

# AI-driven DevSecOps for Embedded Systems

Rainer Poisel, Stefan Riegler

# Overview

- Introduction (5 Min.)
- Usage Scenarios + Demos (15 Min.)
- Outlook (5 Min.)

# About the Speakers

- Rainer Poisel
  - Embedded Systems DevSecOps
  - Embedded Systems CI/CT/CD
- Stefan Riegler
  - SW / Embedded / IIoT Security
  - Reverse Engineering / Forensics



Process Automation, Security Testing  
(IEC 62443, CRA)

# Introduction

- **Original-Title:**

Secure by Design: Wie KI und DevSecOps die Embedded Systems-Entwicklung verändern

# Enter AI-DevSecOps (I)

## Definition of DevOps:

DevOps combines development (Dev) and operations (Ops) to **unite** people, process, and technology in application planning, development, delivery, and operations.

– Source: Microsoft

# Enter AI-DevSecOps (II)

## Definition of DevSecOps:

DevSecOps, which stands for development, security, and operations, is a framework that **integrates** security into all phases of the software development lifecycle.

– Source: Microsoft

# Enter AI-DevSecOps (III)

Our Stack: **LLM + MCP + pytest + labgrid**

- Automated vulnerability analysis
- Intelligent test generation
- Distributed hardware testing

# Motivational Example (I)

- No-Code Security
  - Inspired by “No-Code Development”
  - Security Audits and System Analyses w/o Coding
  - Empowering non-technical Stakeholders
  - Proactive Security Culture

# Motivational Example (II)

- Product Certification Process
  - IEC 62443
  - CRA / EUCC

# Large Language Models (LLMs)

- Most used type of a probabilistic generative model
- Can understand & generate:
  - natural language,
  - code, and
  - structured data.
- Key enabler for **automation in DevSecOps**

