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

Three Levels of SDD

Level 1: Spec-first

- You write a spec before you write code.
- The **spec becomes the input** for the AI coding agent.
- **When the feature is done, the spec is not needed anymore.**
- For the next change, you write a new spec.

Level 2: Spec-anchored

- The **spec stays in the repository** after the feature is done.
- When the feature changes, you **update the spec** and the code together.
- The spec is the **long-term reference** for humans and for the AI.

Level 3: Spec-as-source

- The **spec is the only thing** a human edits.
- The **code is always (re-)generated** from the spec.
- **Humans never touch the code**.

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!

Steps of AI Adoption

- https://claude.ai/code/artifact/bfdfaef9-bc62-4dfe-ba9e-c58a26c9accf

Choosing a Method

How to Choose

- **Greenfield or brownfield**
  - Brownfield needs existing code as context and more structure
- **Lifetime of the system**
  - A prototype needs less process than a system that lives ten years
- **Regulation and audit**
  - Traceability from requirement to code can be mandatory
- **Team size and maturity**
  - More people means more need for shared artifacts
- **Size of the context**
  - Large monoliths break every method

Not the Same Category

Category	Examples	Best for
Process and methodology	AI Unified Process	Enterprise, brownfield, long lived systems
Workflow toolkit	BMad, GitHub Spec Kit	Greenfield, developer-centric, small to medium projects
Spec centric IDE	Amazon Kiro	Single developer or small team, one tool
Own lightweight setup	Your own commands	Fits your toolchain, but has no method behind it

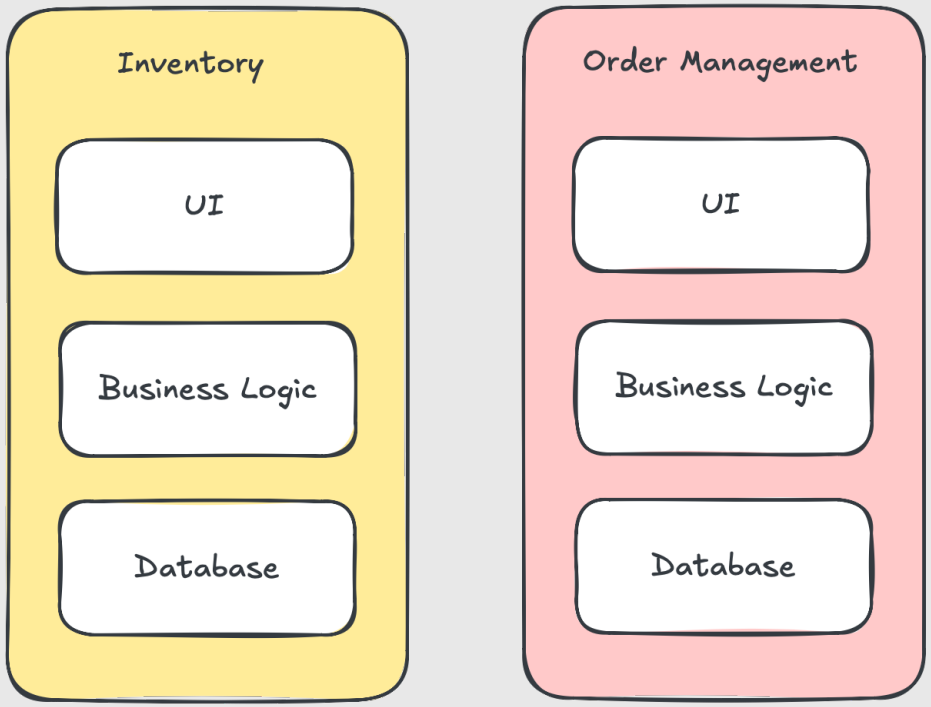
Lessons from Customer Projects

Context Is the First Bottleneck

- **Large legacy code does not fit**
  - The agent only sees what you give it
- **Modularity is now a requirement**
  - Self-contained systems keep the context small
- **Specs replace reading code**
  - A short use case spec is cheaper than a whole module
- **Bad structure costs money**
  - Every extra file in the context is paid for in every turn

