CSE 565: Software Verification and Validation Portfolio Submission

Gautham Vijayaraj
gvijaya6@asu.edu
Arizona State University
Tempe, Arizona, USA

Abstract

This report summarizes the key learnings and outcomes from five comprehensive assignments completed in this course (CSE 565: Software Verification and Validation). Each assignment focused on specific aspects of software verification and validation, leveraging tools, methodologies, and testing paradigms. The assignments strengthened my understanding of concepts like unit testing, combinatorial testing, code coverage analysis, endurance testing, and reliability prediction. This portfolio reflects the application of theoretical knowledge to practical scenarios, equipping me with essential skills for software quality assurance.

Keywords

Software Testing, Unit Test Automation, Generative AI, Pairwise Testing, Code Coverage Analysis, Static Code Analysis, Endurance Testing, Reliability Prediction, Machine Learning in Testing, Software Quality Assurance

ACM Reference Format:
Gautham Vijayaraj. 2024. CSE 565: Software Verification and Validation Portfolio Submission. In *ACM SIGCONF Conference (Conference name).* ACM, New York, NY, USA, 4 pages. https://doi.org/10.xxxx/xxxxxxx.xxxxxxx

1 Assignment 1: Generative AI for Unit Testing

1.1 Introduction

This assignment explored the use of an AI-driven tool suggested by ChatGPT [19] to generate unit tests, and compared its functionality with the manual approach of writing test cases in JUnit.

1.2 Explanation of the Solution

- Selected Diffblue Cover [4], an AI-based unit test generation tool, for its ability to analyze Java code and generate comprehensive test cases.
- Integrated Diffblue Cover into a Java project using IntelliJ.
- The Quick Sort algorithm was used as the testing target.
- Diffblue Cover generated baseline test cases, covering edge cases, normal scenarios, and exception handling.
- The generated tests were executed, and results showed high test coverage for the Quick Sort implementation.
- Manually fine-tuned specific tests in JUnit to address domain-specific scenarios, ensuring comprehensive testing.
- Test results demonstrated the utility of AI tools in automating repetitive testing tasks while maintaining quality.

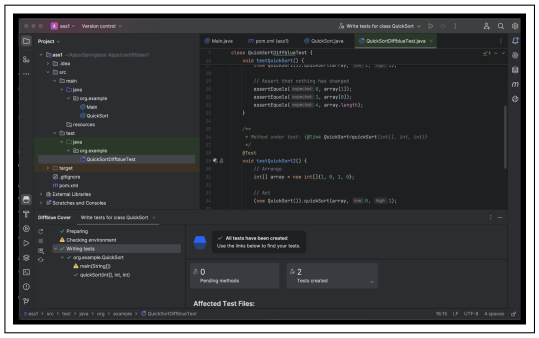
1.3 Summary of Results

- Diffblue Cover successfully automated the generation of unit test cases in Figure 1 for a Quick Sort implementation, providing comprehensive coverage for edge cases, boundary values, and normal scenarios.
- Generated test cases in Figure 2 enhanced the efficiency of the test suite, covering multiple execution paths, and reducing manual effort.
- Comparative analysis showed that while Diffblue Cover automates baseline testing, JUnit allows for manual fine-tuning of complex, domain-specific test cases.

1.4 New Skills and Knowledge

- Learned to integrate AI tools into the development lifecycle for test automation.
- The complementarity of automated and manual testing approaches was investigated.
- Gained proficiency in identifying scenarios where AI can accelerate development without compromising test accuracy.

1.5 Output Screenshots

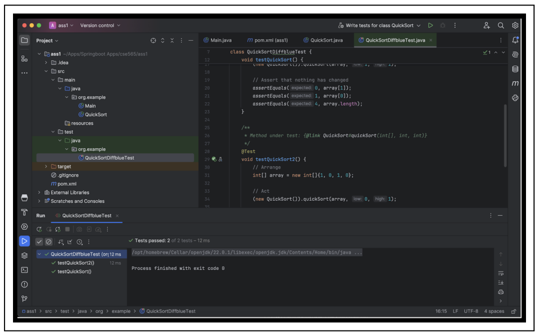


```
class QuickSortDiffblueTest {
    void testQuickSort() {
            (new QuickSort()).quickSort(array, low: 1, high: 1);

            // Assert that nothing has changed
            assertEquals( expected: 0, array[1]);
            assertEquals( expected: 1, array[0]);
            assertEquals( expected: 4, array.length);
        }

        /**
         * Method under test: {@link QuickSort#quickSort(int[], int, int)}
         */
        @Test
        void testQuickSort2() {
            // Arrange
            int[] array = new int[]{1, 0, 1, 0};

            // Act
            (new QuickSort()).quickSort(array, low: 0, high: 1);
```

All tests have been created
Use the links below to find your tests.

