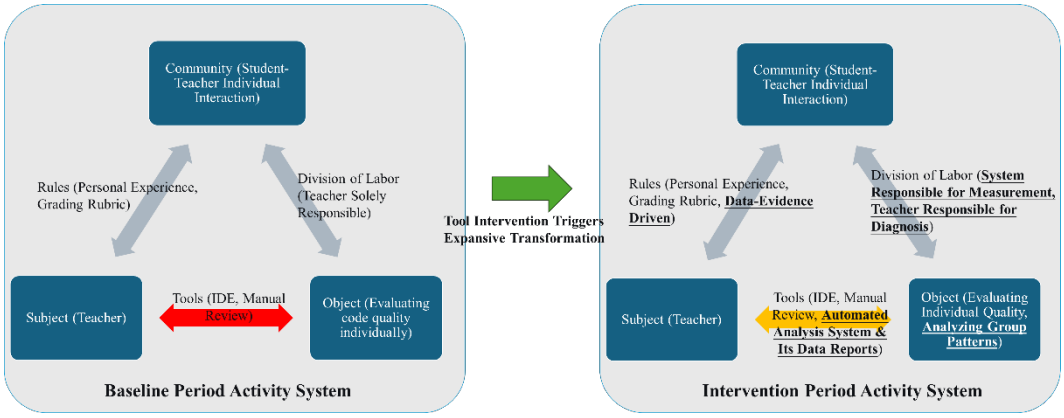

Figure 1: Conceptual model of change in teacher assessment practice based on Cultural-Historical Activity Theory.

2.4. From Tool Use to Professional Learning

Vocational teachers' professional knowledge is often tacit and context-bound; it develops mainly through workplace learning [12]. Automated analysis tools, especially the aggregated reports they generate, can serve as powerful "boundary objects" [13]. They externalize and materialize teaching challenges that were previously scattered across numerous assignments—patterns such as "most students avoid using interfaces" or "cyclomatic complexity is generally high."

When teachers confront these visualized group patterns, they enter a situation that demands reflective practice [14]: they must interpret the data, connect it to teaching experience, question prior assumptions, and form new pedagogical judgments. This process itself is a form of deep situated learning. More importantly, it may reshape professional identity—shifting from an evaluator who judges individual assignments to a diagnostic designer who uses data insights to steer the class's learning direction [15]. The study therefore focuses not only on efficiency gains but on how the tool functions as a cognitive and cultural medium that supports continuous teacher learning.

3. Research Context and Methods

3.1. Research Context

The study was conducted at a vocational university focused on application-oriented medical informatics. The course "Java Object-Oriented Programming" is a core, mandatory course for second-year students in the Health Information Management program. Course objectives require students not only to produce code free of syntax errors but also to develop high-quality, maintainable software using object-oriented principles in simulated medical-informatics scenarios such as patient information management systems or medical data query tools. Consequently, assessment shifts from "whether the program runs" to "how the code is constructed," focusing on design rationality, resource efficiency, appropriate use of object-oriented features (encapsulation, inheritance, polymorphism), control of code complexity, and adherence to professional conventions (naming, commenting).

3.2. Research Tool: The Automated Code Quality Analysis System

To address the high cognitive load of such complex assessment, the researcher independently designed and developed an automated code-quality analysis and measurement system for Java projects. Moving beyond the syntax-correction functions typical of tools for beginners, its core innovation lies in conducting multidimensional, quantitative structural analysis and quality evaluation of functional code. The specific analytical dimensions are: (1) Scale and Structure Metrics: total lines of code (LOC), number of classes/files, number of resource files. (2) Java Fundamentals and Object-Oriented Analysis: using static analysis to count the frequency of specific keywords and structures—such as new (instantiation), extends/implements (inheritance/implementation), interface, abstract, inner classes, threads

ISSN 2522-6398 Vol. 9, Issue 7: 51-62, DOI: 10.25236/FER.2026.090706

(Thread/Runnable), etc.—to quantify the breadth and depth of students' application of core language features. (3) Data Structure and Algorithm Complexity Analysis: detecting explicit use of standard-library data structures such as Stack, Queue, and Collection; calculating the cyclomatic complexity of methods to assess the complexity of program logic paths. (4) Code Style and Conformance Checking: checking variable and class names for adherence to English camelCase naming conventions based on preset rules, and calculating comment density.

Tables 1–4 detail the four main categories of analysis performed by the system on functional Java code and their specific indicators. These indicators aim to move beyond syntactic correctness, providing a quantitative assessment of structural quality, design level, and adherence to norms from a software-engineering perspective, thereby offering teachers multidimensional diagnostic evidence.

Table 1: Scoring details for scale and structure.

