Software Verification and Validation Portfolio

Lucy Zhang
Ira A. Fulton Schools of Engineering, Arizona State University
lzhan370@asu.edu

Abstract*—This comprehensive report details the methodologies, processes, and outcomes of three distinct software testing projects, each focusing on different testing strategies and objectives. Project 1 delves into specification-based testing, with an emphasis on equivalence partitioning, boundary value analysis, and cause-effect testing, alongside an advanced exploration into the Design of Experiments (DOE) approach. Project 2 shifts focus to structural-based testing, offering insights into code coverage and data flow analysis within different application scenarios. Project 3 presents a practical examination of Graphical User Interface (GUI) testing involving the development of an application and the subsequent testing using a selected automation tool. Across all projects, the report highlights the theoretical foundations, practical applications, and critical analyses of various testing techniques and tools, underscoring their significance in achieving robust, reliable, and efficient software solutions.

Keywords*—equivalence partitioning, boundary value analysis, cause and effect testing, code coverage, statement coverage, decision coverage, software validation, JPM Pro, unittest, and DOE.

I. Introduction

In the ever-evolving field of software development, rigorous testing methodologies are crucial in ensuring application robustness, functionality, and user satisfaction. This report encapsulates the journey through three meticulously designed projects, each embodying key software testing paradigms.

Project 1: specification-based testing, introduces foundational testing strategies such as equivalence partitioning, boundary value analysis, and cause and effect testing, followed by an in-depth application of Design of Experiments (DOE) in Part 2. This systematic approach enhances understanding and application of statistical methods in test design. Project 2: structural-based testing, navigates through the realm of code coverage and data flow analysis, emphasizing uncovering hidden defects and enhancing code quality. Lastly, Project 3 takes a pragmatic turn with a hands-on Graphical User Interface (GUI) Testing exercise, where a GUI testing tool's selection, utilization, and assessment are performed in a controlled environment.

Each project's distinct approach and learning outcomes contribute to a holistic understanding of software testing's diverse landscape.

II. Project 1 part 1

A. Introduction

In Project 1Part 1, learner delve deep into the practical application of foundational software testing methodologies taught in class. This project requires learners to employ key techniques such as equivalence partitioning, boundary value analysis, and cause and effect testing, [7] using them to evaluate a provided software program critically. The program in question is a database system designed to provide discounts for a product based on various user factors. However, it comes with a challenge: at least ten seeded defects within the program. The primary objective is for students to develop a comprehensive set of test cases that spot these defects and validate the software's functionality against specified criteria.

B. Explanation of the solution

This project necessitates stringent adherence to specified input requirements, ensuring the robustness of the test cases developed (Figure 1). For the 'User Name,' create cases for strings adhering to a 5-10 character limit, exclusively comprising alphabets and void of numerics, hyphens, or underscores. The 'Age' parameter compels to test for integer inputs, specifically validating the program's acceptance of legal ages (18 and above). When dealing with 'User Status,' 'Rewards Member Status,' 'Season Bought,' and 'Product Category,' emphasis is placed on valid entries, negating the necessity for invalid test cases. Critical testing scenarios involve the program's response to strings in place of expected integers for both 'Age' and 'Rating,' testing its robustness against incorrect data types.

The project's second phase focuses on output requirements, specifically the accuracy of discount vouchers based on diverse user inputs. The system is designed to provide discounts under precise conditions, with specific scenarios categorically excluded from discounts. Therefore, validating the software's precision in recognizing various combinations of 'User Status,' 'Rewards Member Status,' 'Season Bought,' and 'Product Category' and issuing the correct discount percentages is critical. This involves the creation of specific test cases that include valid and diverse sets of inputs corresponding to the different purchasing scenarios and user types. The critical examination is whether the program adheres to the business logic, such as denying discounts to new users and managing different reward

statuses, ensuring that only the appropriate discount vouchers are issued upon execution.