0 Pending methods
2 Tests created

Figure 1: Test cases generated by Diffblue Cover



```
class QuickSortDiffblueTest {
    void testQuickSort() {
        // Assert that nothing has changed
        assertEquals( expected: 0, array[1]);
        assertEquals( expected: 1, array[0]);
        assertEquals( expected: 4, array.length);
    }

    /**
     * Method under test: {@link QuickSort#quickSort(int[], int, int)}
     */
    @Test
    void testQuickSort2() {
        // Arrange
        int[] array = new int[]{1, 0, 1, 0};

        // Act
        (new QuickSort()).quickSort(array,  low: 0,  high: 1);
```

```
Tests passed: 2 of 2 tests – 12 ms
/opt/homebrew/Cellar/openjdk/22.0.1/libexec/openjdk.jdk/Contents/Home/bin/java ...

Process finished with exit code 0
```

Figure 2: Test cases passed

2 Assignment 2: Design of Experiments and Pairwise Testing

2.1 Introduction

This assignment used the PICT (Pairwise Independent Combinatorial Testing Tool) [11] to optimize the generation of test cases for an e-commerce platform with multiple combinations of variables.

2.2 Explanation of the Solution

- Chose PICT [8], a command-line tool by Microsoft, for generating pairwise test cases. Installed it on macOS using the Mono framework.
- Defined five factors for an e-commerce system: authentication methods, payment methods, shipping options, device types, and user account types.
- Created a .pict file in Figure 3 to specify the variables and their levels.
- Ran the PICT tool to produce a compact set of 20 pairwise test cases, ensuring that all feature pairs were tested.
- Test cases in Figure 4 were systematically verified to uncover potential interactions and conflicts.
- PICT significantly reduced the test space compared to exhaustive testing, ensuring efficient use of resources without compromising coverage.

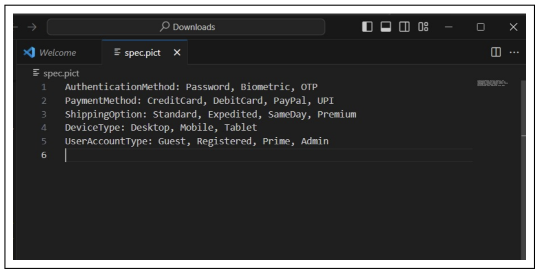
2.3 Summary of Results

- PICT reduced an exhaustive testing set to 20 pairwise combinations, ensuring comprehensive interaction coverage while minimizing effort [20].
- The key features tested included combinations of authentication methods, payment systems, shipping options, device types, and user account types.
- Generated compact test cases were effective in exposing potential interactions and conflicts between system components.

2.4 New Skills and Knowledge

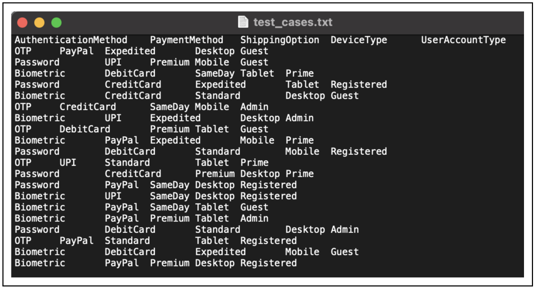
- Mastered the application of Design of Experiments (DOE) [2] to software testing.
- Learned to leverage pairwise testing to optimize test case design without sacrificing coverage.
- Acquired hands-on experience using PICT on macOS through Mono integration.