Impact on the Architecture

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



Review Is the Second Bottleneck

- **Generating is fast, understanding is not**
  - The team produces more code than it can review
- **Reviews need a spec**
  - Without a spec you review against nothing
- **Small units help**
  - One use case per session keeps the diff reviewable
- **Plan the review time**
  - Teams underestimate this in every project

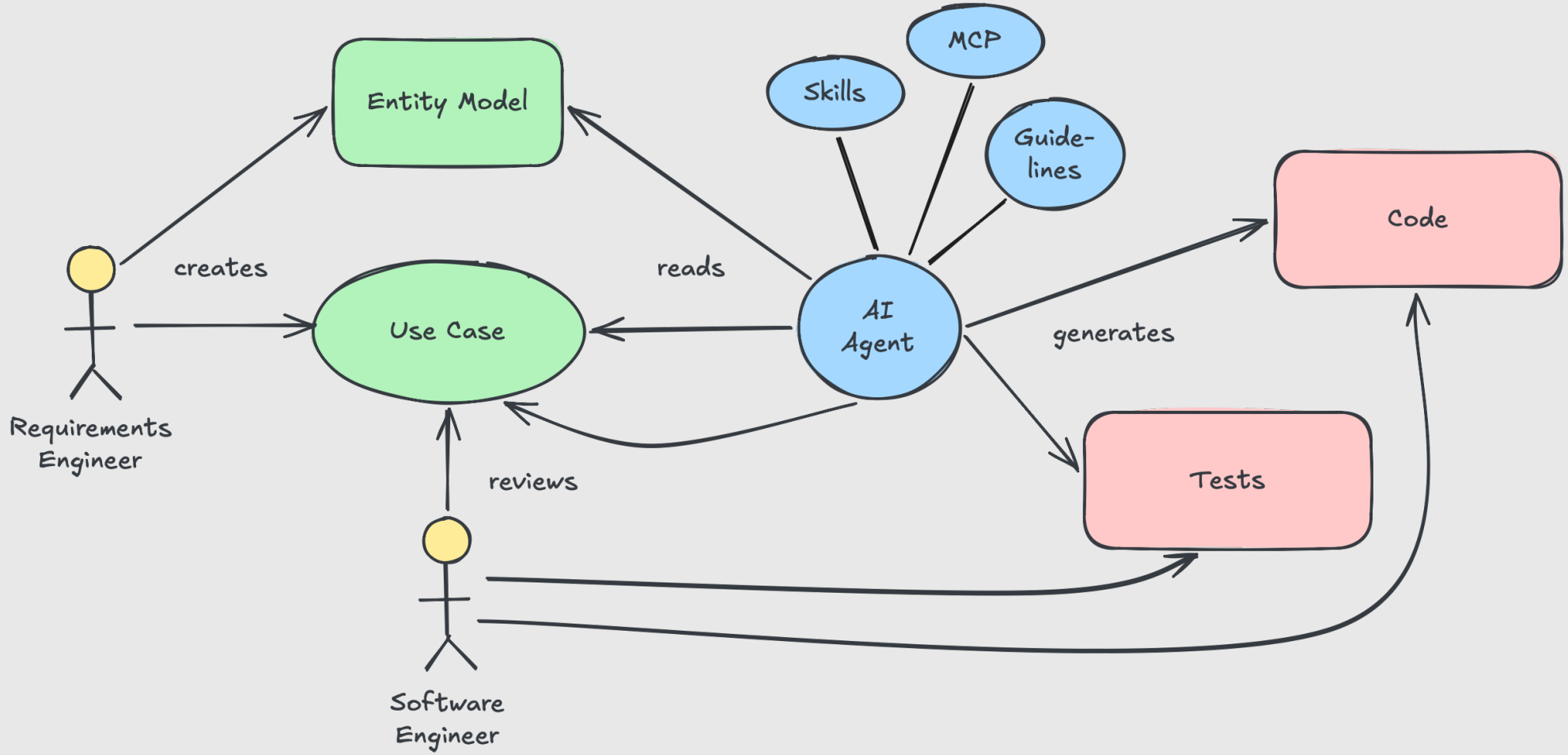
Recurring Challenges

- **Spec drift**
  - Specs get old when nobody owns them
- **Non determinism**
  - Same prompt, different result, so use quality gates and not hope
- **Data protection**
  - No production data, on premise models for sensitive code
- **Tool volatility**
  - Everything changes every few weeks, so method before tool
- **Trust is not balanced**
  - Seniors doubt the output, juniors trust it too much

