

Agent-Driven DevSecOps: Transforming Embedded Software Development

Rainer Poisel

DevSecOps

Embedded Focus / honeytreesLabs

St. Poelten, Austria

hello@embedded-focus.com

Abstract—Traditional monolithic AI coding assistants are IDE-locked, vendor-specific, and lack auditability which are critical shortcomings for compliance-driven workflows. This paper presents a multi-agent development infrastructure based on open standards: the Agent Client Protocol (ACP) for agent orchestration, the Model Context Protocol (MCP) for tool integration, and Agent Skills for domain expertise. Each agent runs in an isolated container with restricted filesystem and network access, ensuring security and auditability. As an example, we present our Labgrid MCP server which provides controlled access to embedded devices from our AI-driven infrastructure, enabling faster human-in-the-loop failure triage. This protocol-driven approach reflects a shift from ad-hoc assistant usage to workflows where multiple specialized agents can take over dedicated tasks. It enables agent specialization while maintaining vendor independence, with a roadmap toward agent-to-agent collaboration via the Agent2Agent (A2A) protocol.

Embedded systems development faces unprecedented complexity as industry standards and regulations demand increasingly rigorous security validation. The EU Cyber Resilience Act (CRA) [18] and IEC 62443 [23], [24] require dedicated workflows to ensure system safety, security, and resilience. Traditional manual CI/CD maintenance struggles to keep pace with these demands, creating bottlenecks in the software development life cycle (SDLC) that slow adaptation to evolving requirements. To address these challenges, this paper builds on several foundational concepts and emerging standards. The following paragraphs explain these concepts.

DevOps emerged to bridge the gap between development and operations through automation, collaboration, and continuous delivery. According to Ebert and Serrano, DevOps emphasizes rapid, reliable software delivery by integrating agile development, infrastructure as code, automated testing, and continuous monitoring [22]. DevSecOps extends this paradigm by embedding security practices throughout the SDLC rather than treating them as an afterthought [30]. A key principle is “shift security left” which addresses security concerns during test preparation and development, not just at deployment [4]. This approach aligns directly with IEC 62443’s mandate for

structured, repeatable validation steps integrated into development workflows.

Large Language Models (LLMs) are increasingly used to automate parts of these workflows. LLMs are large neural language models trained on massive corpora (often with next-token prediction objectives), which enables them to understand and generate natural language, code, and structured data [26], [33]. When augmented with tool access and decision-making capabilities, LLMs can act as *AI agents* that interact with systems and execute multi-step workflows rather than merely generating text.

However, current AI coding assistants such as GitHub Copilot and Cursor are often IDE-locked and vendor-specific, lacking standardized interfaces for system integration. This creates several limitations: vendor lock-in prevents teams from choosing a suitable agent for each task, lack of auditability makes it difficult to trace agent actions for compliance, inability to collaborate means multi-agent scenarios require custom integrations, and reproducibility suffers without standard protocols for tool access.

This protocol-driven approach reflects a shift in developer-AI collaboration. Our architecture treats agents as constrained collaborators, each running in an isolated container with explicit network and filesystem boundaries, communicating through standardized protocols that enable auditing, reproducibility, and vendor independence. This framing positions AI assistance closer to a “transparent partner” operating under controlled conditions that aim to satisfy compliance requirements.

This paper makes the following contributions:

- 1) A containerized multi-agent development infrastructure using ACP, MCP, and Agent Skills, with container isolation, filesystem restrictions, and audit trails for compliance-driven workflows.
- 2) A labgrid [7] MCP server implementation that enables AI-assisted CI/CD failure triage with controlled embedded device access as an example for how to integrate additional tools into the system.

- 3) Practical lessons learned and a roadmap toward collaborative multi-agent scenarios using the Agent2Agent (A2A) protocol.

Section I presents background on DevSecOps compliance automation and introduces the open AI protocols (ACP, MCP, Agent Skills, A2A) that form the foundation of our approach. Section II presents our multi-agent infrastructure and practical applications. The last part concludes with lessons learned and future directions.

I. BACKGROUND AND OPEN STANDARDS

This section provides the conceptual foundation for the approach presented in the remainder of the paper. It briefly introduces the open AI protocols that enable a multi-agent, tool-integrated workflow and frames why they matter for embedded DevSecOps. The goal is to establish context for the architecture and use cases described in Section II.

