MARTINELLI
Keep IT simple

Spec-driven Development Workshop

https://martinelli.ch/sdd/

Simon Martinelli

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

Introduction

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**."

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 enhancing developer throughput

Spec-driven Development (SDD)

- **Start** with the **specification**, not the code

- **Specifications**
  - Define the **intended behavior** of the system
  - **Serve as shared contract** between business and development
  - Are the single **source of truth**
  - **Reduce guesswork** and improve quality

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

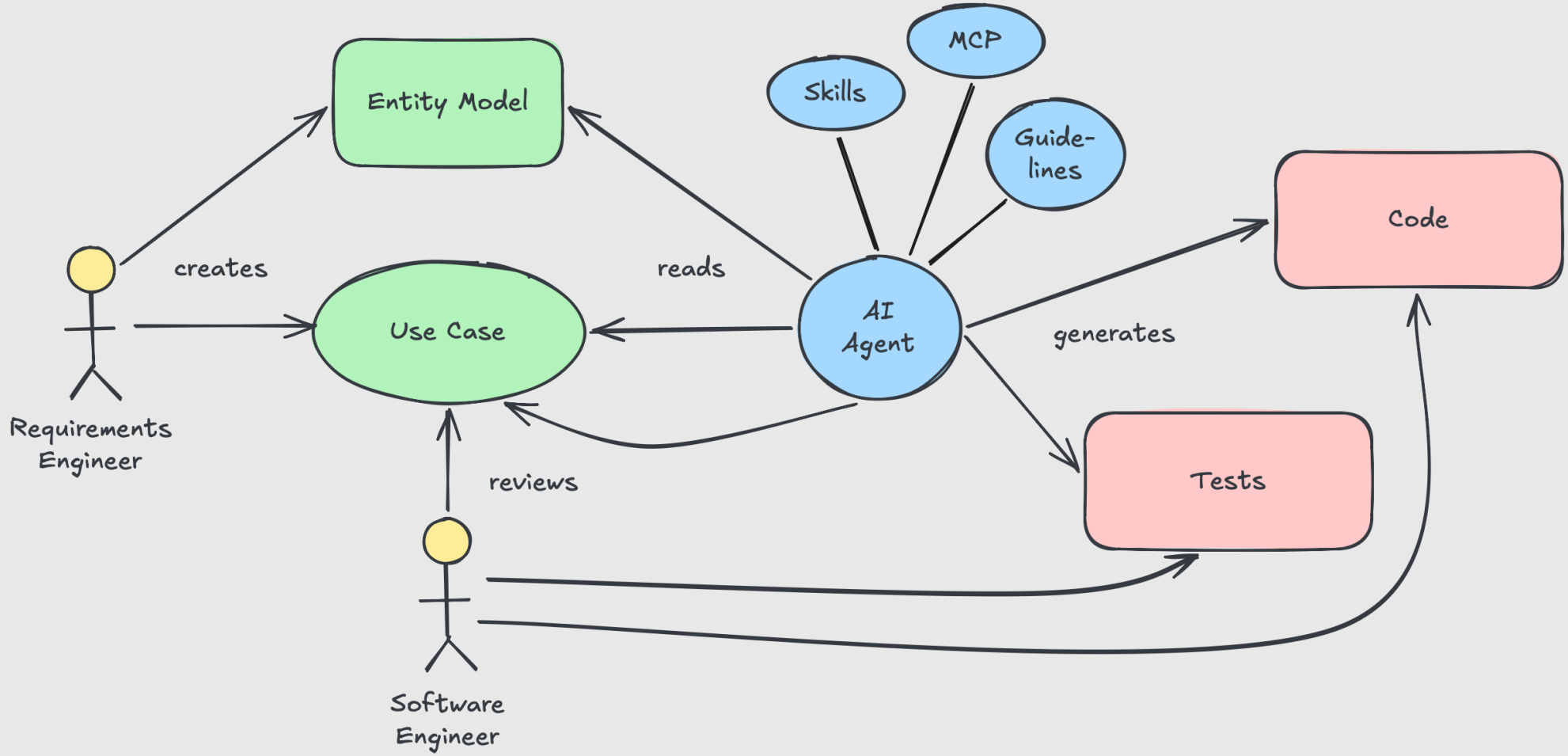
- AI is **non-deterministic** and makes mistakes
- **You are responsible** for the output!

