MARTINELLI
Keep IT simple

Spec-driven Development Workshop

Simon Martinelli

About Me

- >30 years in **Software Engineering**
- >25 years with **Java**
- **Self-employed** since 2009
- **Teaching** at four universities
- Co-lead Berne, **JUG Switzerland**

Java Champions

vaadin Champion

Oracle ACE Pro



Agenda

Part 1

- Why Spec-Driven Development? From developer to builder
- Choosing a method: OpenSpec, BMAD, Spec Kit, Kiro, AI Unified Process
- Lessons from customer projects
- AI Unified Process, the harness and use cases
- The agent: Claude Code and security
- Hands-on: from vision to running code

Part 2

- Hands-on: tests, coverage and changing requirements
- Hardening the harness: guardrails, sensors and constraints
- Brownfield: from code to specs
- Token economy
- Risks, recommendations and wrap-up

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
  - **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

- **Step 0: Gated**
  Only older models approved, access to AI tools is gated and process-heavy
- **Step 1: Assisted**
  You + one agent as a pair, you review almost every change
- **Step 2: Parallel**
  You orchestrate 5–10 agents, Claude checks its own work
- **Step 3: Supervised autonomy**
  Manager of managers, ~100 agents, Claude writes nearly all the code
- **Step 4: AI-native**
  Steering by intent, 1,000+ agents mostly kicked off by Claude

From Developer to Builder

Before	Now
Writes every line of code	Writes specifications and reviews code
Follows a ticket	Owns architecture and quality
Knowledge in the head	Knowledge in the repository
Uses the IDE	Builds and tunes the harness
Gives the agent rigid commands	Sets goals, context and feedback loops

The developer does not disappear. The work moves up one level.

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
Agent-team method	BMAD	Product teams who want a full agile flow with AI roles
Workflow toolkit	OpenSpec, GitHub Spec Kit	Developer-centric, small to medium projects
Spec-centric IDE	Kiro (AWS)	Single developer or small team, one tool
Skills registry	Tessl	Sharing versioned specs and skills across agents and teams
Own lightweight setup	Your own commands	Fits your toolchain, but no method behind it

Five Frameworks at a Glance

Framework	Core idea	Main artifacts	Sweet spot	Checks code against spec
OpenSpec	Change proposals on top of living specs	Proposal, spec deltas, tasks	Changes in existing code, lightweight	/opsx:verify, AI review (optional)
BMAD	A team of agent personas (analyst, PM, architect, dev)	PRD, architecture, stories	Greenfield, full agile flow with AI roles	Parallel AI code review
GitHub Spec Kit	Constitution, then specify, clarify, plan, tasks, implement	Constitution, spec, plan, tasks	Greenfield features, works with many agents	/analyze, /converge, AI review
Kiro (AWS)	Spec-centric IDE with steering and hooks	Requirements (EARS), design, tasks	One developer or small team, one tool	Property-based tests from requirements
AI Unified Process	Iterative process based on use cases	Requirements, use cases, entity model, test cases	Enterprise, brownfield, long-lived systems	/coverage-check per flow and rule, sensors in CI

Tools change every few weeks. Check the current versions before you decide.

Which Fits When?

A change in an existing repository

OpenSpec

New product, the team likes roles and agile ceremonies

BMAD

New feature, any agent, structured flow

GitHub Spec Kit

One developer, everything in one tool

Kiro (AWS)

Long-lived enterprise system, many stakeholders, audit

AI Unified Process

Customer already uses other tools

AI Unified Process as the method, keep the tools

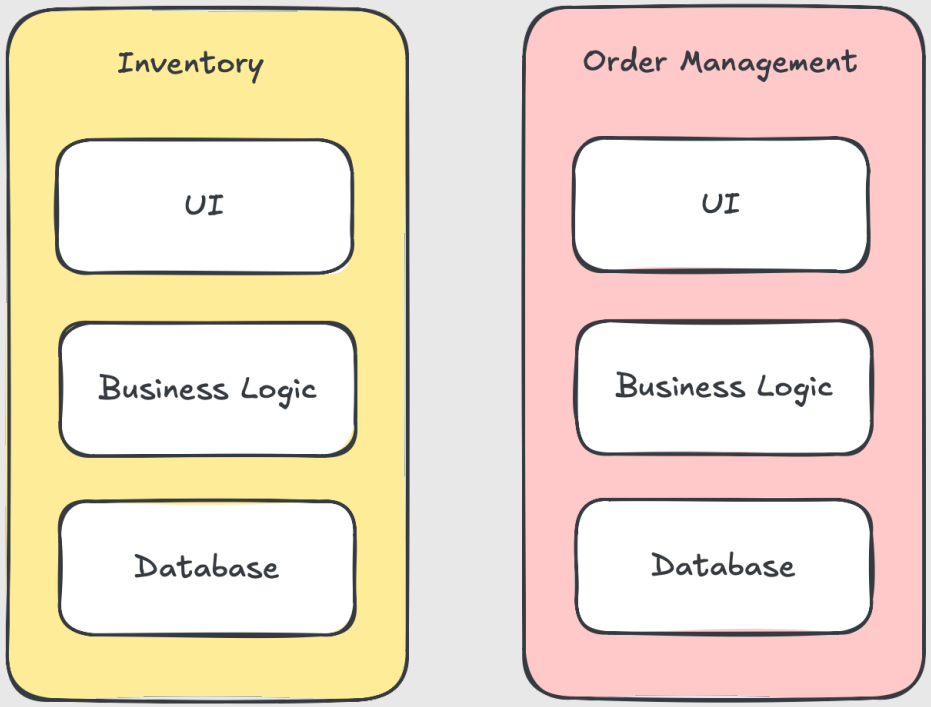
Lessons from Customer Projects

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

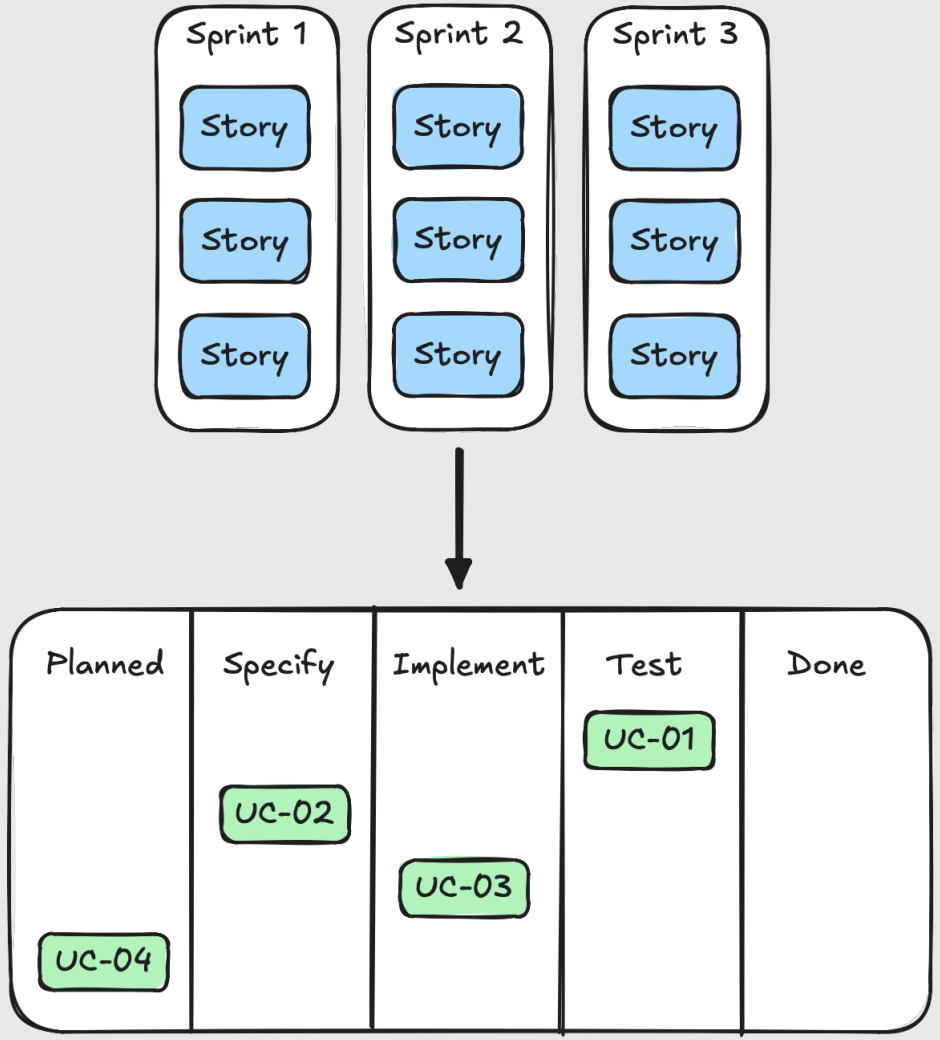
Impact on the Architecture

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



Impact on Teams

- **Smaller team size**
  - 1-2 developers per self-contained system
- **No more sprints**
  - Continuous flow
  - From Scrum to Kanban
  - Progress tracking on use cases



One Team, Shared Decisions

- **Shared harness**
  - Same CLAUDE.md, plugins and skills for everybody, from the repository
- **Decide first, then generate**
  - Write the ADR before the first use case that needs the decision
- **One owner per use case**
  - Two agents never work on the same specification
- **One branch or worktree per use case**
  - Small pull requests that can be reviewed
- **Review against the specification**
  - The pull request links the use case it implements
- **Spec changes are reviewed like code**

Workflow for Several Developers

1 Spec reviewed
/spec-review is green

→

2 Owner assigned
One developer per use case

→

3 Branch or worktree
Isolated session

→

4 Agent implements
Code and tests

→

5 Quality gates
Build, ArchUnit, tests

→

6 Review against spec
/coverage-check

→

7 Merge
Status: Implemented

Progress is tracked per use case on a Kanban board, not in story points.