Metric objective	Metric standard	Instructional diagnostic significance	Scoring criteria
Code quantity (length)	Count the total lines of all Java source files.	Assess project size and workload; identify unusually large or small assignments.	Score = min(LOC / 10, 100), max 100.
Code quantity (size)	Count the total number of classes defined in the project.	Understand the degree of modularization and basic structure of the project.	Score = min(Classes × 10, 100), max 100.
Code resources	Count non-code files such as images, music, and configuration files.	Evaluate the completeness and resource integration capability of the project.	Score = min(ResourceFiles × 10, 100), max 100.

Table 2: Scoring details for Java fundamentals and object-oriented.

Metric objective	Metric standard	Instructional diagnostic significance	Scoring criteria
Java basic	Count occurrences of basic type and feature keywords: (1) data types: `byte`, `short`, `int`, `long`, `float`, `double`, `char`, `boolean`, `String`; (2) control flow: `if`, `else`, `switch`, `case`, `default`, `for`, `while`, `do`, `break`, `continue`, `return`; (3) variables: `var`, `null`, `new`, `void`; (4) package and import: `package`, `import`.	Quantify the richness and variety of variable types and basic language features used.	Score = (Type count × 10) + (Keyword count × 1), max 100.
OOP basic	Count occurrences of OOP basic keywords: (1) access control and modifiers: `public`, `private`, `protected`, `static`, `final`, `abstract`, `synchronized`, `volatile`, `transient`; (2) class and interface: `class`, `interface`; (3) inheritance and implementation: `extends`, `implements`; (4) references: `this`, `super`; (5) type checking: `instanceof`.	Quantify the breadth and depth of students' use of core OOP concepts.	Score = (Type count × 10) + (Keyword count × 5), max 100.
OOP application	Count occurrences of OOP advanced keywords: (1) file operations: `File`, `Path`, `Files`; (2) I/O streams: `InputStream`, `OutputStream`, `Reader`, `Writer`, `BufferedInputStream`, `BufferedOutputStream`, `BufferedReader`, `BufferedWriter`, `Serializable`; (3) multithreading: `Thread`, `Runnable`, `synchronized`, `volatile`, `wait`, `notify`, `notifyAll`, `join`, `yield`, `sleep`; (4) exception handling: `try`, `catch`, `finally`, `throw`, `throws`, `Exception`, `Error`, `RuntimeException`, `Throwable`; (5) date and time: `Date`, `Calendar`, `SimpleDateFormat`, `LocalDate`, `LocalTime`, `LocalDateTime`, `Instant`, `Duration`, `Period`, `DateTimeFormatter`; (6) network programming: `Socket`, `ServerSocket`, `DatagramSocket`, `URL`, `URLConnection`, `HttpURLConnection`, `InetAddress`; (7) math operations: `Math`, `StrictMath`, `BigInteger`, `BigDecimal`, `Random`.	Quantify the breadth and depth of students' use of OOP application features.	Score = (Category count × 10) + (Keyword count × 5), max 100.

Table 3: Scoring details for data structures and complexity.

Metric objective	Metric standard	Instructional diagnostic significance	Scoring criteria
Data structure usage detection	Count occurrences of data structure keywords: (1) list: `List`, `ArrayList`, `LinkedList`, `Vector`, `Stack`; (2) set: `Set`, `HashSet`, `LinkedHashSet`, `TreeSet`; (3) queue/deque: `Queue`, `Deque`, `PriorityQueue`, `ArrayDeque`, `LinkedList`; (4) map: `Map`, `HashMap`, `LinkedHashMap`, `TreeMap`, `Hashtable`, `ConcurrentHashMap`; (5) array: `[]`; (6) generics: `<>`.	Understand students' preferences for problem-solving tools and algorithmic thinking.	Score = (Type count × 10) + (Usage count × 5), max 100.

ISSN 2522-6398 Vol. 9, Issue 7: 51-62, DOI: 10.25236/FER.2026.090706

Cyclomatic complexity	Calculate the cyclomatic complexity of each method using the following decision point values: `{"if": 1, "else if": 1, "else": 0, "switch": 0, "case": 1, "default": 0, "for": 1, "while": 1, "do-while": 1, "foreach": 1, "&&": 1, "||": 1, "? :": 1, "catch": 1}`. Cyclomatic complexity of a method = number of decision points + 1.	Evaluate the logical complexity of methods; identify potentially high-risk or hard-to-maintain code.	Score = min(100, (15 − AverageComplexity) × 10), max 100.

Table 4: Scoring details for code style and standardization.

Metric objective	Metric standard	Instructional diagnostic significance	Scoring criteria
Naming convention check	Calculate the proportion of variable and class names that follow CamelCase conventions and are in English. e.g., `int studentID = 1;` `public class EnhancedSearch{}`.	Assess code readability and adherence to basic professional standards.	Score = min((legal names / all names) × 100, 100), max 100.
Comment density	Calculate the percentage of comment lines relative to the total lines of code. e.g., `// set student ID` `int studentID = 1;`.	Evaluate code maintainability and documentation awareness.	Score = min(CommentPercentage × 1000, 100), max 100.