# Model Context Protocol (MCP)

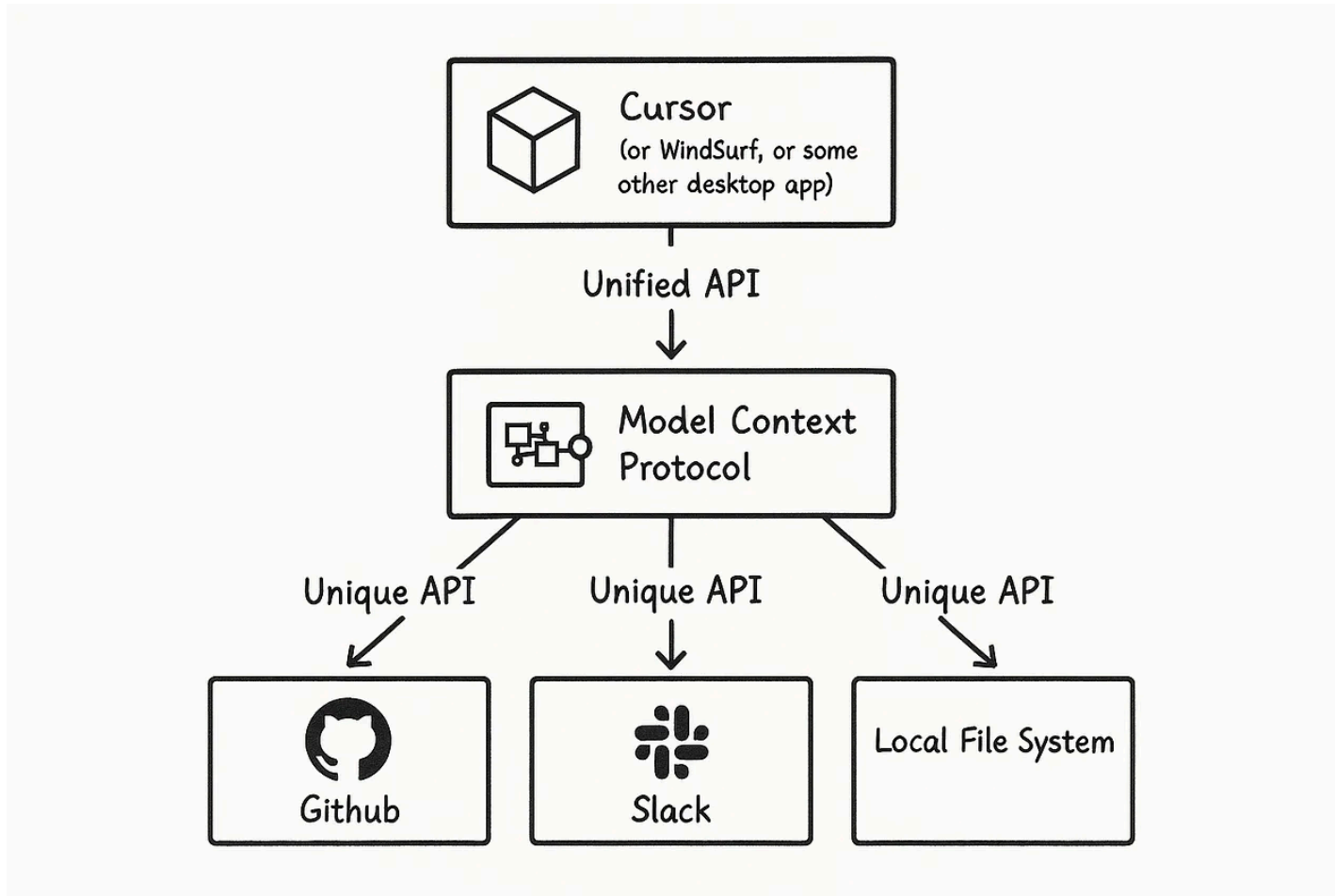


Figure 1: MCP Overview (Addy Osmani - MCP: What It Is and Why It Matters)

# Model Context Protocol (MCP)

- Open protocol to connect LLMs with tools, data & workflows
- Think of it as “**IPC for AI systems**”
- Ensures reproducibility, interoperability, and integration into pipelines

Note: Still a long way to go in comparison to IPC (authorization, versioning, etc.)

# What We Demonstrate Today

- **pytest**: automated testing framework
- **labgrid**: hardware abstraction and device management
- **MCP**: communication with external services
  - Issue Tracker
  - Execution Environment / DUTs
  - Code Repository

# Test Framework

# pytest

- Python-based
- 13.1k Stars on GitHub
- Well-known to LLMs
- Easy to use

# labgrid (I)

- Local/Distributed Infrastructure
- Pytest Integration
- CLI/Library Usage
- Drivers
- Strategies
- Virtualization Support