SDD in Pratice

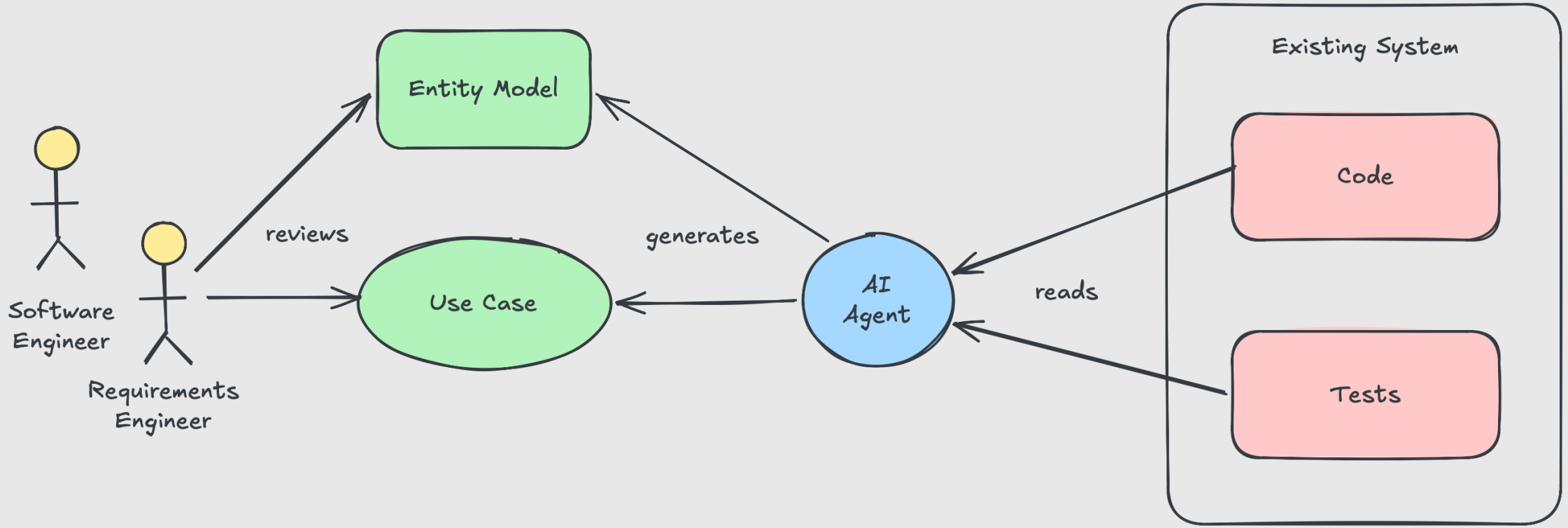
- **Process-centric**
  - AI Unified Process (AIUP) https://unifiedprocess.ai

- **Tools-centric**
  - Amazon Kiro https://kiro.dev
  - GitHub Spec Kit https://github.com/github/spec-kit
  - BMad https://github.com/bmad-code-org
  - ...