A. Agent Client Protocol (ACP)

The Agent Client Protocol [14] enables IDE-independent orchestration of multiple AI agents within a single development environment. ACP addresses fragmentation by providing a common interface between editors and agents, analogous to how the Language Server Protocol (LSP) standardized language server integration. This allows any ACP-compliant agent to work with any ACP-compatible editor, reducing redundant integration work and vendor lock-in.

For DevSecOps, ACP's key value is agent specialization: different agents can be selected for different task types, such as architectural analysis, firmware debugging, security review, or test generation. Each agent runs in an isolated environment and communicates through a uniform protocol, allowing teams to choose the best agent per workflow step without switching tools or contexts. This modularity directly supports the multi-agent use cases described later.

B. Model Context Protocol (MCP)

The Model Context Protocol [17] provides a standardized interface for AI agents to access tools, data sources, and external systems. In embedded DevSecOps, MCP enables agents to interact with hardware test infrastructure, version control systems, and build environments in a type-safe, auditable manner. Conceptually, MCP is comparable to remote procedure calls (RPCs), but procedures are invoked by language models based on context rather than by software components. Industry interest in these standards is reflected by the Thoughtworks Technology Radar Vol. 33 [31], which places MCP and AI coding tools such as Claude Code in the Trial ring as technologies to consider for projects.

For our use case, MCP provides a standardized interface to expose existing labgrid capabilities to AI agents via tools, without granting arbitrary shell or SSH access. Our implementation relies on the official labgrid APIs and adds a thin adapter layer between labgrid and FastMCP [5], enabling access with auditability that builds the foundation of the labgrid MCP server described later.

C. Agent Skills

Agent Skills [16] are modular capability packages that extend AI agent functionality through bundled instructions, metadata, and optional resources such as scripts or templates. Unlike MCP and ACP, which define how agents communicate, Agent Skills define what agents can do by encoding domain expertise and organizational workflows in reusable, version-controlled formats.

In embedded DevSecOps, Skills package compliance knowledge (e.g., IEC 62443 or CRA checklists), secure coding guidelines, or standardized test procedures so agents can apply consistent workflows across projects. In our infrastructure, Skills complement ACP and MCP by providing the domain procedures that agents invoke during CI/CD triage and compliance-related tasks described later.

D. Agent2Agent Protocol (A2A)

The Agent2Agent Protocol [25] is an open standard for communication and collaboration between AI agents. Unlike ACP, which focuses on human-to-agent orchestration, A2A enables agents to discover each other and delegate subtasks securely. A2A complements MCP: MCP standardizes agent-to-tool access, while A2A standardizes agent-to-agent collaboration.

For embedded DevSecOps, A2A enables automated delegation: an orchestrator agent can hand off security analysis, test generation, or infrastructure diagnostics to specialized agents without exposing sensitive internals. This capability motivates the multi-agent collaboration use case discussed later. Together with ACP, MCP, and Agent Skills, A2A contributes to a flexible automation layer on top of established tools such as pytest [10] and labgrid [7], allowing AI agents to assist with code changes, log analysis, and other error-prone tasks in a controlled workflow.

II. USE CASES

This section presents three use cases that demonstrate the practical application of the protocols introduced in Section I-A. First, we present our agent-driven development infrastructure where containerized agents collaborate through ACP, accessing tools via MCP and leveraging Agent Skills for domain expertise. Second, we describe a labgrid MCP server implementation that provides AI agents with controlled access to embedded hardware test infrastructure. Third, we explore how the Agent2Agent protocol can automate inter-agent collaboration, addressing the limitation that developers must currently select agents manually for each task.

A. Agent-Driven Development Infrastructure

This section presents our multi-agent infrastructure for embedded DevSecOps, demonstrating how the protocols introduced in Section I enable practical, security-focused automation workflows. Figure 1 provides an overview of how the aforementioned protocols and components relate to each other.

The IDE acts as the client side of the ACP connection. Editors such as Zed, Neovim, and JetBrains IDEs natively

support ACP [2], while Emacs users can connect to agents via the agent-shell package [32]. VS Code uses a proprietary communication protocol and does not natively support ACP, though an open GitHub issue requests this functionality [12]. VS Code users can alternatively build on the editor’s native agent capabilities [11] or enable ACP through a community extension [27], though the latter is not officially supported by Microsoft.

