Regression Testing Skill Transfer to Industry: A Preliminary Study in Higher Education

Andrada-Mihaela-Nicoleta Moldovan* and Andreea Vescan*
*Computer Science Department, Faculty of Mathematics and Computer Science, Babeș-Bolyai University, Cluj-Napoca, Romania
{andrada.moldovan, andreea.vescan}@ubbcluj.ro

***Abstract*—Regression testing is essential for software maintenance, but is often underrepresented in higher education. This study examines the understanding and application of regression testing among 176 third-year computer science students. Using the Goal-Question-Metric (GQM) framework, we conducted an empirical study with three tasks: test suite creation, bug fixing, and regression test selection, alongside questionnaires with open-ended, closed-ended, and Likert-scale responses. Results show that students mainly selected tests based on changes, affected methods, and dependencies, while reporting positive experiences with teamwork, problem-solving, and skill development. The findings highlight the value of hands-on, collaborative activities in reinforcing both technical and soft skills, suggesting that structured regression exercises can bridge the gap between academic instruction and industry needs.**

***Index Terms*—Regression Testing, Test Case Design, Software Testing Education**

I. Introduction

Software testing [1] is a fundamental step of the Software Development cycle in modern development. Its main purpose covers the quality, reliability, functionality, performance, and usability of the products delivered. With shorter development cycles and higher user expectations, testing has emerged as a crucial component of successful software delivery. In addition to identifying flaws, it also helps developers avoid them by providing insightful feedback.

Regression Testing (RT) [2] is a particular type of testing that involves ensuring that previously validated software components are not affected by new changes in ways that could cause unintended side effects. In the context of Continuous Integration (CI) and Continuous Delivery (CD) [3], where code artifacts are constantly changing and evolving, this step is crucial. Agile environments have set the standard for rapid feedback loops and frequent update handling, enabling developers and QA teams to apply efficient and scalable testing methods.

Although there is extensive literature on the effectiveness of automated and regression testing in industrial settings [4], [5], there is a significant gap in the preparation of higher education students to understand and apply these concepts in real-world contexts. Most university programs emphasize fundamental software development principles, but testing, especially regression testing, is often covered only superficially. This article addresses this gap by analyzing the level of familiarity and understanding of regression testing among over 200 higher education students. The goal is to identify knowledge deficiencies and offer recommendations for aligning academic curricula with the current needs of the software industry.

ORCID: 0009-0006-6289-9869
ORCID: 0000-0002-9049-5726

We aim to assess, through a simple practical application, the readiness of higher education students regarding software testing, with a focus on regression testing, in the context of modern practices such as CI/CD and Agile development. Additionally, we highlight the importance of education in developing the necessary and up-to-date skills demanded by the industry, as well as the significance of teamwork in the software testing process. The study is based on an assignment completed by 198 students, of which 176 answered questions related to their previous experience, familiarity with regression testing, and perceived readiness for future employment in the IT field.

Our research questions target key aspects of how students engage in software testing in an educational context. They address the strategies that students use when designing test cases, the approaches they take in selecting regression tests, and their reflections on the learning activity.

The limited presence of regression testing in higher education is addressed. To our knowledge, it is the first to examine the knowledge of senior computer science students about regression testing through a practical assignment, while documenting the results. We analyze their previous experience, perceived task difficulty, solution strategies, and reflective feedback to provide insights and recommendations for improving software testing education and aligning graduates' skills with industry needs.

The paper is organized as follows: Section II introduces background concepts, and Section III reviews related work. Section IV presents the methodology, including research questions, participants, and activities. Section V reports the results, followed by lessons learned (Section VI), threats to validity (Section VII), and conclusions and future work (Section VIII).

II. Background and Theoretical Concepts

This section introduces the theoretical foundations relevant to our study. We first present key concepts of regression testing, including its role in software maintenance and optimization strategies commonly adopted in practice. We then discuss the importance of testing in higher education, highlighting its

2151-0849/25/$31.00 ©2025 IEEE
DOI 10.1109/ASEW67777.2025.00034

contribution to student learning, skill development, and alignment with current curricular recommendations. Together, these perspectives provide the necessary background to understand the practical activity analyzed in this study.

A. Regression testing

In Software Engineering [1], testing is a fundamental activity to ensure the quality of applications. It verifies whether a system meets its specifications and helps identify defects before the product reaches the end user.

Test case design [2] often involves Black-box and White-box approaches. While Black-box testing focuses on functionality without knowing the internal code, White-box testing relies on understanding the code structure. These concepts are also reflected in the students' assignment, where they are required to specify the nature of the defined regression test.

