ON THE ASSESSMENT OF STUDENTS' SOFTWARE TESTING AND PROGRAMMING LANGUAGE SKILLS – CAMEROON CONTEXT: AN EXPERIMENTAL APPROACH

DJAM Xaveria Youh, Aminou Halidou, TAPAMO Kenfack Hyppolyte Michel, Atsa Roger Etoundi

ABSTRACT

Despite decades of researches and technological advancements in software development, there is still a wide gap between good programmers, good software testers, software industry and the academic world. Higher institutions continue to rely on traditional teaching methods which fails to leverage recent technological improvements. The complexity of software systems has made traditional assessment methods inadequate to meets new programming demands. Over the last decades, the fast-growing complexity of software systems has forced academic world to improve to a certain extent on teaching techniques in order to meet up with industrial demands. Nevertheless, there is still a wide gap as theory does not match practical. This paper presents a novel performance assessment method for assessing students' practical work not only in Software Testing courses but equally Programming Languages courses, by establishing an eye-tracking practical monitoring system in order to improve the rate of students' performance. We developed a novel assessment algorithm that used five performance criteria that were used in two Software testing and programming language courses for two consecutive semesters in 2023-2024 academic year in University of Yaoundé 1, Yaoundé, Cameroon. The assessment was done in two levels: Students-self assessment and Lecturer assessment of students' group practical using five features for assessment. After using this approach in two courses for two consecutive semesters, we observed significant improvements of students' learning outcome in various aspects. The results revealed that, the use of scoring elements with linguistics variables can also produce an evidence-based students' growth assessments direction showing whether a student is either improving or not in an iterative assessment.

Index Terms: *Students, University, Performance Assessment, Coverage Testing, Software Testing, Programming Language, Testing Criteria.*

Reference *to this paper should be made as follows****:*** *DJAM Xaveria Youh, Aminou Halidou, TAPAMO Kenfack Hyppolyte Michel, Atsa Roger Etoundi, (2025), "On The Assessment of Students' Software Testing and Programming Language Skills – Cameroon Context: An Experimental Approach" Int. J. Electronics Engineering and Applications, Vol. 7, No. 2, pp. 21-41.*

Biographical notes:

DJAM Xaveria Youh - KIMBI *is currently a Senior Lecturer in Department of Computer Science, Faculty of Science, University of Yaounde I, Cameroon. She holds a Ph.D. in Computer Science from Modibbo Adama University of Technology, (MAUTECH) Yola which she obtained in 2011. Her research interests include Search-based Software Engineering, Information System Modelling, Software Testing and Cloud Computing.*

Hippolyte Michel Tapamo Kenfack *is a professor in the Department of Computer Science of the Faculty of Sciences of the University of Yaoundé I, Cameroon. He obtained his Ph.D. in Computer Science from University of Yaoundé I, Cameroon. He is a specialist in computer vision and artificial intelligence. He is interested in medical imaging and image analysis for the prevention of natural disaster risks such as floods and landslides.*

Aminou Halidou *is a computer science expert born in Cameroon in 1979. He holds a Ph.D. in Computer Science from Huazhong University of Science and Technology (HUST) in Wuhan, China, which he obtained in 2014. His research expertise lies in computer vision and image processing. Since 2014, Aminou has been a lecturer at the University of Yaoundé 1, where he currently serves as the Head of the Computer Science Department.*

Roger Atsa Etoundi *is a distinguished academic affiliated with Université de Yaoundé I in Cameroon, where he has contributed significantly to the field of information systems and technology since 2005. His research primarily focuses on the intersection of information and communication technology (ICT) and public administration,*

particularly within the context of developing countries. Etoundi's recent work addresses pressing contemporary issues, such as the impact of ICT in managing road traffic control during the COVID-19 pandemic, showcasing his commitment to applying theoretical frameworks to real-world challenges. His publications reflect a deep engagement with topics like e-government, process optimization, and the development of the digital economy in Cameroon

---

1. INTRODUCTION

Software development is a resource-intensive, time-consuming, labor-intensive and error-prone process. Despite decades of researches and technological advancements in software development, there is still a wide gap between good programmers, good software testers, software industry and the academic world (Djam and Blamah,[1]). Higher institutions continue to rely on traditional teaching and assessment methods which fails to leverage recent technological improvements. The complexity of software systems have made traditional assessment methods inadequate to meets new programming demands.

Among various aspects of education, the assessment of learning outcomes is critical in ensuring good pedagogical quality and has always been difficult. Because computer programming is problem-solving oriented and very practical, the assessment of programming learning performance is challenging. The traditional assessment is not easy to be adapted to various new developments in computer programming education. New teaching approaches such as collaborative learning, project-based learning, e-Learning and m-Learning are endeavors of exploring new ways to boost learning outcomes

Different factors have influenced poor students' performances in software testing and programming courses through specific surveys (Yanqing, et al., [2], Adzima [8]). Amongst several, one of the most important is the lack of efficient practical follow-up methods to assess students' programming skills. This paper presents a new assessment method for assessing students' practical work not only in Software Testing courses but equally Programming Languages courses, by establishing an eye-tracking practicals monitoring system in order to improve the rate of students' performance. We developed an assessment desgin strategy with performance criteria that were used in two Software testing and programming language courses for two consecutive semesters in 2023-2024 academic year in University of Yaounde 1, Yaounde, Cameroon. The approach was used as a formal assessment approach in those courses. The assessment approach covered the following five features: test case document, coverage testing, functionality and execution, user experience and object-oriented programming concepts compliance.

This paper is structured as follows: Section 3 presents a background work on software testing and programming language learning based on peer code review. Section 3 describes different aspects of the proposed practical assessment approach: an assessment algorithm, performance assessment criteria, approach features and quality assurance strategies. Section 3 presents analytic results of two case studies while Section 5 gives the conclusion and perspectives.

2. SOFTWARE TESTING AND PROGRAMMING LEARNING BASED ON PEER CODE REVIEW

The complexity of software systems have made traditional assessment strategies inadequate. Among various aspects of education, the assessment of learning outcomes is critical in ensuring good pedagogical quality and has always been difficult. Because computer programming is problem-solving oriented and very practical, the assessment of programming learning performance is challenging. The traditional assessment is not easy to be adapted to various advanced technological developments in computer programming. New teaching approaches such as collaborative learning, project-based learning, e-Learning and m-Learning are endeavors of exploring new ways to boost learning outcomes thereby meeting industrial demands (Abdul-Rahman and Boulay, [3]; Bejarano et al.,[4]; Alexandron et al.,[5]; Pregitzer and Clements,[6], Akbulut et al., [9]) investigated software testing practicals through analytical online learning while preserving academic integrity. Furthermore, (Rowe [10]) applied code review process in software testing courses to minimize the difficulties in learning and teaching software testing and programming courses.

The traditional assessment approach, in which one single written examination counts toward a student's total score, no longer meets new demands of Software teesting and programming language courses (Yanqing, et al., [2]). Observing the successful practices of code review in software industry, many computer science

researchers have shown their interests in introducing peer code review processes to their courses to minimise academic dishosnety (Gehringer, et al., [12]; Sitthiworachart and Joy, [13]; Trytten, [14]). A good number of studnets' learning outcomes have been reported to assess students' performance in software testing courses (Li [17]; Turner and Pérez-Quiñones, [16]). Turner [15] described positives effects of code review in teaching freshmen concepts of object oriented programming lanaguage. Li [17] assessed coding quality when students used code review in programming language. According to (Li, [18]) incorporating a code review process into the students' assessment for evaluating the rate of performance reduced the act of copied and pasting codes in a source code window.

In light of the above, different factors have influenced students poor performances in software testing courses. There exist a wide research gap in the above mentioned researchers' articles due to the fact that no accurate assessment performance strategy was developed in their articles to improve students performance taking into cognizance branch coverage, statement coverage, function coverage and other pivotal assessment criteria that could bridge the gap between software industry and the academic world for software-intensive systems.

3 SOFTWARE TESTTING AND PROGRAMMING LANGUAGE SKILLS: ASSESSMENT DESIGN APPROACH

3.1 Features for Practical Assessment Approach

The proposed assessment approach has five desired features described in Table-1 and five performance criteria described in Table-2.