Test Case Number	Name	Age	User Status	Reward Member Status	Season Bought	Product Category	Rating	Pass/Fail	Requirements Mapping
Test Case #1	Mark	24	Returning	Silver	Spring	Electronics	5	Pass	1.1
Test Case #2	markk	24	New	Bronze	Summer	Unknown	5	Fail	9.1
Test Case #3	markk	24	Returning	Bronze	Spring	Electronics	5	Fail	9.4.1
Test Case #4	markk	24	Returning	Silver	Fall	Electronics	5	Fail	9.6.1
Test Case #5	markk	24	Returning	Gold	Winter	Electronics	5	Fail	9.5.1
Test Case #6	markkkkkkkk	24	New	Bronze	Summer	Unknown	5	Fail	1.1
Test Case #7	markk0	24	New	Bronze	Summer	Unknown	5	Pass	1.2
Test Case #8	markk-	24	New	Bronze	Summer	Unknown	5	Pass	1.3
Test Case #9	markk_	24	New	Bronze	Summer	Unknown	5	Fail	1.3
Test Case #10	markk	17	New	Bronze	Summer	Unknown	5	Fail	2.1
Test Case #11	markk	a	New	Bronze	Summer	Unknown	5	Fail	2.1
Test Case #12	markk	24	New	Bronze	Summer	Unknown	0	Fail	7.1
Test Case #13	markk	24	New	Bronze	Summer	Unknown	11	Pass	7.1
Test Case #14	markk	24	New	Bronze	Summer	Unknown	a	Fail	7.1
Test Case #15	markk	24	Returning	Gold	Summer	Unknown	5	Pass	9.2
Test Case #16	markk	24	Returning	Gold	Summer	Electronics	5	Pass	9.7
Test Case #17	markk	24	Returning	Bronze	Summer	Electronics	5	Pass	9.7
Test Case #18	markk	24	Returning	Silver	Summer	Electronics	5	Pass	9.7
Test Case #19	markk	24	Returning	Gold	Spring	Electronics	5	Pass	9.7
Test Case #20	markk	24	Returning	Silver	Spring	Electronics	5	Pass	9.7
Test Case #21	markk	24	Returning	Bronze	Winter	Electronics	5	Pass	9.7
Test Case #22	markk	24	Returning	Silver	Winter	Electronics	5	Pass	9.7
Test Case #23	markk	24	Returning	Gold	Fall	Electronics	5	Fail	9.7
Test Case #24	markk	24	Returning	Bronze	Fall	Electronics	5	Pass	9.7

Figure 1 (Test Cases for Project 1 Part 1)

C. Result

The defect tracking revealed several critical oversights, with 11 defects undetected, indicating gaps in the testing process (Figure 2). The result comprised five output-related test cases and six associated with input validation, reflecting a need for a more comprehensive testing strategy. Particularly concerning was the program's failure to display error messages for non-integer inputs in 'age' and 'rating' fields, a fundamental requirement for maintaining data integrity and ensuring user input validity. This outcome underscores the necessity for enhanced test scenarios, especially those that rigorously challenge the software's ability to handle and respond appropriately to anomalous or unexpected input types.

Defect Number	Defect Description	Requirement Mapping	Test Cases Mapping
D1	Output discount given 10% instead of 0%	9.1	Test Case #2
D2	Output discount given 5% instead of 10%	9.4.1	Test Case #3
D3	Output discount given 10% instead of 15%	9.6.1	Test Case #4
D4	Output discount given 20% instead of 25%	9.5.1	Test Case #5
D5	No error messages for incorrect username longer than 10 characters	1.1	Test Case #6
D6	No error messages for incorrect username contain underscore	1.3	Test Case #9
D7	No error messages for incorrect age under 18	2.1	Test Case #10
D8	No error messages for incorrect age not integer	2.1	Test Case #11
D9	No error messages for incorrect rating less than 1	7.1	Test Case #12
D10	No error messages for incorrect rating not integer	7.1	Test Case #14
D11	Output discount given 10% instead of 0%	9.7	Test Case #23

Figure 2 (Defect Tracking for Project 1 Part 1)

D. Contribution

This was an individual project, wherein the author was solely responsible for the ideation, development, and documentation of all test cases and tracking defects. The GUI utilized for this intricate task was furnished by the CSE 565 course from Arizona State University, providing a structured environment that facilitated this comprehensive exercise in software testing methodologies.