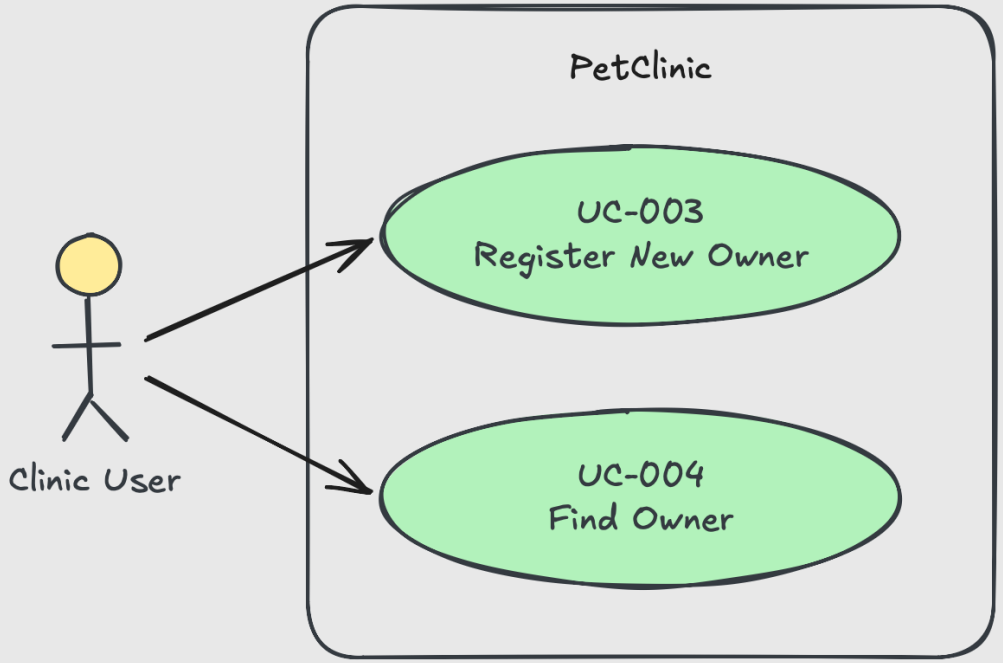
Greenfield Workflow



Brownfield Workflow



Why Use Cases?



Overview
(ID, Name, Actors, Goal, Status)

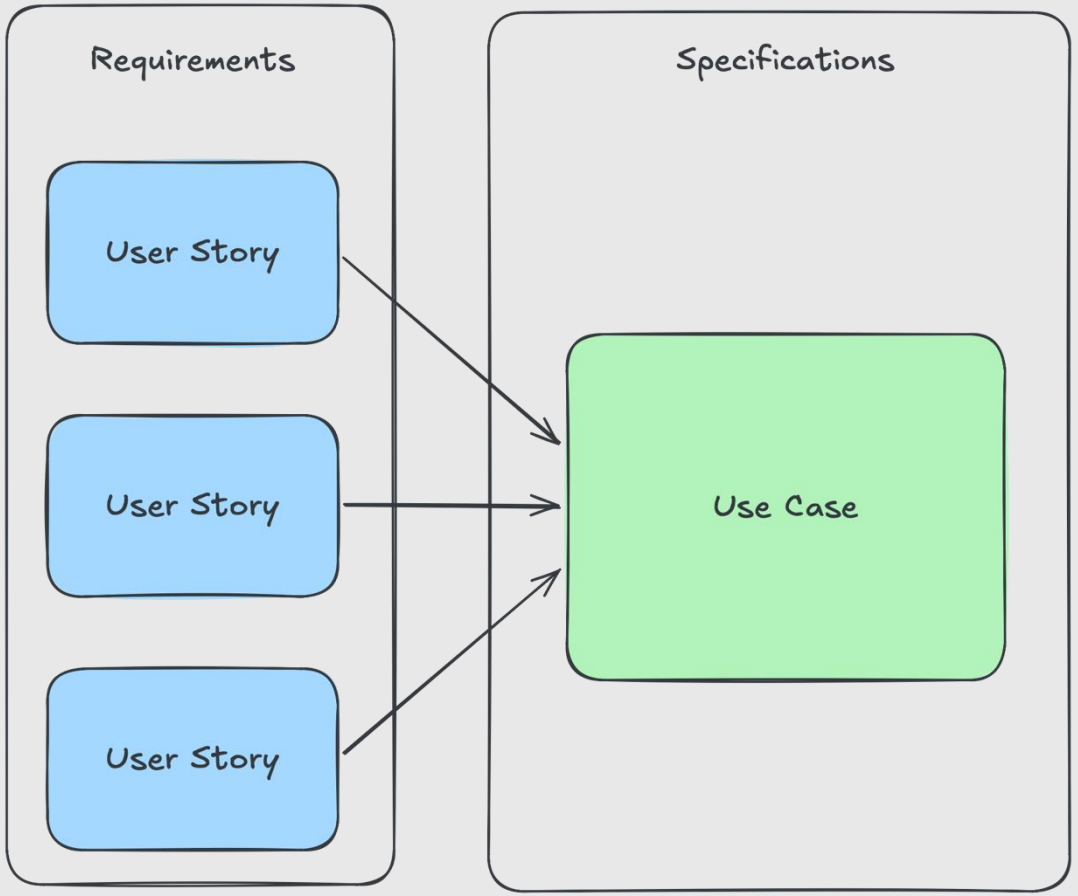
Preconditions

Main Success Scenario

Alternative Flows (optional)

Postconditions

How Are Use Cases and User Stories Related?