To ensure continuous follow-up of students' programming skills in Software Testing and programming courses in University of Yaounde 1, Yaounde, Cameroon in order to improve the rate of performance, we used a novel approach to assess and monitor students' performances. The assessment was done in two levels: Students-self assessment and Lecturer assessment of students' group practical using five features for assessment (Test Case Document, Coverage Testing, Functionality, User Experience and Object-oriented Programming Concepts).
A teacher role is responsible for giving assignments and setting schedules. Only a teacher role or a teacher assistant role can inspect and grade student submission. The assessment process begins with a new assignment from a teacher and ends with score check by students. The review process is specified by an activity diagram depicted in (Figure-1) and the corresponding flow diagram in (Figure-2). In Figure-2, SUI represents System Under Investigation while GP1, GP2, - - - GPn represent the various students' groups.

- Test Case document:
- Coverage Testing:
- Functionality and Execution:
- User Experience:
- Object-oriented Programming (OOP) Concepts:

In this study, two sets of practical exercises were given to three groups of students, each group was made up of 25 students each to perform software testing in JavaScript using ViTest as Unit Testing Tool.

First Practical Assignment: Results Management System

Second Practical Assignment: Unit Testing in ViTest to test the following scenario in Unit testing.

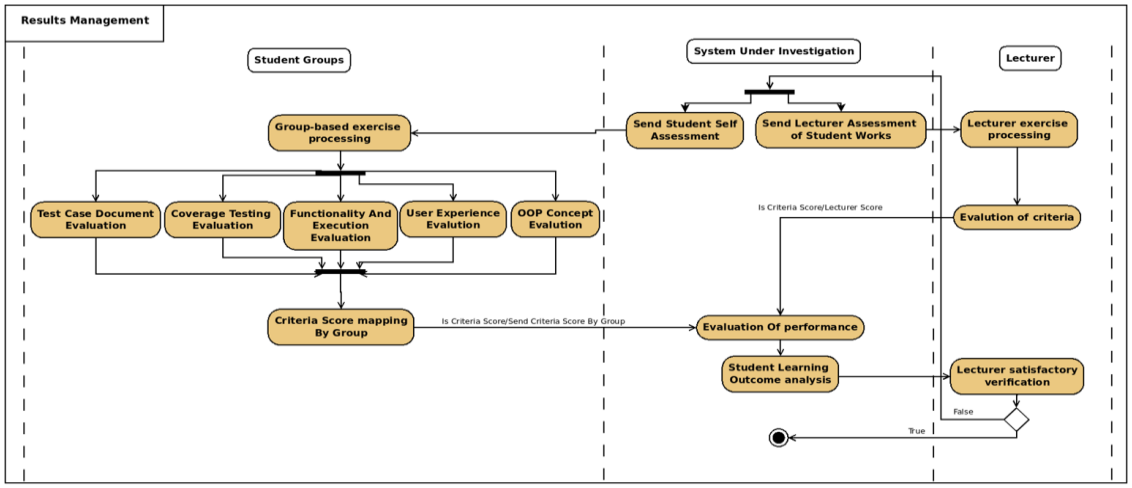


Figure-1: Activity Diagram for Students' Growth Performance Assessment

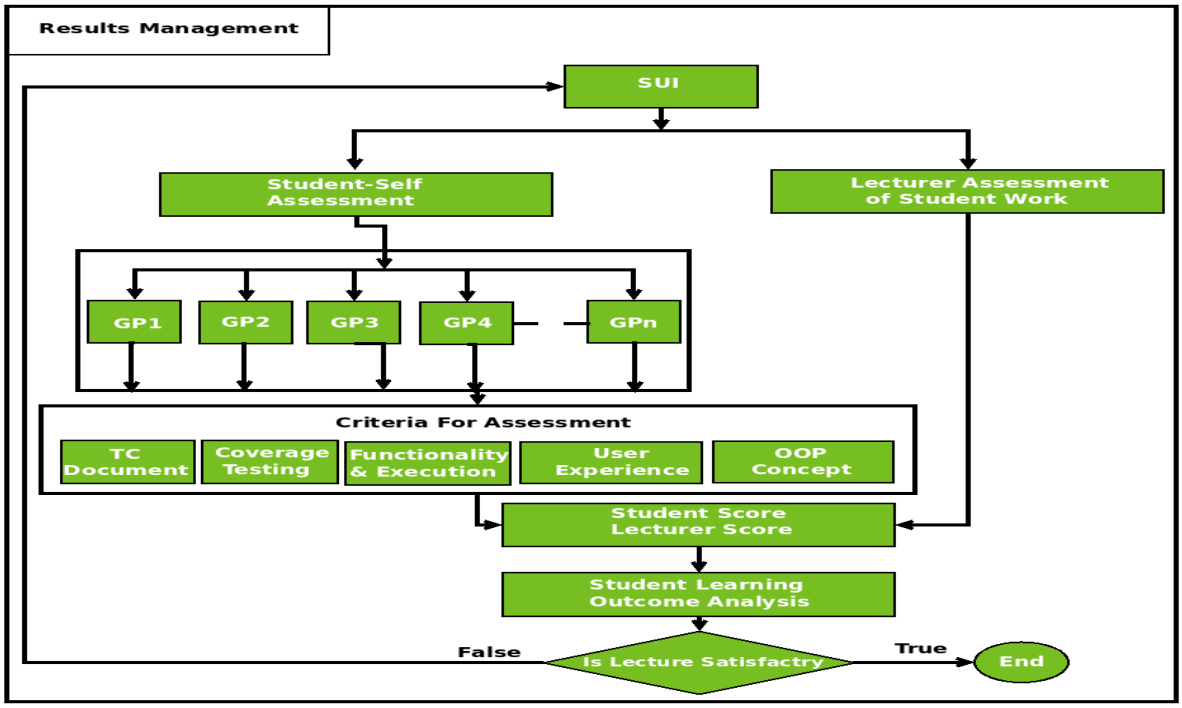


Figure-2: Students' Growth Performance Assessment

All the three groups were asked to produce individuals test case documents which comprised of test case description, preconditions, Expected results (including valid and valid input data) that depicts that functionalities of the the two practical assignments with Unit Testing in ViTest inclusive.

It is worthy of note that, coverage testing (statement coverage, branch coverage and function coverage) was considered pivotal in each test case document .

3.3 Performance Assessment Growth Algorithm

The algorithm starts by initializing the SUI (System Under Investigation), then, it iterates through the different groups of students (GP1 to GPn) and performs the students' and lecturer evaluations. The student and lecturer scores are calculated. Finally, the analysis of the students' learning outcomes is carried out, and a final decision is made based on the satisfaction of the lecturer.

Table 3.1 represents the proposed assessment algorithm, which comprises of the following elements:

SUI (System Under Investigation): This is the system used for the study.

Student-Self Assessment: Assessment carried out by the students themselves.

Lecturer Assessment of Student Work: Assessment of student work by the lecturers.

GP1 to GPn: Groups of students (G1, G2, G3, G4, ..., Gn).

Criteria For Assessment: Criteria used for evaluation, such as test coverage, functionality and execution, user experience, and object-oriented programming (OOP) concept.

Student Score, Lecturer Score: Scores obtained by the students and the score given by lecturers.

Student Learning Outcome Analysis: Analysis of the students' learning outcomes.

Is Lecture Satisfactory?: Final decision on the satisfaction of the lecture.

Table-1: Performance Assessment Algorithm

Inputs Parameters : sc (statement coverage), bc (branch coverage), fc (function coverage), N=number of modules, linguistic variables (lv)
Output Parameter : P(i) = Performance Assesstment

1 Begin

2 Initialize the SUI system

3 For each group of students GP_1 to GP_n:
4 Perform student evaluation (Student-Self Assessment) :
5 Perform lecturer evaluation (Lecturer Assessment of Student Work)
Compute Student Score and Lecturer Score :
6 $P(i) = (\sum^{n}_{i=1}(\text{Testing Criteria}_i)/N)$

7 End for
8 Analyze the students' learning outcomes (Student Learning Outcome Analysis)
9 If the learning outcome analysis is satisfactory:
10 Return "True" (End of the lecture)
11 Else:
12 Return "False" (Lecture is not satisfactory)
13 End if
14 End

3.4 Ranking of Students' Growth Performance Assesstment Criteria

In software testing and programming in general, a lot of imprecise and uncertainty exist when assessing students' skills (performance growth). In view of the above, Domain Experts were consulted in assessing students' performance based on the System Under Investigation (SUI). On the basis of Domain Experts' opinion, students' growth performance assessment criteria selected for this research are : Test Case Document, Coverage Testing, Functionality and Execution, User Experience, Objected-oriented Programming Concepts.

(Excellent = 1, Average = 0.5, Insufficient = 0) as shown in Table-2. The overall performance evaluation analysis takes values in the interval [0,4] based on Domain Experts opinion as shown Table -3 respectively.