A crucial part of software maintenance is regression testing, which makes sure that new code does not adversely affect already-existing functionality. Repeatedly running the whole test suite may get costly and time-consuming as software develops. Test Case Selection (TCS), Test Case Minimization (TCM), and Test Case Prioritization (TCP) are some of the optimization strategies that have been proposed to solve this. These techniques seek to increase regression testing's efficiency without sacrificing effectiveness.

TCS concentrates on selecting only the tests that are affected by the changes, whereas TCP attempts to run the most crucial tests first in regression testing. For increased efficiency, they are frequently merged into hybrid techniques.

B. Learning and Skill Development

Students use their theoretical knowledge in a real-world context by actively participating in a bug-fixing task, which helps them reinforce key testing concepts through active learning. In doing so, it stimulates technology literacy and encourages creative thinking, as well as analytical thinking, which are considered to be among the core skills for 2030 according to the *Future of Jobs Report*, published by the World Economic Forum [6].

Both the 2020 [7] and 2023 [8] Computing Curricula highlight Verification and Validation as essential competencies in undergraduate Computer Science degree programs. Core skills include designing and executing black-box tests, performing regression testing, using appropriate testing tools, and planning effective test case design. The 2020 report emphasizes that testing should be embedded across the curriculum, rather than treated as a standalone topic. The updated 2023 guidelines further stress the relevance of testing by integrating objectives related to test types, usability, reliability, security, and the distinction between testing and production code.

Following Bloom's taxonomy [9], students move from remembering and understanding concepts introduced in lectures, seminars, and labs to applying them in the assessment. Through bug-fixing, they analyze code, evaluate testing approaches, and create solutions, advancing from foundational knowledge to higher-order thinking skills.

III. Related Work

In this section, we survey research on regression testing approaches and on how software testing is addressed in higher education, highlighting the gap between them.

The academic community has extensively investigated RT, yet a gap persists between research advances and their adoption in practice. Greca et al. [10], in a live systematic literature review, observed that despite the breadth of research on regression testing, practical applicability remains limited, highlighting the disconnect between academic output and industrial requirements. This emphasizes the importance of better aligning educational content with real-world expectations. Marappan and Raja [11] identified ongoing challenges in test case selection, prioritization, and optimization, areas that are rarely addressed in-depth within undergraduate or postgraduate software engineering programs. Furthermore, Mafi and Mirian-Hosseinabadi [12] explored the selection of regression tests within test-driven development, showcasing its relevance in structured development workflows and strengthening the pedagogical value of including regression testing concepts in academic instruction. These studies collectively argue for regression testing not only as a valuable research area but also as a crucial topic to be taught in higher education, ensuring students are equipped with practical, in-demand testing competencies upon entering the workforce.

Numerous recent studies have examined how software testing is experienced and taught in educational environments. Jokisch et al. [13] and Barrett et al. [14] are two examples of broad curriculum analyses that highlight discrepancies in the way testing subjects are taught in higher education institutions, with some important areas—like RT—frequently being underrepresented. As a result, several pedagogical strategies have been developed to enhance testing education. For example, gamified approaches to teaching and testing were investigated by Straubinger et al. [15], [16], Nurue et. al [17], and Silvis-Cividjian et al. [18]. They introduced serious games and playful tools aimed at a variety of learner groups, ranging from elementary school students to college students. The goal of these initiatives is to make abstract testing ideas more approachable and interesting. In parallel, Fang et al. [19] investigated the role of AI-based support and structured checklists in helping students grasp unit testing principles, while Cammaerts et al. [20] and Iudean et. al [21] studied students' behavior in model-based testing and storytelling-driven CI/CD education, respectively. Despite these significant contributions, most programs focus on unit testing, test understanding, or engagement strategies. Regression testing is still commonly overlooked in professional workflows and software maintenance, despite its critical importance.

Although prior research highlights the importance of regression testing and testing education separately, we uniquely combine practical tasks with reflection to promote a better understanding of the subject. This empirical approach to active learning complements studies on the theory-practice gap [10] in RT [13].

IV. Methodology

This research investigates student strategies for test case design, regression test selection, and reflective evaluation of learning activities. This section presents the approach, research questions, participants, activities, and data collection methods (that were chosen based on the objective of the study, namely, using activities to evaluate the applied of concepts learned by students as in Bloom taxonomy [9].

A. Overview

Our study employs an empirical course-based methodology to examine how computer science students engage in regression testing. We combine hands-on programming and testing assignments with the collection of questionnaire responses and reflective feedback. The research design follows three main steps. First, students work in teams to complete a programming and testing assignment that involves bug fixing, unit test creation, and regression test selection. Second, their solutions and documented reasoning are analyzed to extract patterns of test design and regression testing strategies. Third, we collect the perceptions, experiences, and reflections of the students through a questionnaire that integrates open-ended and Likert-scale questions.