Figure 2 provides an example of an individual student's scoring results. It corresponds to Tables 1–4 and also presents statistics on the application of various object-oriented features.

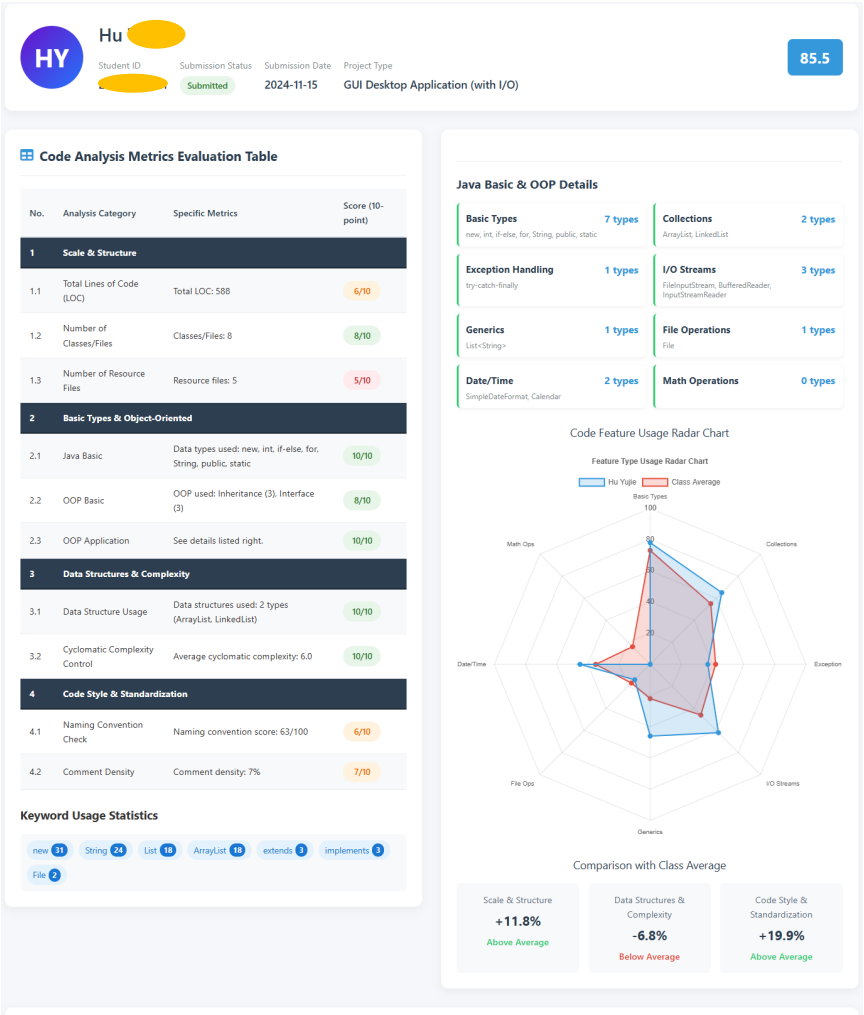

Figure 2: Example of individual student score details in the automated assessment system.

The study extends beyond automated scoring of individual submissions to foreground the aggregated diagnostic reports generated by the system, which function as a macro-level instructional dashboard. By performing horizontal comparison and cluster analysis across all class submissions, the system

ISSN 2522-6398 Vol. 9, Issue 7: 51-62, DOI: 10.25236/FER.2026.090706

synthesizes visual analytics such as: (1) an object-oriented feature application spectrum that maps the distribution and density of advanced programming constructs (e.g., interfaces, polymorphism, abstract classes); (2) a data structure and cyclomatic complexity heatmap that illuminates student preferences in technical implementation alongside gradients in logical design sophistication; (3) a project architecture and scale cluster analysis that classifies macro-level technical pathways (e.g., GUI desktop applications, GUI-based network programming) and correlates them with project scope; (4) targeted improvement suggestions derived from systematic knowledge-gap analysis across assignment segments.

Figure 3 shows the overall class situation, allowing the teacher to grasp the class's overall tendencies in design thinking at a glance. These group-based, patterned data insights are the key mediating artifacts for the teacher's instructional diagnosis and decision-making transformation in this study.