Table-2: Students' Growth Performance Assessment Linguistics Variables

Students' Growth Performance Assessment Linguistics Variables			
Testing Criteria	**EXCELLENT = 1**	**AVERAGE = 0.5**	**INSUFFICIENT =0**
Test Case (TC) Document	Produce a test case docuement that conforms to the desire functionality of the SUI.	Test case document needs some modifcations	Test case document cannot implement the desire functionalities of the SUI
Coverage Testing	Able to fully perform software verifcation and validation of statement coverage, branch coverage and fucntional coverage to an apppreciative degree of the SUI	Moderate verifcation and validation of statement coverage, branch coverage and fucntional coverage of the SUI.	Incomplete verifcation and validation of statement coverage, branch coverage and fucntional coverage of the SUI .
Functionality and Execution	SUI fully functional with the ability expand further	The function is normal with some few unexpected results	The function can't be excuted normally
User Experience	Easy to interract with the system	Needs others to explain the interraction of the system	Difficult to interract with the system
Object-oreiented Programming Concepts	Abstraction encapsulation inheitance polymorphism	Abstraction polymorphism	Polymorphism only

Table-3: Overall Performance Evaluation Analysis

Evaluation Metrics	**Overall Performance Ranking**
At-risk Students	$0 \leq x < 0.5$
Low-grade Performance	$0.5 \leq x < 1.0$
Moderate Performance	$1.0 \leq x < 1.5$
Good Performance	$1.5 \leq x < 2.5$
High-grade Performance	$2.5 \leq x \leq 4.0$

4. RESULTS AND DISCUSIONS

We have applied the performance assessment design approach presented in the previous section in two software testing and programming language courses for two consecutive semesters in 2023-2024 academic year in University of Yaounde 1, Yaounde, Cameroon.

4.1 Subject Programs

Two sets of practical exercises were given to three groups of students, each group was made up of 25 students each and they were expected to perform software testing on their practical projects written in JavaScript using ViTest as a Unit Testing Tool.

First Practical Exercise : Student Results Management System

Second Practical Exercise : Unit Testing and Mocking Practicals in ViTest

4.1.1 First Practical Exercise: Student Results Management System

As part of this research, the various groups (GP) of students performed coverage testing (statement coverage, branch coverage and functional coverage) to test different parts of SUI (System Under Investigation) for Student Results Management Systems in JavaScript using ViTest as Unit Testing Tool. Each group was tasked with evaluating specific features, and the test results are presented below.

Groups and Their pratical Contributions

Group 1: Testing Routes and app.js

- **Objective:** Test the routes defined in app.js to ensure the proper functioning of the APIs (see Figure 4.1 and Figure 4.2).
- **Tests Performed:**
  - User authentication.
  - Creation and retrieval of study classes.
  - Management of students, teachers, and subjects.
- **Results obtained for Code Coverage Testing:**
  - **app.js (see Figure 4.1):**
    - Statements: 96.96% (32/33)
    - Branches: 50% (2/4)
    - Functions: 80% (4/5)
    - Lines: 96.96% (32/33)
  - **db.js (see Figure 4.1):**
    - Statements: 0% (0/11)
    - Branches: 100% (0/0)
    - Functions: 0% (0/1)

This gave us the following coverage report for the two server modules:

- Statements: 72.72% (32/44)
- Branches: 50% (2/4)
- Functions: 66.66% (4/6)

For an overall coverage of 63.13% for the server module.

Observations: In this article, Group 1 focused on testing the app.js file, which contains the application's routes. They achieved good code coverage for this element (96.96% of the lines). However, the db.js module, which manages the database connection, (see Fugure-3) was not tested at all by this group. The observations indicated that data validation issues were identified in some routes, resulting in 400 software errors and the students were reluctant to test further. The coverage report indicates that the server module was partially tested, with relatively high coverage for lines (72.72%) and functions (66.66%) (see Figure-4), but lower coverage for branches (50%). This suggested that the tests performed on this module covered the majority of the code, but there were still some test cases to be added, particularly to improve the coverage of different execution paths (branch coverage).

This is the github link to access for overall coverage document for Figure-3 and Figure-4 :https://github.com/SoftWare-Testing-Evaluation/GestionDeNotes1/blob/main/coverage/index.html

Figure-3: App Coverage Document

File ▲		Statements		Branches		Functions		Lines	
src		0%	0/18	0%	0/6	0%	0/4	0%	0/18
src/server		72.72%	32/44	50%	2/4	66.66%	4/6	72.72%	32/44
src/server/config		0%	0/4	100%	0/0	100%	0/0	0%	0/4
src/server/controllers		0%	0/444	0%	0/118	0%	0/50	0%	0/442
src/server/middleware		0%	0/20	0%	0/4	0%	0/4	0%	0/20
src/server/models		0%	0/62	0%	0/2	0%	0/2	0%	0/62
src/server/routes		0%	0/96	0%	0/4	0%	0/1	0%	0/96
src/services		0%	0/29	0%	0/8	0%	0/5	0%	0/29
src/slices		0%	0/330	0%	0/20	0%	0/120	0%	0/316
src/store		0%	0/1	100%	0/0	100%	0/0	0%	0/1
src/utils		0%	0/68	0%	0/48	0%	0/14	0%	0/60

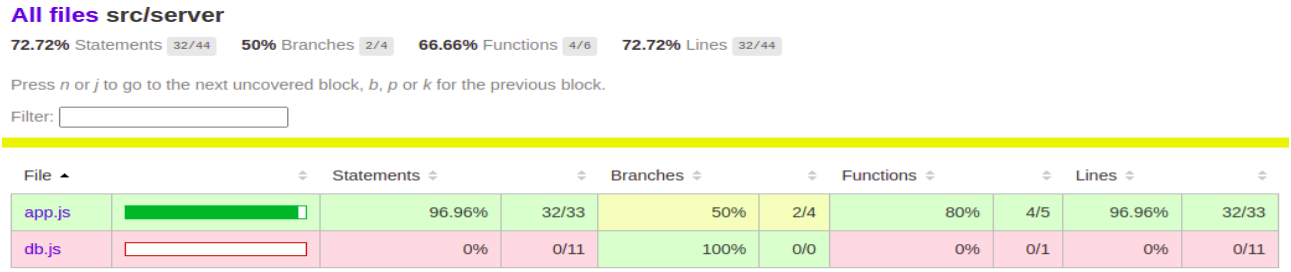


Figure-4: Server Coverage Document in Line 2

Overall performance assessment P(i) calculation for group 1:

By applying the formula described in Table-1 for three modules (N=3) and three parameters for unit testing, we have:

$P(i) = (\sum^{n}_{i=1}(\text{Testing Criteria}_i)/N)$

$P(i) = (\sum^{3}_{i=1}(SCi+BCi+FCi)/3)$

$P(i) = [(0.96+0.5+0.8)+(0+1+0)] /3$

P(i) **= 1.08,** which indicates that (group 1) has a Moderate Performance according to Table-3 for both Students'-self Evaluation and Lecturer Evaluation.

Group 2: Testing app.js and db.js Functionalities

- **Objective:** Verify the application's functionalities and the connection to the database.
- Tests Performed:
  - Creation of new records (students, teachers, subjects).
  - Retrieval of information from the database.
  - Synchronization of models with the database.

Results obtained for Code Coverage Testing for three modules:

- **For the app.js file:** The overall code coverage testing is as follows (see Figure-5):
  - 96.96% coverage for Statements
  - 50% coverage for Branches
  - 80% coverage for Functions

- **For the db.js file:** The overall code coverage testing is as follows (see Figure-5):
  - 90.9% coverage for Statements
  - 100% coverage for Branches
  - 100% coverage for Functions
- **src/server files :**The overall code coverage for all src/server files is (see Figure-6):
  - 95.45% for Statements
  - 50% for Branches
  - 83.33% for Functions For an overall coverage of 85.50%
- **Observations**: The code coverage is generally good, with coverage exceeding above 80% for most metrics (see Figure-5). However, the low score of 50% for branches in app.js indicates that additional test cases need to be added by this group. The tests validated the main functionalities related to data management (creation, retrieval, synchronization) and the connection to the database(see Figure-6).