E. New skill

Engaging in this project endows the practical skill of developing a test case matrix [1], a critical tool in software testing. This matrix not only streamlined the organization and execution of test scenarios but also became instrumental in recording the nuanced outcomes of each case. Also, one of the objectives of this project is to compare actual results against expected ones, a process that is fundamental in identifying disparities and understanding software behavior. This hands-on experience has been invaluable, as it transcended textbook knowledge, instilling a comprehensive understanding of real-world applications in software quality assurance and the subtle art of creating compelling, robust test strategies.

F. Lessons learned

This project imparted several critical lessons in software testing methodologies. One notable strategy acquired was defect-based testing, which emphasizes the formulation of test cases strategically crafted to pinpoint specific categories of defects, enhancing the efficacy and focus of the testing process. Furthermore, selecting test cases that rigorously examine boundary conditions is critical. This practice involves crafting scenarios that align with the direct boundaries of input and output equivalence partitions and those that exist just above and below these edges, ensuring a comprehensive assessment.

Additionally, the project illuminated the efficiency of making informed assumptions about related partitions during cause-and-effect testing. By hypothesizing relationships between different partitions, it is feasible to significantly pare down the number of test cases, optimizing resource allocation without compromising the thoroughness of the testing phase. These insights underscore the nuanced complexities of software testing and highlight the need for strategic thinking in effectively uncovering and addressing potential system vulnerabilities.

III. PROJECT 1 PART 2

A. Introduction

This project centered on the innovative application of the Design of Experiments (DOE) technique, specifically in generating effective pairwise combination test cases for a mobile application. By harnessing statistical methodologies, the initiative adopted a systematic strategy that delved into the intricate design of test cases, aiming to critically analyze the multifaceted impacts of various input factors on the application's operational behavior. Several critical objectives anchored the endeavor are the practical application of experimental design principles to craft and refine test protocols, an in-depth exploration to identify a fitting tool specialized for DOE testing and the meticulous development of pairwise combination test cases. This methodical approach underscores the project's commitment to elevating the quality and reliability of mobile software through sophisticated testing paradigms.

B. Explanation of the solution

In this project, various tools underwent rigorous evaluation based on criteria such as feature set, application scope, user-friendliness, and installation simplicity. JMP Pro emerged as the preferred choice, owing to its sophisticated Design of Experiments (DOE) capabilities and proficiency in generating potent pairwise combination test cases. The project outlined specifications for mobile application testing across five distinct variables with multiple levels, encompassing phone type, parallel task execution, connectivity, memory size, and battery level (Figure 3).

Leveraging JMP Pro's capabilities, a structured DOE plan was created, creating pairwise combination test cases.

Furthermore, JMP Pro's advanced DOE technique condensed what would traditionally be 800 individual test scenarios derived from comprehensive variable combinations into a more manageable total of 25. This reduction was achieved without sacrificing test thoroughness, as the tool's output ensured that each possible variable pairing was evaluated at least once. Such an approach is pivotal in unearthing any interactive effects between variables, potentially critical in influencing the mobile application's performance.

Figure 3 (Report for Project 1 Part 2)

C. Result

Significant advantages were discerned through a meticulous evaluation of the test cases in line with Design of Experiments (DOE) principles. DOE allows for the analysis of individual factors and their various combinations, necessitating specific values or ranges to be established for each element under scrutiny. Experiments are conducted using different factor combinations to discern their systemic impact. The test cases, curated by the tool, underwent a rigorous assessment to determine their adherence to DOE guidelines and overall validity. They were deemed both valid and exhaustive, encapsulating all conceivable pairwise factor combinations, thus fulfilling DOE criteria efficiently.