Claude Code

- **Terminal-based** AI code assistant
- Integrated in Claude App and https://claude.ai
- **Plugins** for JetBrains tools and Visual Studio Code
- GitHub and GitLab **integration**
- **Features**
  - Skills
  - MCP
  - Subagents

- Documentation: https://docs.claude.com/en/home

Claude Code Security Model

- Claude Code provides built-in sandboxing, but it is **not full isolation**
- **Key mechanisms**
  - Filesystem restrictions (limit accessible folders)
  - Network controls (allow / deny outbound access)
  - Permission system (asks before critical actions)
  - Configurable policies per project
- **Important**
  - Sandbox can be relaxed or bypassed with approval
  - It is a soft boundary, not a hard security layer
- **Takeaway**
  - Good default protection, but not enough for untrusted execution

From Sandbox to Real Isolation

- Stronger security requires **external isolation**
- **Option 1: Docker Container**
  - Run agent inside container
  - Limit volumes and secrets
  - Control network access
- **Option 2: Docker Sandbox (MicroVM)**
  - Full isolation from host
  - Separate filesystem and runtime
  - No access to local machine
  - Environment can be reset anytime
- **Benefits**
  - Safe autonomous agent execution
  - Prevents data leaks to host system
  - Enables multi-agent setups securely

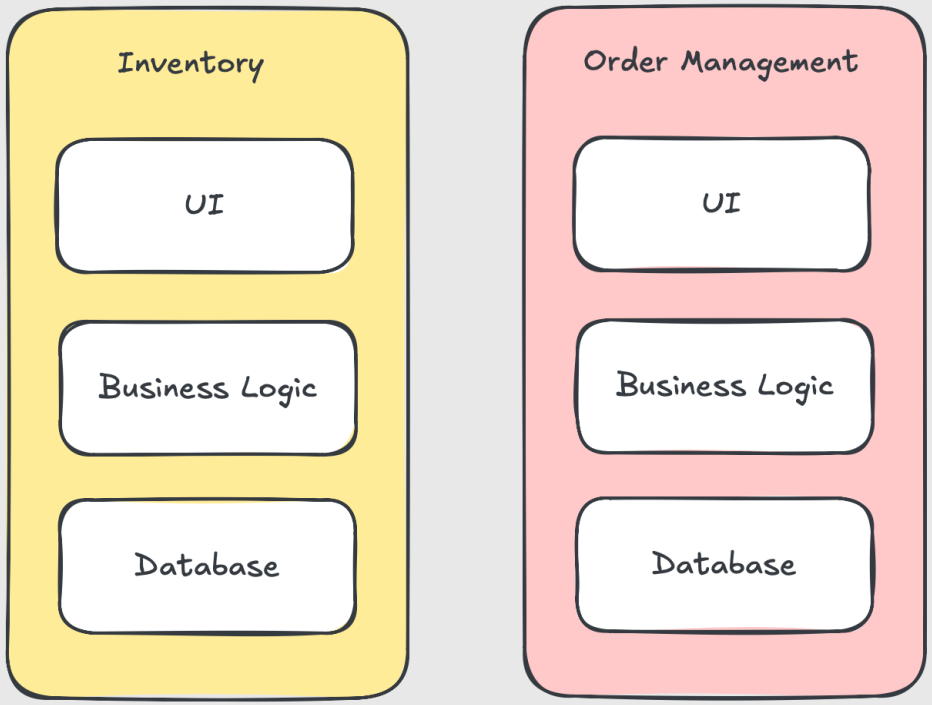
Guidelines and Guardrails

- **Guidelines** define rules
  - Standards, architecture, testing
- **Skills** capture repeatable tasks
  - Implementation, testing, refactoring, reviews
- **MCP** connects tools and context
  - Documentation, code examples, quality checks

- Together they create **guardrails**
- **Human review** stays essential

Impact on the Architecture

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



Hotel Reservation System

Vision

- We are the proud owners of a small, ten-room hotel.
- We need a system to manage reservations, guest check-in/check-out, and room cleaning.
- Actors
  - Guest (Online Booking)
  - Receptionist (On-site Management)
  - Manager (Reporting and Administration)
  - Housekeeping (Room Service)

0. Set the Stage

1. **Clone the project**
   https://github.com/simasch/hrs
   - main: PostgreSQL with Testcontainers (requires Docker)
   - h2: H2 (no Docker required)

2. **Install the Claude Code Marketplace**
   https://github.com/AI-Unified-Process/marketplace/