# labgrid (II)

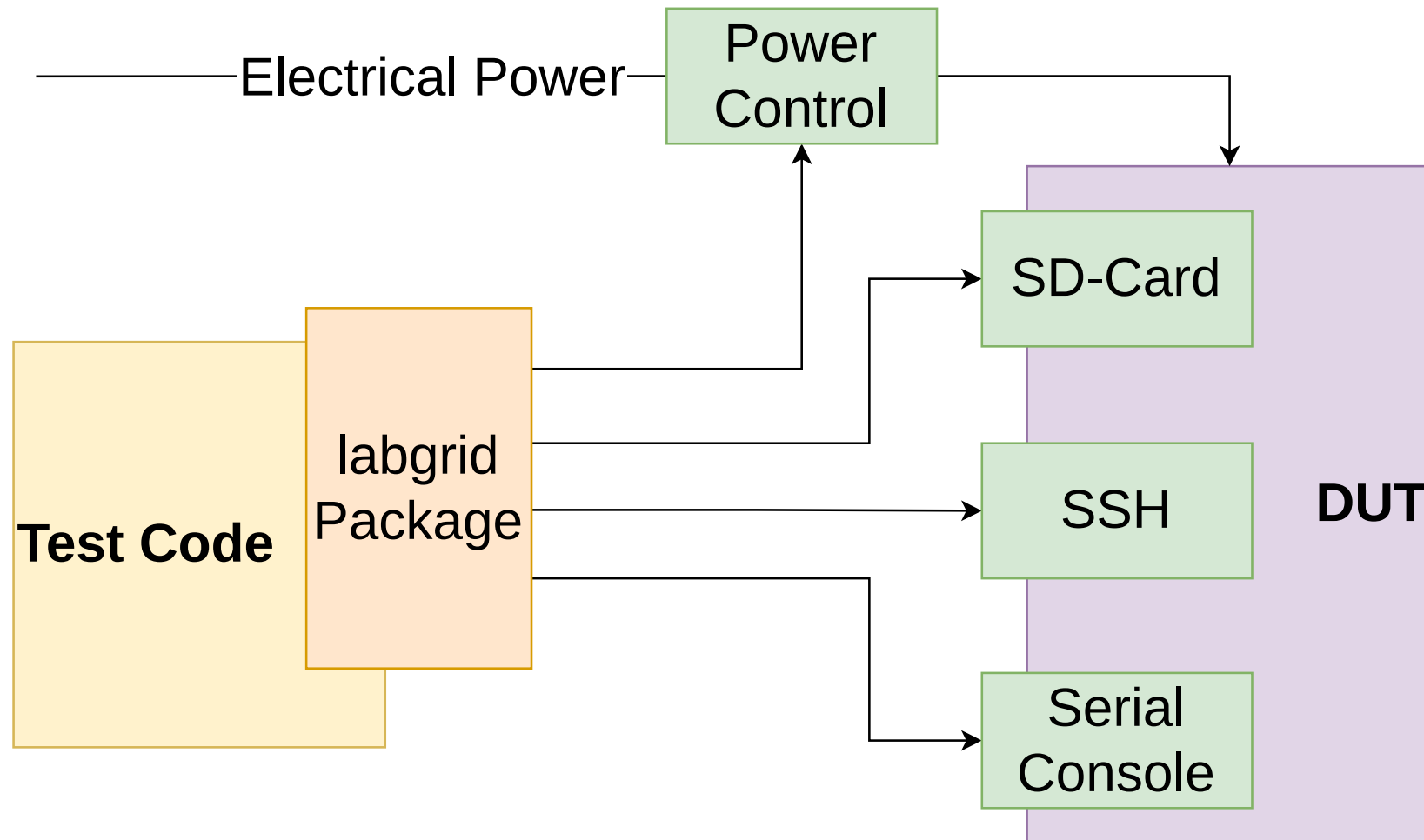


Figure 2: Labgrid Local Architecture

# labgrid (III)

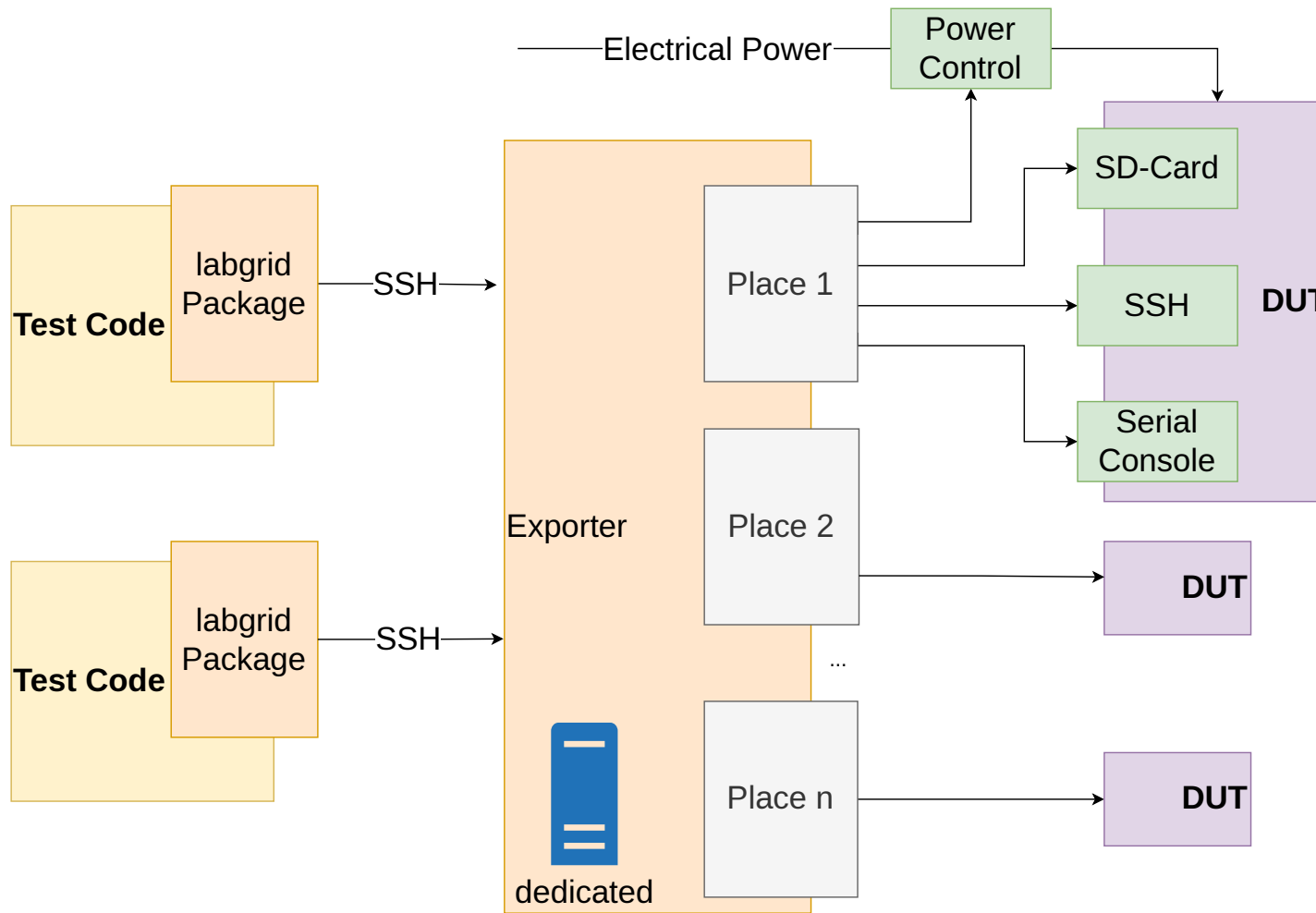


Figure 3: Labgrid Distributed Architecture

# labgrid: Example (I)

**Task:** Check that `uname -s` returns the string `Linux`

```
1 from labgrid.util.ssh import SSHConnection
2
3 def test_uname(ssh_cmd: SSHConnection) -> None:
4     stdout = ssh_cmd.run_check("uname -s")
5     assert stdout == "Linux"
```

Figure 4: Simple pytest/labgrid sample

# labgrid: Example (II)

Not in the code:

- Power on
- Pass bootloader
- Wait for login
- Wait for shell
- Configure DHCP client (if required)
- Determine IP