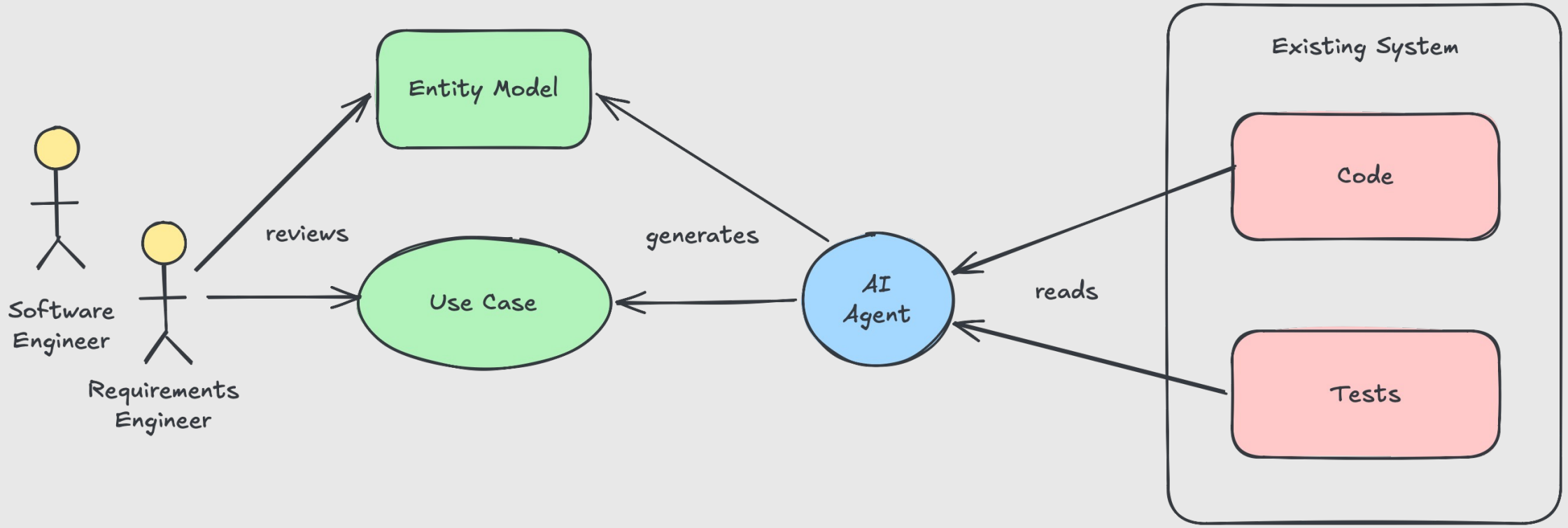
AI Unified Process

https://unifiedprocess.ai

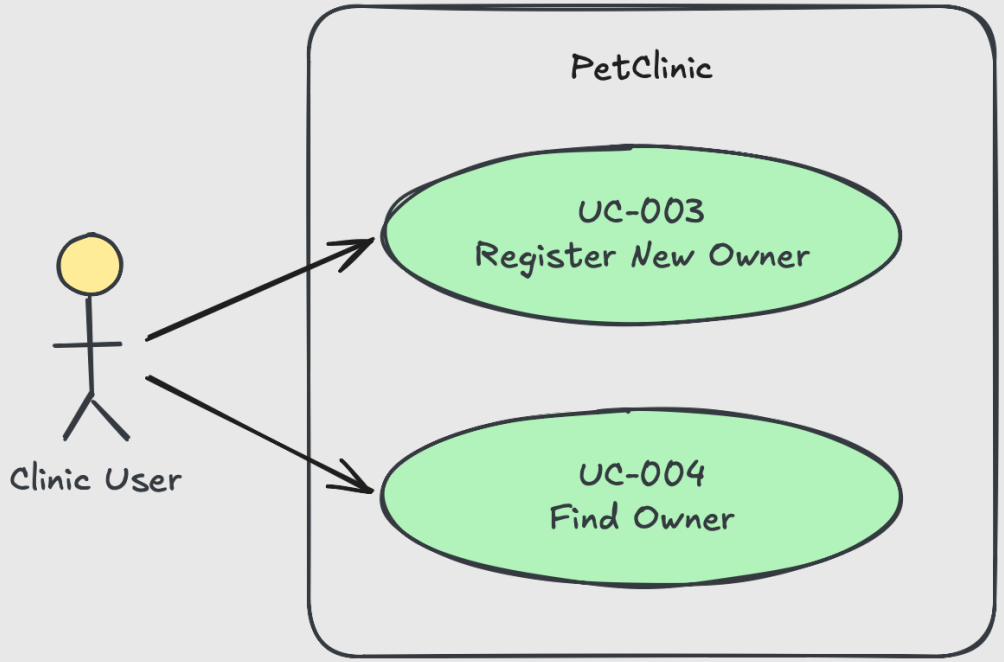
Greenfield Workflow



Brownfield Workflow



Why Use Cases?