3. Run **`./mvnw spring-boot:test-run`**

1. From Vision to Requirements

1. Create the vision markdown file **`docs/vision.md`**

2. Run **`/requirements`**

3. Review **docs/requirements.md** and make adjustments if needed

Example Requirements

ID	Description	Prio	Status	Size
FR-001	Guests can search online for available rooms	High	Open	M
FR-002	Guests can reserve rooms online	High	Open	L
FR-003	The system automatically generates booking confirmation emails	High	Open	S
FR-004	Receptionists can check in walk-in guests	High	Open	M
FR-005	Receptionists can check out guests and generate invoices	High	Open	L
FR-006	The system manages room status (available, occupied, cleaning, maintenance)	High	Open	M
FR-007	Guests can cancel reservations up to 24 hours before arrival	Medium	Open	S
FR-008	Managers can generate occupancy reports	Medium	Open	XL
FR-009	The system sends reminder emails before arrival	Low	Open	S
FR-010	Housekeeping can update room status	High	Open	S

2. Derive Entity Model

1. Run **`/entity-model`**

2. Review **`docs/entity_model.md`** and make adjustments if needed

3. Create Use Cases Diagram

1. Run **`/use-case-diagram`**

2. Review **`docs/use_cases.puml`** and make adjustments if needed

4. Use Case Specification

1. Run **`/use-case-spec UC-###`**
   ### = the number of the use case you want to generate the specification for

2. Review **`docs/use_cases/UC-###`** and make adjustments if needed

5. Create Database Migrations

1. Run **`/flyway-migration`**

2. Review the files generated in **`src/main/resources/db/migration`** and make adjustments if needed

3. Run **`./mvnw compile`**

6. Generate Code

1. Run **`/implement UC-###`**
   ### = the number of the use case you want to generate the code for

2. Run the application in the IDE or by running
   `./mvnw spring-boot:test-run`

3. Review the code in **`src/main/java`** and make adjustments if needed

7. Integration Test

1. Run **`/browserless-test UC-###`**
   ### = the number of the use case you want to generate the code for

2. Review the generated test in **`src/main/test`** and make adjustments if needed

3. Run the tests **`./mvnw test`**

7. E2E Test

1. Run **`/playwright-test UC-###`**
   ### = the number of the use case you want to generate the code for

2. Review the generated *IT class in **`src/main/test`** and make adjustments if needed

3. Run the tests **`./mvnw verify`**

Risks and Recommendations

Technical Risks

- **Prompt injection attacks**
  - Malicious input manipulates the agent’s behavior
- **Dependency confusion**
  - Agents may import unsafe or unnecessary libraries
- **Unsafe generated code**
  - AI may produce vulnerable or insecure implementations
- **Over-permissive configurations**
  - Too many permissions increase attack surface

Quality Risks

- **Incorrect or insecure code**
  - Agents often produce plausible but wrong or unsafe code
- **Silent logic errors**
  - Code compiles and passes basic tests but behaves incorrectly
- **Architecture drift**
  - Generated code might violate architectural rules or conventions
- **Poor test coverage**
  - Agents may generate minimal or trivial tests
- **Performance issues**
  - Agents don’t optimize unless explicitly prompted

Security and Compliance Risks

- **Data leakage**
  - Source code, data, credentials, or business logic sent to external APIs (LLM, MCP etc)
- **Insecure code patterns**
  - Hardcoded secrets, missing input validation, or unsafe serialization
- **Supply chain contamination**
  - Generated code might copy licensed or unknown-source snippets

Legal and IP Risks

- **Copyright uncertainty**
  - You may not fully own the generated code
- **License incompatibility**
  - Agents may generate code under a restrictive license
- **Auditability**
  - Hard to trace which parts of code were human vs. AI generated

Organizational and Process Risks

- **Skill erosion**
  - Developers stop understanding what they use
- **Developers stop understanding what they use**
  - Code grows faster than governance processes
- **False confidence**
  - Teams assumes “AI created it, so it’s fine”
- **Mismatch with documentation**
  - Code changes faster than specs or architecture diagrams

Mitigation Strategies

Area	Action
Governance	Constantly **review everything** that is generated
Architecture and Security	**Scaffold** the application **yourself** Use tools like ArchUnit to **enforce structure** automatically **Run static analysis** (SonarQube, OWASP Dependency Check)
Data protection	**Don’t use production data** while at development time Prefer on-prem or self-hosted models for sensitive code
Testing	Create **test case examples**
Training	**Educate developers** on AI limits and prompt engineering

Conclusion

- **Specs help** to
  - **Reduce** non-determinism
  - Make development **sustainable**

- 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