B. Research Questions

The Goal Question Metric (GQM) framework, as provided by Basili and Rombach [22], was used to develop the study questionnaire. This framework outlines the crucial steps for setting goals prior to beginning any software measurement activity. As a result, the initial measurement *Goals* is established, and then a suitable model related to the items under evaluation is selected. Following the formulation of *Questions* to ascertain the attainment of each aim, the relevant *Metrics* are picked in accordance with the model that has been selected.

The goals of the study are the following:

- G1 - To determine the criteria, heuristics, and knowledge the students apply when choosing which test cases to re-execute during software maintenance tasks in the regression testing process.
- G2 - To understand and characterize the strategies, patterns, and decision-making processes they employ during test case creation, concerning their effectiveness, completeness, and alignment with taught testing techniques.
- G3 - To characterize and analyze the reflective self-assessments provided by students after completing the activity, for the purpose of understanding their perceptions, emotional responses, evaluations of the experience, identified learning outcomes, and intended improvements (using the Gibbs Reflective Cycle [23]).

The research questions are as follows:

RQ1: How do students select regression tests?

RQ2: What strategies do students use to design test cases?

RQ3: What are the students' reflective opinions about the performed activity (activity, feelings, positive and negative elements, learned concepts, skills)?

We use the following four metrics to answer the three research questions.

- M1: *Open-ended questions that are coded, quantified, and categorized* about how students identified the tests to be re-used.
- M2.1: *Frequency* of using the Black-Box Testing (BBT) and White-Box Testing (WBT) strategies by the student teams.
- M2.2: *Likert scale results* about various aspects of the activity performed.
- M3: *Open-ended questions that are coded, quantified, and categorized* about perceptions, emotional responses, evaluations of the experience, identified learning outcomes, and intended improvements.

The first research question will be investigated by coding and analyzing the students' explanations for the selection of regression tests in assignments and survey responses (M1).

The second research question will be explored by examining the implemented test suites together with the justifications the teams were required to provide as part of the assignment.

The third research question is addressed by applying Gibbs' reflective cycle [23] to open-ended reflections and responses on the Likert scale of students.

C. Course and Participants

1) Course Description: The study was carried out in the Department of Computer Science, Faculty of Mathematics and Computer Science, Babes-Bolyai University, as part of a third-year computer science course called *Software Systems Verification and Validation (SSVV)* on software verification and validation topics. Students learn about the procedures of software delivery and testing in the context of CI/CD pipelines by covering Test-driven development, Behavior-driven development, deployment to production, staging environments, and automation tools for building and testing.

2) Participant Demographics: Third-year computer science students enrolled in the SSVV course served as the study's participants. A total of 210 students participated in the SSVV class, of which 198 completed the tasks (being part of the evaluation). Among them, 176 answered questionnaires and provided feedback on the instructional strategy. All students performed the tasks; however, only a few completed also the questionnaire. Demographic information is provided in Table I. The university's Scientific Council ethically approved the research.

Nearly half (47.16%) of the respondents said they had worked in the sector for less than a year, while just over a

TABLE I: Demographic information of the respondents.

Information	Number
No. of Respondents	176
Female	46
Male	125
Other	5
Age	20 (1), 21 (99), 22 (70), 23 (5), 25 (1)

quarter (26.70%) said they had never worked in the business. 21.59% claimed about one year in the industry, and only 4.54% had three years or more of industry experience.

Regarding experience with the JUnit Framework, 61.36% of the students report using it for less than a year, including course activity. Only 10.80% have been using it for 2 years, 23.86% for about 1 year, while just 3.98% claim to have been using it for 3 or more years.

Most of the participants had minimal prior experience with regression testing, which makes them suitable to assess the learning approach under study. Their background reflected a broad range of general software skills, but few had substantial hands-on exposure to structured regression testing practices: 90.11% had less than a year of experience, 7.14% had about one year, and only 2.75% had two years of practice. From a proficiency standpoint, the overwhelming majority could be considered *novices*, with a small proportion at the *beginner* level, and a rare few reaching *competent*. None could be classified as proficient or expert.

D. Description of Activities

The activity was conducted in teams of up to five students based on how we approximated the time needed to solve the tasks) in a Java program (Version A) that had to be modified to fix a known bug (Version B), with materials available in figshare [24]. (1) Students analyzed the program, fixed the bug, and reflected on its impact. (2) They designed and implemented a JUnit test suite for Version A. (3) From this suite, they selected tests to run on Version B to verify the fix and detect regressions, documenting the results throughout.

E. Collecting Data

Figure 1 presents the timeline for the learning activities dedicated to designing test cases and regression testing, which include lectures, seminars, and laboratory activities. The activities performed, as described above, are also encapsulated in the three tasks. The questionnaire is filled in by the respondents at the end of the performed activities, each student for all the formed teams.

