SE333 - Demonstration Report

Sofia C. Barrios
SE333 Software Testing
Professor Dong Jae Kim

***Abstract*—This report presents a written demonstration of an AI-assisted agent for automated JUnit test generation in Java Maven projects that goes hand-in-hand with the provided recording of the demonstration. The agent uses the Model Context Protocol (MCP) framework and an LLM to generate, execute, and iteratively refine tests, improving coverage and enforcing coding standards. GitHub Actions integration enables automatic generation of JaCoCo and Checkstyle reports for continuous feedback.**

I. Project Overview & Objectives

Goal: Implement an AI-assisted agent to automatically generate, run, and iteratively improve JUnit tests for a Java Maven project.

Motivation: Manual testing is time-consuming and prone to errors; AI-assisted testing improves coverage and regression safety.

Technical Stack:

- **Language:** Java (JUnit), Python (AI agent)
- **Frameworks/Tools:** MCP framework, LLM, JaCoCo (coverage), Checkstyle (coding standards)
- **CI/CD:** GitHub Actions for automatic coverage and style reporting

II. Feature Demonstration

A. User Input → Processing → Output

- **Input:** User triggers the agent with `perform tester prompt`.
- **Processing:**
  1) Reads `main.py` for project configuration.
  2) Parses JaCoCo coverage report to identify low-coverage classes/methods.
  3) Generates new tests using boundary, equivalence, decision table, and contract-based strategies.
  4) Executes tests and iteratively refines coverage.
  5) Uploads updates to GitHub, triggering GitHub Actions to generate updated JaCoCo and Checkstyle reports.
- **Output:** Updated JUnit test suite, coverage metrics, and automated repository update with CI/CD reports.

B. MCP Commands and Configuration

- Trigger command: `"perform tester prompt"`.
- Configuration includes project paths, test generation parameters, and automated commit settings.
- Rationale: Enables full automation from user command to coverage improvement with minimal manual intervention.

C. LLM-Based Reasoning and Automation

- Determines which methods or classes require additional tests based on coverage gaps.
- Dynamically selects appropriate test types according to method signatures.
- Generates assertions and iteratively refines tests over multiple runs.
- Pushes updates to GitHub, triggering CI/CD for automatic coverage and style reporting.

D. Design Decisions, Trade-offs, and Constraints

- **Design:** Fully automated AI-assisted workflow using MCP and LLM.
- **Trade-offs:** Reduces manual effort but may produce redundant or incomplete tests; runtime increases for larger projects.
- **Constraints:** Limited to Maven runtime execution; coverage and static analysis depend on JaCoCo and Checkstyle outputs.

III. Metric Improvement Showcase

Before vs. After Coverage:

Metric	**Before**	**After**	**Improvement**
Line coverage	93.53%	93.57%	+4 lines
Branch coverage	90.27%	90.27%	0
Instruction coverage	94.04%	94.08%	+21 instructions
Method coverage	93.78%	93.86%	+2 methods

Qualitative Insights:

- AI-generated tests explored new paths and edge cases.
- High-impact classes (e.g., `Conversion.java`, `ToStringBuilder.java`) showed meaningful coverage gains.
- Some redundancy occurred, but regression safety improved overall.

GitHub Integration:

- Automated commits push updates to the repository.
- GitHub Actions generate updated JaCoCo and Checkstyle reports without manual intervention.

IV. Reflection & Insights

Challenges:

- Debugging AI-generated test failures.
- Prompt engineering for LLM to produce meaningful assertions.

AI-Assisted Development:

- **Helpful:** Rapid, iterative test generation; improved coverage; automated CI/CD integration.
- **Manual Intervention:** Reviewing edge cases and removing redundant or incomplete tests.

Future Enhancements:

- Integrate SpotBugs for additional static analysis.
- Organize JaCoCo and Checkstyle reports into a dedicated folder.
- Add visual coverage summaries to highlight gaps in the codebase.

Most Impressive Feature: The MCP tools that created Specification-Based tests