# Automatic Test Case Generation

# Components

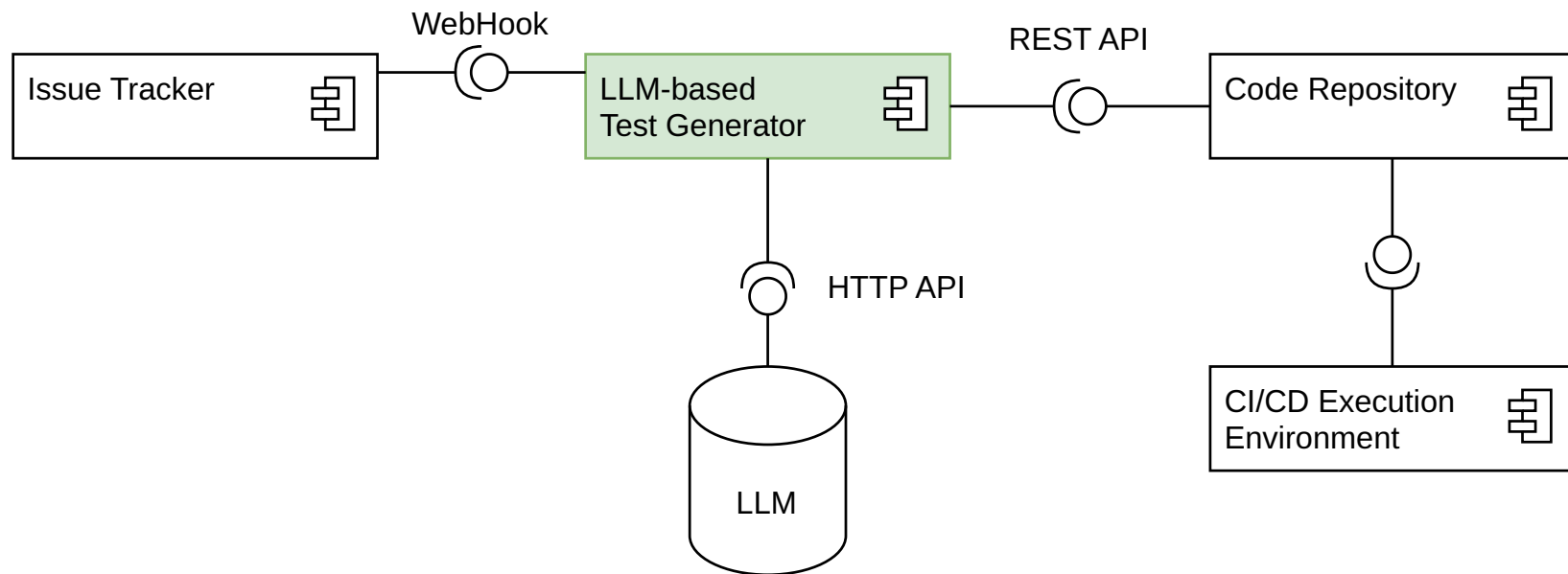


Figure 5: Components Diagram

# Prepared Command Prompt (I)

```
1 Generate a pytest from the following description in markdown format:
2 ```markdown
3 {description}
4 ```
5
6 Terminology:
7
8 < ... >
9
10 Important:
11
12 < ... >
13
14 Code Guidelines:
15
16 < ... >
```

Figure 8: Prepared Command Prompt

# Prepared Command Prompt (II)

- **Terminology:** Abbreviations
- **Important:**
  - High-Level Test Code Requirements
  - General Structure of generated Code
- **Code Guidelines:**
  - Description of the test infrastructure
  - Libraries/Techniques to follow/avoid

# Large Language Model

- Open-Source Models
  - Self-Hosted: Privacy
  - Hosted: Low computing resources available
- Models evaluated:
  - `meta/meta-llama-3.1-405b-instruct`
  - `meta/meta-llama-3-70b-instruct`

# Software Engineering and Architecture Roles

- System/Software Architect
- Security Architect
- Software Developer
- Quality Assurance:
  - Framework/Library Developer
  - Test Developer

# Demo: The Test Case

- Determine Listening TCP Ports
  - Expected Listening Ports: [22, 80, 443]
- Determine Listening UDP Ports
  - Expected Listening Ports: []