The first layer captured assignment results with JUnit tests for Version A and re-execution on Version B, showing how tests were designed and reused. The second layer included the' written explanations of the students' reasoning, test choices, and regression selection. The third layer came from questionnaires with open-ended answers, ratings, and time estimates.

Table II summarizes key survey questions related to applied concepts and students' perceptions of the learning process. Our exploration focused on three main aspects: the regression

Fig. 1: Overview research study activities.

testing aspect, the learning aspects, and reflective questions about learning. The questionnaire was composed of four parts, as provided in Table II.

TABLE II: Student's Questionnaire Investigated Aspects

Qs. Set	Questions's Target Aspects
RToQs	Regression testing related questions using open-ended types of questions.
LIQs	Learning activity related questions using Likert scale types of questions.
TvQs	Time for each activity performed by the students using a grid with many options.
RoQs	Participants' perceptions of the overall learning experience.

RToQs: Regression testing related questions using open-ended types of questions (in the context of Research Question 1). These elements asked students to describe how they selected regression tests and which criteria guided their decisions, providing qualitative evidence of the strategies used for test reuse and selection.

LlQs: Learning related and activity-related questions using Likert scale types of questions (in the context of Research Question 1 and Question 2). These elements focused on the perceived understanding of regression testing concepts by students, as well as their evaluation of the difficulty and usefulness of the assignment. The results offered a quantitative measure of how well the activity supported learning objectives.

TvQs: Time for each activity performed by students using a grid with many options. These items asked participants to estimate the time invested in coding, test design, debugging, and documentation. Based on the results, we analyzed the effort distribution between tasks and compared workload perceptions between teams.

RoQs: Perceptions of participants about the overall learning experience (in the context of Research Question 3). These elements encouraged reflection on positive and negative aspects of the assignment, as well as concepts and acquired skills.

They helped by providing insight into students' emotions, evaluations, and suggestions for improvement.

More information on the questionnaire and the included questions is available in figshare [24].

V. Results

This section presents the main findings of our study.

A. General Results with team formation

A total of 68 teams were created, with the distribution of team sizes shown in Figure 3 (green column). For example, there were 11 teams consisting of 1 student, 22 teams of 3 students, and 11 teams of 5 students. The recommendation for completing the tasks was to form teams of at least 3 and at most 5 students. A notable challenge was that some students did not comply with this recommendation.

Students from different teams created a variable number of test cases, the lowest average recorded by single-member teams (an average of 7 test cases, as shown in Figure 2) and the highest average by four-member teams (14 test cases). Teams of three and five members had the same average number of test cases, namely 10.

Regarding the number of test cases selected for regression testing, the average across all teams was lower than the average number of test cases designed for version A. However, it should be noted that some teams maintained the same number of test cases (adopting a retest-all strategy). The largest difference between the average number of designed test cases and the average number of selected test cases for regression testing was observed in four-member teams (5 test cases), followed by five-member teams (4 test cases), and two- or three-member teams (2 test cases on average). Single-member teams had the smallest difference, with an average of only 1 test case.

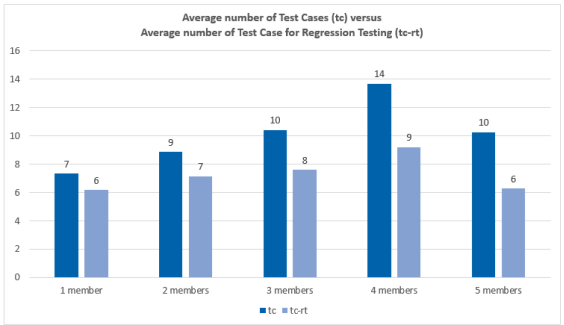

Fig. 2: Average number of test cases and test cases for regression testing.

With regard to the design techniques used, Figure 3 illustrates the number of teams in each category that used only BBT, only WBT, or both methods. It can be seen that the highest number of teams applying both methods is found among the three-member teams, where 13 teams adopted this approach. Among the 15 teams of two-members, 9 employed both methods. WBT techniques were used only by two and

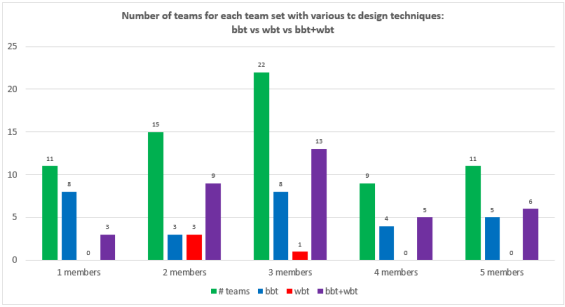

Fig. 3: Test case design techniques used by students for each set of teams (bbt=black-box testing, wbt = white-box testing).

three-member teams, in relatively small proportions: 3 two-member teams and 1 three-member team. Although many teams in each category applied BBT, a larger number of teams combined both methods (as indicated by the purple columns being taller than the blue columns).