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

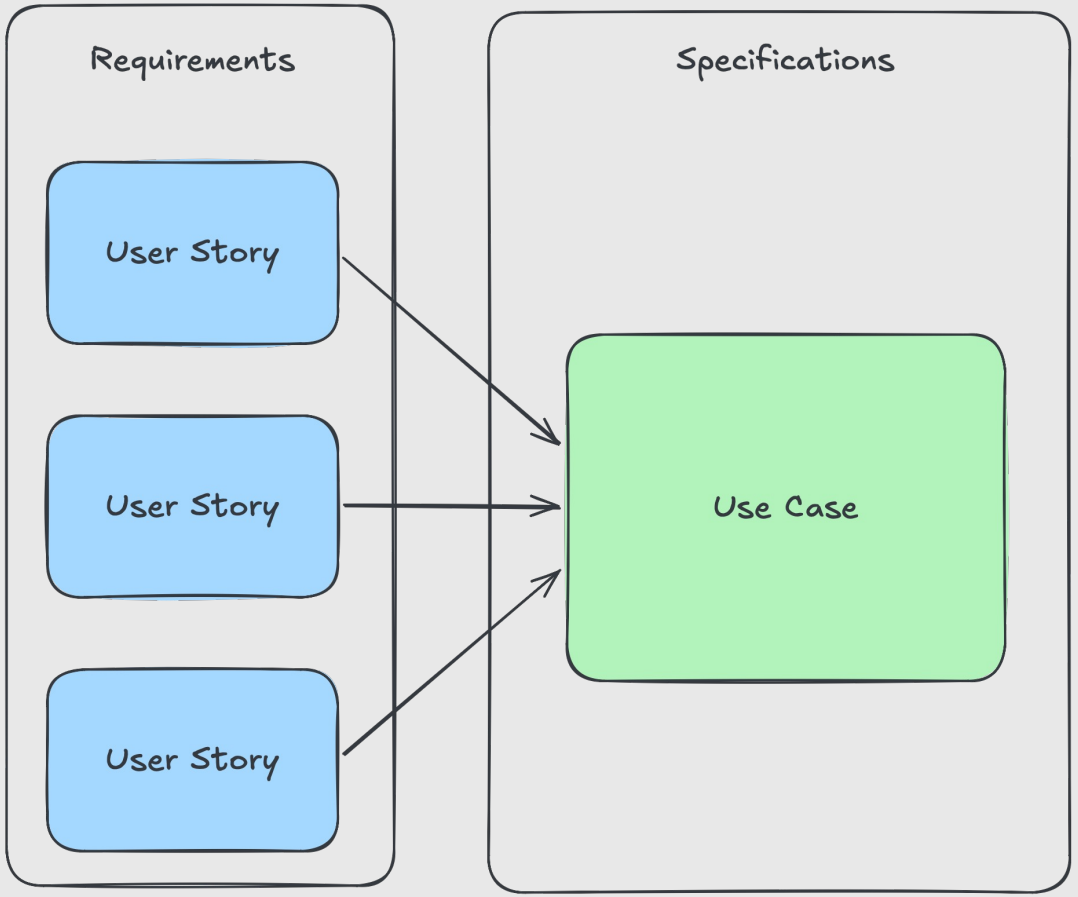
Preconditions

Main Success Scenario

Alternative Flows (optional)

Postconditions

How Are Use Cases and User Stories Related?



The Agent

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: VM**
  - Run agent on a VM (i.e. Hetzner, Fly.io Sprites)
  - Optional: Connect with Claude Code Remote Control
- **Option 2: Sandbox** (see next slide)
  - 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

Sandbox & Isolation Solutions – Comparison

sbx vs. nono.sh vs. NVIDIA OpenShell

CATEGORY	CRITERIA	Docker sbx	nono.sh	NVIDIA OpenShell
ISOLATION	Isolation Model	MicroVM (Firecracker)	Kernel-based Sandbox	Container + Kubernetes (K3s)
	Security Level	Very High ★★★★★	High ★★★★☆	High ★★★★☆
	Escape Protection	Very High ★★★★★	High ★★★★☆	High ★★★★☆
	Network Isolation	Policies (Allow/Deny) ★★★★	Domain Allowlist ★★★★★	Kubernetes Policies (OPA) ★★★★
	Resource Limits	Yes (CPU, RAM, Disk, Net) ★★★★	Yes (CPU, RAM, Disk, Net) ★★★½☆	Yes (CPU, RAM, Disk, Net) ★★★★
DEVELOPMENT & USAGE	Installation	Easy ★★★★	Very Easy (Single Binary) ★★★★	Complex ★★½☆
	Docker Required	Yes (Docker + Login)	No	Docker or Podman
	Startup Time	Fast (Seconds) ★★★★	Very Fast (Milliseconds) ★★★★	Long (K3s Start) ★★★☆
