MARTINELLI
Keep IT simple

Spec-driven Development

Simon Martinelli

REPORT 15 SEP 2025

Kambundji shocks in Tokyo as Swiss peaks to take 100m hurdles gold

About Me

- 30 years in software engineering
- 25 years with Java
- Self-employed since 2009
- Teaching at two Universities
- Co-lead Berne, JUG Switzerland

Java Champions

vaadin Champion

Oracle ACE Pro

What is Vibe Coding?

I just see things, say things, run things, and copy-paste things, and it mostly works.

- Andrej Karpathy, OpenAI Co-founder, former Tesla AI lead, February 2025

- A **programming approach** where you
  - Use natural language to describe what you want
  - Let AI (LLMs) write all the code
  - Accept code without full understanding
  - Focus on the goal, not the code itself

Is Vibe Coding Any Good?

Ok-ish

- Prototypes
- Quick experiments
- Hobby projects

- Non-programmers building apps

Risky

- Business applications
- Complex systems
- Security-critical apps

- Non-programmers building apps

Quotes by Martin Fowler

"I think the appearance of LLMs will change software development to a similar degree as the change from **assembler** to the first **high-level programming languages**.

The further development of languages and frameworks increased our abstraction level and productivity, but didn't have that kind of impact on the nature of programming.

"LLMs are making that degree of impact like high-level languages had versus the assembler.

The distinction is that LLMs are not just **raising the level of abstraction**, but also forcing us to consider what it means to program with **non-deterministic tools**."

Caution: AI Is Not a Compiler

- **AI** code generation **isn't a compiler** - it's an assistant!
  - Many developers expect AI to work like a compiler
  - Precise input → perfect output
  - But that's not how it works
  - It’s not comparable to Model-Driven Development

- AI is **non-deterministic** and makes mistakes

- **You are responsible for the output!**

AI Native Development

- **Spec-Centric Development**
  - Clear intent/specs guide AI to generate meaningful code
- **Context-Aware Development**
  - AI agents understand full codebase context
- **Agent Experience (AX)**
  - Autonomous AI tasks (bug fixes, PRs, tests) enhancing developer throughput

https://ainativedev.io/

Spec-driven Development (SDD)

- **Start with the spec**, not code
- Spec is the contract and **source of truth** for tools and AI
- Leads to **less guesswork**, **fewer surprises**, **better code**

SDD in Pratice

- **Process**
  - AI Unified Process  https://unifiedprocess.ai

- **Tools**
  - AIUP Claude Code Plugin https://unifiedprocess.ai/tools.html
  - Amazon Kiro https://kiro.dev
  - GitHub Spec Kit https://github.com/github/spec-kit
  - BMad https://github.com/bmad-code-org
  - ... and others