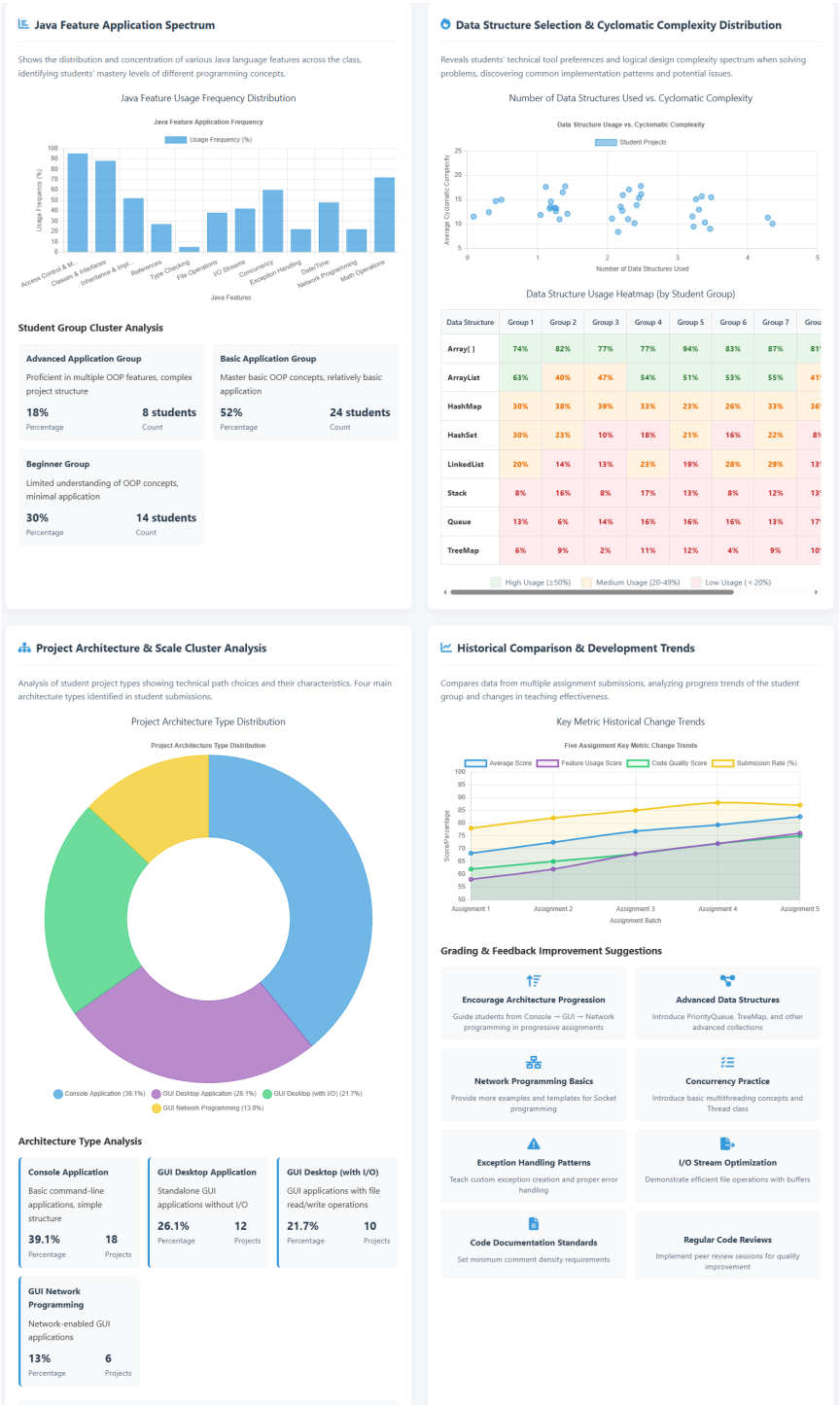

Figure 3: Example of class programming statistics results.

ISSN 2522-6398 Vol. 9, Issue 7: 51-62, DOI: 10.25236/FER.2026.090706

4. Research Design and Data Collection

The study employs an interpretive, longitudinal comparative case study design, systematically tracking and comparing the professional practice of the same teacher (the researcher) under two different assessment modes to explore the mediating role of the technological tool. The study naturally formed two comparative phases.

The baseline period (Spring Semester 2023–2024) employed fully traditional, in-depth manual grading. The teacher had to manually download, unzip, and open each student's project in an IDE, run it, read the code file by file, and use a scoring rubric to provide comprehensive judgment and feedback on code quality, design, and norms.

The intervention period (Spring Semester 2024–2025) employed automated code-analysis system-assisted grading. After students submitted their projects, the system automatically performed all the aforementioned code measurement and analysis, generating individual analysis reports and class-aggregated reports. The teacher's starting point shifted to reviewing these reports and basing instructional decisions on the insights they provided.

Researcher role and reflexive positioning. The researcher embodies the triple role of system builder, tool user (teacher), and research subject. This "builder-practitioner-researcher" positioning afforded the study a deep, embodied understanding of the tool's design logic, adaptive adjustments during classroom integration, and the teacher's internal cognitive shifts. To maximize rigor and trustworthiness, the following strategies were employed: (1) collecting comparable objective traces of work from both phases, forming the basis for triangulation; (2) using the intervention period's system reports and baseline period's memory anchors as "stimuli" to engage in a careful, distanced reconstruction and tracing of one's own decision-making processes; (3) ensuring the analysis process was a continuous dialogue among data, theory, and researcher reflection, rather than a simple narrative of personal experience.