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-premises 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

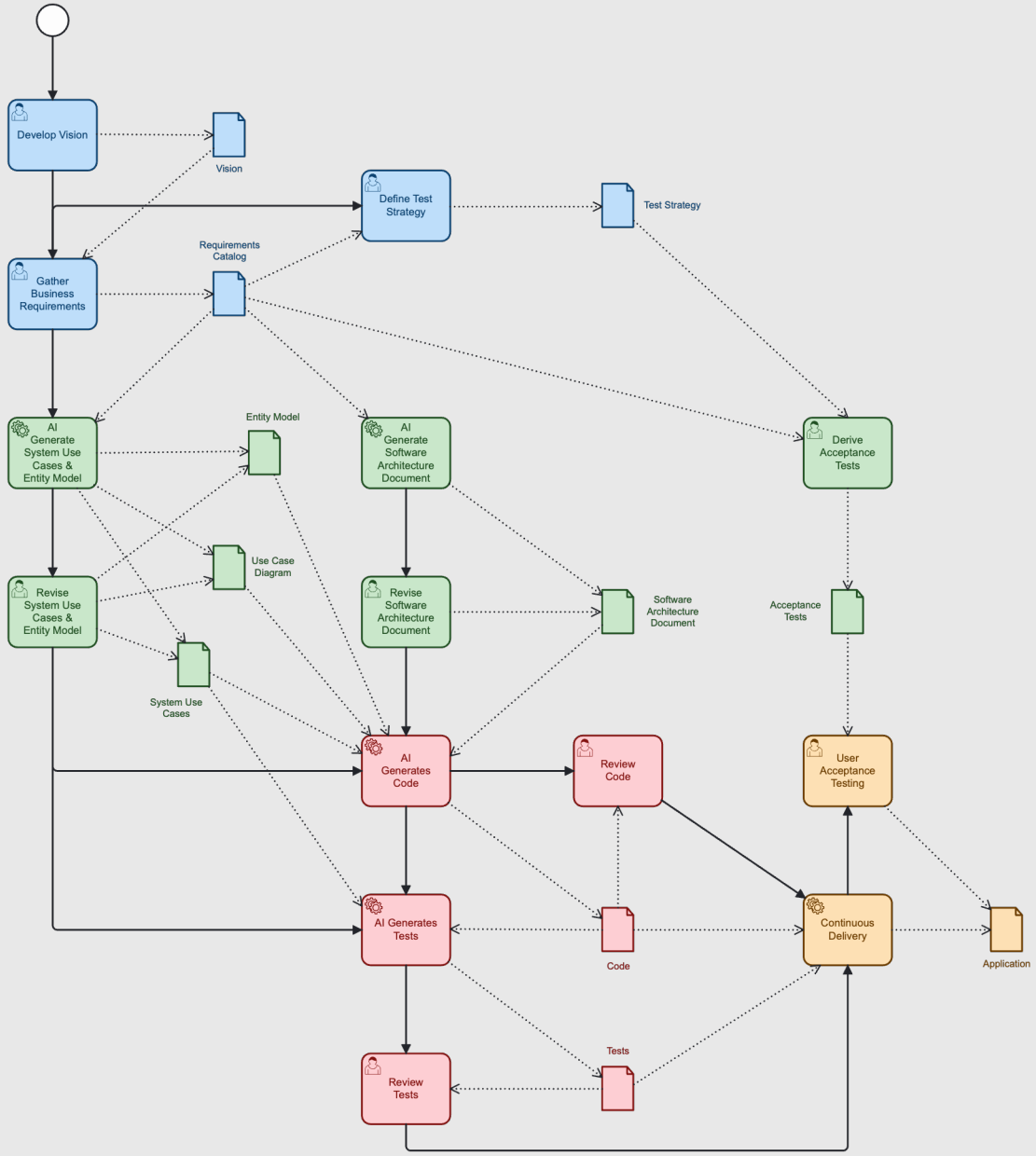


AI Unified Process

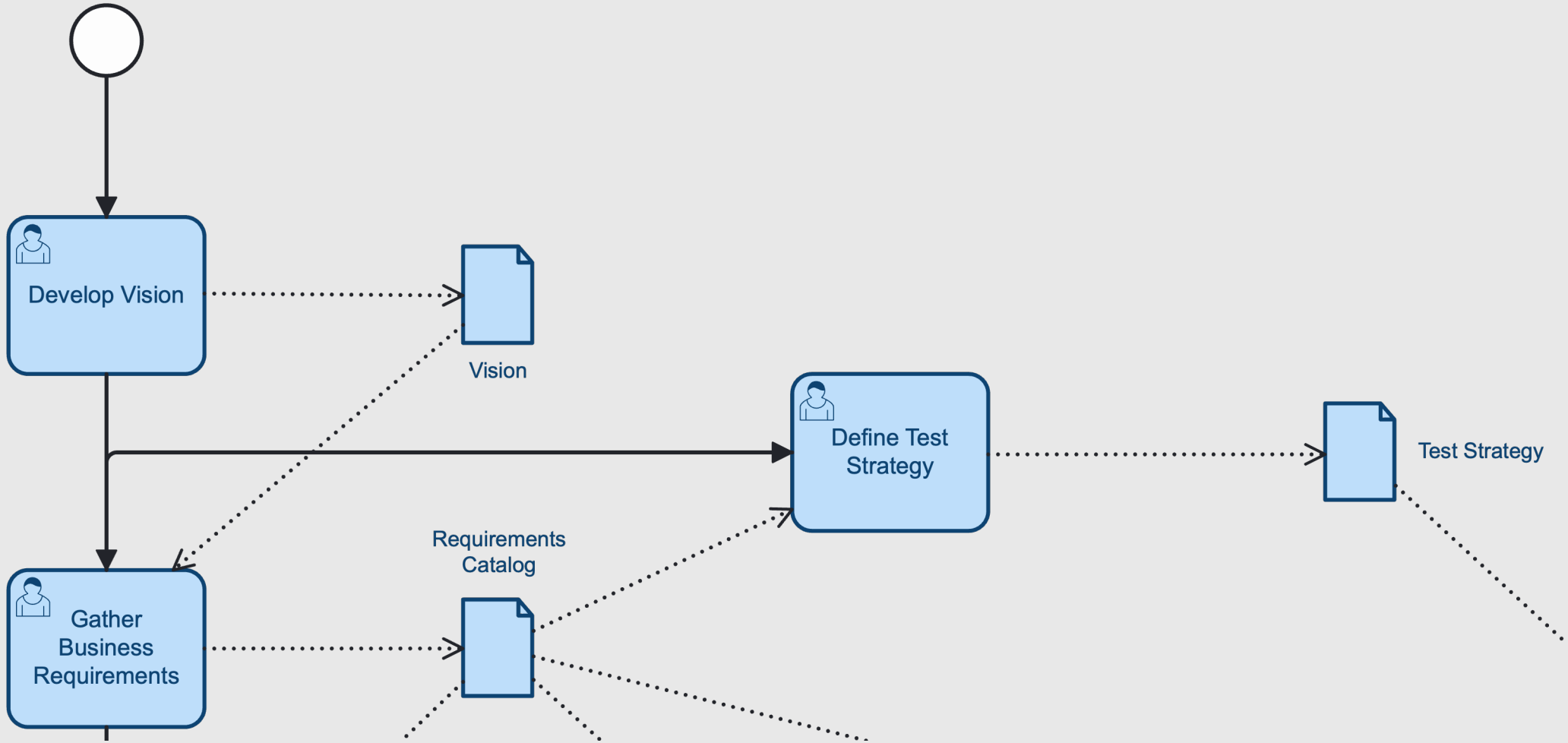
https://unifiedprocess.ai

At a Glance

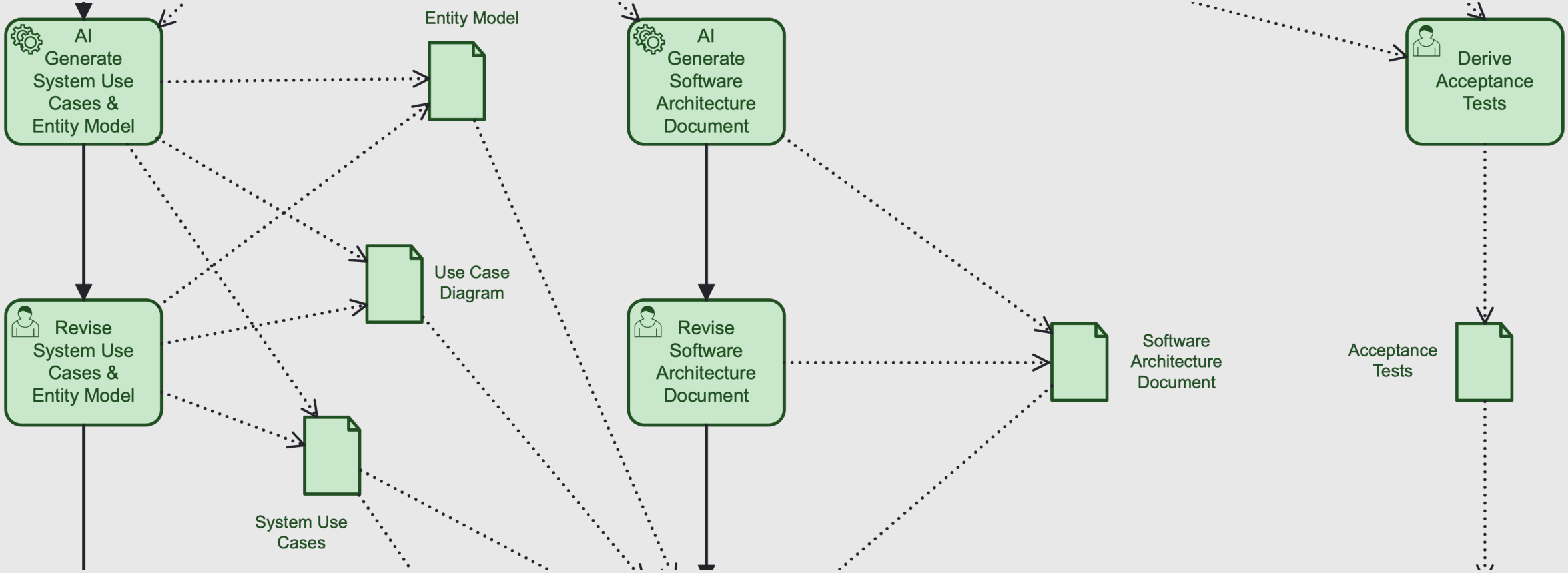
- Vision and requirements
- Use cases, entity model, architecture, test strategy
- AI generates code and tests
- Delivery and acceptance