On the agent side, developers can achieve ACP support either through extensions to agent CLIs such as the ACP adapter for Claude Code [3] and for OpenAI Codex [9], or through agents that natively implement the protocol such as Vibe CLI [15] and Gemini CLI [6]. ACP adapters are typically built on the AI agents vendors’ software development toolkits (SDKs) for their models. The Agent Client Protocol project website maintains a comprehensive list of compatible agents [1].

The ACP protocol interface also serves as the boundary between the IDE running natively and the agents running in containers. Container isolation provides several security benefits critical for DevSecOps: network isolation in combination with customized firewall rules prevents agents from accessing external services or exfiltrating code; filesystem restrictions ensure only the workspace directory is mounted, protecting host system files and credentials; comprehensive audit trails log all agent actions via the ACP protocol for compliance reviews; and the principle of least privilege ensures each container has only the minimal permissions required for its tasks.

Within these containers, agents access tools through the Model Context Protocol (MCP) and utilize Agent Skills to encode domain-specific workflows. Currently, Agent Skills

implementations vary by vendor, though standardization efforts are underway [16].

For embedded DevSecOps, MCP servers provide low-level access to development infrastructure and tooling. A labgrid MCP server exposes hardware test capabilities such as device power control, serial console access, and firmware deployment to devices under test. A git MCP server enables agents to query version history, generate diffs, and perform blame analysis for root cause investigation during CI/CD failures. A build system MCP server grants access to compilation logs, linker maps, and build artifacts, allowing agents to correlate test failures with recent changes or identify memory layout issues.

At a higher abstraction level, Agent Skills encode domain expertise and multi-step workflows as reusable instruction sets. A secure coding guidelines skill packages CERT C and MISRA C rules with violation detection patterns, enabling agents to identify unsafe constructs during code review. A firmware update security skill provides templates for implementing cryptographic signature verification and rollback protection mechanisms compliant with IEC 62443 SR 3.4. A CI/CD debugging skill encodes systematic failure analysis procedures, guiding agents through log correlation, environment validation, and regression identification workflows.

Figure 2 illustrates a practical application of these protocols and components, showing the relationship between the IDE and multiple agents accessing a shared source repository. We implemented this architecture in Agent Circus [28], an open-source repository containing containerized configurations for Claude Code, OpenAI Codex, and Mistral Vibe agents.