Data were systematically collected across both semesters from the following triangulated sources: (1) Teacher reflective practice logs and time audit records: captured precise temporal metrics, in-process observations, emerging questions, representative cases, and pedagogical reflections for five comparable project assignments. (2) System-generated aggregated analysis reports: from the intervention period, comprising comprehensive visual and raw datasets for all class submissions that served as both analytical artifacts and contextual scaffolds for instructional decisions. (3) Stimulated recall interview: conducted post-intervention, wherein practice logs and system reports functioned as cognitive stimuli to retrospectively trace how systemic patterns (e.g., low polymorphism adoption) dialogically engaged with prior individual case experiences to generate actionable pedagogical insights and design responsive classroom interventions.

5. Data Analysis

Data analysis adopted a hybrid strategy combining longitudinal comparative analysis with reflexive thematic analysis, with the entire process being iterative and reflective.

The analysis began with a quantitative comparison of efficiency and behavioral patterns. Descriptive statistics and comparative analysis were applied to time audit records from both phases to objectively assess changes in grading efficiency. Simultaneously, teacher feedback on student assignment samples underwent systematic content categorization—for instance, quantifying the proportion of comments focused on syntax or runtime correctness, code structure and design quality, and connections to professional norms or group patterns. This quantitative approach provided measurable evidence of shifts in the teacher's feedback focus across the two periods.

For the qualitative examination of practice transformation, reflexive thematic analysis was employed to analyze teacher practice logs and stimulated recall interview transcripts [16]. The process unfolded through three interconnected phases: (1) phase-specific immersion and open coding: conducted separately for baseline and intervention data to capture emergent meanings; (2) cross-phase comparison: codes were systematically examined to identify recurring and distinctive patterns in the intervention period—particularly those linked to system-generated reports. These patterns were clustered into preliminary candidate themes. (3) Theory-guided refinement: Cultural-Historical Activity Theory was introduced to engage in iterative dialogue among the candidate themes, raw data, and system reports. This focused attention on the evolution of core activity system components—such as tools, object, and division of labor—before and after the intervention, leading to the continuous refinement of themes and

ISSN 2522-6398 Vol. 9, Issue 7: 51-62, DOI: 10.25236/FER.2026.090706

the eventual identification of three core themes that articulate the underlying mechanisms of change.

To ensure analytical rigor, a strategy of triangulation and iterative synthesis was sustained throughout. The analysis formed an ongoing dialogic loop in which qualitative practice data from both periods, objective system reports, and activity theory constructs were continually brought into conversation. This iterative process enabled mutual validation between qualitatively derived themes and quantitatively observed patterns, ultimately converging into a coherent explanatory account of how technology mediates and reshapes teacher professional practice.

6. Findings

Systematic analysis of teaching practice logs, time records, stimulated recall interviews, and student assignment samples across two semesters reveals three ways in which the automated code analysis system restructured the teacher's professional practice. These transformations manifest across three interrelated dimensions: workflow, cognitive approach, and teacher-student interaction.

6.1. Theme One: Workflow Reorientation—From "Digging Into Details" to "Optimizing the Process"

The most visible impact was the transformation of grading workflow structure and time allocation. For comprehensive project assignments of comparable scale and complexity (approximately 50 submissions), total grading time dropped from an average of 12–15 hours in the baseline period to approximately 2–3 hours in the intervention period. More importantly, the nature and distribution of that time changed fundamentally.

Baseline period: "individual deep dive" linear review. The teacher's logs detail a linear, repetitive process: download, unzip, open in IDE, run tests, read code file by file, mentally reconstruct project structure, evaluate against scoring rubric item by item, write comments by hand. Nearly 80% of the time was consumed by reading files line-by-line to understand individual code logic and structure. Descriptors like "exhausted" and "difficult to maintain consistent attention and standards toward the end of grading" frequently appear in the logs. The teacher's role resembled that of a quality inspector conducting a deep scan on each assignment.

Intervention period: efficient diagnosis via a panoramic view. The workflow was restructured into: system automatically analyzes and generates reports, teacher quickly reviews individual reports for basic scoring, focuses on studying the class-aggregated report, designs teaching interventions based on report patterns. The saved 10+ hours were reallocated. For instance, the teacher noted: "The system's cyclomatic complexity distribution chart showed that the core methods in 15 assignments exceeded a complexity of 10—a high-risk signal. Instead of hunting for them one by one, I used that time directly to prepare a mini-workshop on method refactoring to reduce complexity and specifically scheduled one-on-one tutorials with those 15 students." The teacher's role transformed into that of a tactical commander, using the radar map (system report) to identify hot zones (common patterns) and launch precise strikes (targeted interventions).