# Demo: Workflow

# MCP: Use-Cases

# MCP Labgrid Server: Components

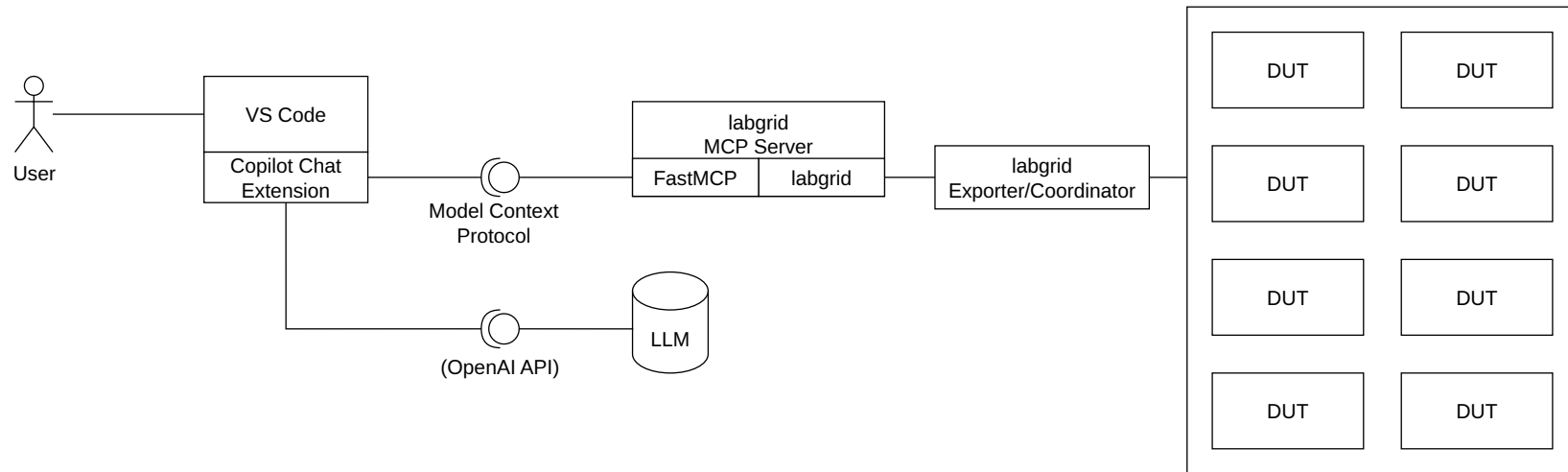


Figure 9: Components Diagram

# MCP Labgrid Server: Sequence

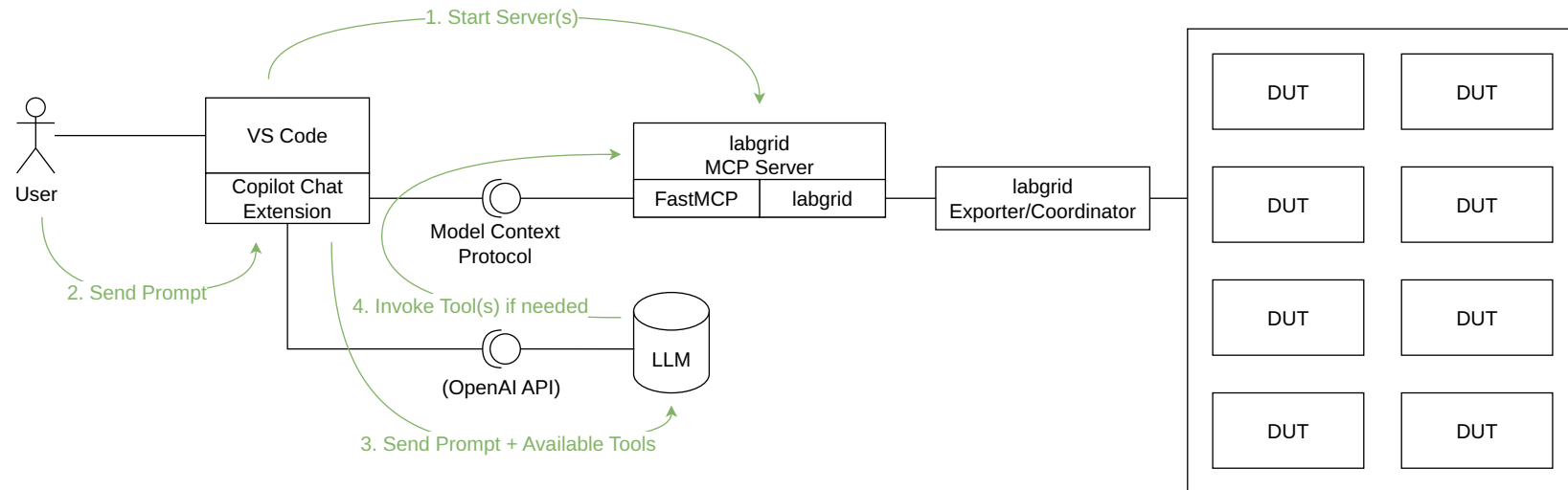


Figure 10: Components Diagram with Sequence

# MCP Labgrid Server: Demo

# Future Vision

# Agent Communication Protocol (ACP)

- Standardizes communication between AI agents and clients
- Complements MCP by focusing on agent orchestration & interoperability
- Enables multi-agent, multi-stage DevSecOps workflows

