Reflection Report: MCP Automated Testing Agent

Leonardo Diaz, Hammy Siddiqui

Abstract

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

1 Introduction

For this assignment, we created a custom MCP (Model Context Protocol) server that connects with VS Code Chat and allows us 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 we are still learning about automation and testing, this project helped us understand how these tools can speed things up and reduce mistakes we would normally make by hand.

2 Methodology

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

To evaluate how useful the generated tests were, we compared the original test suite with the new tests created by the MCP tool. We 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, we 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.

We 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 we learned is that AI tools are effective at handling repetitive tasks, such as writing basic test cases. Instead of us having to manually write each test, the MCP tool could generate them instantly. This saved time and reduced the chances of forgetting important edge cases.

We also learned that AI can help students like us understand testing strategies more clearly. By examining the structure of the tests generated by the tool, we gained a better understanding of how equivalence classes and boundary value analysis work in practice. Even though the AI is not perfect, it provides a starting point that we can improve.

3.3 Future Enhancements

There are several improvements we would like to add in the future. First, we would expand the test generation tool so that it can analyze the source code and detect functions automatically, instead of supporting only one function. We would also include more advanced forms of testing, such as decision coverage or mutation testing, to improve fault detection.

Another helpful improvement would be stronger integration with GitHub Actions. Currently, the MCP agent runs tests locally, but it would be useful if it could also check CI results or automatically update pull requests.

Finally, adding a reporting tool that displays coverage percentages or failed test trends would make the agent even more helpful for debugging and learning.

4 Conclusion

Overall, this project taught us a lot about automating software development using MCP tools. We saw how AI can support testing, improve coverage, and streamline common tasks such as running Maven or committing changes. Even as beginners, we were able to build useful automation features with relatively simple code. In the future, expanding the agent with more advanced testing strategies and CI integrations would make it even more powerful.