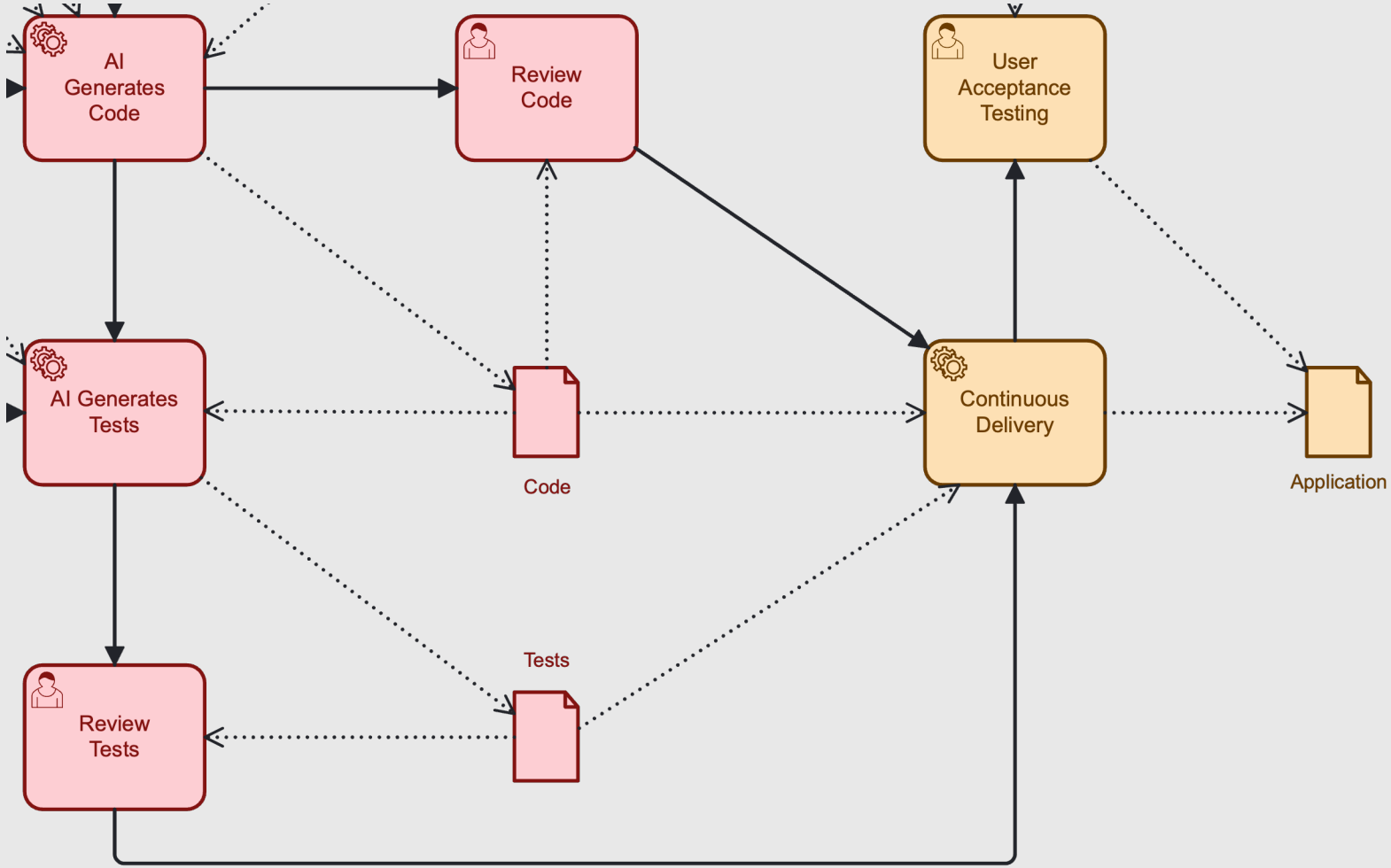
Vision and Requirements



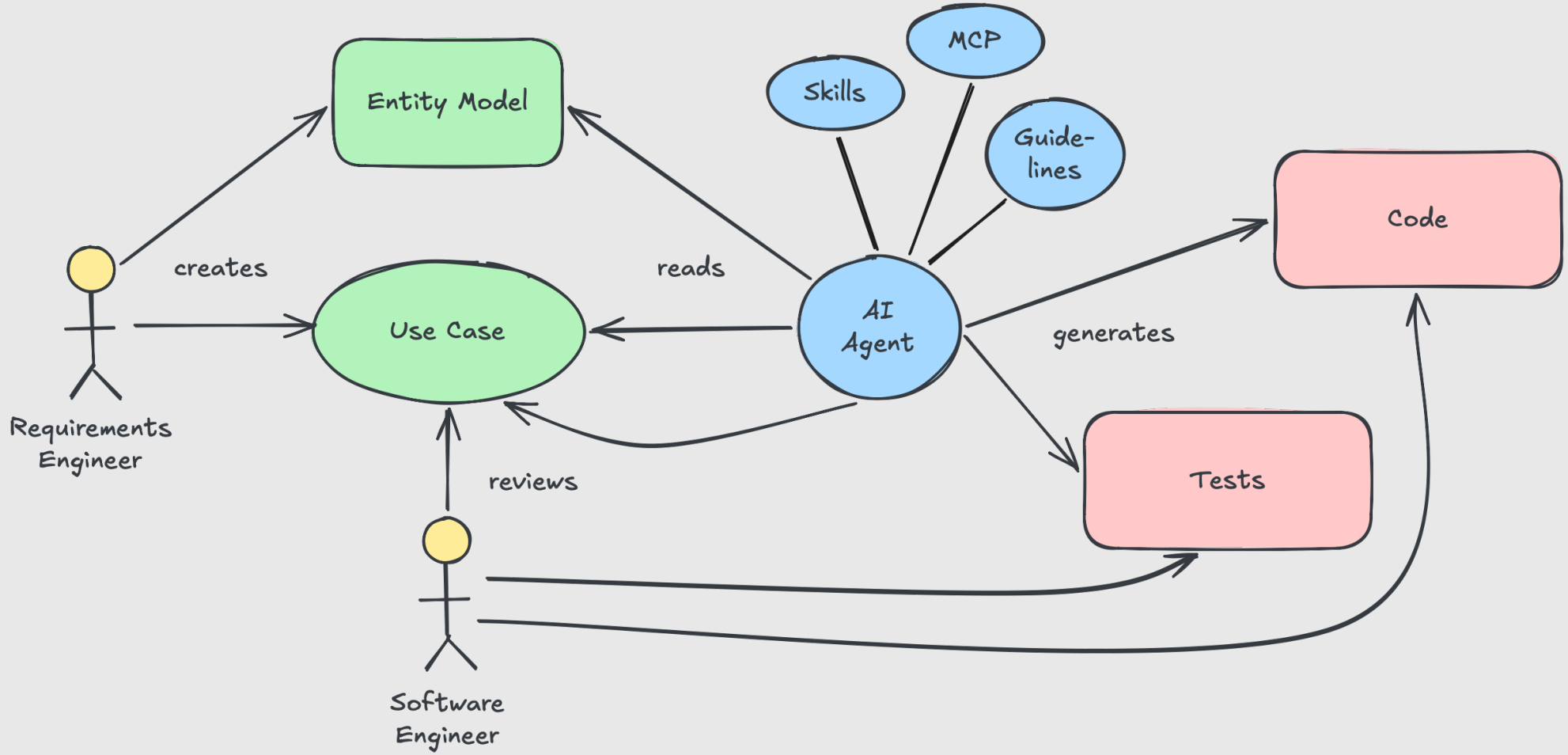
Use Cases and Architecture



Code, Tests and Delivery



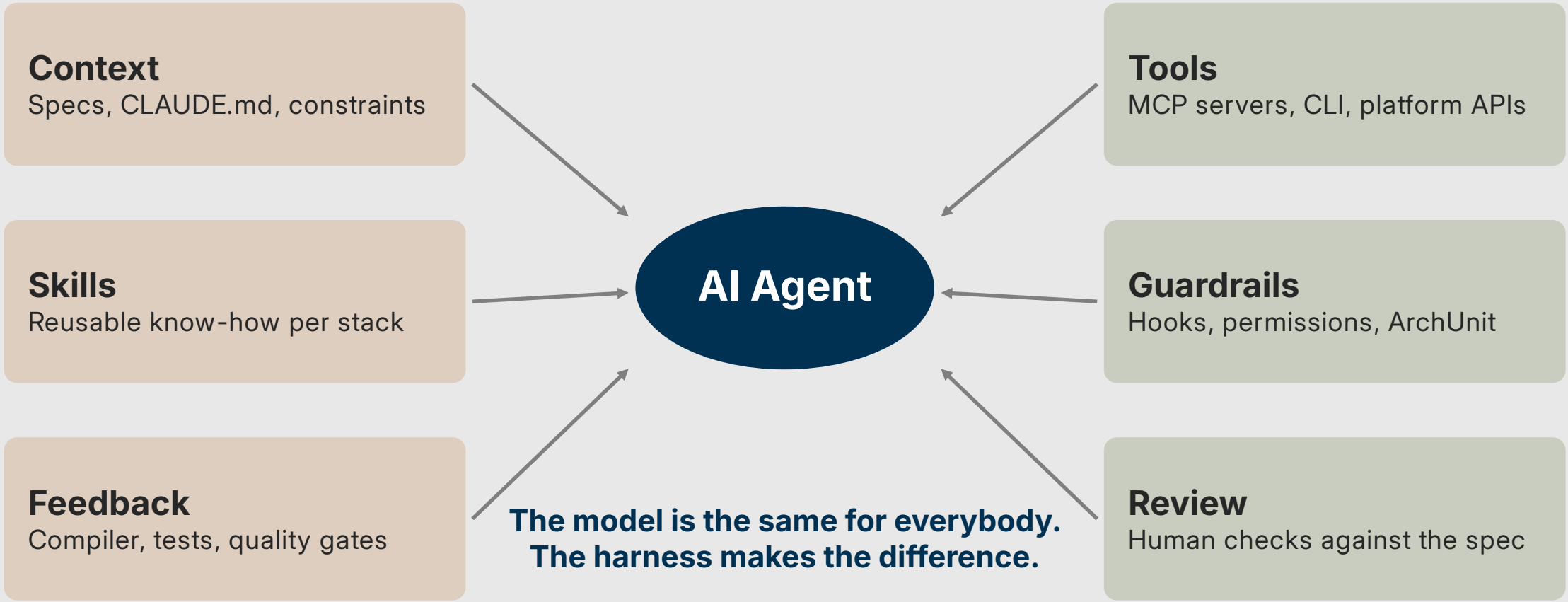
Greenfield Workflow



Harness Engineering

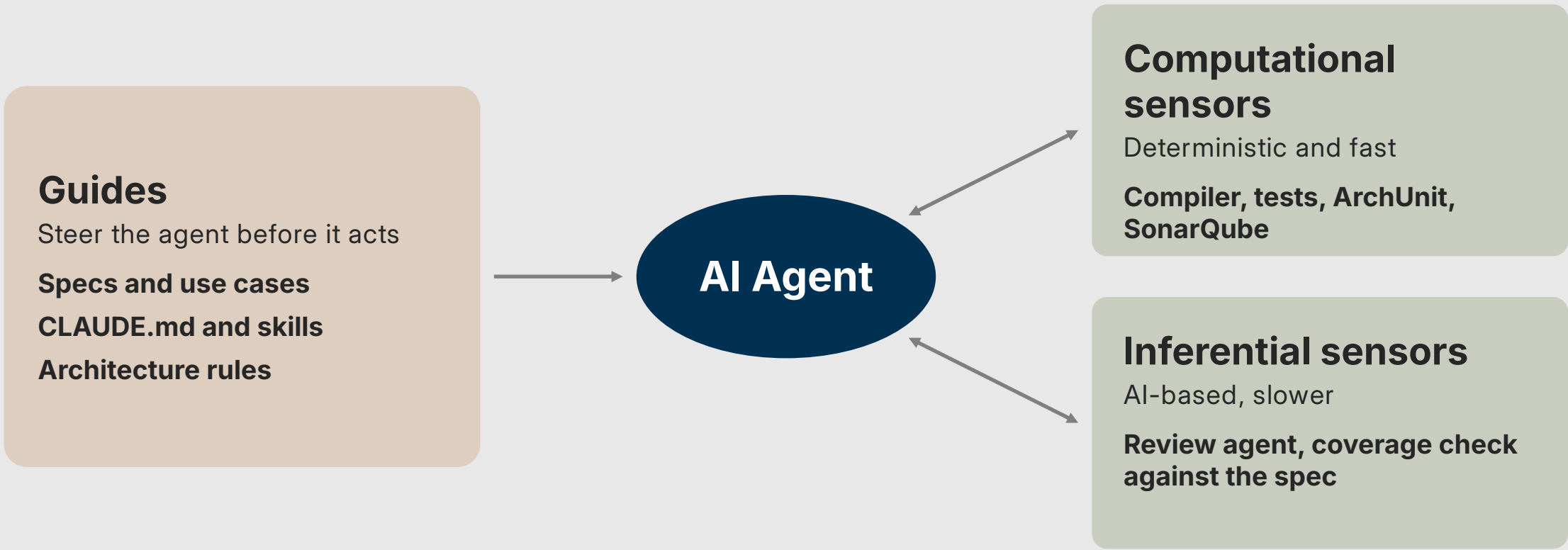
- **Harness**: Everything around the model that makes the agent reliable
- **Guides**: Steer the agent before it acts
- **Sensors**: Check after the agent acts so it can self-correct
  - **Computational**: Deterministic and fast
  - **Inferential**: AI-based, e.g. a review agent
- **The human steers**: When a mistake happens more than once, improve the harness
- **Goal**: Less review effort and higher quality

What Is the Harness?