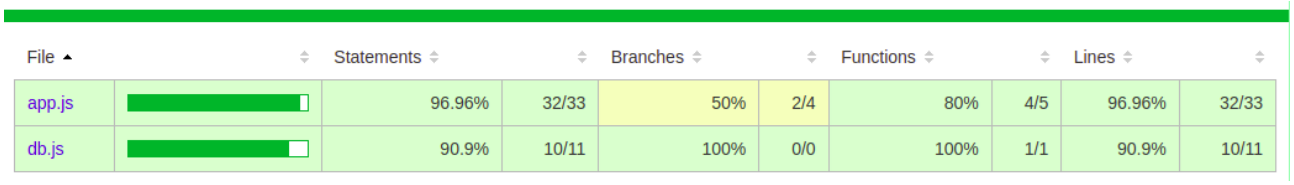


Figure-5: App and Db Coverage

File		Statements		Branches		Functions		Lines	
src		0%	0/18	0%	0/6	0%	0/4	0%	0/18
src/server		95.45%	42/44	50%	2/4	83.33%	5/6	95.45%	42/44
src/server/config		0%	0/4	100%	0/0	100%	0/0	0%	0/4
src/server/controllers		0%	0/444	0%	0/118	0%	0/50	0%	0/442
src/server/middleware		0%	0/20	0%	0/4	0%	0/4	0%	0/20
src/server/models		0%	0/62	0%	0/2	0%	0/2	0%	0/62
src/server/routes		0%	0/96	0%	0/4	0%	0/1	0%	0/96
src/services		0%	0/29	0%	0/8	0%	0/5	0%	0/29
src/slices		0%	0/330	0%	0/20	0%	0/120	0%	0/316
src/store		0%	0/1	100%	0/0	100%	0/0	0%	0/1
src/utils		0%	0/68	0%	0/48	0%	0/14	0%	0/60

Figure-6: Server Coverage Document in Row 2

Overall performance assessment P(i) calculation for group 2:

By applying the formula described in Table 3.1 for three modules (N=3) and three parameters for unit testing, we have:

$P(i)= (\sum^{n}_{i=1}(\text{Testing Criteria}_i)/N)$

$P(i)= (\sum^{3}_{i=1}(SCi+BCi+FCi)/3)$
$P(3) = ((0.96+0.5+0.8)+(0.9+1+1))/3$

P(3) =1,72 which indicates that (group 2) has High-grade performance according to Table-4, for both Students'-self Evaluation and Lecturer Evaluation.

Group 3: Testing app.js, db.js, and Student Average Calculation

- **Objective:** Evaluate the complete integration of the application, including the calculation of student averages.
- **Tests Performed:**
  - Verification of average calculations for different sequences.
  - Tests of the grade management logic.
  - Evaluation of the synchronization between the application and the database.
- Results: The tests performed by Group 3 gave us the following results (see Figure-7 and Figure-8):

Results obtained for Code Coverage Testing (see Figure-7 and Figure-8)**:**

- **For the app.js file:** The overall code coverage testing is as follows (see Figure-7):
  - Statements: 96.96%
  - Branches: 50%
  - Functions: 80%

Coverage of db.js:

- Statements: 90.9%
- Branches: 100%
- Functions: 100%

This gives us a coverage of 85.5% for the server module.
Coverage of calculMoyenneClasse.js:

- Statements: 73.07%
- Branches: 29.16%
- Functions: 100% File calculerStatistiquesClasses.js: not tested

This gives us a coverage of 53.54% for the utils module.

Observations: The average calculation tests showed consistent results, but some edge cases were not taken into account. The results for app.js and db.js are similar to those of Group 2, with good overall coverage. However, the calculMoyenneClasse.js file shows gaps, particularly at the branch level (29.16%) and line level (69.56%). The calculerStatistiquesClasses.js file was not tested.

This is the github link to access for overall coverage document for Figure-7 and Figure-8: https://github.com/SoftWare-Testing-Evaluation/GestionDeNotes3/blob/main/coverage/index.html

File		Statements		Branches		Functions		Lines	
src		0%	0/18	0%	0/6	0%	0/4	0%	0/18
src/server		95.45%	42/44	50%	2/4	83.33%	5/6	95.45%	42/44
src/server/config		0%	0/4	100%	0/0	100%	0/0	0%	0/4
src/server/controllers		0%	0/444	0%	0/118	0%	0/50	0%	0/442
src/server/middleware		0%	0/20	0%	0/4	0%	0/4	0%	0/20
src/server/models		0%	0/62	0%	0/2	0%	0/2	0%	0/62
src/server/routes		0%	0/96	0%	0/4	0%	0/1	0%	0/96
src/services		0%	0/29	0%	0/8	0%	0/5	0%	0/29
src/slices		0%	0/330	0%	0/20	0%	0/120	0%	0/316
src/store		0%	0/1	100%	0/0	100%	0/0	0%	0/1
src/utils		55.88%	38/68	29.16%	14/48	78.57%	11/14	53.33%	32/60

Figure-7: Server and Utils Coverage Document

All files src/utils

55.88% Statements 38/68 **29.16%** Branches 14/48 **78.57%** Functions 11/14 **53.33%** Lines 32/60

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

Filter:

File		Statements		Branches		Functions		Lines	
calculMoyenneClasse.js		73.07%	38/52	29.16%	14/48	100%	11/11	69.56%	32/46
calculerStatistiquesClasses.js		0%	0/16	100%	0/0	0%	0/3	0%	0/14

Figure-8: Utils Coverage Document

Overall performance assessment P(i) calculation for group 3:

By applying the formula described in Table-1 for three modules (N=3) and three parameters for unit testing, we have:

$P(i) = (\sum^{n}_{i=1}(\text{Testing Criteria}_i)/N)$

$P(3)= (\sum^{3}_{i=1}(SCi+BCi+FCi)/3)$

$P(3)= ((0.96+0.5+0.8)+(0.9+1+1)+(0,73+0,29+1))/3$

P(3) =2,393333333 which indicates that (group 3) has High-grade Performance according to Table-3, for both Students'-self Evaluation and Lecturer Evaluation.

4.1.2 Second Practical Exercise: Unit Testing and Mocking Practicals in ViTest

Secenario Description: In this practical assignment, it was requested that, the three groups of students conduct tests on a new mobile application for a local restaurant, allowing users to create a basket of products and access promotions, while ensuring the quality and reliability of the application before its public release. Each group presented their results, highlighting their findings, challenges encountered, and the overall effectiveness of the tests performed.

The following students' growth performance assessment criteria with their corresponding Linguistics Variables (see Table-2) were used for this case study:

(1) Test Case document: Excellent = 1, Average = 0.5, Insufficient = 0
(2) Coverage Testing: Excellent = 1, Average = 0.5, Insufficient = 0
(3) Functionality and Execution: Excellent = 1, Average = 0.5, Insufficient = 0
(4) User Experience: Excellent = 1, Average = 0.5, Insufficient = 0
(5) Object-oriented Programming (OOP) Concepts: Excellent = 1, Average = 0.5, Insufficient = 0

Group 1: Unit Testing And Mocking Function

The group 1 results showed that 3 out of 14 test files failed, with 24 failed tests out of a total of 129 tests. Additionally, 2 errors were detected during the testing process. These results indicate that group 1 encountered issues in the implementation of certain user account-related functionalities (see Figure-9).