Questions set TvQs – Time for each activity performed by the students using a grid with many options.

Each student estimated the time they spent on each job, although the activities were carried out in groups. Four questions were answered: (1) implementing test cases for Version A, (2) fixing the bug, (3) assessing whether the fix affected functionality, and (4) deciding which tests to reexecute. For each, students selected one of six predefined time ranges (*less than 5 minutes*, *5 minutes*, *10 minutes*, *15 minutes*, *20 minutes*, or *more than 20 minutes*) from a multi-option grid.

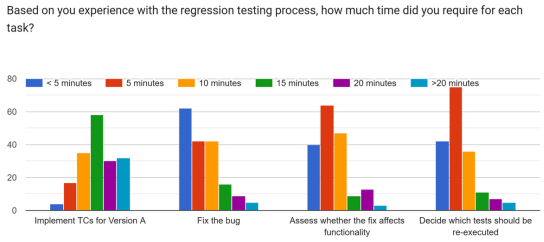

Fig. 4: Responses on the time for the various aspects and tasks performed.

Figure 4 outlines the time required for various tasks in the research study activity. For *Implementing test cases for Version A*, most students reported needing around 15 minutes, suggesting that designing a complete set of tests required careful consideration. For *Fixing the bug*, the majority indicated less than 5 minutes, likely because the error had already been identified and the codebase was relatively simple. For *Assessing whether the fix affected functionality*, the most common response was 5 minutes, reflecting the straightforward verification required after the change. Finally, for *Deciding which tests should be reexecuted*, the dominant choice was also 5 minutes, indicating that test selection was perceived as a quick and intuitive step

once the suite was in place.

B. Results on test design and regression testing using Open-ended and Likert scale questions

The results reveal a variety of approaches to test design and regression testing, the majority of which are related to relationships between changes and methods. Open-ended answers may reinforce or reevaluate students' confidence regarding their solutions, while Likert-scale data reflect their perceived effectiveness in applying testing concepts.

RToQs questions (Q1, Q2) - Regression testing related questions using open-ended types of questions.

Q1: How did you choose the tests to use for the regression?

We identified 6 main categories of codes that summarize the reasoning patterns students reported when justifying their test selection. These include: *Relation to* (e.g., method, changes, bug fixes, dependencies), *Explanations general* (same tests, all functionalities, unaffected tests), *Answers not Ok* (no or incorrect explanation, unrelated tests only), *Techniques used* (WBT, branch coverage, BBT), *Versions* (tests from version A, version B, or both), and *Tests that failed* (initial failures or failures in version B). More information regarding students' choices and a mapping of the keywords identified in their answers is available in our Figshare package [24].

The distribution of the main codes found in the' responses of the students was also analyzed (due to space limitations, the figure is included in the Figshare package [24]). The most frequent category by far is *Relation to*, accounting for 71% of responses (143 mentions). In contrast, other categories appear less frequently: *Answers not Ok* (9%, 18 mentions), *Versions* (7%, 15 mentions), *Tests that failed* (6%, 12 mentions), *Techniques used* (4%, 9 mentions), and *Explanations general* (2%, 5 mentions).

Relation to appears as the most frequently invoked argument for test case selection, encompassing justifications based on links to affected methods, changes, bug fixes, functionalities, dependencies, or impact. More detailed word cloud graphics for each of the four main subcategories within Relation to (methods, changes, bug fixes, dependencies) are included in the Figshare package provided [24].

Q2: Did the bug fix (change) affect other existing functionalities?

The coding of the responses to this open question shows that most students were able to create a fix that would affect other functionalities. Most of the answers were coded as *No* (131) or *yes, no explanations* (7), indicating that many students simply confirmed that they were running tests without articulating a rationale. Only 18.18 % referred to specific elements of the system, such as the *PaymentProcessor* (8), the *final amount* (11), or *pricing calculation* (13). Since the goal of regression testing is to verify that the existing functionality remains correct, there is no single "right" yes or no answer. Interestingly, some fixes did affect other functionalities while others did not, as this reflects how interconnected different parts of the system can be and emphasizes the importance of carefully selecting tests that can reveal unintended side effects.

LlQs questions (Q3-Q7) - Learning activity-related questions using Likert scale types of questions.

We included five **Likert-scale questions** to capture students' perceptions of task clarity, difficulty, and ease. These elements, listed in Table III, address understanding the instructions, fixing the bug, implementing and reusing tests, and documenting design techniques.