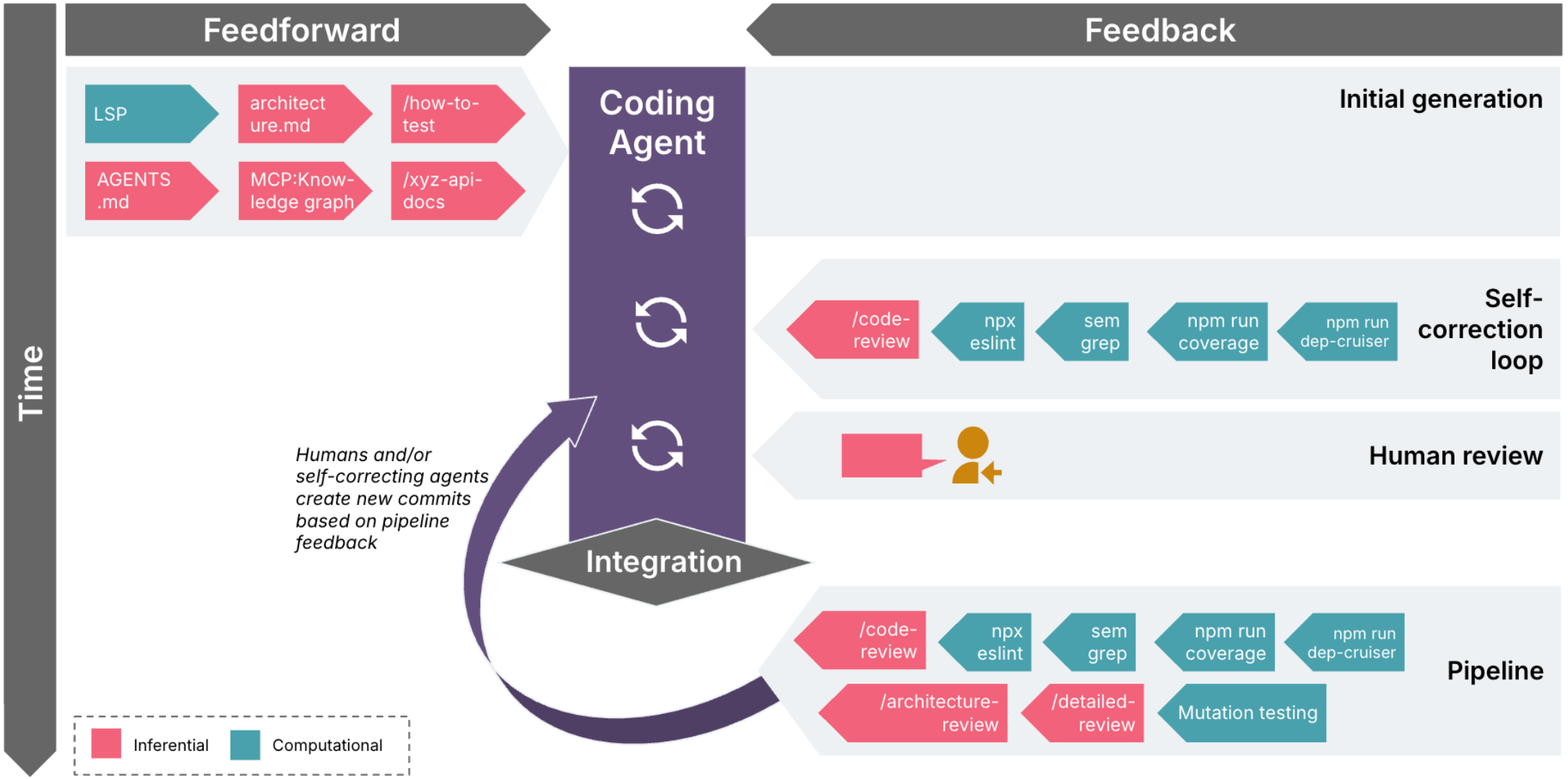
Source: Harness Engineering for Coding Agent Users, Birgitta Böckeler

Guides and Sensors



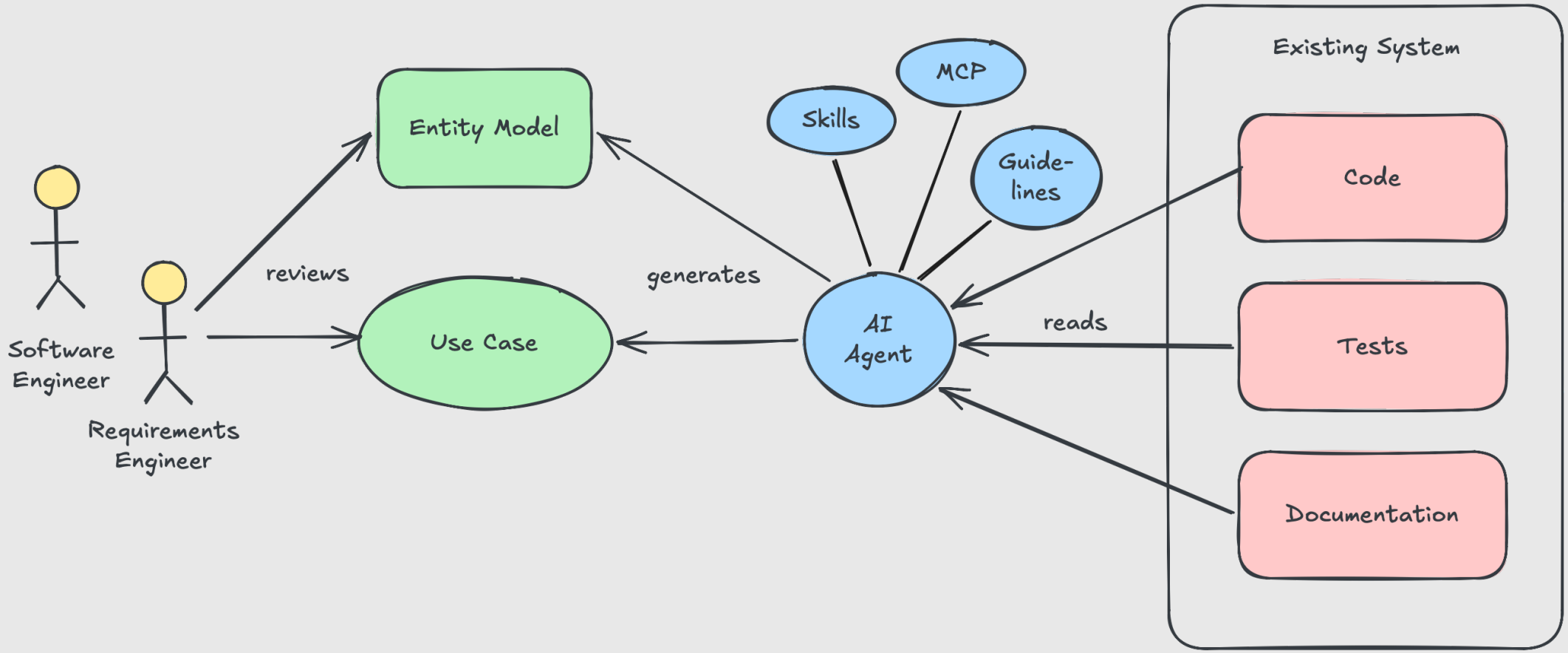
Sensors close the loop: the agent fixes its own mistakes before you review.

Examples: Feedforward and feedback in the change lifecycle

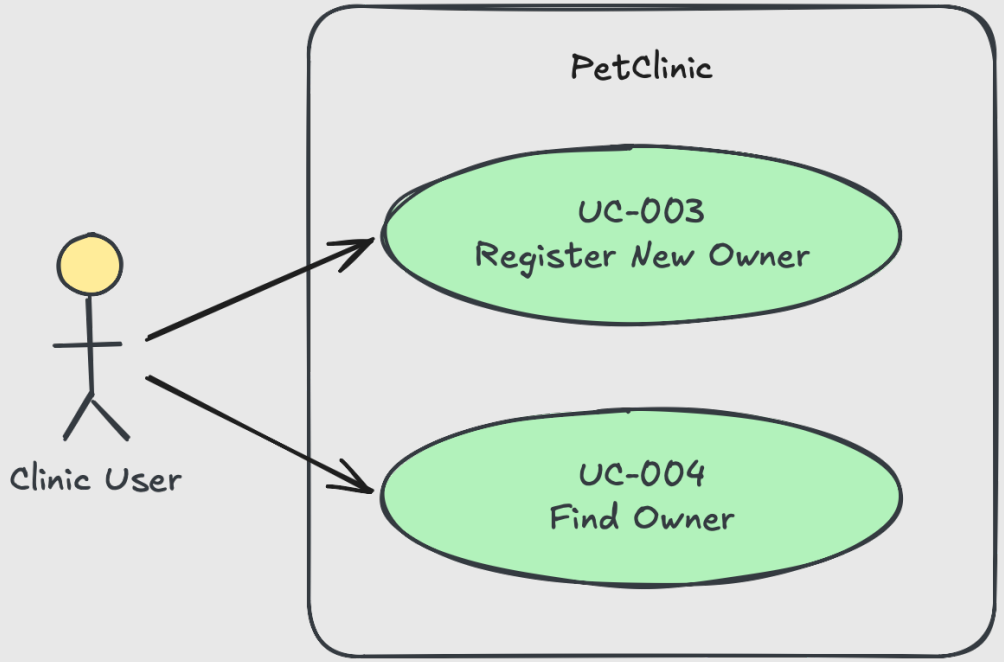


https://martinfowler.com/articles/harness-engineering.html

Brownfield Workflow



Why Use Cases?