# Test Generation: Feedback Loop

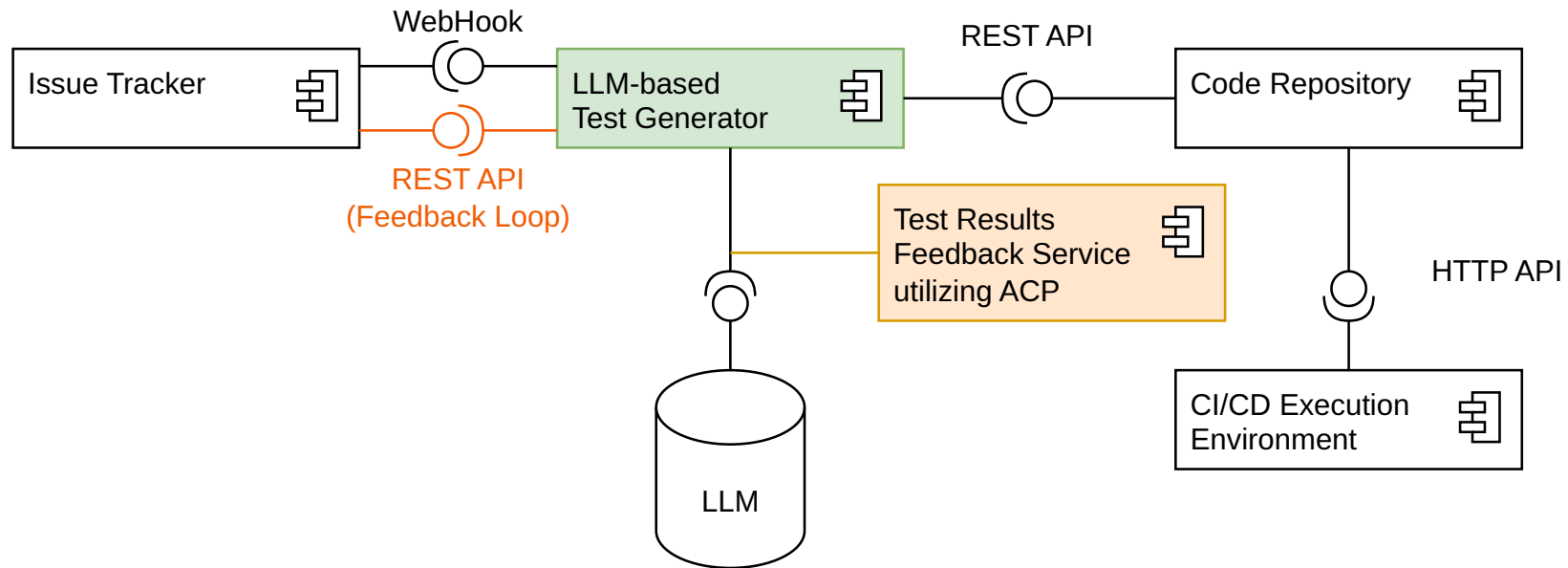


Figure 11: Components with Feedback Loop

# Call to Action

- Say Hi 🙌
- Connect on LinkedIn:
  - [in/rainer-poisel](#)
  - [in/stefan-riegler-oe3sha](#)

# Addendum

# References (I)

## **pytest**

<https://pytest.org>

## **labgrid**

<https://pengutronix.de/en/software/labgrid.html>

## **labgrid QEMU Sample**

<https://github.com/Embedded-Focus/labgrid-qemu-sample>

# References (II)

## **Model Context Protocol (MCP)**

<https://modelcontextprotocol.io/>

## **FastMCP**

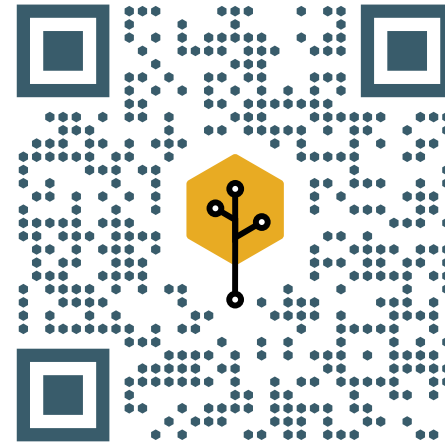
<https://gofastmcp.com/>

## **Agent Communication Protocol (ACP)**

<https://agentcommunicationprotocol.dev/>

# Our Services

- Embedded CI/CD
- Software Development
- DevSecOps Trainings
- Security Audits
- Reverse Engineering
- Forensics
- Automation
- Modernization
- Supply Chain Security