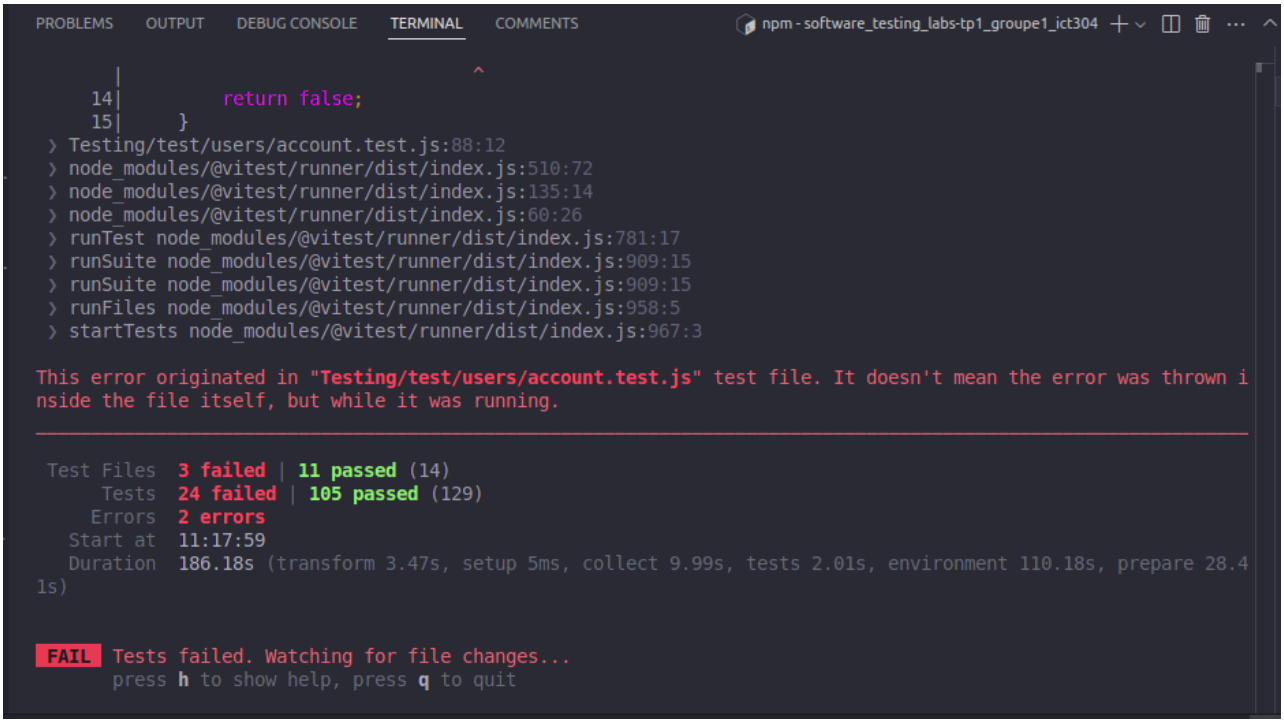


Figure-9 : Unit Testing of Group 1

The provided code utilizes various mocking techniques to test the different functionalities of the system in isolation. It mocks the access to external modules such as analytics, currency management, email sending, payments, and security. It also simulates interactions with web pages, signup and login processes, as well as

date management and discounts. These mocking techniques allow for the independent testing of functions, facilitating unit testing and the integration of the system's various components (see Figure-10).

This is the Github link to acces for mocking function :https://github.com/SoftWare-Testing-Evaluation/UnitTest2/blob/main/mocking.js

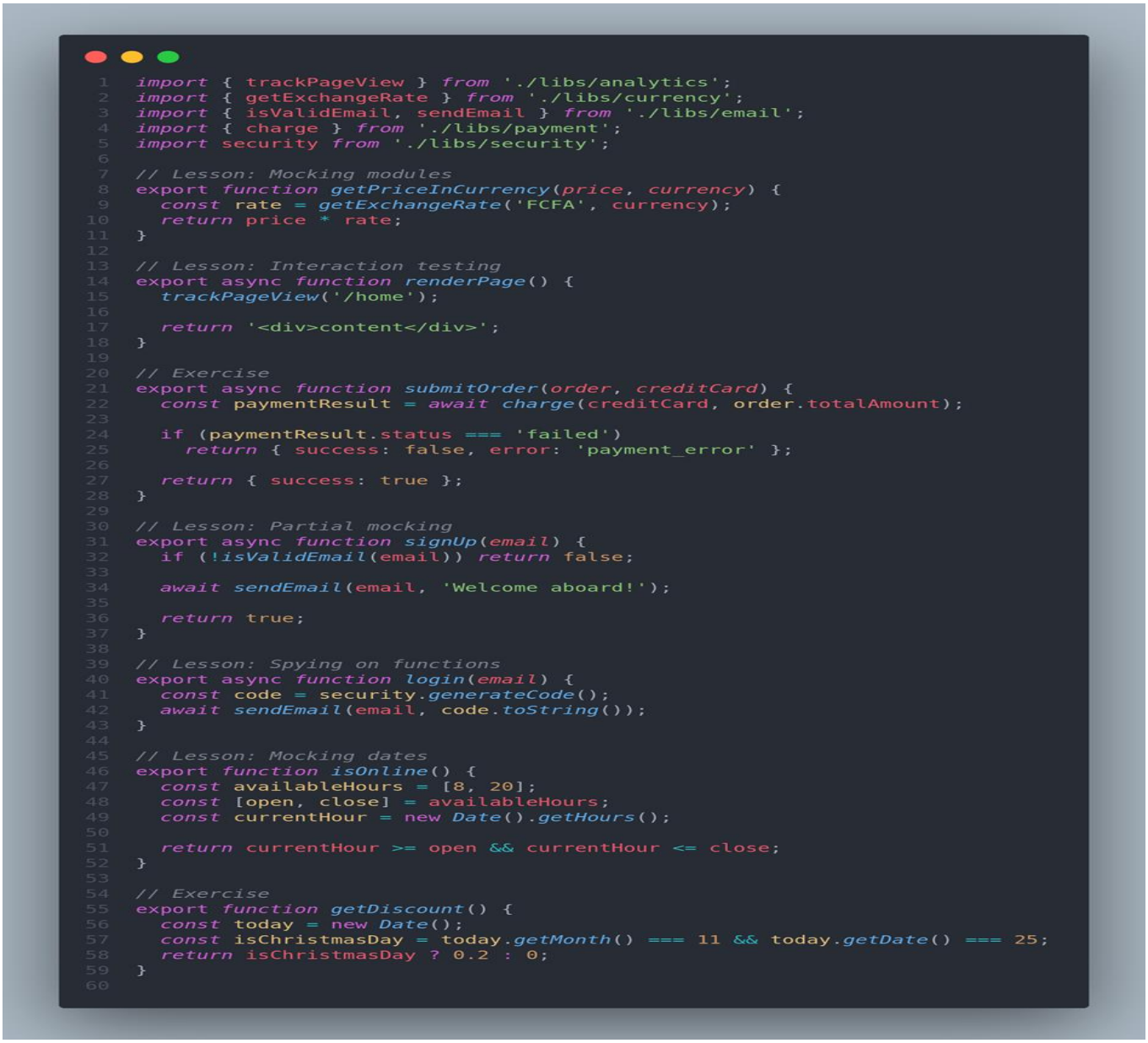
Based on students'-self assessment and lecturer assessment, the overall students' performance growth assessment is depicted in Table-4 for group 1:

Table-4 : Computation for Overall Students' Performance Growth Assessment (Unit Testing and Mocking): Group 1

Testing Criteria (Group 1)	**Linguistics Variables**	
	Students-self Assesstment (Students' Score)	**Lecturer's Assestment (Lecturer's Score)**
Test Case document	Excellent = 1	Excellent = 1
Coverage Testing	Average = 0.5	Insufficient = 0
Functionality and Execution:	Average = 0.5	Average = 0.5
User Experience	Average = 0.5	Average = 0.5
OOP Concepts	Excellent = 1	Excellent = 1
Total	**3.5/2=1.85 (Good Performance)**	**2/2 = 1 (Moderate Performance)**

By applying the formula described in Table-1 for two modules (N=2) and five parameters for unit testing, we have the following Overall Students' Performance Growth Assessment for group 1 (see Table-4):

$P(i) = (\sum^{n}_{i=1}(\text{Testing Criteria}_i)/N)$
P(i) = 3.5/2=1.85 for Students-self Assesstment (Students' Score Assessment) indicating moderate performance
P(i) = 2/2 = 1 for Lecturer's Assestment (Lecturer's Score Assessment)
Group 1 had 1.85 (Good Performance) for Students' Score Assessment and 1.0 (moderate Performance) for Lecturer's Score Assessment, implying more practicals skills are needed for group 1 students to excel in Software Testing and programming courses with more monitoring and follow-up (Table-4).



```
import { trackPageView } from './libs/analytics';
import { getExchangeRate } from './libs/currency';
import { isValidEmail, sendEmail } from './libs/email';
import { charge } from './libs/payment';
import security from './libs/security';

// Lesson: Mocking modules
export function getPriceInCurrency(price, currency) {
  const rate = getExchangeRate('FCFA', currency);
  return price * rate;
}

// Lesson: Interaction testing
export async function renderPage() {
  trackPageView('/home');

  return '<div>content</div>';
}

// Exercise
export async function submitOrder(order, creditCard) {
  const paymentResult = await charge(creditCard, order.totalAmount);

  if (paymentResult.status === 'failed')
    return { success: false, error: 'payment_error' };

  return { success: true };
}

// Lesson: Partial mocking
export async function signUp(email) {
  if (!isValidEmail(email)) return false;

  await sendEmail(email, 'Welcome aboard!');

  return true;
}

// Lesson: Spying on functions
export async function login(email) {
  const code = security.generateCode();
  await sendEmail(email, code.toString());
}

// Lesson: Mocking dates
export function isOnline() {
  const availableHours = [8, 20];
  const [open, close] = availableHours;
  const currentHour = new Date().getHours();

  return currentHour >= open && currentHour <= close;
}

// Exercise
export function getDiscount() {
  const today = new Date();
  const isChristmasDay = today.getMonth() === 11 && today.getDate() === 25;
  return isChristmasDay ? 0.2 : 0;
}
```