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

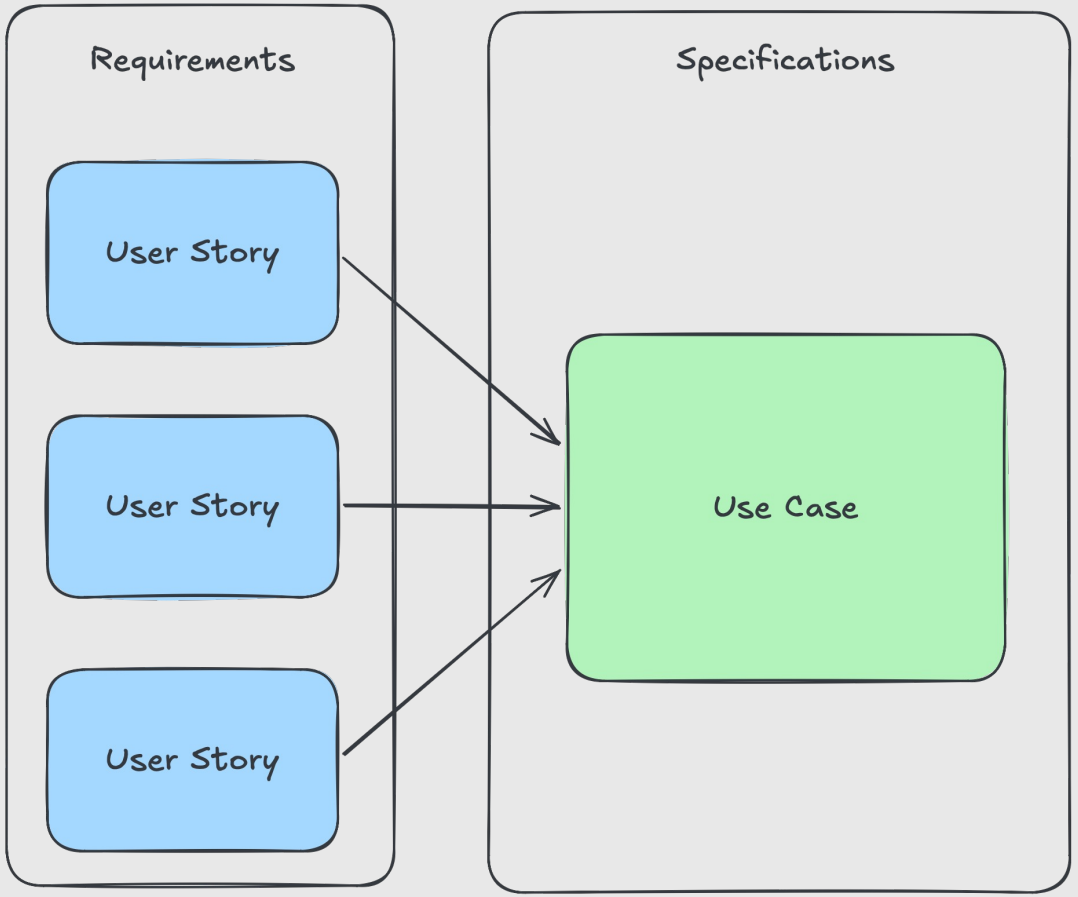
Preconditions

Main Success Scenario

Alternative Flows (optional)

Postconditions

How Are Use Cases and User Stories Related?



From Overview to Code

Levels of detail after **Alistair Cockburn** — every altitude has its own artifact and notation.



OVERVIEW — **Process map** context, not a flow — OVERVIEW

one process of the map



CLOUD — **Business process** across applications — BPMN

is an activity in the business process



KITE — **Process within one application** detailed process — BPMN

is an activity in the detailed process



SEA — **System Use Case** goal of an actor — MARKDOWN

absorbed into a step or a business rule



FISH — **Subfunction** no document of its own — PART OF UC

is implemented as



CLAM — **Code** implementation — REPOSITORY

contains / decomposes into

context, not strict decomposition

Behavior, Not Code

Code-level requirement

“Add POST /reservations/{id}/cancel and a column cancelled_at to the reservation table.”

- Describes the how
- Only developers can read it
- Outdated after the next refactoring
- Repeats what the framework already does

Behavior specification

“A guest can cancel a reservation up to 24 hours before arrival. The system then makes the room available again.”

- Describes the what
- Every stakeholder can read it
- Stays valid when the code changes
- Can be tested from the outside



https://leanpub.com/writing-use-cases-for-ai

Readable by Every Stakeholder

```
UC-002 Reserve Room
Actor: Guest
Precondition: The guest has found an available room

Main Success Scenario
1. The guest selects the room and the dates.
2. The system shows the total price.
3. The guest enters name and email and confirms.
4. The system saves the reservation and sends
   a confirmation email.

Alternative Flows
4a. The room was booked in the meantime. The system
    informs the guest and shows other rooms.

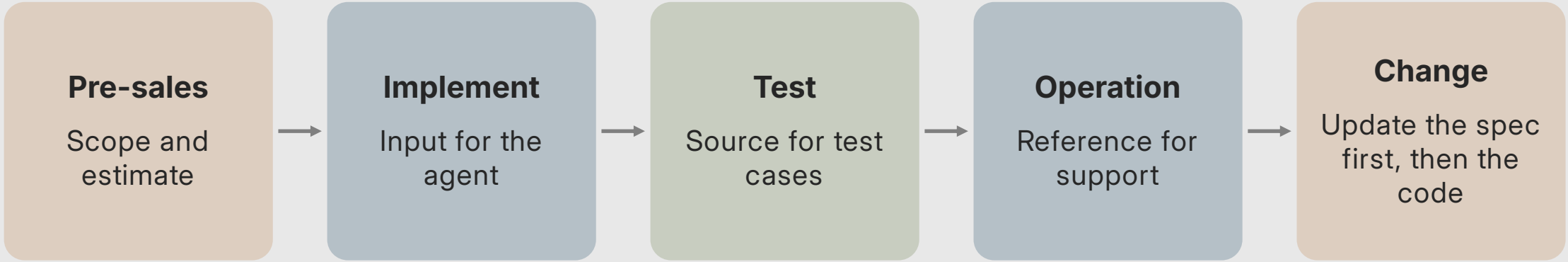
Business Rules
BR-001 A reservation is at least one night.
```

Who reads it?

- **End user:** is this how I work?
- **Business:** are the rules right?
- **Developer and agent:** what to build
- **Tester:** what to test

No screens, no APIs, no tables in a use case.

One Spec for the Whole Lifecycle



- **Spec-anchored:** the spec lives in the repository next to the code
- **One owner per spec:** otherwise the spec drifts away from the code
- **Status per use case:** Draft, Reviewed, Approved, Implemented, Tested, Done, Obsolete

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://code.claude.com/docs

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.

Where the Architecture Comes From

Starter project

Spring Boot, Vaadin, jOOQ, Flyway and H2/PostgreSQL are already set up in the starter project you clone in lab 0

Stack plugin

The plugin's skills (e.g. /implement in lab 7) decide the structure: view, jOOQ data access, DTO records, Flyway migrations, test layers

Existing code

The skills follow the patterns they find. The first use case sets the style for all the others

Your decisions

ADRs in docs/architecture/adr and constraints in requirements.md. /implement reads both

The stack plugin is executable architecture. Choosing a plugin is an architecture decision.

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 (Codespaces)



1. **Open the project in GitHub Codespaces**

   https://github.com/martinelli-workshops/sdd-lab

   - Code → Codespaces → Create codespace on main
   - Nothing to install on your machine

0. Set the Stage (Local)



1. **Prerequisites:** JDK 25, Claude Code, Docker (only for the postgresql branch)

2. **Clone the project**
   https://github.com/martinelli-workshops/sdd-lab

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

0. Set the Stage (contd.)



4. Install the AI Unified Process plugins

```
/plugin marketplace add AI-Unified-Process/marketplace

/plugin install aiup-core@ai-unified-process-marketplace

/plugin install aiup-vaadin-jooq@ai-unified-process-marketplace
```

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. Review the Specifications



1. Run **`/spec-review`**

2. Optional: fix the findings in **`docs/`**
   - Which findings surprised you?

AI4RE Prompt Guide

What it is

Free prompt templates for requirements engineering, written by the IREB special interest group #AIREB

- 23 prompts, and the guide keeps growing
- Works with any LLM; examples made with Copilot
- For inspiration and discussion, not a standard

Each prompt guide contains

- The RE task, the AI approach, and a conclusion
- A prompt template with variables to configure
- Pitfalls and tips, each with a mitigation
- A filled example with sample output

Two rules: never put sensitive data in a prompt, and always validate the output before you use it.

https://ireb.atlassian.net/wiki/spaces/airebpromptguide

23 Prompts Along the RE Activities

Elicitation
7 prompts

Explore
Introduce a new domain
Identify stakeholders
Prepare interview
Plan workshop

Transcribe
Refine transcription
Transcribe whiteboard images

Extract
Identify requirements

Documentation
9 prompts

Formulate
Specify requirement
Generate synonym table
Decompose requirement

Translate
Model from text
Text from model
Explain in simple language
Generate persona
Generate test cases
Requirements from source code

Validation
4 prompts

Verify
Compare spec with source code
Analyze requirements consistency
Check model consistency

Validate
Generate wireframe

Management
3 prompts

Align
Analyze change request impact

Assign
Tag requirements
Trace requirements

Anatomy of a Prompt Template

Prompt Guide: Specify Requirement

```
AI's role: Requirements specification expert
My role:   Requirements engineer
Purpose:   Requirement formulated for readability
           and comprehensibility
Task:      Optimize the formulation of the
           following requirement.
Formulation standard: [FORMULATION]
Quality standard:     [QUALITY]
Step 1: Identify the standard of the input.
Step 2: If another standard fits better, stop
        and ask which one to use.
Step 3: Formulate the requirement.
Step 4: Validate it against the quality standard;
        ask for missing information.
Format: [FORMAT]
Input:  [REQUIREMENT]
```

Building blocks

- **Roles:** who the AI is and who you are
- **Purpose and task:** what good output looks like
- **Variables:** user story, use case, IEEE, SOPHIST; INVEST, ISO/IEC/IEEE 29148, IREB CPRE
- **Checkpoint:** stop and ask instead of guessing
- **Format:** for example a table with a quote from the source

Our skills use the same structure: role, steps, template and output format.