TABLE III: Statements from Q3–Q7 regarding LlQs aspects.

Q	Statement
Q3	The tasks I had to performed were perfectly clear to me.
Q4	Fixing the bug was easy.
Q5	Implementing regression tests for Version A was easy.
Q6	Identifying the tests to be re-used for Version B was easy.
Q7	Recording what design techniques to use was easy.

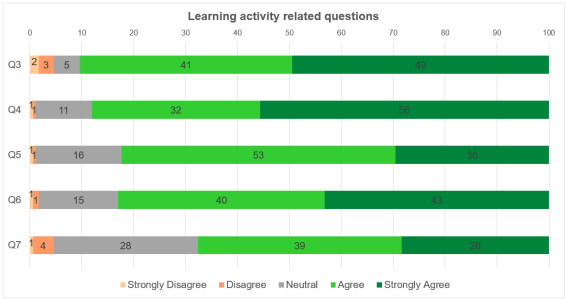

Fig. 5: Responses on the statements from LlQs aspects (Likert scale).

For most of the statements, the percentages of students who responded with *Agree* or *Strongly Agree* are dominant. For example, regarding the statement *The tasks I had to perform were perfectly clear to me,"* 90% of students agreed. For *Fixing the bug was easy,"* disagreement was almost absent, with only 2% selecting this option. A similar trend can be seen for *Implementing regression tests for Version A was easy,"* where the vast majority (83%) expressed agreement. Likewise, in the case of *Identifying the tests to be reused for Version B was easy,"* disagreement was again minimal, at just 2%. Finally, for *"Recording what design techniques to use was easy,"* agreement was still the prevailing response (67%), although in this case the distribution was slightly more balanced compared to the other items. Figure 5 illustrates these results in detail.

C. Results on the perception on the performed activity using Open-ended questions

RoQs questions (Q8-Q13) - Participants' perceptions of the overall learning experience

The second set of reflective questions (Q8–Q13) focused on students' experiences during the activity, their emotional responses, evaluations of what went well or poorly, and the skills they considered necessary for future improvement [23].

The reflections of the students on Q8–Q13 (see Table IV) highlight both their concrete experiences during the assignments and their perceptions of teamwork, problem-solving, and skill development.

TABLE IV: Statements from Q8–Q13 regarding RoQs aspects.

Q	Statement
Q8	What happened during the activity (solving the assignments)? What did you do? What was the outcome?
Q9	What were you feeling during the situation? How do you feel about the situation now?
Q10	What went well? What didn't go well? What positive or negative things did you (or other people from your team) contribute to the situation?
Q11	Why did (or didn't) things go well? What theories or research can help you better understand the situation?
Q12	What did you learn from this situation? If this situation happened again, what would you do differently?
Q13	What skills do you need to develop to handle a situation like this better? How will you develop the skills you need?

As shown in Figure 6, generated with MAXQDA 2022 [25] using the *Smart Coding* and the *Analyze Sentiments* functionalities, most of the responses were categorized as Positive or Slightly Positive.

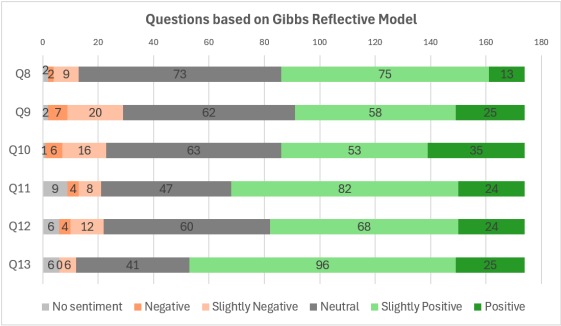

Fig. 6: Responses on the statements from RoQs aspects, Participants' perceptions of the overall learning experience.

The students described the activity as an iterative process of testing and debugging, where collaboration and task division facilitated functional outcomes and gradually transformed initial stress into satisfaction and confidence. Positive experiences were largely related to effective teamwork, communication, and the application of established testing principles, while challenges such as documentation, setup, or time constraints were generally resolved through mutual support. The reflections further emphasized lessons on planning, early testing, and systematic coverage of edge cases, alongside the need to strengthen both technical skills (e.g., debugging, testing strategies, Java proficiency) and soft skills (e.g., communication, critical thinking, time management), underscoring the importance of continuous practice and real-world application.

The responses on the last question "Please leave your final thoughts. Anything that you want to share and we did not ask about this activity." were analyzed using MAXQDA 2022 [25] using the *Smart Coding* functionality and the *Analyze Sentiments* option. Most responses were Positive or Slightly Positive, while the few entries categorized as Negative were minimal and non-substantive. More details are available in figshare [24].