2.5 Output Screenshots



```
AuthenticationMethod: Password, Biometric, OTP
PaymentMethod: CreditCard, DebitCard, PayPal, UPI
ShippingOption: Standard, Expedited, SameDay, Premium
DeviceType: Desktop, Mobile, Tablet
UserAccountType: Guest, Registered, Prime, Admin
```

Figure 3: The PICT file with the specifications



```
AuthenticationMethod    PaymentMethod   ShippingOption  DeviceType      UserAccountType
OTP     PayPal  Expedited       Desktop Guest
Password        UPI     Premium Mobile  Guest
Biometric       DebitCard       SameDay Tablet  Prime
Password        CreditCard      Expedited       Tablet  Registered
Biometric       CreditCard      Standard        Desktop Guest
OTP     CreditCard      SameDay Mobile  Admin
Biometric       UPI     Expedited       Desktop Admin
OTP     DebitCard       Premium Tablet  Guest
Biometric       PayPal  Expedited       Mobile  Prime
Password        DebitCard       Standard        Mobile  Registered
OTP     UPI     Standard        Tablet  Prime
Password        CreditCard      Premium Desktop Prime
Password        PayPal  SameDay Desktop Registered
Biometric       UPI     SameDay Desktop Registered
Biometric       PayPal  SameDay Tablet  Guest
Biometric       PayPal  Premium Tablet  Admin
Password        DebitCard       Standard        Desktop Admin
OTP     PayPal  Standard        Tablet  Registered
Biometric       DebitCard       Expedited       Mobile  Guest
Biometric       PayPal  Premium Desktop Registered
```

Figure 4: The output test cases generated

3 Assignment 3: Design of Code Coverage Analysis and Static Source Code Analysis

3.1 Introduction

This assignment combined JaCoCo [7] for decision code coverage and SonarLint [9] for static source code analysis to evaluate a Java-based Quicksort implementation.

3.2 Explanation of the Solution

- Integrated [5] JaCoCo into a Maven-based project [21].
- Configured it to measure branch, decision, and line coverage for the Quick Sort algorithm.
- Identified untested logical paths and modified test cases to achieve complete decision coverage.
- Installed SonarLint as an IntelliJ plugin to identify coding issues during development.
- Detected unused variables, dead code, and reassignment errors.
- Combined JaCoCo's runtime analysis with SonarLint's static checks to ensure robust and maintainable code.
- Modified the codebase [24] to address issues flagged by SonarLint, improving overall quality.

- Achieved a balanced approach to testing by leveraging both dynamic and static tools.
- Generated detailed coverage and quality reports to demonstrate improved code reliability.

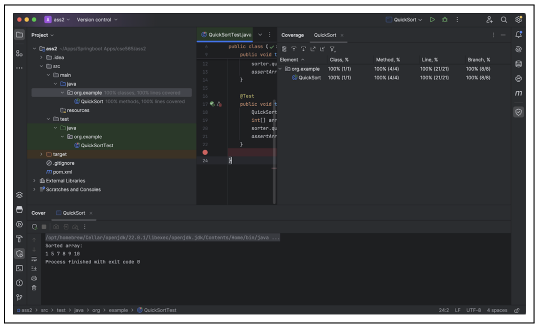
3.3 Summary of Results

- JaCoCo identified untested branches in Figure 5 and logical paths, enabling targeted test enhancements to achieve full decision coverage.
- SonarLint highlighted issues like unused variables, dead code, and reassignment errors, ensuring adherence to coding best practices in Figure 6.
- Combined, these tools provided a comprehensive overview of both runtime behavior [18] and static code quality.

3.4 New Skills and Knowledge

- Developed expertise in integrating JaCoCo into a Maven project to visualize and improve test coverage.
- Gained proficiency in real-time static code analysis with SonarLint, enhancing code maintainability.
- Understood the synergy between dynamic and static analysis tools for robust software testing.

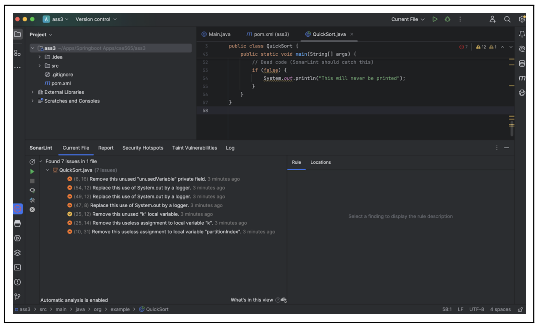
3.5 Output Screenshots



Element	Class, %	Method, %	Line, %	Branch, %
org.example	100% (1/1)	100% (4/4)	100% (21/21)	100% (8/8)
QuickSort	100% (1/1)	100% (4/4)	100% (21/21)	100% (8/8)

```
/opt/homebrew/Cellar/openjdk/22.0.1/libexec/openjdk.jdk/Contents/Home/bin/java ...
Sorted array:
1 5 7 8 9 10
Process finished with exit code 0
```