Evaluating the tool itself, JMP Pro, several criteria were considered, including its features and functionalities, areas of usage, and user accessibility. Known for its extensive statistical suite, interactive data visualization, and user-friendly interface, JMP Pro stands out with its robust DOE capabilities. Due to its formidable statistical analysis tools, its applicability spans diverse industries such as engineering, pharmaceuticals, chemicals, semiconductors, and manufacturing. Furthermore, JMP Pro enjoys an intuitive design, rendering it accessible to individuals with varying degrees of statistical acumen, supplemented by helpful guides and tutorials. The installation process posed no substantial hurdles, adding to its convenience. JMP Pro proved instrumental in this project, satisfying all prerequisites and enabling the efficient generation of comprehensive test cases based on pairwise combinations.

D. Contribution

This was an individual project, wherein the author was solely responsible for the ideation, development, and documentation of all results. The application utilized for this intricate task was furnished by JPM Pro, providing a structured environment that facilitated this comprehensive exercise in software testing methodologies.

E. New skill

This project was instrumental in cultivating several advanced skills pivotal in software testing, such as applying the principles of Design of Experiments (DOE) in a practical setting. This approach significantly enhanced the ability to structure tests more effectively and to understand the intricate dynamics involved in testing processes. This methodology underscored the importance of systematic arrangement and observation of tests to evaluate the various factors affecting a response variable. Secondly, the research skills have improved by identifying and selecting the appropriate tool for conducting DOE testing. This phase was crucial as it required a comprehensive understanding of the tool's capabilities concerning the project's needs. Finally, the new skill gained is developing pairwise combination test cases. This technique is compelling in reducing the number of test cases required while still maintaining a high level of effectiveness in identifying errors and enhancing overall proficiency and confidence in software testing.

F. Lessons learned

Throughout this project, there were some invaluable insights from utilizing JMP Pro to implement a Design of Experiments (DOE) plan. By leveraging JMP Pro's advanced functionalities [2], strategically plan experiments that methodically alter input variables, allowing for a detailed analysis of each component's effect on the overall system. This rigorous planning phase was crucial in ensuring the comprehensiveness and effectiveness of the testing process. Furthermore, a significant revelation was understanding and applying the concept of pairwise combination in test cases. This technique, facilitated by JMP Pro, involved generating test scenarios that covered all possible pairs of input variable combinations, significantly enhancing the efficiency of the test suite. Therefore, this project was pivotal in deepening the understanding of sophisticated testing methodologies' strategic and technical facets.

IV. Project 2

A. Introduction

This project focuses on an intensive investigation and assessment of code coverage, drawing on the nuanced approaches of structural-based testing techniques. There were several key objectives. First, to delve into code coverage analysis through the dual lenses of statement and decision coverage methodologies, providing a comprehensive review of the code's robustness and reliability. Secondly, it involves meticulously creating a solid suite of test cases, each designed and refined based on specific criteria to ensure they contribute meaningfully to the coverage analysis. Furthermore, the project allows participants to deepen their understanding of various data flow anomalies, enhancing their ability to identify and address potential irregularities in code behavior. The endeavor is structured into two pivotal phases, each dedicated to honing specific skills integral to software testing and rigorous code evaluation, ensuring a well-rounded competence in these critical aspects of software engineering.

B. Explanation of the solution

Part 1 delves into a Vending Machine Application's comprehensive code coverage analysis, utilizing the JaCoCo tool within the Eclipse IDE environment. Recognized for its extensive application in the Java landscape, JaCoCo facilitates an in-depth examination of code through various lenses of coverage metrics.[3] This includes Statement Coverage, ensuring each line of the code executes during the testing phase, and Decision Coverage, verifying that every branch of each control structure is tested. A series of nuanced test cases were meticulously created for this analysis, focusing on the VendingMachine.java file. These scenarios were diverse, encompassing exact change, overpayment, insufficient funds, and erroneous inputs, thereby promising a thorough evaluation of the application's robustness in handling a broad spectrum of user interactions.