D. Answers to the Research Questions

Answer to RQ1: Students most often selected tests based on their relation to the modified code, bug fix, or dependencies, but strategies also included rerunning all tests for assurance, using testing techniques, focusing on failed tests, or leveraging version-specific choices. They find selecting suitable regression tests slightly more difficult than fixing the bug and initially implementing the suite, with 83% of them agreeing on the task being easy. Justifying the choice was reported as the most difficult task, proving that students are more experienced with software development than testing.

Answer to RQ2: While many teams used black-box testing, the majority combined both black-box and white-box approaches, particularly among three-member teams. The number of test cases created varied with team size, ranging from an average of 7 (single-member teams) to 14 (four-member teams).

Answer to RQ3: Students reflected that the activity was a practical and collaborative process of testing and debugging, which initially seemed difficult but soon led to satisfaction and confidence. Teamwork, communication, and systematic testing were key to success, while challenges such as setting up and managing time were resolved mainly through cooperation. They emphasized lessons on better planning, early testing, and understanding the broader impact of code changes. In general, the activity was valued as a meaningful, hands-on, and less stressful alternative to traditional exams.

VI. Lessons Learned

This chapter summarizes key insights, challenges, and observations from the regression testing activity.

Challenges Faced. A notable challenge was that some students did not comply with the team formation recommendation in terms of the number of members. Another limitation was the imbalance between students with and without prior experience, which may have influenced the results.

Lesson 1: RT selection. Students selected tests based on changes or affected methods. A notable pattern is that students tend to adopt a localized perspective in their test selection. They primarily concentrate on the immediate sites of *change*, including the modified method or its direct dependencies, showing an awareness of structural coupling, but their reasoning rarely extends to indirect or transitive effects.

Lesson 2: Test case design and RT. Students mainly used mixed black-box/white-box methods when designing test cases. The prominence of *method* in students' responses reveals a method-centric mindset. This suggests that regression testing is frequently equated with the verification of individual

units of code, which may be a consequence of the way that programming and testing are introduced in educational contexts.

Lesson 3: Bug fix. When discussing tests related to bug fixes, students often invoked terms related to functionality, impact, and coverage, suggesting that they considered not only the structural location of the fix but also its behavioral implications.

Implications for Academic and Industry Settings. Structured regression exercises effectively build practical testing skills, teamwork, and reflective practices, preparing students for real-world software development and supporting efficient test maintenance in industry.

VII. Threats to validity

Construct Validity: Different students may interpret measuring terms such as "easy" differently, and self-reported measures may be prone to bias. Students who performed the exercise correctly may have perceived it as more difficult compared to those who found it easy, but may have failed to detect and address the required changes thoroughly.

Internal Validity: Although background experience is captured in the first four questions, these variations may have a significant impact on task difficulty ratings.

External Validity: Because the participants are all third-year CS students working in a controlled academic setting with Java and JUnit, the findings may not be generalized to industry practitioners, real-world project constraints, or regression testing practices in other technologies.

VIII. Conclusion and Future Work

The study examines the strategies of the students for test design, regression test selection, and self-evaluation using the GQM framework. Participants fixed a Java bug, designed tests, and assessed changes, with data gathered through questionnaires. The results show that the students performed the tasks effectively, benefiting from teamwork, structured methods, and prior knowledge. Reflections indicate that the activity was meaningful and less stressful than traditional assessments. Future work should explicitly teach regression testing and offer practice in labs, seminars, and project-based contexts, and also trace the development of RT skills from higher education to industry.

References

[1] J.-F. Collard and I. Burnstein, *Practical Software Testing*. Springer-Verlag New York, Inc.,, 2003.

[2] P. Ammann and J. Offutt, *Introduction to software testin*. Cambridge University Press, 2016.

[3] H. Leskinen, "Teaching and learning devops," Ph.D. dissertation, Aalto University, 2024. [Online]. Available: https://aaltodoc.aalto.fi/items/18963e8c-72d1-4dec-8b3c-810617f2130c

[4] N. B. Ali, E. Engström, M. Taromirad, M. R. Mousavi, N. M. Minhas, D. Helgesson, S. Kunze, and M. Varshosaz, "On the search for industry-relevant regression testing research," *Empirical Software Engineering*, vol. 24, no. 4, pp. 2020–2055, 2019.

[5] N. M. Minhas, K. Petersen, J. Börstler, and K. Wnuk, "Regression testing for large-scale embedded software development–exploring the state of practice," *Information and software technology*, vol. 120, p. 106254, 2020.

[6] T. Leopold, A. Di Battista, X. Jativa, S. Sharma, R. Li, and S. Grayling, "Future of jobs report 2025," in *World Economic Forum. https://www.weforum. org/reports/the-future-ofjobs-report-2025*, 2025.