Figure 5: Analysis provided by JaCoCo (coverage reports)



```java
public class QuickSort {
    public static void main(String[] args) {
        // Dead code (SonarLint should catch this)
        if (false) {
            System.out.println("This will never be printed");
        }
    }
}
```

```
Found 7 issues in 1 file
  QuickSort.java (7 issues)
    (6, 16) Remove this unused "unusedVariable" private field. 3 minutes ago
    (54, 12) Replace this use of System.out by a logger. 3 minutes ago
    (49, 12) Replace this use of System.out by a logger. 3 minutes ago
    (47, 8) Replace this use of System.out by a logger. 3 minutes ago
    (25, 12) Remove this unused "k" local variable. 3 minutes ago
    (25, 14) Remove this useless assignment to local variable "k". 3 minutes ago
    (10, 31) Remove this useless assignment to local variable "partitionIndex". 3 minutes ago
```

Figure 6: Analysis provided by SonarLint (quality reports)

4 Assignment 4: Endurance Testing

4.1 Introduction

This assignment focused on endurance testing [15], analyzing the prolonged stability of systems like e-commerce platforms and banking applications under sustained load.

4.2 Explanation of the Solution

- Selected Apache JMeter [1] and LoadRunner [16] for simulating sustained load scenarios.
- Designed endurance tests for two systems: E-commerce Platform and Banking System.
- Simulated continuous user actions like browsing, checkout, and payment processing on e-commerce platforms.
- Mimicked online transactions and account queries under prolonged operational loads on baking systems.
- Ran long-duration tests to monitor metrics like memory usage, response times, and database connections.
- Identified critical issues such as memory leaks and response degradation.
- Used built-in monitoring tools to pinpoint bottlenecks and resource allocation inefficiencies.
- Suggested strategies [22] for database connection pooling and memory management.
- Detailed performance [3] reports highlighted potential vulnerabilities and offered actionable insights for improvement.

4.3 Summary of Results

- Used tools like Apache JMeter and LoadRunner to simulate real-world conditions, such as peak user traffic and continuous transaction processing.
- Identified critical issues [17], including memory leaks, database connection failures, and response time degradation, in long-duration tests.
- Proposed actionable strategies to ensure system reliability under extended operational loads.

4.4 New Skills and Knowledge

- Acquired hands-on experience with endurance testing tools [10] for prolonged stress simulations.
- Learned to identify and address system bottlenecks that emerge under continuous usage.
- Gained insight into designing realistic load scenarios for stability testing.

5 Assignment 5: Reliability Prediction Approaches

5.1 Introduction

This assignment compared traditional reliability models like the Musa-Okumoto model [6] with modern, data-driven approaches such as machine learning.

5.2 Explanation of the Solution

- Gathered historical and contextual data [23] for both models, ensuring accurate comparisons.
- Compared their accuracy, scalability, and adaptability across various datasets.
- Highlighted the limitations of Musa-Okumoto [14] in dynamic environments and the flexibility of machine learning in handling heterogeneous systems.
- Demonstrated how machine learning enhances reliability predictions in modern software development contexts [12].

5.3 Summary of Results

- Musa-Okumoto Model proved effective in scenarios with well-documented failure data but lacked adaptability to dynamic systems [25].
- Machine learning-based methods demonstrated flexibility, leveraging diverse datasets like code metrics and usage logs for predictions.
- Highlighted the trade-offs between the data dependency of traditional models and the scalability of machine learning approaches [13].

5.4 New Skills and Knowledge

- Gained an understanding of the logarithmic Poisson execution time model for reliability prediction.
- Learned to apply machine learning algorithms for dynamic and heterogeneous software systems.
- Developed the ability to evaluate predictive models based on context, data availability, and system requirements.

6 Conclusion

These assignments collectively enhanced my understanding of software testing paradigms, instilling both theoretical knowledge and practical skills. From leveraging AI for automated testing to optimizing reliability predictions, I gained a holistic view of ensuring software quality. Tools like Diffblue Cover, PICT, JaCoCo, SonarLint, JMeter, and LoadRunner became instrumental in developing a rigorous and efficient testing approach. This learning experience prepares me to tackle complex testing challenges in real-world projects.