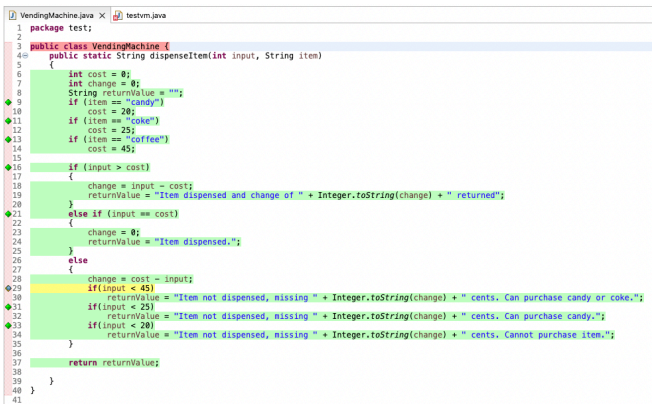

Figure 4 (Code for Project 2 Part 1)

In Part 2, the project transitions into static source code analysis, employing SonarLint, a highly regarded extension known for its seamless integration with various IDEs and real-time feedback capabilities. SonarLint, functioning on the foundational principles consistent with SonarQube, specializes in identifying various anomalies within the Java code. The issues detected include hampering future maintenance, explicit bugs potentially derailing the application's intended functionality, and vulnerabilities that pose significant security risks. Moreover, the tool proves instrumental in highlighting data flow anomalies, pinpointing areas where data dynamics within the application are either inefficient or superfluous, and guiding strategic refinement of the codebase.

C. Result

The results of Part 1 underscored the effectiveness of the test cases employed, with the code coverage metrics illustrating a comprehensive engagement with the code. Specifically, the Statement Coverage reached a full 100%, indicating a complete execution of all lines of code during the testing process (Figure 5). Decision or Branch Coverage was also high at 95.8%, reflecting that nearly all possible branching points were tested for both true and false conditions. This meticulous approach confirms a substantial exploration of the code's various logical pathways, underscoring the testing process's robustness.

Figure 5 (Coverage report and test code for Project 2 Part 1)

In Part 2, the utilization of SonarLint for static code analysis revealed specific inefficiencies within the StaticAnalysis.java file (Figure 6). Among these were two declared but unused variables, 'length' and 'weight.' SonarLint has proved invaluable, and its real-time analysis feature within IDE. [4] Its compatibility with SonarQube ensures uniformity in code quality standards and its support for multiple programming languages beyond Java. The tool demonstrated proficiency in identifying various issues, from syntax errors and code smells to complex vulnerabilities and data flow anomalies. Moreover, SonarLint enhanced the debugging process with clear, actionable feedback on issues,

and its flexible configuration options allow for project-specific customization. This blend of features affirms SonarLint's status as an indispensable asset in pursuing high-caliber, efficient, and secure software development.

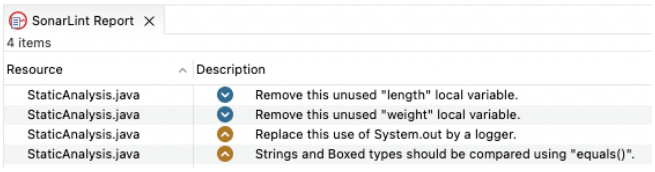

Figure 6 (Report for Project 2 Part 2)

D. Contribution

This was an individual project, wherein the author was solely responsible for the ideation, development, testing, and documentation of all results. The application utilized for this intricate task was furnished by SonarLint, providing a structured environment that facilitated this comprehensive exercise in software testing methodologies.

E. New skill

Through this project, several new skills were gained, such as analyzing code coverage, specifically employing statement and decision coverage techniques [5], which illuminated the importance of comprehensive testing in identifying untested code areas. Additionally, it improves skills in developing detailed, scenario-based test cases adhering to precise requirements. Furthermore, skills such as delving into the complexities of data flow anomalies, understanding their nuances, and recognizing this knowledge's critical role in detecting potential inefficiencies or errors in code logic were adapted as new skills.