[7] A. Clear, A. S. Parrish, J. Impagliazzo, and M. Zhang, "Computing curricula 2020: Introduction and community engagement," in *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, ser. SIGCSE '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 653–654. [Online]. Available: https://doi.org/10.1145/3287324.3287517

[8] A. N. Kumar, R. K. Raj, S. G. Aly, M. D. Anderson, B. A. Becker, R. L. Blumenthal, E. Eaton, S. L. Epstein, M. Goldweber, P. Jalote, D. Lea, M. Oudshoorn, M. Pias, S. Reiser, C. Servin, R. Simha, T. Winters, and Q. Xiang, *Computer Science Curricula 2023*. New York, NY, USA: Association for Computing Machinery, 2024.

[9] "Using bloom's taxonomy," https://tips.uark.edu/using-blooms-taxonomy/, accessed: 2025-08-15.

[10] R. Greca, B. Miranda, and A. Bertolino, "State of practical applicability of regression testing research: A live systematic literature review," *ACM Computing Surveys*, vol. 55, no. 13s, pp. 1–36, 2023.

[11] R. Marappan and S. Raja, "Recent trends in regression testing: Modeling and analyzing the critiques in selection, optimization, and prioritization," *National Academy Science Letters*, pp. 1–7, 2025.

[12] Z. Mafi and S.-H. Mirian-Hosseinabadi, "Regression test selection in test-driven development," *Automated Software Engineering*, vol. 31, no. 1, p. 9, 2024.

[13] C. Jokisch, K. Schramm, S. Hobert, L. Wilhelmi, and M. Schumann, "Structured analysis of software testing education in higher education in germany," in *2025 IEEE/ACM 37th International Conference on Software Engineering Education and Training (CSEE&T)*. IEEE, 2025, pp. 98–107.

[14] A. A. Barrett, E. P. Enoiu, and W. Afzal, "Gaps in software testing education: A survey of academic courses in sweden," in *2025 IEEE/ACM 37th International Conference on Software Engineering Education and Training (CSEE&T)*. IEEE, 2025, pp. 108–117.

[15] P. Straubinger, T. Greller, and G. Fraser, "Sojourner under sabotage: A serious testing and debugging game," in *Proceedings of the 33rd ACM International Conference on the Foundations of Software Engineering*, 2025, pp. 738–748.

[16] P. Straubinger, L. Bloch, and G. Fraser, "Teaching loop testing to young learners with the code critters mutation testing game," in *2025 IEEE/ACM 37th International Conference on Software Engineering Education and Training (CSEE&T)*. IEEE, 2025, pp. 347–358.

[17] H. D. Nurue and J. Gray, "Test early: An approach to introduce younger students to unit testing with graphical comparison in scratch," in *2025 IEEE/ACM 37th International Conference on Software Engineering Education and Training (CSEE&T)*. IEEE, 2025, pp. 129–134.

[18] N. Silvis-Cividjian, J. Veltman, A. Buchel, E. Link, J. Kenyon, and M. Oey, "Bug hunting games to add enthusiasm in software testing and programming classes," in *2025 IEEE/ACM 37th International Conference on Software Engineering Education and Training (CSEE&T)*. IEEE, 2025, pp. 359–364.

[19] Z. Fang, J. Li, A. Liang, G. R. Bai, and Y. Huang, "A comparative study on chatgpt and checklist as support tools for unit testing education," in *Proceedings of the 33rd ACM International Conference on the Foundations of Software Engineering*, 2025, pp. 871–882.

[20] F. Cammaerts, B. Marín, and M. Snoeck, "Exploring how students test models in model-driven engineering," in *Proceedings of the 2025 CSEE&T Software Engineering Education Track*, Track: CSEE&T 2025 Software Engineering Education, Session: Automated Feedback and Code Quality Chair(s): Sandro Speth, Apr. 2025.

[21] B. Iudean and A. Vescan, "Layered learning: Teaching ci/cd and software testing through storytelling," in *Proceedings of the 33rd ACM International Conference on the Foundations of Software Engineering*, 2025, pp. 722–731.

[22] V. Basili and D. Rombach, "The tame project: towards improvement-oriented software environments," *IEEE Transactions on Software Engineering*, vol. 14, no. 6, pp. 758–773, June 1988.

[23] G. Gibbs, *Learning by Doing: A guide to teaching and learning methods*. Oxford Polytechnic: Oxford, 1988.

[24] A. Vescan and A.-M.-N. Moldovan, "Regression testing study," 2025, available at https://doi.org/10.6084/m9.figshare.29976346.

[25] VERBI Software, "Maxqda 2022," Computer software, Berlin, Germany, 2021. [Online]. Available: https://www.maxqda.com