References

[1] Apache. 2024. Apache JMeter Documentation. Docs. https://jmeter.apache.org/
[2] ASQ. 2024. Design of Experiments (DOE) Glossary. Docs. https://asq.org/quality-resources/design-of-experiments#:~:text=Quality%20Glossary%20Definition%3A%20Design%20of,parameter%20or%20group%20of%20parameters
[3] QA Source Blog. 2024. QA Testing Insights. Docs. https://blog.qasource.com/a-complete-guide-to-endurance-testing
[4] Diffblue. 2024. Diffblue Cover Documentation. Docs. https://docs.diffblue.com/
[5] Eclemma. 2024. JaCoCo Installation Steps. Docs. https://www.eclemma.org/jacoco/
[6] Geeks for Geeks. 2024. Musa-Okumoto Logarithmic Model Definition. Docs. https://www.geeksforgeeks.org/musa-okumoto-logarithmic-model/
[7] GitHub. 2024. JaCoCo GitHub Repository. Repository. https://github.com/jacoco/jacoco
[8] GitHub. 2024. PICT Tool GitHub Repository. Repository. https://github.com/microsoft/pict
[9] Jetbrains. 2024. SonarLint Documentation. Docs. https://plugins.jetbrains.com/plugin/7973-sonarlint
[10] Software Testing Material. 2024. Endurance Testing Guide. Docs. https://www.softwaretestingmaterial.com/endurance-testing/
[11] Microsoft. 2024. PICT Documentation. Docs. https://learn.microsoft.com/en-us/windows-hardware/drivers/taef/pict-data-source
[12] Gao S. Pattipati Morales, R. 2021. A machine learning approach for software reliability prediction. Journal of Systems and Software, 176. http://dx.doi.org/10.1109/DAC18072.2020.9218543
[13] Singh M. P. MSingh, A. 2014. Software reliability growth modeling using machine learning techniques: A survey. International Journal of Computer Applications, 97(21), 1-7.. https://www.researchgate.net/publication/332172546_Survey_on_software_reliability_prediction_using_machine_learning_techniques
[14] J. D. Musa. 1980. Software Reliability Engineering: More Reliable Software Faster and Cheaper. McGraw-Hill. https://books.google.com/books/about/Software_Reliability_Engineering.html?id=FL1QAAAAMAAJ
[15] Art of Testing. 2024. Endurance Testing Definition. Docs. https://artoftesting.com/endurance-testing
[16] Perfmatrix. 2024. LoadRunner Documentation. Docs. https://www.perfmatrix.com/micro-focus-loadrunner-tutorial/
[17] Load View Testing. 2024. Endurance Testing for System Stability. Docs. https://www.loadview-testing.com/learn/endurance-testing/
[18] Gautham Vijayaraj. 2024. ChatGPT Prompt 2 for assignment 3. ChatGPT. https://chatgpt.com/share/6700baa7-265c-8013-bea3-7f8fb26dc1b4
[19] Gautham Vijayaraj. 2024. ChatGPT Prompt for assignment 1. ChatGPT. https://chatgpt.com/share/59fdc4f2-1582-431e-aa9c-b48e02b4ce72
[20] Gautham Vijayaraj. 2024. ChatGPT Prompt for assignment 2. ChatGPT. https://chatgpt.com/share/66ee14c5-9e2c-8013-b23d-bee513a2d447
[21] Gautham Vijayaraj. 2024. ChatGPT Prompt for assignment 3. ChatGPT. https://chatgpt.com/share/6700b42b-8134-8013-9cfd-af877f08689f
[22] Gautham Vijayaraj. 2024. ChatGPT Prompt for assignment 4. ChatGPT. https://chatgpt.com/share/6722db30-8e7c-8013-b938-0e0c7b6405a8
[23] Gautham Vijayaraj. 2024. ChatGPT Prompt for assignment 5. ChatGPT. https://chatgpt.com/share/673c23e6-7690-8013-80f4-446962e9c206
[24] Gautham Vijayaraj. 2024. Gemini Prompt for assignment 3. Gemini. https://g.co/gemini/share/ff7619b14594
[25] Li Y. Wang S. Zhang, X. 2020. Machine learning methods for software reliability prediction. IEEE Transactions on Reliability, 69(3). https://doi.org/10.1109/QRS-C.2017.115