Using container orchestration with DevContainers [20] and

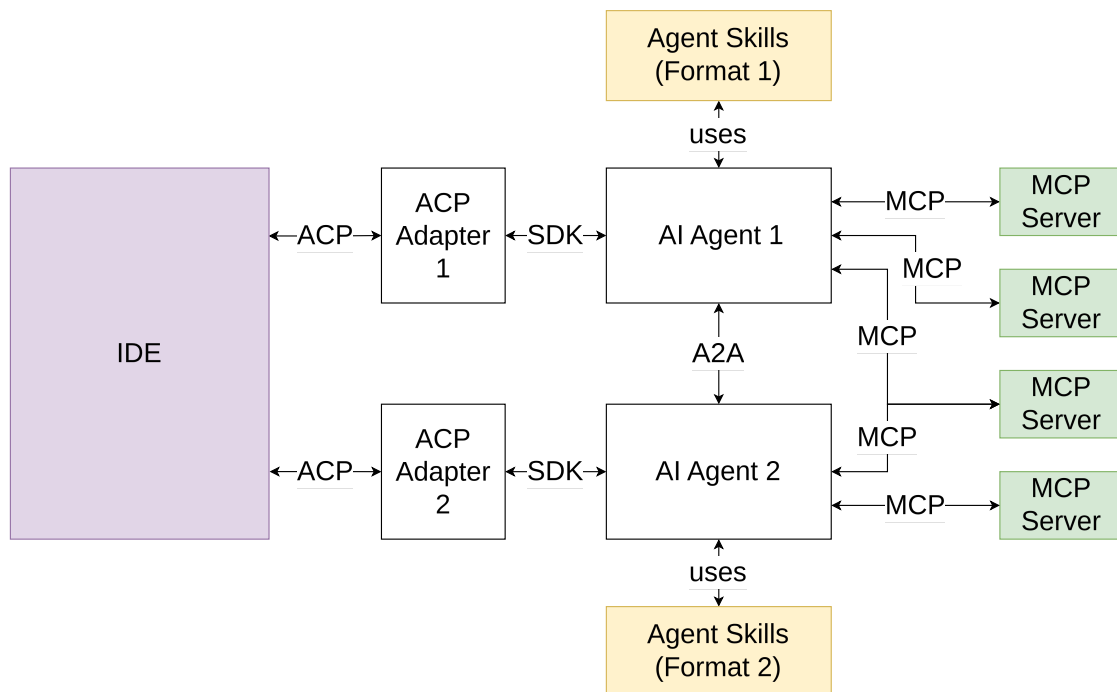


Fig. 1. AI Architecture Using Open Standards for Artificial Intelligence

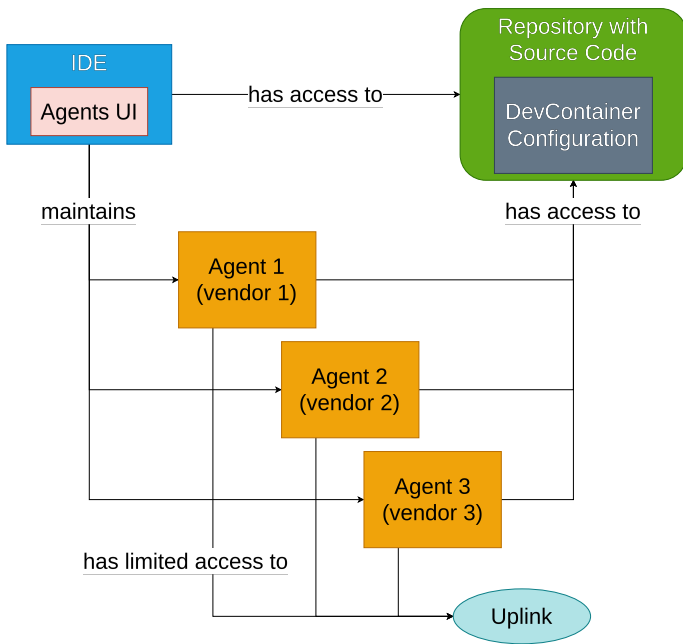


Fig. 2. Containerized multi-agent architecture with ACP-based IDE integration, showing isolated agent environments with controlled network and filesystem access.

the DevContainer CLI [19], each agent runs in a dedicated service within a Docker Compose [21] setup. Agents are isolated in Docker containers with `NET_ADMIN` capabilities to enforce per-container firewall rules (e.g., via `iptables` and `ipset` whitelists), while bind-mounted workspace directories provide source access without exposing host system files. This setup has been used in daily development workflows, enabling switching between agents depending on task requirements such as architectural analysis with Claude Code, boilerplate generation with Codex, while maintaining security boundaries.

The ACP-based multi-agent architecture enables task-specific agent selection. In our experience, Claude Code works well for analyzing codebases and suggesting structural improvements, making it suitable for architectural refactoring tasks. OpenAI Codex has been effective for generating repetitive test fixtures and configuration code in our workflows, while future specialized agents may focus on vulnerability detection and compliance validation. Developers can thus choose an appropriate agent for each task without changing tools or workflows, avoiding the limitations of monolithic assistants. A current limitation of this architecture is that developers must manually select which agent to invoke for each task. The emerging Agent2Agent (A2A) protocol [25] addresses this by enabling agents to discover each other’s capabilities and delegate subtasks autonomously, potentially removing the need for human orchestration in routine workflows.

B. Test Infrastructure Access with a Labgrid MCP Server

Labgrid [13] is a distributed embedded testing framework that manages hardware resources through a central coordinator. It supports power control, serial console interaction, and de-

vice bootstrapping across multiple test stations. To enable AI agents to interact with this infrastructure, we implemented an MCP server [29] using FastMCP [5] that exposes four tools: `get_places` retrieves available test hardware; `acquire` reserves a specific place for exclusive access; `release` returns the place to the pool; and `shell_cmd` executes commands on the acquired device. The MCP Inspector [8] facilitated development and debugging through an interactive interface for testing tool invocations before connecting actual AI agents.