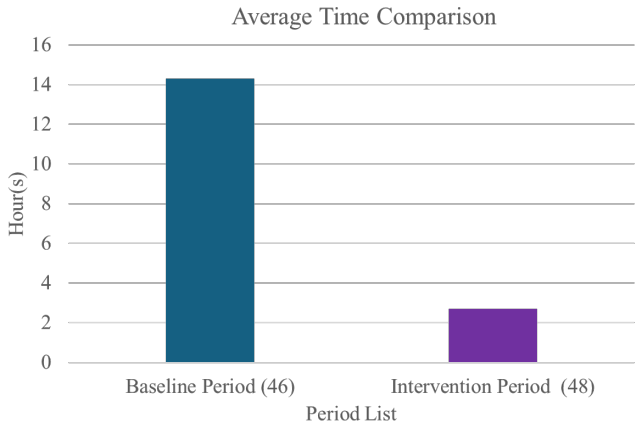

Figure 4: Time spent grading approximately 50 students' Java SE project assignments.

Figure 4 illustrates the comparison of average grading time per assignment between the baseline and intervention periods, with 46 students in the baseline class and 48 in the intervention class. It clearly

ISSN 2522-6398 Vol. 9, Issue 7: 51-62, DOI: 10.25236/FER.2026.090706

shows that using the system saved the teacher approximately 12 hours, time that could be redirected to more advanced instructional design activities.

6.2. Theme Two: Shift in Cognitive Insight—From "Case Accumulation" to "Data-Driven"

The aggregated analysis reports shifted the teacher's cognitive focus from qualitative impressions of individual cases to attention to and interpretation of group-level quantitative patterns.

Baseline period: induction and guessing based on dispersed cases. Insights arose from "interesting cases" or "common mistakes" stumbled upon during grading. For example, a log entry states: "Saw three assignments in a row using ArrayList instead of HashMap for fast queries; students might not be familiar with this data structure." Such insights were fragmented, retrospective, and their prevalence was hard to gauge. Instructional decisions relied more on intuition and prior experience.

Intervention period: deduction and confirmation based on panoramic data. The system reports externalized previously implicit patterns, making the teacher's insights proactive, systematic, and verifiable. In the stimulated recall interview, the teacher recalled a pivotal moment: "The object-oriented feature spectrum clearly showed that interface usage was below 20%, while concrete inheritance usage was as high as 70%. This visually confirmed my vague feeling from the baseline period—that students avoid abstraction and prefer concrete implementation. But the data told me this wasn't an isolated phenomenon; it was a class-wide design thinking tendency. This directly prompted me to shift the focus of the next chapter's teaching from 'how to implement an interface' to 'why we need interfaces: design evolution from concrete to abstract,' and I designed comparative cases." Here, data not only verified experience but also quantified the scale of the issue and guided the specific instructional direction.

From Figure 2 (the system's detailed individual report), one can infer how the system summarizes the class's performance across various Java knowledge domains—from object-oriented application and data structures to program architecture and suggested guidance. Previously, this required manual, deep inspection of all Java project assignments, extensive grading, and intuition-based summarization. Now, the system provides objective results directly through automated quantification and statistical visualization, allowing the teacher to manage the overall class situation more conveniently.

6.3. Theme Three: Qualitative Evolution of Feedback—From "Corrective Annotation" to "Developmental Dialogue"

As the system took over the objective measurement of dimensions like code structure and norms, the content, nature, and purpose of the teacher's written and oral feedback underwent a qualitative leap.

Baseline period: correction focused on "what's missing or wrong." Student assignment samples from the baseline period show that feedback mostly consisted of corrective annotations targeting specific lines of code, e.g., "Private variable should be used here," "This method is too long; consider splitting it," "Switch statement lacks a default branch." The core of the feedback was rectification, pointing to past, completed work that did not meet standards.

Intervention period: navigation toward "possibility" and "professional growth." Feedback samples from the intervention period show the teacher more frequently referencing system data and professional scenarios. A typical feedback example is: "Your project passed all functional tests. According to the system report, your code structure is clear (low cyclomatic complexity), but within the advanced feature application spectrum, no interfaces or abstract classes were used. In future iterations or new projects, consider: if requirements change and a new report type needs to be added, how would you modify your current ReportGenerator class? Try introducing a report interface (IReportGenerator) to make your system better align with the professional design principle of being closed for modification but open for extension."

This type of feedback places the student's work within the context of the class-wide technology choice spectrum and future career development. Its purpose is no longer to correct a past mistake but to initiate a forward-looking, professional development dialogue based on comparison and reflection. In the stimulated recall interview, the teacher stated, "What I give students now is no longer just a report card, but a professional health check report and a growth recommendation letter."

In summary, these three themes sketch a clear picture of transformation. The automated code analysis system, by taking over standardized measurement tasks (workflow reorientation), freed up space for the teacher to engage in higher-order cognitive activities. Simultaneously, by providing a group-level data

ISSN 2522-6398 Vol. 9, Issue 7: 51-62, DOI: 10.25236/FER.2026.090706

lens (cognitive shift), it significantly enhanced the precision and scope of the teacher's diagnosis of learning issues. Ultimately, these two aspects collectively empowered the teacher, enabling feedback to transcend immediate correction of the individual and evolve into a professional dialogue that guides students toward career-minded reflection and development (evolution of feedback nature). This series of shifts illustrates the empowering essence of a technological tool moving from replacing human effort to augmenting human intelligence.