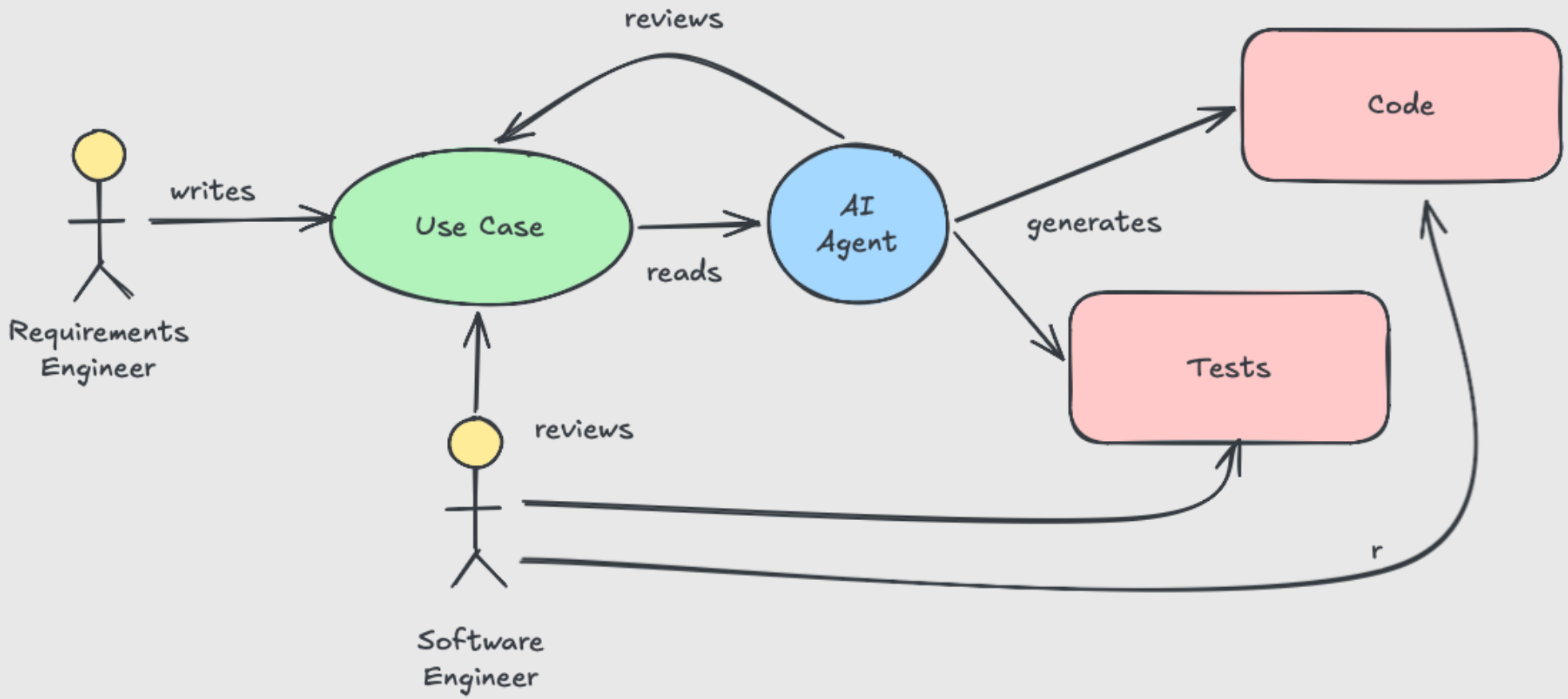
How Do the Approaches Differ?

- **AI Unified Process** is primarily a **process and methodology** with tooling support, emphasizing requirements and spec quality across the entire lifecycle
- **Amazon Kiro** is a **spec-centric AI IDE** with strong integrated automation where AI drives from spec to code within one environment
- **GitHub Spec Kit and BMad** are open-source **workflow toolkits** that formalizes phases and integrates with different AI coding tools rather than providing its own agentic environment

What Are Executable Specs?

- An executable spec describes **observable system behavior**, not implementation details
- It acts as a **stable contract** between business intent and technical implementation
- It can be **systematically turned into code and tests**, with minimal interpretation
- It helps prevent **drift between requirements, code, and tests** over time

Use Case Workflow



https://github.com/simasch/ai-track-and-field

Impact on the Architecture

- **Modular architecture reduce the context size**
  - Self-contained Systems
- **Single ecosystem simplify guardrails**
  - Full-stack frameworks

Guardrails

- **Consistency across sessions**
  - AI has no memory, so guardrails ensure the same behavior in every conversation
- **Encoding project knowledge**
  - Capture coding standards, architecture decisions, and conventions
- **Prevent AI drift**
  - Without rules, AI might suggest patterns or libraries that don't match your project
- **Reducing Repetition**
  - Define skills once instead of explaining them in every prompt
- **Team Alignment**
  - All developers get similar, project-appropriate suggestions from the AI
- **Quality Control**
  - Enforce standards like testing requirements, documentation rules, and code style

Recommendations

- **Scaffold** the application **yourself**
- Set **guardrails**
- Constantly **review everything** that is generated
- Create **test case examples**
- **Use/create skills**
- Use **MCP Servers** but take care of the context size

Conclusion

- **Specs help** to reduce non-determinism
- AI feels like a teammate but **it's just a tool**
- It accelerates development, but **you must**:
  - **Review**, **understand**, and **test** the output
  - **Know** your **architecture** and **domain**

Thank you!

- **Web**
  martinelli.ch
- **E-Mail**
  simon@martinelli.ch
- **Bluesky**
  @martinelli.ch
- **X/Twitter**
  @simas_ch
- **LinkedIn**
  https://linkedin.com/in/simonmartinelli