This integration allows AI agents to assist in diagnosing problems on devices under test such as analyzing network connectivity issues, investigating boot failures, or troubleshooting peripheral malfunctions which results in reducing the need for engineers to manually enter commands. Agents can correlate system logs with recent code changes to identify root causes of test failures, run benchmarks, and verify firmware versions across multiple devices. A developer can ask “What is the uptime of my DUT?” and the agent invokes the appropriate tool or command, parses the response, and presents the result in human-readable form.

In a recent CI failure, the test run broke due to a device infrastructure issue rather than a code regression. Previously, triage required manual log inspection, online research for labgrid commands, and interactive investigation on the DUT, taking roughly 15 minutes. With the MCP server in place, the developer copied the CI error message into an LLM and asked it to investigate the DUT directly. The agent invoked labgrid tools through MCP, discovered a missing physical uplink, and summarized the cause. The analysis took about 2 minutes with no further manual interaction. This case provides an anecdotal reduction in diagnostic time while preserving controlled access and auditability. Future work is to automate CI log analysis and trigger such device investigations without manual prompting.

Beyond time savings, the workflow change is qualitative: agents can act as force multipliers when developers can delegate well-scoped tasks and critically evaluate outcomes. Experienced embedded engineers can offload repetitive triage steps, device interrogation, or compliance checks while retaining responsibility for judgment and safety, while developers with limited domain knowledge may struggle to frame effective tasks or validate results, increasing the risk of incorrect conclusions. This shifts the skill set required for embedded DevSecOps toward precise task decomposition, verification discipline, and system-level understanding alongside traditional coding skills, and it can manifest in faster incident response through direct device inspection, reduced context switching by delegating routine diagnostic steps to agents, and improved auditability because tool access is mediated and logged via MCP. Together, these changes can move embedded DevSecOps from ad hoc, engineer-driven investigation toward a more repeatable, agent-augmented workflow.

A potential extension is automated CI/CD failure triage. When a test run fails, an agent could analyze build logs to classify the failure as infrastructure-related, test code defect, or actual regression in the tested subject. For infrastructure

failures, such as unresponsive devices or network timeouts, the CI system can keep the DUT reserved and hand it over to an agent with context about the suspected failure mode. The agent then uses the MCP server to interact with the device directly: checking system logs, verifying network connectivity, or inspecting peripheral states to diagnose the root cause before an engineer needs to intervene.

The resource acquisition mechanism ensures exclusive access to hardware, preventing interference between concurrent users or agents. All commands executed through the MCP server are logged with timestamps and correlated to specific tasks, creating audit trails suitable for compliance workflows. This combination of controlled access and comprehensive logging makes the server appropriate for regulated environments where traceability is essential.

C. Automating Agent Collaboration with the Agent2Agent Protocol

The infrastructure described in the previous subsection requires developers to manually select which agent to invoke for each task. While this provides control and predictability, it also creates overhead and limits the potential for complex automated workflows. The Agent2Agent (A2A) protocol [25] addresses this limitation by enabling agents to discover each other's capabilities and delegate subtasks autonomously.

A2A introduces a standardized mechanism for agent interoperability. Each agent publishes an AgentCard that advertises its capabilities, authentication requirements, and available skills. When an orchestrator agent receives a complex task, it can query available agents, identify specialists suited for subtasks, and delegate work without human intervention. Consider a task such as implementing a secure firmware update feature compliant with IEC 62443 SR 3.4. An orchestrator agent could delegate security requirement analysis to a compliance specialist, receive back specific requirements like cryptographic signature verification and rollback protection, then delegate test generation to a testing specialist and implementation to a firmware development agent. The orchestrator coordinates iteratively until tests pass and compliance is verified.

This collaborative model enables meaningful agent specialization for DevSecOps roles. A security auditor agent could continuously monitor code changes for compliance violations, utilizing static analysis tools via MCP and flagging issues before CI runs complete. A test generator agent could specialize in creating hardware test cases from natural language requirements or security specifications. A feature implementer agent could write firmware code following secure coding guidelines encoded in Agent Skills. An infrastructure agent could manage test equipment availability and CI/CD pipeline health, automatically recovering from common failures. Each specialist maintains access to relevant MCP servers and Skills while communicating via A2A to coordinate complex workflows.