	IDE / Tool Integration	Good ★★★★	Good ★★★☆	Advanced ★★☆☆
	Native Execution	No	Yes (Native)	No
AI AGENT SUPPORT	Claude Code	✓	✓	✓
	OpenCode	✓	✓	✓
	MCP Support	✓	✓	✓
	Workspace Persistence	✓	✓	✓
SECURITY & SECRETS	Secret Proxy / Placeholder	Yes (Proxy + Placeholder) ★★★★	Yes (Proxy + Placeholder) ★★★★	Yes (K8s Secrets/Proxy) ★★★★
	Vault Integration	Possible ★★★★	Possible ★★★★	Very Good (Vault, K8s) ★★★★
	Audit Logs	Good ★★★★	Medium ★★★★	Very Good ★★★★
	Policy Engine	Docker Policies ★★★★	Simple (Allowlist) ★★★½	OPA / Rego ★★★★
PERFORMANCE	Startup Time	Fast ★★★★	Very Fast ★★★★	Slow ★½☆☆
	RAM Usage	Medium ★★½☆	Very Low ★★★★	High ★★★★
	CPU Overhead	Medium ★★★☆	Very Low ★★★★	High ★★★★
	Parallelism	Good ★★★★	Good ★★★★	Very Good ★★★★
ENTERPRISE	Multi-User	Limited ★★½☆	No	Yes ★★★★
	RBAC / SSO	No	No	Yes ★★★★
	Central Management	No	No	Yes ★★★★
	Compliance / Audit	Medium ★★★☆	Medium ★★★☆	Very Good ★★★★
PLATFORM	macOS (Apple Silicon)	✓ Yes (Apple Silicon)	✓ Yes (Apple Silicon)	✓ Yes (Apple Silicon)
	Linux	✓ Yes	✓ Yes	✓ Yes
	Windows (WSL2)	✓ Yes (WSL2)	✓ Yes (WSL2)	✓ Yes (WSL2)
OPEN SOURCE & ECOSYSTEM	License	Docker (Proprietary)	Apache 2.0	Apache 2.0
	Community	Large ★★★★	Growing ★★★★	Small but active ★★½☆
	Extensibility / API	Good (CLI, Config, API) ★★★★	Very Good (CLI, Library, API) ★★★★	Very Good (API, Plugins) ★★★★