F. Lessons learned

This project underscored several invaluable lessons, emphasizing the instrumental role of advanced tools like JaCoCo and SonarLint in software testing and quality assurance. Engaging with JaCoCo illuminated the intricacies of effective software testing, extending beyond basic functionality checks to guarantee that every line of code and potential decision path is rigorously evaluated. This comprehensive approach, especially evident in the context of the Vending Machine application test cases, is crucial for mitigating unforeseen defects and functional aberrations within the software architecture.

Moreover, the experience with SonarLint reaffirmed its status as an essential developer ally. This tool's capacity for real-time, detailed feedback and its exhaustive array of coding rules streamlines the pursuit of high-caliber code quality. Specifically, the exposure of redundant variables within StaticAnalysis.java was a testament to SonarLint's meticulous accuracy. This instance accentuated the critical nature of static code analysis in the continuous improvement and sustainability of coding projects, reaffirming that maintaining lean, efficient, and clutter-free code is just as vital as functional correctness in software development.

V. Project 3

A. Introduction

This project is designed to immerse learners in the intricacies of GUI testing within a real-world context. The learner will embark on a comprehensive journey, from developing a simple GUI application to identifying and employing a suitable GUI automation tool for testing purposes. This initiative aims to foster a deeper understanding of the systematic creation of test cases focused on GUI elements and page flows and utilizing automation testing, leveraging the selected tool's capabilities. By engaging in this hands-on approach, learners will apply and solidify their theoretical knowledge and critically analyze the tool's effectiveness based on coverage, test case reuse, results, and overall usability, thereby gaining insights pivotal for making informed decisions in professional testing environments.

B. Explanation of the solution

Unittest, an integral Python library, facilitates rigorous software unit testing, ensuring optimal functionality. [6] Its capabilities encompass automatic test discovery and execution. It enables the establishment of test prerequisites and post-requisites via fixtures, leveraging methods like setUp and tearDown. Moreover, unittest facilitate the organization of tests into suites, empowering users to select and execute particular test subsets in expansive projects strategically. The library boasts a repertoire of assertion methods, including, but not limited to, assertEqual, assertTrue, and assertFalse. Its mocking and patching features are encapsulated in unittest.mock, permits the simulation of objects and methods, making it invaluable for GUI interaction tests.

In order to verify unittest is suitable for testing GUI application using the tkinter module in Python, a simple GUI application is created (Figure 7). The application in focus is an intricately designed GUI, leveraging the capabilities of Python's tkinter module. Its primary objective revolves around streamlining user authentication mechanisms, encompassing pivotal features such as user sign-ins, registrations, and password recovery procedures.

Figure 7 (Application interfaces version 1)

Version 1 of this application is structured around three primary interfaces. The Sign In page, the Sign Up page, and the Forgot Password page. Each page is designed with a user-friendly layout that includes various elements such as an image, multiple buttons, labels, a slider, a checkbox, and input boxes for seamless user interaction. Notably, the Sign In interface has the navigation process by featuring a 'Sign Up' button that conveniently redirects users to the Sign Up page.

Figure 8 (Application interfaces version 2)

Several refinements have been implemented in this GUI application's transition from Version 1 to Version 2 (Figure 8). These modifications include alterations in the canvas dimensions, strategic repositioning of buttons, a restructured navigation flow, and recalibration of the satisfaction scale, among others. Each page has three changes in Version 2. Also, within the Sign-In interface, the 'Sign Up' button now intuitively redirects users to the 'Password Recovery' interface rather than the registration page.