Figure-10: Mocking Function of Group 1

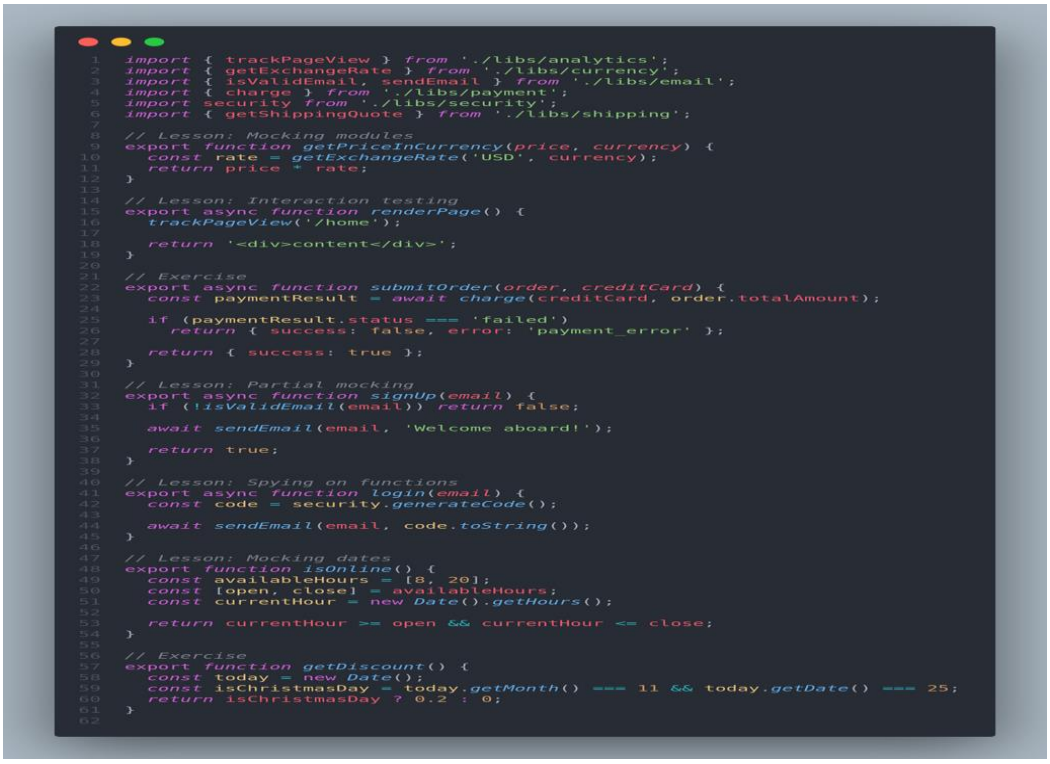
Group 2 : Unit Testing And Mocking Function

The group 2 results showed that 6 test files were executed successfully, with a total of 49 tests passed. The total duration of the tests was 25.20 seconds. The test files cover different modules of the application, such as promotions, basket, events, and filters. (see Figure-11)

Figure-11: Unit Testing of Group 2

The code from group 2 was described as implementing various mocking techniques to test the different functionalities of the system in isolation. It was reported that the code simulated access to external modules such as analytics, currency management, email sending, payments, and security, as well as interactions with web pages, signup and login processes, date management, and discounts(see Figure-12). This would allow for the independent testing of the different functions, facilitating unit testing and integration of the system's components.

This is the Github link to acces for mocking function:https://github.com/SoftWare-Testing-Evaluation/UnitTest3/blob/main/mocking.js



```
import { trackPageView } from './libs/analytics';
import { getExchangeRate } from './libs/currency';
import { isValidEmail, sendEmail } from './libs/email';
import { charge } from './libs/payment';
import security from './libs/security';
import { getShippingQuote } from './libs/shipping';

// Lesson: Mocking modules
export function getPriceInCurrency(price, currency) {
  const rate = getExchangeRate('USD', currency);
  return price * rate;
}

// Lesson: Interaction testing
export async function renderPage() {
  trackPageView('/home');

  return '<div>content</div>';
}

// Exercise
export async function submitOrder(order, creditCard) {
  const paymentResult = await charge(creditCard, order.totalAmount);

  if (paymentResult.status === 'failed')
    return { success: false, error: 'payment_error' };

  return { success: true };
}

// Lesson: Partial mocking
export async function signUp(email) {
  if (!isValidEmail(email)) return false;

  await sendEmail(email, 'Welcome aboard!');

  return true;
}

// Lesson: Spying on functions
export async function login(email) {
  const code = security.generateCode();

  await sendEmail(email, code.toString());
}

// Lesson: Mocking dates
export function isOnline() {
  const availableHours = [8, 20];
  const [open, close] = availableHours;
  const currentHour = new Date().getHours();

  return currentHour >= open && currentHour <= close;
}

// Exercise
export function getDiscount() {
  const today = new Date();
  const isChristmasDay = today.getMonth() === 11 && today.getDate() === 25;
  return isChristmasDay ? 0.2 : 0;
}
```

Figure-11: Unit Testing of Group 2

Based on students'-self assessment and lecturer assessment, the overall students' performance growth assessment is depicted in Table-4 for group 1:

Table-5: Computation for Overall Students' Performance Growth Assessment (Unit Testing and Mocking): Group 2

Testing Criteria (Group 2)	**Linguistics Variables**	
	Students-self Assesstment (Students' Score)	**Lecturer's Assestment (Lecturer's Score)**
Test Case document	Excellent = 1	Excellent = 1
Coverage Testing	Average = 0.5	Average = 0.5
Functionality and Execution:	Excellent = 1	Average = 0.5
User Experience	Average = 0.5	Average = 0.5
OOP Concepts	Excellent = 1	Excellent = 1
Total	**4/2=2.0 (Good Performance)**	**3.5/2 = 1.75 (Good Performance)**

By applying the formula described in Table-1 for two modules (N=2) and five parameters for unit testing, we have the following Overall Students' Performance Growth Assessment for group 2 (see Table-5):

$P(i) = (\sum^{n}_{i=1}(\text{Testing Criteria}_i)/N)$
P(i) = 4/2=2.0 for Students-self Assesstment (Students' Score Assessment) indicating good performance
P(i) = 3.5/2 = 1.75 for Lecturer's Assestment (Lecturer's Score Assessment)
Group 2 had 2.0 (Good Performance) for Students' Score Assessment and 1.75 (Good Performance) for Lecturer's Score Assessment, implying more praticals skills are needed for group 1 students to excel in Software Testing and programming courses (Table-4) with moderate follow-up.

Group 3: Unit Testing And Mocking Function

The test results for group 3 are very positive, with 72 tests passed out of 72 total and a total execution time of 138.32 seconds (see Figure-12). In contrast to group 1, which encountered issues with certain user account-related functionalities, group 3 successfully tested the various application features in-depth, including the shopping cart, promotions, events, and event search. These results indicated that group 3 has done a quality job, and their implementation is robust and reliable.