7. Discussion

7.1. Theoretical Implications

This study finds that in vocational programming education, when student assignments advance to a functional stage, teachers' assessment practices face a fundamental challenge. The longitudinal comparative study clearly reveals that an automated analysis system focused on code quality metrics can empower teacher practice through three mutually reinforcing dimensions: workflow reconstruction, a shift in cognitive insight, and the evolution of feedback nature. This not only resolves the traditional contradiction between "depth" and "scale" in grading but also catalyzes a profound transformation of the teacher's professional role from "code reviewer" to "learning pattern diagnostician" and "professional development coach."

7.2. Dialogue With Theory I: Evidence of Expansive Transformation in the Activity System

This study offers empirical validation of the Cultural-Historical Activity Theory tenet that tool mediation catalyzes expansive learning [11]. Initially, the activity system was characterized by a core contradiction between the teacher-subject's limited temporal and cognitive resources and the object of conducting comprehensive assessment of high-quality code. The introduction of the automated analysis system directly addressed this tension, triggering a cascading reconfiguration across the system's constitutive elements.

First, the object of activity expanded from evaluating individual code submissions toward discerning class-wide patterns in code quality and design practice—shifting from discrete judgment to systemic diagnosis. Second, tools became layered and compounded: while the automated system functioned as the primary operational mediator, the data reports it generated acted as secondary cognitive artifacts that actively shaped pedagogical reasoning and decision-making. Third, the division of labor was reconfigured into a collaborative human–machine arrangement, wherein the system assumed responsibility for objective measurement and data aggregation, freeing the teacher to focus on interpretive, diagnostic, and instructional translation tasks. Finally, the rules governing pedagogical decision-making evolved from reliance on personal intuition and episodic observation toward evidence-informed judgment, establishing data as a legitimate discourse for professional deliberation. Together, these shifts demonstrate that the system operated not as a mere productivity tool, but as a transformative mediator that reconfigured the entire activity system toward a more advanced, collectively oriented form of diagnostic teaching.

7.3. Dialogue With Theory II: Boundary Objects and the Mechanism of Making Teacher Situated Learning Explicit

This study further reveals that the automated system plays a profound role in making cognition explicit, supporting teacher learning in the workplace. Its core mechanism lies in the fact that the aggregated reports generated by the system essentially function as potent boundary objects [13].

This boundary object enables a crucial knowledge transformation: it converts student group programming thinking tendencies—which were originally implicit within numerous scattered assignments and only vaguely perceived by the teacher (e.g., "avoidance of abstraction" or "preference for lengthy methods")—into explicit, shareable, and stable visual objects (e.g., "interface usage spectrum" and "cyclomatic complexity distribution heatmap"). Thus, the collective cognitive state of the students is brought from the shadows of perception into the light of visualization.

Confronted with these materialized boundary objects, the teacher is naturally drawn into a structured cycle of reflective practice [14]. The thought process is clearly discernible: "What pattern does the data reveal?" → "How does this pattern compare with my teaching expectations?" → "What cognitive limitations or thinking habits does this reflect in the students?" → "What intervention should I design in

ISSN 2522-6398 Vol. 9, Issue 7: 51-62, DOI: 10.25236/FER.2026.090706

response?" In the stimulated recall interview, the teacher's in-depth attribution and pedagogical redesign triggered by the data point "low interface usage rate" is a vivid example of this reflective cycle.

Therefore, this system is not merely an assessment tool; it is a cognitive catalyst that triggers and supports evidence-based, situated professional learning for teachers.

7.4. Practical Implications

The findings carry significant implications for pedagogical practice, teacher professional growth, and educational technology design in vocational education.

At the pedagogical level, instructors engaged in advanced skill development should adopt a data-informed teaching approach, deliberately structuring courses and projects to render students' cognitive processes visible, thereby enabling diagnostic analysis and shifting instructional focus from remediating individual errors toward guiding collective developmental trajectories.

Regarding teacher development, educational data literacy must be recognized as a core professional competency, particularly in STEM-related vocational fields. Teacher training should transcend operational tool proficiency to cultivate an integrated capacity for interpreting pedagogical data, linking evidence to theoretical frameworks, and designing responsive interventions—a data-to-insight-to-action pathway exemplified in this study.

For educational technology design, tools intended for higher vocational contexts should evolve from verifying basic correctness toward enabling high-quality analysis and complex competency assessment. This necessitates sustained collaboration between developers and practicing educators to create diagnostic interfaces that deliver layered, interpretable insights rather than merely presenting information.