Test	Page	Changes	Version 1	Version 2
1	Main	Var_name	Take string	Unchange
2	Main	Var_pwd	Take string	Take integer
3	Main	C1 Element	Checkbox	Unchange
4	Main	Sat_Slider	Scale 0-100	Unchange
5	Main	Sat_slider	Slider	Unchange
6	Main	Canvas	Width = 500	Width = 400
7	Main	C1 Location	x = 50, y = 220	x = 50, y = 200
8	Page Flow	Click on Sign Up	Flow to Sign Up	Flow to Forget Password
9	Sign Up	Signup_canvas	Height = 300	Height = 256
10	Sign Up	Signup_name	Take string	Take integer
11	Sign Up	C2 Element	Checkbox	Button
12	Sign Up	C2 Location	x = 50, y = 170	Unchange
13	Sign Up	Signup_slider	Scale 0-100	Unchange
14	Sign Up	Signup_pwd	Take string	Unchange
15	Sign Up	Signup_slider	Slider	Unchange
16	Forget Password	Otp_code	String	Integer
17	Forget Password	Forget_slider	Scale 0-100	Scale 1-100
18	Forget Password	Forget_name	Take string	Unchange
19	Forget Password	C3 Element	Checkbox	Unchange
20	Forget Password	Forget_slider	Slider	Unchange
21	Forget Password	Forget_canvas	Height = 300	Unchange
22	Forget Password	C3 Location	x = 50, y = 180	x = 100, y = 180

Figure 9 (Test cases)

In this comprehensive suite of GUI test cases, rigorous checks are conducted across various pages to ensure data integrity and UI component functionality (Figure 9). On the main page, tests are designed to validate the data types for username and password inputs, confirm the types and features of widgets such as checkboxes and sliders, and verify the canvas size and widget locations. Similar tests are replicated on the signup page, with additional verifications ensuring the proper functionality of user input fields, the adjustability of slider values, and the precise dimensions and positions of GUI elements. The forget page undergoes parallel scrutiny, particularly emphasizing the validation of string entries for username and OTP code, the functionality and adjustability of the slider, and the confirmation of widget types and their specific locations. Beyond individual page elements, the suite culminates with a critical test asserting the seamless navigation flow between pages, triggered explicitly by activating the signup button, a pivotal aspect ensuring a coherent user experience.

C. Result

The comprehensive testing process yielded positive outcomes, with all test scenarios achieving successful pass status (Figure 10). These evaluations, meticulously conducted across the application's three critical interfaces Sign In, Sign Up, and Forget Password were inclusive of both data integrity checks and assessments of layout modifications across the successive application versions. The ability to identify and triumphantly navigate these diverse alterations underscores the efficacy of Unittest as an indispensable asset for quality assurance in Python GUI applications. This success reinforces the utility and reliability of Unittest in the realm of complex software testing environments.

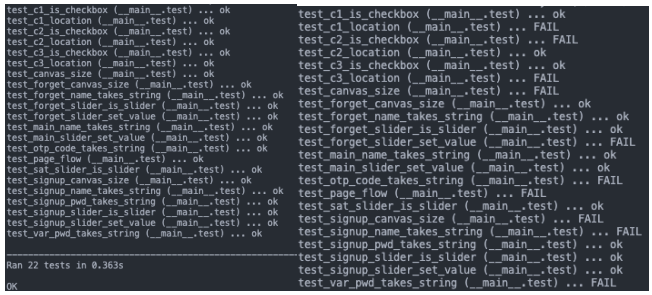

Figure 10 (Test result from Version 1 and Version 2)

D. Contribution

In the context of this research, it is imperative to underscore that the work delineated herein is the sole intellectual product of the author. This comprehensive endeavor encapsulated the genesis of two distinct iterations of a Python-based Graphical User Interface (GUI) application, each illustrating nuanced variations as detailed within this report. The project necessitated an exhaustive exploration and subsequent selection of an apt testing instrument tailored to the developed GUI. This phase was critical, demanding a meticulous analysis of potential tools to ensure the chosen one possessed the requisite features and capabilities to evaluate the application rigorously. Furthermore, the author independently conceptualized and formulated a series of test cases, each designed to assess the selected testing tool's efficacy rigorously. These cases were instrumental in gauging the tool's reliability and performance, particularly in its capacity to interact seamlessly with the GUI application, thereby affirming its validity within this project's scope.

