# Reflection Report: MCP Automated Testing Agent

Leonardo Diaz, Hammy Siddiqui

**Abstract**

This reflection explains how I built an MCP agent that runs Maven tests, manages Git commands, and generates specification-based tests automatically. I describe how I approached the problem, what kind of improvements I noticed in the project's test coverage, and what I learned about using AI to help with software development. I also include recommendations for future improvements to the project.

## 1 Introduction

For this assignment, I created a custom MCP (Model Context Protocol) server that connects with VS Code Chat and allows me to run tests, commit code, and generate new tests automatically. The main idea behind the project was to explore how automated tools and AI can improve the software development workflow. Since I am still learning about automation and testing, this project helped me understand how tools can speed things up and reduce mistakes I would normally make by hand.

## 2 Methodology

My approach was pretty straightforward. First, I set up the MCP server and added tools for running Maven tests and Git commands. Then I made a tool that generates JUnit tests based on simple specifications, like checking positive, negative, and zero inputs. After that, I tested everything using the VS Code Chat interface, which let me call the tools by typing messages.

To evaluate how useful the generated tests were, I compared the original test suite with the new tests created by the MCP tool. I also observed how often the tests caught issues or improved the overall coverage.

## 3 Results

### 3.1 Coverage Improvement Patterns

Even though the agent only produced a small set of tests, I still noticed some coverage improvements. The original project focused mostly on regular input values, but the automatically generated tests forced the program to handle extreme values like `Integer.MAX_VALUE` and `Integer.MIN_VALUE`. These edge cases increased branch and statement coverage in parts of the code that were not originally exercised.

I also saw that using AI-generated tests made it easier to fill in missing equivalence classes. For example, the tool consistently created tests for negative, zero, and positive values, which helped reveal patterns in the function's behavior. This made the overall testing feel more complete, even though the tool was simple.

### 3.2 Lessons Learned About AI-Assisted Development

One of the main things I learned is that AI tools are really good at doing repetitive tasks, like writing basic tests. Instead of me having to write every single test case manually, the MCP tool could generate them instantly. This saved time and also reduced the chances of forgetting edge cases.

I also learned that AI can help students like me understand testing strategies better. For example, by looking at the structure of the tests the tool created, I was able to see how equivalence classes and boundary analysis are supposed to work in practice. Even though the AI is not perfect, it provides a starting point that I can build on.

### 3.3 Future Enhancements

There are several things I would add to improve this project in the future. First, I would expand the test generator tool so that it can analyze the source code and detect functions automatically, instead of supporting only one function name. I would also add more advanced testing strategies, like decision coverage or mutation testing, to improve fault detection.

Another improvement would be better integration with GitHub Actions. Right now, the MCP agent can run tests locally, but it would be useful if it could also check CI results or automatically update pull requests.

Finally, adding a reporting feature that shows coverage percentages or failed test patterns would make the tool much more helpful for debugging and learning.

## 4 Conclusion

Overall, this project taught me a lot about automating software development using MCP tools. I was able to see how AI can support testing, improve coverage, and streamline common tasks like running Maven or committing changes. Even as a beginner, I felt like I could build useful automation features with relatively simple code. In the future, expanding the agent with more advanced testing and CI integration would make it even more powerful.