Reflection Report: MCP Testing Agent for SE333 Final Project

Leonardo Diaz

November 16, 2025

Abstract

This report summarizes the development and evaluation of an autonomous testing agent built for the SE333 final project. The agent uses Maven and Git automation to generate tests, improve code coverage, and integrate specification-based testing. The study evaluates coverage improvements and discusses lessons learned from using AI-assisted development tools.

1 Introduction

The goal of this project was to create an automated testing agent that can generate and run unit tests, collect coverage data, and manage version control tasks. The agent is built using Python with the FastMCP framework, integrating with a Java Maven project. This project also explores the application of specification-based testing to improve coverage for critical methods in the codebase.

2 Methodology

The testing agent performs the following tasks:

- Executes Maven tests and collects JaCoCo coverage reports.
- Stages, commits, and pushes code changes to GitHub.
- Generates specification-based tests, focusing on boundary value analysis and equivalence classes.

The implementation involved:

1. Setting up a FastMCP server (`server.py`) to provide tools like `run_tests()`, `git_status()`, `git_add_all()`, and `git_commit()`.
2. Creating new JUnit test classes to cover untested branches, especially for the `classifyNumber(int)` method.
3. Running tests, collecting coverage via JaCoCo, and verifying all branches were executed.

3 Results

3.1 Coverage Improvement

The agent successfully improved code coverage for the `classifyNumber()` method:

- Branch coverage: 100% (positive, negative, zero branches)
- Line coverage: 100% (8/8 lines)
- Instruction coverage: 100% (17/17 instructions)
- Method coverage: 100% (3/3 methods)

A total of 21 tests ran across four test classes, with zero failures, errors, or skipped tests.

3.2 Analysis

The specification-based tests were effective in covering all branches of the method. The AI-assisted agent simplified repetitive tasks such as generating new tests, running Maven commands, and managing Git operations, saving development time and ensuring consistency.

3.3 Lessons Learned

- AI tools can efficiently assist in generating tests but still require human oversight for meaningful test design.
- Integration of testing, coverage analysis, and version control into a single agent streamlines workflow.
- Clear documentation and incremental testing prevent regression when adding new tests or methods.

3.4 Future Enhancements

- Integrate static analysis tools like SpotBugs or PMD to detect code smells automatically.
- Expand the agent to generate property-based tests for additional classes.
- Automate pull request approval workflow for small changes.

4 Conclusion

The MCP testing agent successfully automated test execution, coverage collection, and Git operations while demonstrating the benefits of specification-based testing. The project highlights the potential of AI-assisted development to enhance productivity and software quality.