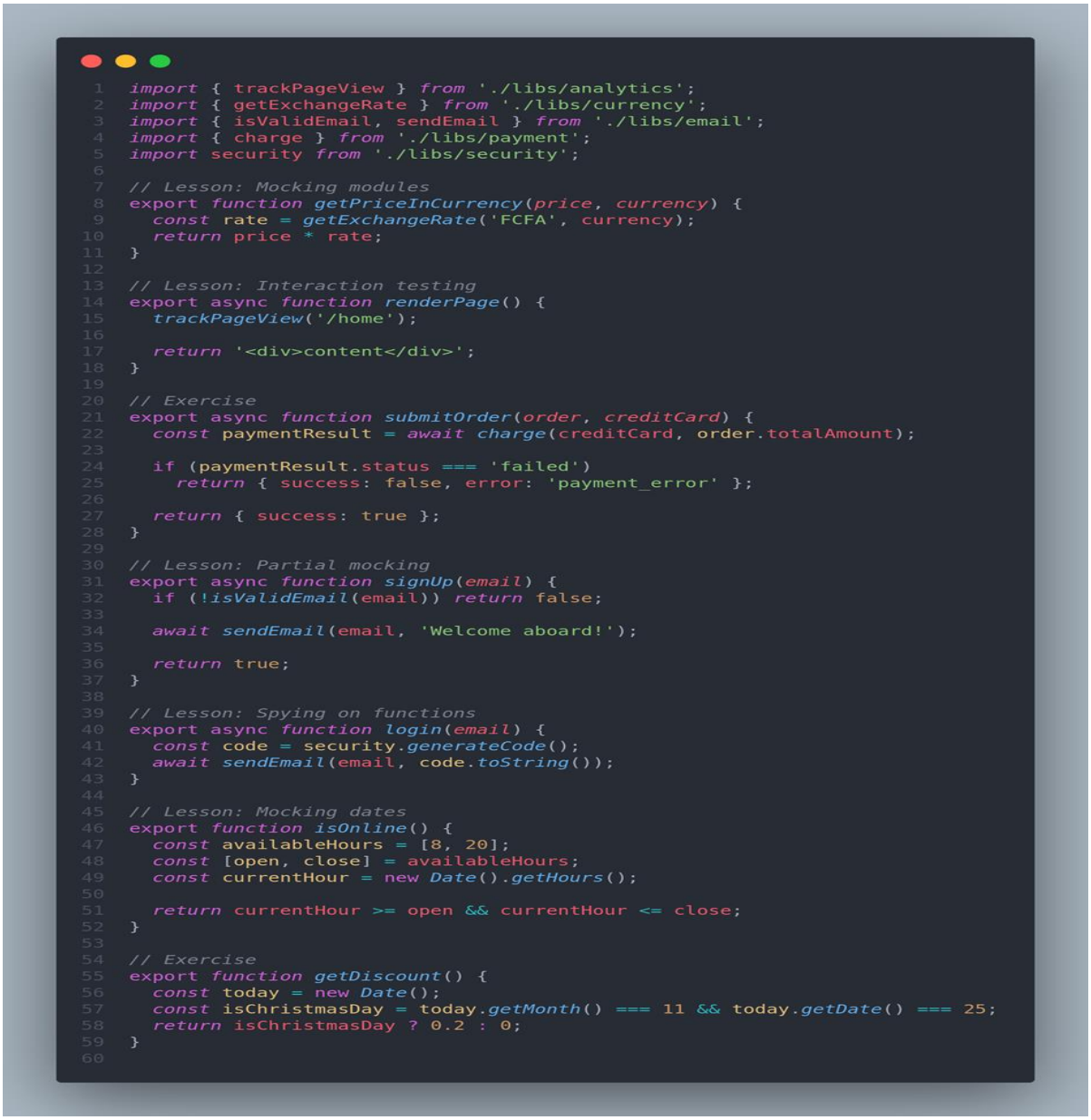
Figure-12: Unit Testing of Group 3

Figure-13 demonstrate the use of various mocking techniques to test the different functionalities of a system in isolation. It simulates access to external modules such as analytics, currency management, email sending, payments, and security, as well as interactions with web pages, registration and login processes, date management, and discounts. This approach allows for independent testing of the different functions, facilitating unit testing and component integration. The code also showcases the use of partial mocking, function mocking, and date mocking to simulate specific behaviors and test the functionalities in isolation.

This is the Github link to access for mocking function: https://github.com/SoftWare-Testing-Evaluation/UnitTesting1/blob/main/mocking.js

Figure-13: Mocking Function of Group 3

Based on students'-self assessment and lecturer assessment, the overall students' performance growth assessment is depicted in Table-6 for group 3:



```
import { trackPageView } from './libs/analytics';
import { getExchangeRate } from './libs/currency';
import { isValidEmail, sendEmail } from './libs/email';
import { charge } from './libs/payment';
import security from './libs/security';

// Lesson: Mocking modules
export function getPriceInCurrency(price, currency) {
  const rate = getExchangeRate('FCFA', currency);
  return price * rate;
}

// Lesson: Interaction testing
export async function renderPage() {
  trackPageView('/home');

  return '<div>content</div>';
}

// Exercise
export async function submitOrder(order, creditCard) {
  const paymentResult = await charge(creditCard, order.totalAmount);

  if (paymentResult.status === 'failed')
    return { success: false, error: 'payment_error' };

  return { success: true };
}

// Lesson: Partial mocking
export async function signUp(email) {
  if (!isValidEmail(email)) return false;

  await sendEmail(email, 'Welcome aboard!');

  return true;
}

// Lesson: Spying on functions
export async function login(email) {
  const code = security.generateCode();
  await sendEmail(email, code.toString());
}

// Lesson: Mocking dates
export function isOnline() {
  const availableHours = [8, 20];
  const [open, close] = availableHours;
  const currentHour = new Date().getHours();

  return currentHour >= open && currentHour <= close;
}

// Exercise
export function getDiscount() {
  const today = new Date();
  const isChristmasDay = today.getMonth() === 11 && today.getDate() === 25;
  return isChristmasDay ? 0.2 : 0;
}
```

Table-6: Computation for Overall Students' Performance Growth Assessment (Unit Testing and Mocking): Group 3

Testing Criteria (Group 3)	**Linguistics Variables**	
	Students-self Assesstment (Students' Score)	**Lecturer's Assestment (Lecturer's Score)**
Test Case document	Excellent = 1	Excellent = 1
Coverage Testing	Excellent = 1	Excellent = 1
Functionality and Execution:	Average = 0.5	Average = 0.5
User Experience	Excellent = 1	Excellent = 1
OOP Concepts	Excellent = 1	Excellent = 1
Total	**4.5/2=2.25 (High-grade Performance)**	**4.5/2 = 2.25 (High-grade Performance)**

By applying the formula described in Table-1 for two modules (N=2) and five parameters for unit testing, we

have the following Overall Students' Performance Growth Assessment for group 1 (see Table-4):

$P(i) = (\sum^{n}_{i=1}(\text{Testing Criteria}_i)/N)$
P(i) = 4.5/2=2.25 for Students-self Assesstment (Students' Score Assessment) indicating High-grade performance
P(i) = 4.5/2 = 2.25 for Lecturer's Assestment (Lecturer's Score Assessment)
Group 3 had 2.25 (High-grade Performance) for both Students' Score Assessment and Lecturer's Score Assessment, implying less praticals skills are needed for group 3 students to excel in Software Testing and programming courses (see Table-6) with less follow-up.

4.2 Overall Performance Evaluation Analysis

To ensure continuous follow-up of students' programming skills in Software Testing and programming courses in general, we used our novel performance assesstment approach and obtained the in section 4.1.
we observed significant improvements of student learning outcome in various aspects. In this article, we analyze the overall performances of the three groups based on the performance evaluation metric described in (Table-1) in section

4.21 Impacts on Students Performances

(1.) Promoting Peer Group Learning Among Students : The assessment-based strategy proposed in this research is an open process that gives students a lot of flexibility in completing their assignments. It was observed that most students were willing to openly discuss programming problems, especially in the review phase. The collaborative nature of the assessment process made students feel connected to their classmates: students were encouraged to carefully write their own programs and to equally review programs written by other students because it is collaborative and mutually beneficial process.

(2) Compliance with Object-oriented Coding Standards. A good numbers of students were able to comply with well-accepted object-oriented coding standards. Students learned new coding styles coding styles from review comments and were able to find similar mistakes when they played the reviewer role. At the end of the course, students were found to improve a lot in complying with coding standards and the average number of violations of coding standards in a program was reduced from more than 5 at the beginning to almost zero at the end of the course.

(3) Student Programming Skills. Compared with traditional assessments that concentrate on theoretical study, our approach puts a high priority on practical programming competence. After using this approach in two courses in two consecutive semesters in University of Yaounde 1, Yaounde, Cameroon, we observed significant improvements of student learning in various aspects. In the coding stage of an assignment, students were required to complete many handson practicals including software verification and validation of all statements, branches and functions inclusive. This assessment approach demonstrated the need for more practicals in assessing students' learning outcomes in Software Testing courses and Programming Languages in general.

(4) Degree of Student Satisfaction : The students were generally satisfied with the new assessment approach as depicted in Figure-14. Compared with traditional assessment approaches without adequate pratical sessions, about 80% students ('learning objective met') considered that the assessment approach is superior in several aspects, about 28 % students ('progressive learning outcome') considered the learning outcome in a progressive incremental manner and about 4 % students (students not interested) were not interested at all in the new approach (see Figure-14).

Through interview fact finding review process, we gathered and understood the following students' opinions on the new assessment approach:

- This novel and interesting learning approach can enhance and stimulate their interests in Programming Language skills and enhance their awareness in the process of active learning.