One potential application is agent-orchestrated test-driven development for embedded security features. A test developer agent writes failing security tests based on compliance requirements and verifies they fail on hardware via the labgrid MCP

server. It then delegates implementation to a feature implementer agent, which writes code and commits changes. The test developer agent re-runs tests, providing feedback if failures remain. This cycle continues until all tests pass, helping ensure security requirements drive implementation while maintaining human oversight through git-based code review before merge.

Infrastructure self-healing represents another A2A application. When CI jobs fail due to environmental issues such as offline test devices or network timeouts, an infrastructure agent can detect the failure pattern, diagnose the root cause, and either resolve it automatically or escalate to human operators with detailed diagnostics. This reduces the burden on engineers to investigate transient infrastructure problems and keeps development pipelines flowing.

CONCLUSION

This paper presented a multi-agent development infrastructure for embedded DevSecOps based on open AI standards. Our containerized architecture builds on the Agent Client Protocol (ACP) for agent orchestration, the Model Context Protocol (MCP) for hardware test infrastructure integration, and Agent Skills for packaging domain expertise such as IEC 62443 and CRA compliance patterns.

The key contributions of this work are: (1) a security-focused multi-agent architecture with container isolation and network restrictions; (2) practical AI-assisted CI/CD failure triage using a labgrid MCP server that provides controlled access to embedded devices under test; (3) a demonstration of agent specialization in our setup, enabling teams to select appropriate AI agents for different task types without vendor lock-in; and (4) a roadmap toward agent-to-agent collaboration via the Agent2Agent (A2A) protocol.

Based on our implementation experience, we recommend teams begin with MCP for tool integration before adopting more complex multi-agent workflows. Containerizing agents early establishes security boundaries, while comprehensive logging of agent actions is valuable for both debugging and compliance certification. Human oversight remains essential. Agents should suggest rather than autonomously execute changes to production code or hardware configurations. Our labgrid MCP integration highlights an area requiring further work: agents with broad access to test infrastructure can potentially affect systems beyond their intended scope. Mitigating such agent breakout scenarios through fine-grained capability restrictions and sandbox enforcement remains an open challenge for future research.

Looking ahead, A2A integration could enable collaborative multi-agent workflows where specialized agents coordinate on complex tasks. Future directions include self-healing CI/CD pipelines, agent-orchestrated test-driven development for security features, and cross-project Agent Skill libraries that capture organizational security expertise in reusable, version-controlled formats. By treating agents as constrained collaborators rather than opaque tools, the open standards approach presented here provides embedded development teams with a practical

path to AI-assisted DevSecOps without sacrificing security, auditability, or flexibility.

