

# Reflection Report: MCP Automated Testing Agent

Leonardo Diaz      Hammy Siddiqui

November 17, 2025

## Abstract

This report summarizes our development of an MCP-based automated testing agent that runs Maven tests, executes Git commands, and generates specification-driven JUnit tests. We describe the setup process, interaction with VS Code Chat, coverage improvements observed from generated tests, and the challenges experienced while relying on AI during development. We conclude with lessons learned and recommendations for future enhancements.

## 1 Introduction

Our project involved building a custom Model Context Protocol (MCP) server that integrates with VS Code Chat to automate software development tasks. The agent can run Maven tests, handle Git operations, and create basic JUnit tests based on equivalence-class specifications. Our goal was to understand how automation and AI tools could reduce manual effort and improve testing quality. Working through this project also exposed us to how unpredictable AI-assisted workflows can be when context is unclear or when tasks require multiple steps.

## 2 Methodology

We began by configuring the MCP environment and implementing tools for test execution and Git commands. Afterwards, we added a simple test-generation tool that produced JUnit templates using categories such as negative, zero, and positive inputs. We evaluated the generated tests by running them on a Java project and checking for changes in coverage. All interactions with the agent occurred through VS Code Chat, which allowed us to call tools directly but also introduced challenges when the agent misinterpreted instructions or ended tasks prematurely.

## 3 Results & Discussion

### 3.1 Coverage Improvement Patterns

Even though the test generator was limited, it increased coverage in meaningful ways. The generated tests exercised edge cases like integer extremes and missing equivalence classes, improving both branch and statement coverage in parts of the code not originally tested. The AI consistently

added negative, zero, and positive cases, which expanded the testing surface beyond what was manually included.

## 3.2 Insights from AI-Assisted Development

Working with VS Code Chat taught us that AI tools can speed up repetitive tasks but are extremely fickle. The agent frequently required very specific context, and small phrasing differences caused it to stop early, skip steps, or forget previous instructions. Despite this, the generated tests helped reinforce testing concepts such as boundary analysis and equivalence classes. AI assistance was most useful for producing quick templates and for helping us debug unclear behaviors, though it often needed corrections or clarifications.

## 3.3 Recommendations for Future Enhancements

Future improvements could include automated detection of functions to test, deeper coverage strategies such as mutation or decision coverage, and improved error handling within the MCP tools. Integrating GitHub Actions or CI verification would help connect the agent to real development pipelines. Adding automated reporting tools for coverage trends or repeated failures would also make the agent more practical for debugging.

# 4 Conclusion

Developing the MCP testing agent showed us the strengths and weaknesses of AI-assisted workflows. The automation improved coverage by introducing edge cases and reducing manual testing effort, while also revealing how sensitive AI tools can be to missing context. Overall, the project improved our understanding of testing strategies, automation infrastructure, and the importance of precise instructions when relying on AI tools.