• Compared with traditional assessment processes, this approach does not only add more fairness to students' continuous assessment but also helps to achieve learning objectives more efficiently in software testing and programming courses.

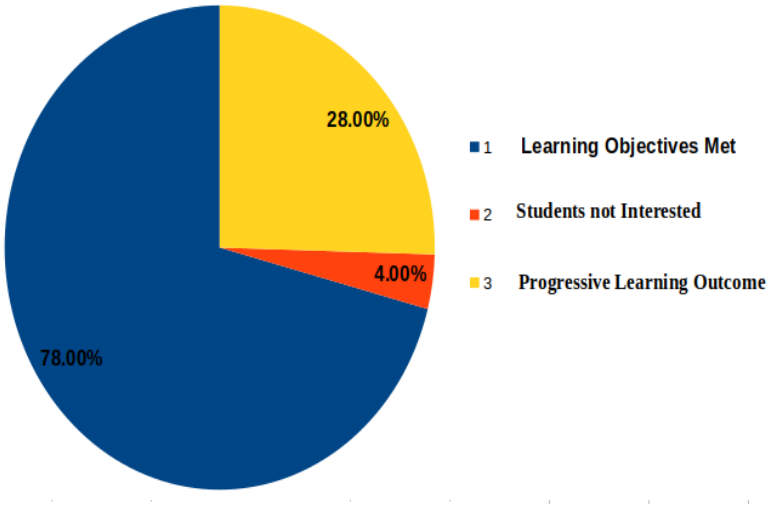


Figure-14: Impacts on Students' Performances

5 CONCLUSION AND PERSPECTIVES

This paper presents a novel performance assessment design approach with performance criteria that were used in two Software testing and programming language courses for two consecutive semesters in 2023-2024 academic year in University of Yaounde 1, Yaounde, Cameroon. The assessment was done in two levels: Students-self assessment and Lecturer assessment of students' group practicals using five features for assessment (Test Case Document, Coverage Testing, Functionality, User Experience and Object-oriented Programming Concepts).

Overall, the tests performed by the three groups have covered a large part of the code for both applications, with satisfactory results indicating performance growth.
However, some weaknesses were identified and will need to be improved.

Group 1 focused on testing the routes defined in the app.js file, achieving very good coverage of 96.96% for the lines of code. However, the db.js module, which manages the database connection, was not tested at all. Data validation issues were also identified in some modules, resulting in 400 errors. These issues will need to be addressed.

Group 2 conducted more comprehensive tests on the application's functionalities, covering both app.js and db.js. The overall coverage of the server module reached 85.50%, which is very satisfactory. However, the branch coverage in app.js remains low at 50%, indicating that additional test cases will need to be added.

Finally, Group 3 evaluated the complete integration of the application, including the calculation of students' averages. The results for app.js and db.js are similar to those of Group 2, but the calculMoyenneClasse.js file shows gaps, particularly at the branch level (29.16%) and line level (69.56%). The calculerStatistiquesClasses.js file was not tested at all.

In conclusion, the tests performed have validated the majority of the application's functionalities and identified areas for improvement, particularly regarding branch coverage and handling of edge cases. Additional efforts will be necessary for the groups to ensure optimal reliability and robustness of their applications.

Notwithstanding, however, the results revealed and provided some insights for further investigation Due to inherent challenges in assessment of Software Testing and Programming Language courses, there are some interesting research topics to be investigated in future research. We want to focus on

evivdence-based quality assurance in the assessment process for better students' pratical learning outcome. The future research in evivdence-based quality assurance will address following topics:

(1) Inclusive modern anti-plagiarism mechanism: Successful adoption of modern anti-plagiarism techniques in conjunction with Artificial Intelligent techniques can be used in future system to effectively fight plagiarism and improve the assessment quality in Software Testing and Programming courses at large in order to minimize academic dishonesty.

(2) Automatic Assessment Technique : Overloading lecturers is major setback of this new approach assessment process. A possible solution is to use automatic assessment techniques by employing search-based optimization techniques to minimize while optimzing the assessment procecedure (Harman and Jones, [11]).

REFERENCES:

[1] Djam, X.Y and N.V. Blamah (2024). Have we Gotten There ? Mutation Analysis as an Auntomated Program Analysis Technique for Linux Kernel's Regression Test Suites Generation. International Journal of Computer Engineering and Applications, Vol 14 (5), pp 9-23.

[2] Yanqing, W., Hang, L., Feng, Y.O., Ying, L. (2012). Assessment of Programming Language Learning Based on Peer Code Review Model: Implementation and Experience Report. Computers & Education. 59(2):412-422

[3] Abdul-Rahman,S.S. and Boulay, B.D (2014). Learning Programming Via Worked-examples: Relation of learning styles to cognitive load. Computers in Human Behavior, vol. 30, pp. 286-298.

[4] Bejarano, A.M., García, L. E. and Zurek, E. E. (2015). Detection of source code similitude in academic environments," Computer Applications in Engineering Education, vol. 23, no. 1, pp. 13-2.

[5] Alexandron, G., Yoo, L. Y., Ruipérez-Valiente, J. A., Lee, S., & Pritchard, D. E. (2019). Are MOOC Learning Analytics Results Trustworthy? With Fake Learners, They Might Not Be! *International Journal of Artificial Intelligence in Education*, 29(4), 484–506. https://doi.org/10.1007/s40593-019-00183-1

[6] Pregitzer, M. and Clements, S. N. (2013). Bored with the Core: Stimulating Student Interest in Online General Education," Educational Media International, vol. 50, no. 3, pp. 162-176.

[7] Amigud, A., Arnedo-Moreno, J., Daradoumis, T., & Guerrero-Roldan, A. E. (2017). Using Learning Analytics for Preserving Academic Integrity. *The International Review of Research in Open and Distributed Learning*, 18(5), 192–210. https://doi.org/10.19173/irrodl.v18i5.3103

[8] Adzima, K. (2020). Examining Online Cheating in Higher Education Using Traditional Classroom Cheating as a Guide. *Electronic Journal of e-Learning*, 18(6), 476–493. https://doi.org/10.34190/jel.18.6.002

[9] Akbulut, Y., Şendağ, S., Birinci, G., Kılıçer, K., Şahin, M. C., & Odabaşı, H. F. (2008). Exploring the Types and Reasons of Internet-triggered Academic Dishonesty Among Turkish Undergraduate Students: Development of Internet-Triggered Academic Dishonesty Scale (ITADS). *Computers & Education*, 51(1), 463–473. https://doi.org/10.1016/j.compedu.2007.06.003

[10] Rowe, I. M. A. G. (2020). Difficulties in Learning and Teaching Programming — Views of Students and Tutors," Education and Information Technlogies, vol. 7, no. 1, pp. 55-66.

[11] Harman, M. and Jones, B. F. (2001). Search-based software engineering. Inf. Softw. Technol., vol. 43, no. 14, pp. 833–839.

[12] Gehringer, E. F., Chinn, D. D., Pérez-Quiñones, M. A., & Ardis, M. A. (2005). Using Peer Review in Teaching Computing. In SIGCSE '05 Proceedings of the 36th SIGCSE technical symposium on computer science education (pp. 321–322). St. Louis, MO, USA: ACM New York, NY, USA

[13] Sitthiworachart, J., and Joy, M. (2004). Effective peer assessment for learning computer programming. In ITiCSE '04 Proceedings of the 9th annual SIGCSE conference on innovation and technology in computer science education (pp. 122–126). Leeds, United Kngdm: ACM New York, NY, USA.

[14] Trytten, D. A. (2005). A design for team peer code review. In SIGCSE '05 Proceedings of the 36th SIGCSE technical symposium on computer science education (pp. 455–459). St. Louis, Missouri: ACM New York, NY, USA

[15] Turner, S. (2009). Peer review in CS2: the effects on attitudes, engagement, and conceptual learning. Doctoral Dissertation of Virginia Polytechnic Institute and State University.

[16] Turner, S., & Pérez-Quiñones, M. A. (2019). Peer review in CS2: conceptual learning. In SIGCSE '10 Proceedings of the 41st ACM technical symposium on computer science education (pp. 331–335). Milwaukee, WI, USA: ACM New York, NY, USA.

[17] Li, X. (2006). Using peer review to assess coding standards–a case study. In Frontiers in education conference, 36th annual (pp. T2D-9–T2D-14). San Diego, CA, USA.

[18] Li, X. (2007). Incorporating a code review process into the assessment. In 20th annual conference of the National Advisory Committee on Computing Qualifications (NACCQ 2007) (pp. 125–131). Nelson, New Zealand.