# Backup Slides

# AI-based Log File Analysis

# Retrieval Augmented Generation

**Use-Case:** Talk to your log files.

1. **Retrieve:** Find relevant log entries
2. **Augment:** Add context to prompt
3. **Generate:** Use LLM to generate human-readable answer

# Retrieval Augmented Generation

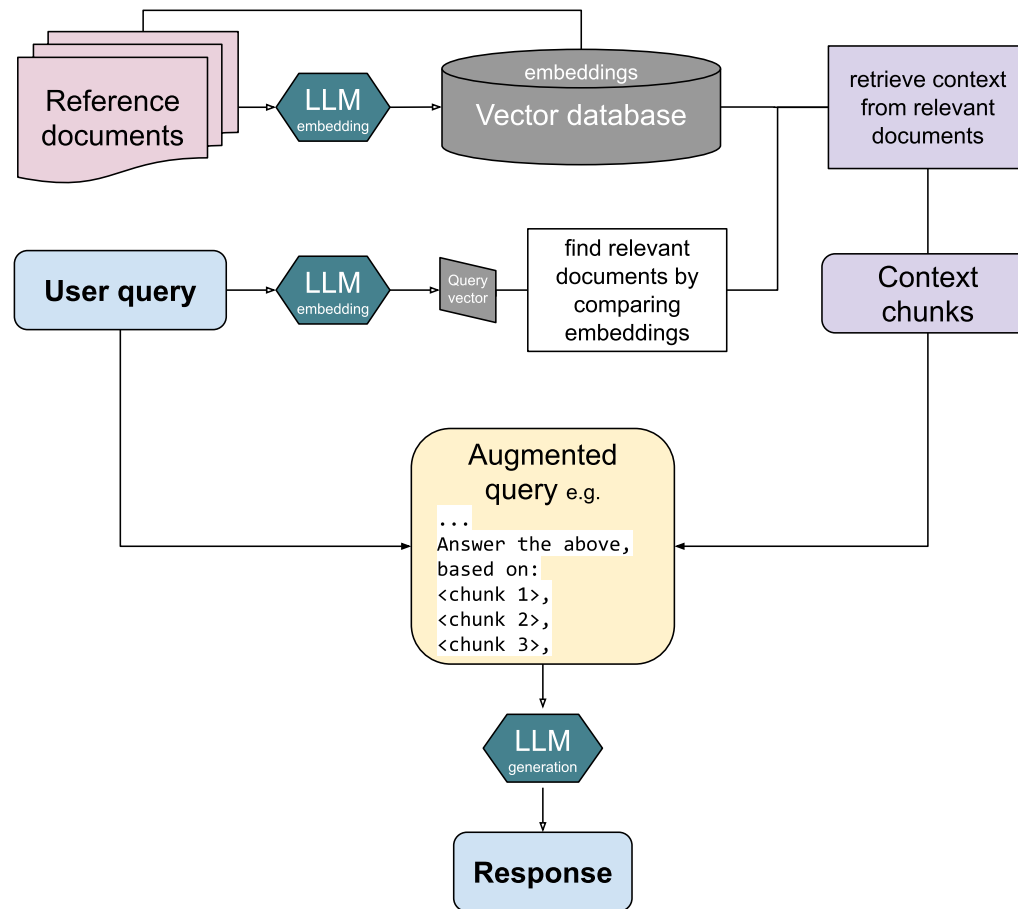


Figure 12: RAG Concept (Wikimedia Commons)

# Pattern Matching: Supervised Learning

## Use-Case:

Classify test run log files by **known** error conditions.

1. **Collect:** Log files from previous test runs
2. **Classify and Learn:** Determine causes of failures
3. **Apply:** Classify new log files