REFERENCES

- [1] Agent Client Protocol Agents. <https://agentclientprotocol.com/overview/agents>. Accessed: 2026-01-11.
- [2] Agent Client Protocol Clients. <https://agentclientprotocol.com/overview/clients>. Accessed: 2026-01-11.
- [3] Claude Code ACP Extension. <https://github.com/zed-industries/claude-code-acp>. Accessed: 2026-01-11.
- [4] DevSecOps Best Practices: A Practical Guide. <https://www.kroll.com/en/insights/publications/cyber/devsecops-best-practices-a-practical-guide>. Accessed: 2025-01-23.
- [5] FastMCP. <https://gofastmcp.com/>. Python framework for building MCP servers. Accessed: 2026-01-11.
- [6] Gemini CLI. <https://geminicli.com/>. Accessed: 2026-01-11.
- [7] Labgrid – Pengutronix. <https://pengutronix.de/de/software/labgrid.html>. Accessed: 2026-02-01.
- [8] MCP Inspector. <https://modelcontextprotocol.io/docs/tools/inspector>. Accessed: 2026-01-11.
- [9] OpenAI Codex ACP Extension. <https://github.com/zed-industries/copex-acp>. Accessed: 2026-01-11.
- [10] pytest: helps you write better programs. <https://pytest.org>. Accessed: 2025-01-26.
- [11] Using Agents in Visual Studio Code. <https://code.visualstudio.com/docs/copilot/agents/overview>. Accessed: 2026-01-11.
- [12] VS Code GitHub Issue: Request for ACP Support. <https://github.com/microsoft/vscode/issues/265496>. Accessed: 2026-01-11.
- [13] Welcome to labgrid. <https://labgrid.org>. Accessed: 2025-01-26.
- [14] Agent Client Protocol Community. Agent Client Protocol: Standardized Editor-Agent Interface. <https://agentclientprotocol.com/>, 2025. Accessed: 2026-01-17.
- [15] Mistral AI. Vibe CLI. <https://github.com/mistralai/mistral-vibe>. Accessed: 2026-01-11.
- [16] Anthropic. Agent Skills: Open Standard for Modular AI Capabilities. <https://agentskills.io/>, 2025. Accessed: 2026-01-17.
- [17] Anthropic. Model Context Protocol: Universal Standard for AI-System Integration. <https://modelcontextprotocol.io/>, 2025. Accessed: 2026-01-17.
- [18] European Commission. Regulation of the European Parliament and of the Council on horizontal cybersecurity requirements for products with digital elements and amending Regulation (EU) 2019/1020. <https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=celex:52022PC0454>, 2022.
- [19] Development Containers Community. Dev Container CLI: Reference Implementation for the Dev Container Spec. <https://github.com/devcontainers/cli>, 2025. Accessed: 2026-01-17.
- [20] Development Containers Community. Development Containers: An Open Specification for Enriching Containers with Development-Specific Content. <https://containers.dev/>, 2025. Accessed: 2026-01-17.
- [21] Docker, Inc. Docker Compose: Multi-Container Application Definition and Orchestration. <https://docs.docker.com/compose/>, 2025. Accessed: 2026-01-17.
- [22] Christof Ebert, Gorka Gallardo, Josune Hernantes, and Nicolas Serrano. DevOps. *IEEE software*, 33(3):94–100, 2016.
- [23] International Electrotechnical Commission (IEC). IEC 62443 Series: Industrial communication networks - Network and system security. Standard, 2018. Available from <https://webstore.iec.ch/>.
- [24] ISA Global Cybersecurity Alliance. Quick Start Guide: An Overview of ISA/IEC 62443 StandardsGlobal Cybersecurity Alliance. <https://gca.isa.org/hubfs/ISAGCA%20Quick%20Start%20Guide%20FINAL.pdf>.
- [25] Linux Foundation. Agent2Agent Protocol (A2A): Open Standard for AI Agent Interoperability. <https://a2a-protocol.org>, 2025. Accessed: 2026-01-17.
- [26] Shervin Minaee, Tomas Mikolov, Narjes Nikzad, Meysam Chenaghlu, Richard Socher, Xavier Amatriain, and Jianfeng Gao. Large language models: A survey. *arXiv preprint arXiv:2402.06196*, 2024.
- [27] Omercnet. VSCode ACP Extension. <https://marketplace.visualstudio.com/items?itemName=omercnet.vscode-acp>. Accessed: 2026-01-11.
- [28] Rainer Poisel. Agent Circus: Containerized Multi-Agent Development Infrastructure. <https://codeberg.org/Embedded-Focus/agent-circus>, 2025. Accessed: 2026-01-30.
- [29] Rainer Poisel. Labgrid MCP Server: Model Context Protocol Interface for Embedded Test Infrastructure. <https://codeberg.org/Embedded-Focus/labgrid-mcp>, 2025. Accessed: 2026-01-30.
- [30] Luís Prates and Rúben Pereira. DevSecOps practices and tools. *International Journal of Information Security*, 24(1):11, 2024.
- [31] Thoughtworks. Thoughtworks Technology Radar: A Guide to the Technology Landscape. <https://www.thoughtworks.com/radar>, November 2025. Volume 33. Accessed: 2026-01-25.
- [32] Xenodium. agent-shell: Emacs ACP client. <https://github.com/xenodium/agent-shell>. Accessed: 2026-01-11.
- [33] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, et al. A survey of large language models. *arXiv preprint arXiv:2303.18223*, 1(2), 2023.

AUTHORS AND AFFILIATIONS

Rainer Poisel is the founder of Embedded Focus and hon-eytreeLabs, specializing in DevSecOps for embedded systems. His expertise lies in designing scalable, high-performance development processes, integrating automation, security, and compliance frameworks into modern engineering workflows. With a strong interdisciplinary approach, he bridges the gap between software engineering, embedded systems, and cybersecurity, enabling organizations to modernize legacy infrastructures, optimize CI/CD pipelines, and ensure regulatory compliance. His research-driven methodology and practical focus contribute to advancing secure and efficient development practices in embedded software engineering.