From Prompt Template to Skill

AI4RE Prompt Guide		AI Unified Process skill	
Identify requirements	→	`/requirements`	Lab 1
Generate model from text	→	`/entity-model, /use-case-diagram`	Lab 2–3
Specify requirement	→	`/use-case-spec`	Lab 4
Analyze requirements consistency	→	`/spec-review`	Lab 5
Generate test cases	→	`/test-case`	
Compare spec with source code	→	`/coverage-check`	Lab 9
Analyze change request impact	→	Change a requirement	Lab 10
Requirements from source code	→	`/reverse-engineer`	Lab 13

Ready for Implementation?

- Every functional requirement is **covered by a use case**
- **`/spec-review`** shows no blocking findings
- The **customer has reviewed** the main use cases
- Constraints for runtime, monitoring and testing are written down
- Key decisions are written as ADRs
- Each open question has an owner

6. Create Database Migrations



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

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

3. Run **`./mvnw compile`**
   Flyway applies the migrations, then jOOQ generates the Java classes from the schema. /implement needs these classes.

7. 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

8. UI Test (Browserless)



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

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

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

9. Check Coverage



1. Run **`/coverage-check UC-###`**

2. Review the matrix: which steps and rules have no code or no test?

3. Close the gaps: add the missing tests or fix the spec
4. Note your token usage for this use case, from spec to test

When Requirements Change

- **Change the spec first**
  - Update the requirement, the use case or the business rule, not the code
- **Update the code from the spec, don’t hand-patch**
  - The agent changes the code based on the new spec
- **Let the tests show the impact**
  - Failing tests point to code that no longer matches the spec
- **Review spec and code together**
  - One pull request with the spec diff and the code diff
- **Check the coverage again**
  - /coverage-check shows where code and spec drift apart

10. Change a Requirement



1. Change a business rule in **`docs/use_cases/UC-###.md`**, e.g. a reservation is at most 14 nights

2. Run **`/implement UC-###`** again and review the diff

3. Run **`/browserless-test UC-###`** to update the tests, then **`./mvnw test`**

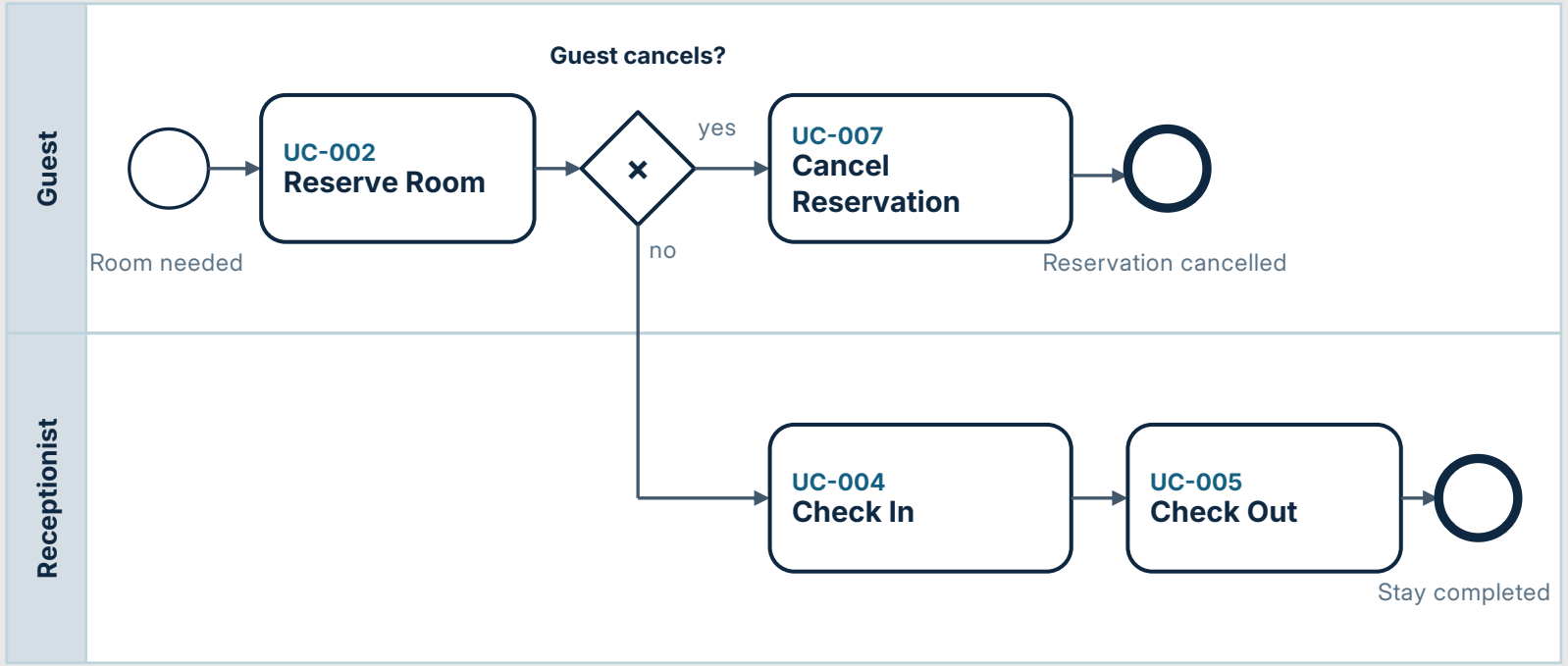
4. Run **`/coverage-check UC-###`** and check that nothing drifted

Bonus:
From Process to E2E Tests





From Process to Test Cases



`/test-case BP-001`

TC-001 `stay-completed.md`

Reserve Room
→ Guest cancels? no
→ Check In
→ Check Out

TC-002 `reservation-cancelled.md`

Reserve Room
→ Guest cancels? yes
→ Cancel Reservation

Activity = use case
The name carries the id, e.g. UC-002 Reserve Room

Lane = role
The lanes become the roles of the test case

Path = test case
Test data forces every gateway decision

Bonus 1: Model the Business Process



1. Run **`/business-process`**
   The agent drafts docs/processes/BP-###-.bpmn from the use cases. Open it in a BPMN modeler, e.g. demo.bpmn.io*
2. Review the diagram: every activity is a use case, e.g. **`UC-002 Reserve Room`**
   One lane per role, e.g. Guest and Receptionist
3. Run **`/spec-review`**
   It reports activities without a use case

Bonus 2: Generate Test Cases



1. Run **`/test-case BP-###`**
   One test case per path from the start event to an end event
2. Review **`docs/test_cases/TC-###`** and make adjustments if needed
   The test data must force every gateway decision
3. Change the process and run **`/test-case BP-###`** again
   Existing test cases are updated, removed paths become Obsolete

Bonus 3: 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/test/java`** and make adjustments if needed

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

Hardening the Harness

The 4+1 View Model

View	Question it answers	In AI Unified Process / HRS
Logical	Which concepts and responsibilities?	Entity model, domain modules
Development	How is the code organized?	Packages and modules, mostly set by the stack plugin
Process	What happens at runtime?	Transactions, emails, integration, concurrency
Physical	Where does it run?	Kubernetes, database, SSO (C-001, C-007)
+1 Scenarios	**Which use cases drive and validate the design?**	**The use cases UC-*.md. We already have them!**

Source: Philippe Kruchten, "The 4+1 View Model of Architecture", IEEE Software, 1995

Guidelines, Guardrails, Architecture

Guidelines

Tell the agent how

- CLAUDE.md or AGENTS.md
- Coding conventions
- Skills with stack knowledge

Soft: the agent can ignore them

Guardrails

Stop the agent when it is wrong

- Hooks before and after tool use
- ArchUnit and static analysis
- Tests and quality gates in CI
- Permissions and sandbox

Hard: enforced automatically

Architecture documents

Explain the decisions

- Architecture document (4+1 views)
- Architecture Decision Records (ADR)
- Module boundaries

Context: the why behind the code

Where It Lives in the Repository

```
CLAUDE.md                  # short rules, links to the rest
docs/
  vision.md
  requirements.md          # FRs, NFRs, constraints
  entity_model.md
  use_cases.puml
  use_cases/UC-001.md      # +1: scenarios view
  test_cases/TC-001.md
  architecture/            # the other 4 views
    logical.md
    development.md         # partly in the stack plugin
    process.md
    physical.md
    adr/ADR-001.md
.claude/
  settings.json            # permissions, hooks
  skills/                  # team skills
```

