**EvoSuite Experience Report**

**Course Name: CIS-565 Software Quality Assurance**

**Assignment 03**

**Student Name: Neeraj Saini**

**UMID - 33580268**

**Introduction**

This report compares the branch coverage of the manually created test suite (measured using Cobertura) and the EvoSuite-generated test suite for the jsoup library. The goal is to analyze the effectiveness of EvoSuite in generating test cases and identify areas where it outperforms or underperforms compared to manually written tests.

**A) Comparison of Branch Coverage**

**Manually Created Test Suite**

* The manually created test suite achieves 81%-line coverage and 77% branch coverage overall, as reported by Cobertura (index.html).
* This test suite is well-structured and covers a wide range of scenarios, including edge cases and complex logic.

**EvoSuite-Generated Test Suite**

* The EvoSuite-generated test suite shows varying levels of branch coverage across different classes, as seen in the statistics.csv file.
* Higher Coverage: EvoSuite achieves higher branch coverage for some classes, such as:
  + org.jsoup.helper.StringUtil (**96.77%**)
  + org.jsoup.nodes.TextNode (**97.78%**)
* **Lower Coverage**: EvoSuite struggles with classes involving complex logic or external dependencies, such as:
  + org.jsoup.helper.HttpConnection (**27.74%**)
  + org.jsoup.parser.HtmlTreeBuilderState (**46.42%**)

**Key Observations**

* EvoSuite excels in generating high coverage for simple, well-defined classes with clear input-output relationships.
* It struggles with complex classes or those involving external dependencies (e.g., network calls, stateful behavior).
* Overall, EvoSuite complements the manually created test suite by covering edge cases and boundary conditions that humans might miss.

**b) EvoSuite's Best Class**

Class: org.jsoup.helper.StringUtil

* Branch Coverage: 96.77% (EvoSuite) vs. Lower in Manually Created Tests.

**Analysis**

* What EvoSuite Tests: EvoSuite generated tests for edge cases and boundary conditions that were not covered by the manually created tests. For example:
  + Testing methods with empty strings, null inputs, or strings with special characters.
  + Exploring edge cases in string manipulation methods, such as trimming, splitting, or encoding.
* Why EvoSuite is Likely to Generate Such Tests: EvoSuite uses automated search-based techniques to explore various input combinations. This allows it to discover edge cases that humans might overlook due to time constraints or oversight.
* **Quality of Tests**:
  + **Functionality**: The tests are effective in covering edge cases and improving overall coverage.
  + **Readability**: The generated tests may lack readability due to their automated nature. Test names and assertions might not be as descriptive as manually written tests.
  + **Debugging**: If a test fails, the failure message might not be as intuitive as a manually written test. However, the test would still point to the specific input or condition that caused the failure, aiding in debugging.

**Conclusion**

EvoSuite's ability to generate high-coverage tests for StringUtil demonstrates its strength in exploring edge cases and boundary conditions. While the tests are functional, their readability could be improved to make them more maintainable.

**c) EvoSuite's Worst Class**

**Class: org.jsoup.helper.HttpConnection**

* Branch Coverage: 27.74% (EvoSuite) vs. Higher in Manually Created Tests.

**Analysis**

* What EvoSuite Misses: EvoSuite struggles to generate tests for classes involving external dependencies (e.g., network calls) or complex logic. In HttpConnection, the class interacts with external resources (e.g., HTTP servers), which EvoSuite cannot easily mock or simulate.
* **Why EvoSuite Struggles**:
  + **External Dependencies**: EvoSuite does not inherently understand how to mock or simulate external systems like HTTP servers.
  + **Complex Logic**: The class may have complex branching logic that EvoSuite cannot fully explore within its search budget.
  + **Stateful Behavior**: Classes that rely on stateful interactions (e.g., maintaining connection state) are harder for EvoSuite to test effectively.
* **Hypothesis**: EvoSuite's inability to handle external dependencies and complex logic limits its coverage for classes like HttpConnection. Manually created tests can explicitly mock external dependencies and test complex scenarios, leading to higher coverage.

**Conclusion**

EvoSuite's low coverage for HttpConnection highlights its limitations in testing classes with external dependencies or complex logic. Manually created tests are better suited for such scenarios, as they can explicitly handle mocking and stateful behavior.

**Overall Reflection**

**Strengths of EvoSuite**

* **Edge Case Discovery**: EvoSuite is effective at discovering edge cases and boundary conditions that humans might miss.
* **Automation**: It reduces the manual effort required to write tests, especially for simple, well-defined classes.
* **Complementary**: It complements the manually created test suite by filling gaps in coverage.

**Weaknesses of EvoSuite**

* **External Dependencies**: It struggles with classes involving external resources (e.g., network calls, databases).
* **Complex Logic**: It may not fully explore complex branching logic or stateful behavior.
* **Readability**: The generated tests may lack readability and maintainability compared to manually written tests.

**Recommendations**

* Use EvoSuite to **augment** the manually created test suite, especially for simple classes and edge case discovery.
* For complex classes or those with external dependencies, rely on **manually written tests** with proper mocking and state management.
* Consider **refactoring** EvoSuite-generated tests to improve readability and maintainability.

**Conclusion**

EvoSuite is a powerful tool for automating test case generation, particularly for simple classes and edge case discovery. However, it has limitations when dealing with complex logic or external dependencies. By combining EvoSuite-generated tests with manually written tests, we can achieve comprehensive test coverage and improve the overall quality of the jsoup library.