E. New skill

Throughout this project, the primary competencies developed were crafting detailed, objective-oriented test cases designed to validate GUI components and the seamless flow from one page to another, ensuring an intuitive and error-free user experience. Furthermore, by delving into automation testing and harnessing the sophisticated features of the GUI testing tool, new skills gained included covering a broad spectrum of user interactions within the application and how to efficiently reusing test cases to maintain consistency and reliability in testing scenarios. Analyzing test results became more nuanced as interpreting outcomes and applying those insights to enhance usability and functionality.

F. Lessons learned

This project has been an invaluable academic venture, providing numerous educational insights. A critical takeaway is the indispensable role of regression testing in the software modification process, an activity inherently fraught with risk. The primary objective of regression testing is to ascertain that functionalities, once developed and tested, continue to operate within the predefined specifications after any alterations in the software. This form of testing becomes pivotal in maintaining system integrity and performance consistency over time.

Applying a defect-seeding strategy was instrumental in developing robust test cases in this project's context. This technique, employed during validating Graphical User Interface (GUI) elements and page flows, facilitated a rigorous approach to automation testing. By intentionally implanting defects, assess the test tool's efficacy by calculating the rate of correctly identified seeded defects.

The composition of test cases demanded a strategic approach. It was imperative to incorporate a comprehensive range of elements within the tests, encompassing both the modified components and a subset of unchanged elements. This strategy was integral to enhancing the thoroughness of the testing process.

The lessons gleaned from this project underscore the complexities inherent in software modification and the paramount importance of meticulous, strategic testing protocols. These insights contribute significantly to the broader academic and practical discourse on effective software development practices.

VI. Conclusion

Upon reflection, the journey through the various facets of software testing provided invaluable insights and practical skills in implementing systematic and efficient testing methods. Each project underscored different dimensions of testing, starting from the nuanced test case development in Project 1 using specification-based techniques and DOE, to a deeper understanding of internal code quality and flow in Project 2, and finally the external functionality and user interaction aspects in Project 3's GUI testing. The hands-on experience of utilizing different tools and strategies illuminated their respective strengths and potential areas for improvement, providing a clear path toward their effective integration in real-world scenarios. This exploration reinforced not only the technical know-how but also the strategic thinking necessary to navigate the complexities of modern software testing, affirming that a multifaceted approach is essential in achieving software excellence.

VII. References

[1] The Nerdy Geek, " Importance of Traceability Matrix in Testing," Browser Stack, Feb. 1, 2023. Accessed on: Oct. 17, 2023. [Online]. Available: https://www.browserstack.com/guide/importance-of-traceability-matrix-in-testing

[2] JMP, " Designed experiments for more understanding and maximum impact." Accessed on: Oct. 16, 2023. [Online]. Available: https://www.jmp.com/en_us/software/capabilities/design-of-experiments.html

[3] Mountainminds GmbH & Co. KG, " Java Code Coverage for Eclipse." Accessed on: Oct. 16, 2023. [Online]. Available: https://www.jacoco.org/userdoc/coverageview.html

[4] SonarSource S.A, "your companion for Clean Code." Accessed on: Oct. 16, 2023. [Online]. Available: https://www.sonarsource.com/products/sonarlint/features/

[5] Thomas Hamilton, “Code Coverage Tutorial: Branch, Statement & Decision Testing," Guru99, Oct. 14, 2023. Accessed on: Oct. 17, 2023. [Online]. Available: https://www.guru99.com/code-coverage.html

[6] Python Software Foundation, " unittest — Unit testing framework." Accessed on: Oct. 16, 2023. [Online]. Available: https://docs.python.org/3/library/unittest.html

[7] Thomas Hamilton, “Boundary Value Analysis and Equivalence Partitioning Testing," Guru99, Oct. 14, 2023. Accessed on: Oct. 17, 2023. [Online]. Available: https://www.guru99.com/equivalence-partitioning-boundary-value-analysis.html