8. Limitations and Future Directions

This study has limitations. First, as a single-case study led by a teacher-developer, while its findings possess depth and internal validity, their external validity needs verification in more diverse contexts—across different institutions and disciplines (e.g., network engineering, data analytics). Second, although the researcher's multiple roles provided deep insight, potential unconscious researcher bias may still exist. Future efforts could involve implementing this system across multiple colleges and with various teachers to obtain more extensive and objective research data.

Despite these limitations, this study, through its rigorous longitudinal comparative design, demonstrates that technological tools hold deep empowering potential beyond mere efficiency gains in reshaping high-level vocational education assessment practices. This finding provides crucial theoretical reference and practical guidance for exploring pedagogical innovation and teacher development in the digital era.

9. Conclusion

Through a rigorous longitudinal comparative case study, this research systematically investigates how an automated code quality analysis system reshapes teachers' professional practice in vocational programming education. By comparing the same teacher's practices across two consecutive semesters—traditional manual grading (baseline period) and system-assisted grading (intervention period)—we move beyond simplistic descriptions of technical efficiency to reveal how the tool, as a mediator, triggers a profound transformation in the teaching activity system. This transformation manifests in three systemic shifts: in workflow, redirecting significant time from code analysis and scoring to more meaningful instructional enhancements; in cognitive insight, elevating from reliance on discrete case experiences to comprehensive diagnosis based on panoramic data; and in the nature of feedback, evolving from corrective annotations pointing out errors to developmental dialogues oriented toward future professional growth.

Theoretically, this study provides empirical support for the activity theory tenet that tools can drive expansive learning. By serving as boundary objects, the system's reports externalize otherwise implicit teaching challenges, prompting the teacher to enter a data–reflection–action cycle of professional learning and ultimately facilitating an identity shift from evaluator to diagnostic designer.

Practically, this work highlights that the higher aim of digital transformation in vocational education

ISSN 2522-6398 Vol. 9, Issue 7: 51-62, DOI: 10.25236/FER.2026.090706

is not the automation of administrative tasks but the enhancement of pedagogical cognition. The use of such tools enables teachers to translate intuition into evidence and to elevate individual experience into group-level instructional strategy. Consequently, teacher professional development must place educational data literacy and evidence-informed instructional design at its core.

In sum, the value of technology lies not in replacing teachers, but in amplifying their capacity to fulfill the educational mission—namely, to cultivate reflective practitioners who can solve complex problems and demonstrate exemplary professional competence. This study offers an integrated view from tool design to classroom change, establishing an empirical foundation for exploring human–machine collaborative and empowering educational pathways.

Acknowledgements

This work was supported by the Chongqing Vocational Education Society [Grant Number 2025ZJXH580065].

References

[1] Organisation for Economic Co-operation and Development (OECD). The Future of Education and Skills: Education 2030[M]. OECD Publishing, 2018.
[2] Hattie J, Timperley H. The Power of Feedback[J]. Review of Educational Research, 2007, 77(1): 81-112.
[3] Black P, Wiliam D. Classroom Assessment and Pedagogy[J]. Assessment in Education: Principles, Policy & Practice, 2018, 25(6): 551-575.
[4] Keuning H, Jeuring J, Heeren B. A Systematic Literature Review of Automated Feedback Generation for Programming Exercises[J]. ACM Transactions on Computing Education, 2018, 19(1): 1-43.
[5] Carless D, Winstone N. Teacher Feedback Literacy and Its Interplay With Student Feedback Literacy[J]. Teaching in Higher Education, 2020, 28(1): 150-163.
[6] Sommerville I. Software Engineering[M]. 10th ed. Pearson, 2015.
[7] Black P, Wiliam D. Developing the Theory of Formative Assessment[J]. Educational Assessment, Evaluation and Accountability, 2009, 21(1): 5-31.
[8] Yin R K. Case Study Research and Applications: Design and Methods[M]. 6th ed. Sage Publications, 2018.
[9] Martin R C. Clean Code: A Handbook of Agile Software Craftsmanship[M]. Prentice Hall, 2008.
[10] Carless D. Excellence in University Assessment: Learning From Award-Winning Practice[M]. Routledge, 2015.
[11] Engeström Y. Learning by Expanding: An Activity-Theoretical Approach to Developmental Research[M]. 2nd ed. Cambridge University Press, 2015.
[12] Eraut M. Informal Learning in the Workplace[J]. Studies in Continuing Education, 2004, 26(2): 247-273.
[13] Star S L, Griesemer J R. Institutional Ecology, 'Translations' and Boundary Objects: Amateurs and Professionals in Berkeley's Museum of Vertebrate Zoology, 1907-39[J]. Social Studies of Science, 1989, 19(3): 387-420.
[14] Schön D A. The Reflective Practitioner: How Professionals Think in Action[M]. Basic Books, 1983.
[15] Wenger E. Communities of Practice and Social Learning Systems[J]. Organization, 2000, 7(2): 225-246.
[16] Braun V, Clarke V. Using Thematic Analysis in Psychology[J]. Qualitative Research in Psychology, 2006, 3(2): 77-101.