Rules of thumb

- CLAUDE.md is loaded in every turn, so keep it short
- Point to the other files, the agent reads them when needed
- Everything is versioned with the code
- Changes are reviewed in pull requests

Turn Guidelines into Guardrails

- **If a rule matters, enforce it**
  - A rule in CLAUDE.md is a wish, a failing build is a fact
- **Architecture rules as tests**
  - ArchUnit checks layers, packages and dependencies
- **Hooks give fast feedback**
  - Run formatter, compiler and tests after each edit of the agent
- **Quality gates in CI**
  - Static analysis, dependency check, test coverage
- **The agent fixes its own errors**
  - It sees the failing check and corrects it in the same session

Example: Sensors in the PetClinic

Sensor	Checks
`UseCaseTraceabilityTest`	Every @UseCase points at a real use case, flow and business rule. Done or Tested needs a test for every flow and rule
`TestCaseTraceabilityTest`	Every journey test matches its test case. Automated only on use cases that are Done
`BusinessRuleTraceabilityTest`	Shared rules (GR-###) and the use cases that use them agree in both directions
`ArchitectureTest`	Layers, packages, stereotypes and transaction boundary (ArchUnit)
`TestLayerConventionsTest`	*Test runs in test, *IT in verify, so no test silently never runs

Hooks Guard the Session

- **Stop hook**
  - No end of turn while src/, docs/ or pom.xml changed after the last green sensor run
- **Status guard**
  - Done or Tested only after the uc-coverage agent has audited the use case
- **Positive evidence**
  - Fresh Surefire reports count, console output does not
- **Fail open**
  - A broken hook must never block the session
- **Rule of thumb**
  - If you can check it by reading the repository, write a test. Only session rules become hooks

11. Run the Sensors



1. Clone **`https://github.com/AI-Unified-Process/petclinic`**, a PetClinic that already has specs and sensors, and run **`./mvnw test -Dgroups=sensor`**

2. Rename an alternative flow in **`docs/use_cases/UC-###.md`** and run the sensors again

3. Ask Claude to set a use case to **`Done`** and watch the hooks refuse it without **`/coverage-check`**

4. Read ADR-007, ADR-010 and ADR-011 in **`docs/architecture/adr`**

Enterprise Constraints Are Input

Runtime

Language and framework versions, platform, hosting, external APIs

Monitoring

Log format, tracing, metrics and alerts per use case

Testing layers

Which tests are required: unit, integration, contract, end-to-end

Security

Single sign-on, roles, data protection, secrets

Integration

ERP, PIM, payment: interfaces, limits, error handling

Operation

Performance, availability, release process

If it is not in the specification, the agent guesses. And it guesses the default.

Example: Constraints in requirements.md

Constraints (C)

ID	Constraint	Category
C-001	Runs on the hotel's Kubernetes	Operational
C-002	Prices only from price list system	Technical
C-003	JSON logs with a correlation id	Operational
C-004	Success + error metric per use case	Operational
C-005	Unit test for each business rule	Technical
C-006	Integration test for each use case	Technical
C-007	Staff login via company SSO (OIDC)	Technical

Why like this?

- Numbered, so use cases and tests can point to them
- Short and testable statements
- Part of the requirements catalog
- Same structure in every project, only the values change

12. Add Constraints and Context



1. Back in hrs, open **`docs/requirements.md`** and add one constraint each for runtime, monitoring and testing

2. Add a short section to **`CLAUDE.md`** that points to the requirements and architecture

3. Write **`docs/architecture/adr/ADR-001.md`** for a decision the plugin made for you, e.g. jOOQ instead of JPA

4. Reference one constraint in a use case

Harness Engineering in Practice

- **Start small**
  - One CLAUDE.md, one skill, one quality gate
- **A mistake twice is a gap in the harness**
  - Fix the context or add a check, not only the code
- **Behaviour is the weakest harness**
  - Specs and test cases fill the gap
- **Make the code harnessable**
  - Strong types, clear module boundaries, known frameworks
- **Share it like code**
  - Versioned and reviewed, in a team marketplace

Brownfield Projects

From Code to Specs

- **Start from what you have**
  - Code, database schema, tests, old documents
- **Reverse engineer the specs**
  - /reverse-engineer creates the use case diagram, the use cases and the entity model
- **Review with the business**
  - The code shows what the system does, not what it should do
- **One module at a time**
  - A large monolith does not fit into the context
- **Then work spec-first**
  - Every new change starts in the spec

13. Reverse Engineering



1. Clone the original Spring PetClinic **`github.com/spring-projects/spring-petclinic`**, which has no specs yet, unlike the one in lab 11

2. Run **`/reverse-engineer`**

3. Review the generated files in **`docs/`** and run **`/spec-review`**

4. Change one use case and run **`/implement UC-###`**

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
- **Governance gap**
  - 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

Where It Helps and Where It Doesn’t

Where it helps

“The rules are clear and the system has to live for years.”

- Business applications with many rules
- Long-lived and regulated systems
- Brownfield modernization
- Teams that need a shared understanding

Where it does not help much

“We do not know yet what we want.”

- Quick prototypes and experiments
- Algorithms and performance tuning
- Visual design, look and feel

Separate the What from the How

What: stack independent

- Vision
- Requirements
- Use cases and business rules
- Entity model (logical)
- Test cases

Written with the customer

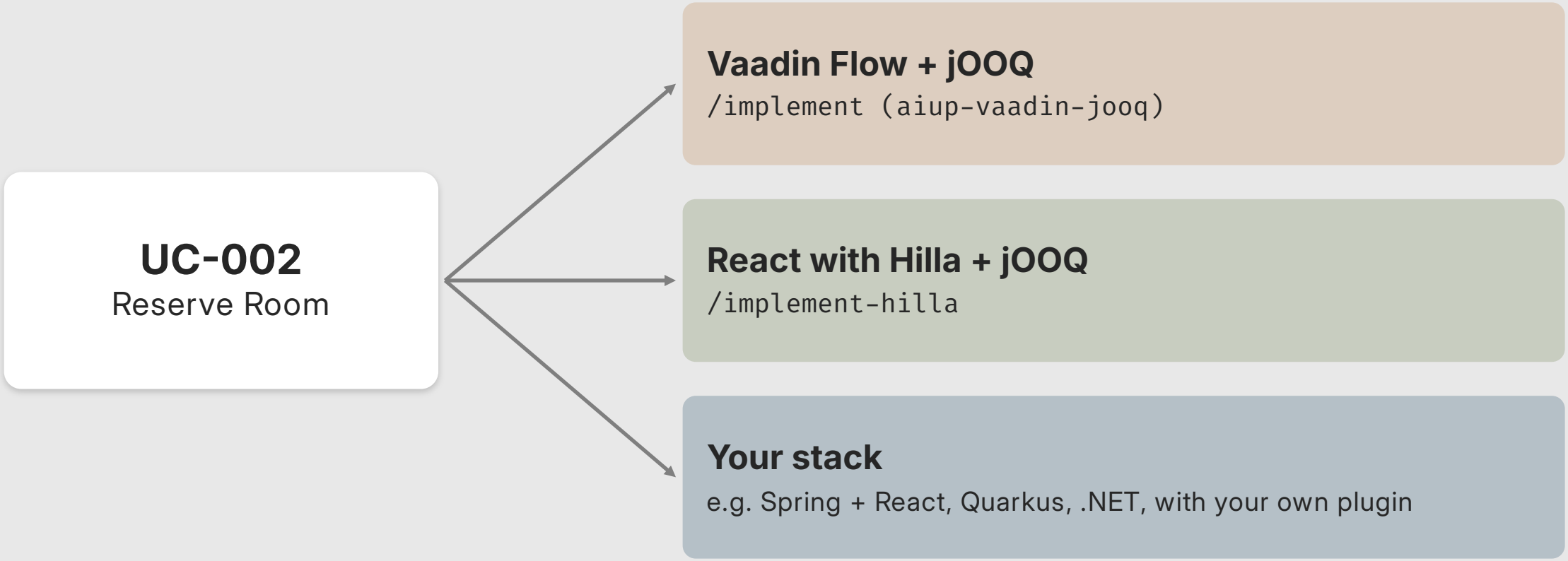
How: stack specific

- Architecture: 4+1 views, ADRs and the stack plugin
- Coding guidelines
- Skills and plugins for the stack
- Runtime constraints

Written by the engineers

Rule: no screens, no APIs, no tables in a use case.

One Spec, Many Stacks



The specification stays. The harness, and with it the architecture, changes.

How to Try It on Your Own Stack

- **Pick one real use case**
  - Small, but with real business rules
- **Write the spec first**
  - Vision, requirements, entity model, one use case
- **Build a minimal harness**
  - CLAUDE.md, one skill for your stack, one quality gate
- **Reuse what exists**
  - AI Unified Process marketplace: unifiedprocess.ai
- **Measure**
  - Rework, review findings and tokens per use case

Conclusion

- **Specs and the harness help** to **reduce** non-determinism and 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