SUMMARY & RECOMMENDATION

Maximum Security for AI Agents

Best isolation through MicroVMs and strong secret handling. Ideal for secure agent execution.

Lightweight & Extremely Fast

Perfect for developers who don't need containers and want maximum speed.

Enterprise Platform for Teams

Central management, policies, K8s-native and scalable for companies and organizations.

TIP In practice, many teams combine multiple solutions – using sbx or nono.sh for secure AI agent execution.

Set the Guardrails

- **Don’t use AI to scaffold** the application
- **Guidelines** define rules that can be **checked deterministically**
  - Standards, architecture, testing
- **Skills** capture repeatable tasks
  - Implementation, testing, refactoring, reviews
- **MCP** connects tools and context
  - Documentation, code examples, quality checks

(Human?) review stays essential!

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: H2 (no Docker required)
   - postgresql: PostgreSQL with Testcontainers (requires Docker)

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

Better copy from website!

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. Create E2E Test Case

1. Run **`/test-case`**

8. Implement E2E Test

1. Run **`/playwright-test TC-###`**
   ### = the number of the test 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`**

Token Economy

Where the Cost Comes From

- **Input tokens drive the cost**
  - Context per turn multiplied by the number of turns
- **Output is the small part**
  - The generated code is rarely the main cost
- **Long sessions get expensive**
  - The whole history is sent again with every turn
- **Letting the agent search is expensive**
  - Scanning the repository burns tokens fast

Levers: Reduce the Context

- **One use case per session**
  - Close the session instead of letting it run forever
- **Modular architecture**
  - Self-contained systems need less context per task
- **Point to files, do not search**
  - Precise references or an MCP that returns the exact information
- **Keep specs short and focused**
  - A use case spec is cheaper than a long document

Levers: Models and Agents

- **Use prompt caching** (often automatically by the agent)
  - The biggest single saving when the context repeats
  - *Changing the model, effort, or changing MCPs destroy the cache*
- **Match the model to the task**
  - Cheap models for boilerplate and tests, strong models for design and review
- **Use subagents**
  - Their own context window keeps the main session clean
- **Be careful with persona chains**
  - Several roles reading the same context multiply the cost

Subscription or API

- **Subscription seats**
  - Predictable cost, good for daily development work
- **API billing**
  - Pay per token, needed for automation and pipelines
- **Who pays**
  - The customer can provide the keys, which also helps with data protection
- **Set a budget**
  - Cost per developer per month is easier to defend than cost per prompt

Measure Cost per Use Case

- **Take a baseline**
  - Track the tokens for one complete use case, from spec to test
- **Compare, do not guess**
  - Numbers from your own project beat any benchmark
- **Watch the trend**
  - Cost per use case should fall when specs and structure improve
- **Exercise**
  - Note your token usage after each step of the hands-on part

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,** in combination with guardrails, **help** to
  - **Reduce** non-determinism
  - Make development **sustainable**
- It **accelerates development**, but **you are responsible for the quality**:
  - **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