The Evolution of Coding

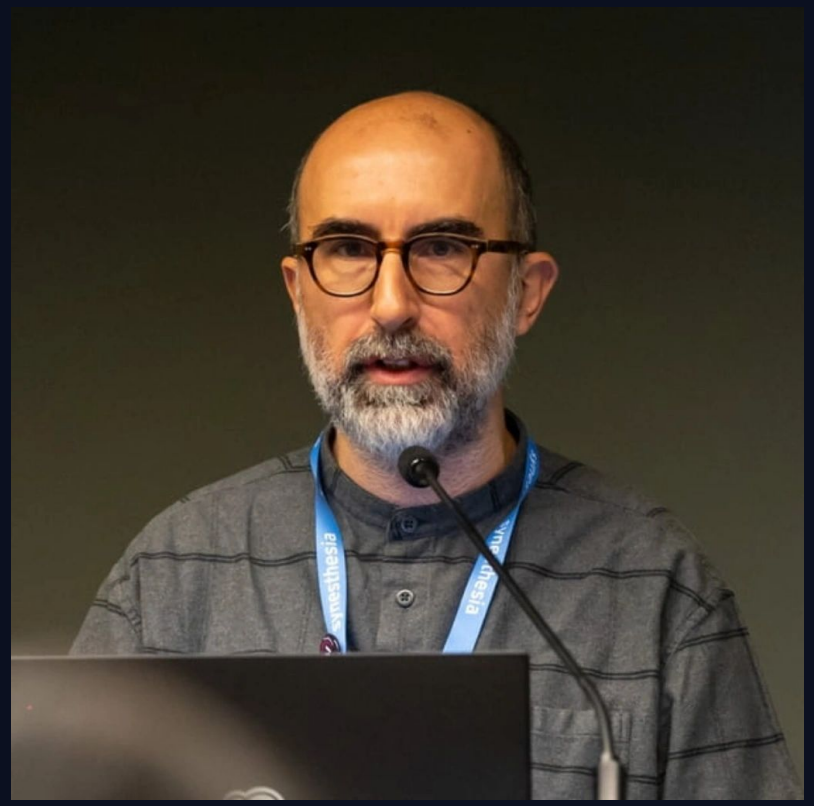
Spec-Driven Development in the Age of Agentic AI

Enrico Zimuel, Adj. Professor at Unito & Uniroma3

Ai Heroes 2026 - June 29, Milan

About me

- Adj. Prof. of *Machine Learning* and *Computer Science* at University of Turin
- Adj. Prof. of *Agentic AI and RAG architectures* at University of Roma Tre
- Former *Tech Lead and Principal Software Engineer* at Elastic (USA)
- Former *Senior Software Engineer* at Zend (USA)
- Former CTO and Director of Engineers (Italy)
- TEDx and international speaker in +140 conferences
- Author of articles and books, e.g. “Anatomia di una mente artificiale” libroia.it



More info: www.zimuel.it

Why this talk now?

Agentic AI is changing the unit of software work

- Generative AI moved from autocomplete to autonomous task execution
- Developers increasingly delegate implementation loops to tools
- The bottleneck shifts from:
  “Can we code it?”
  to:
  “Did we specify it well?”

The future skill is not less engineering, it is more explicit engineering

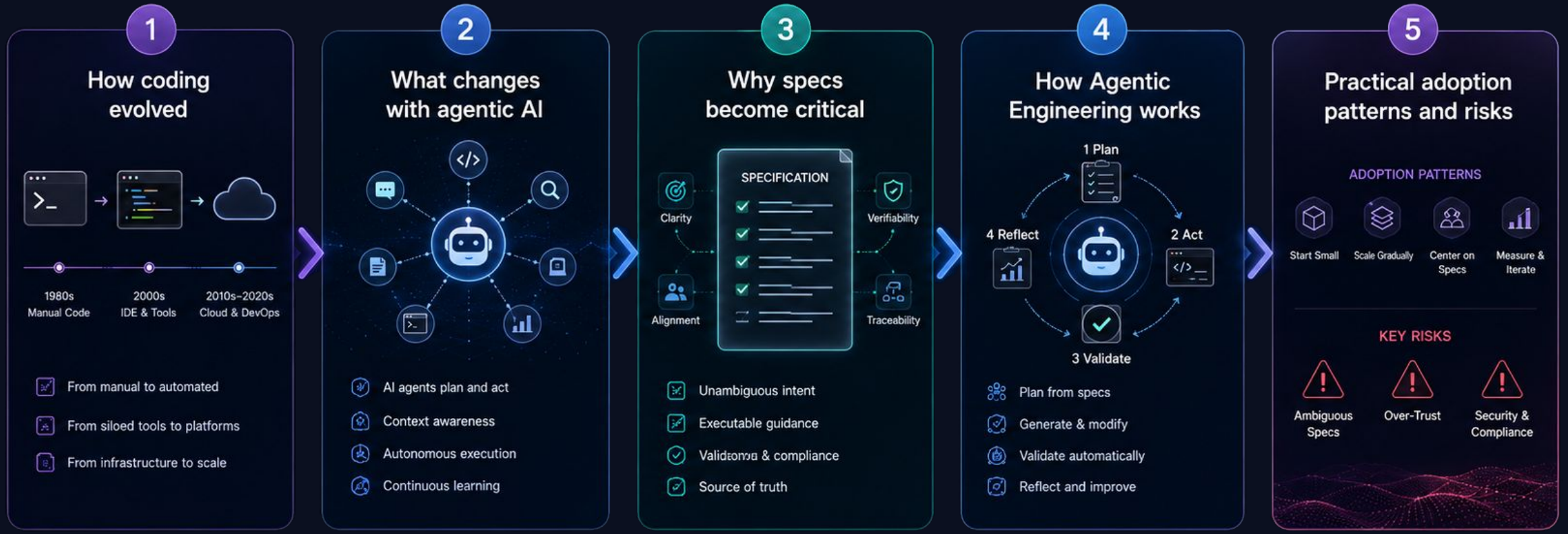
Talk thesis

- AI-assisted coding improves speed at the line and function level
- Agentic engineering changes the workflow, feedback loops, and accountability model
- Spec-Driven Development provides the control surface for agentic systems

Specs become the interface between human intent and AI execution

Agenda

What we'll cover today



Better Specs → Smarter Agents → Better Software

01

How coding evolved

From machine instructions to reusable abstractions to AI-mediated development

A short history of abstraction

Every coding era raised the level of intent

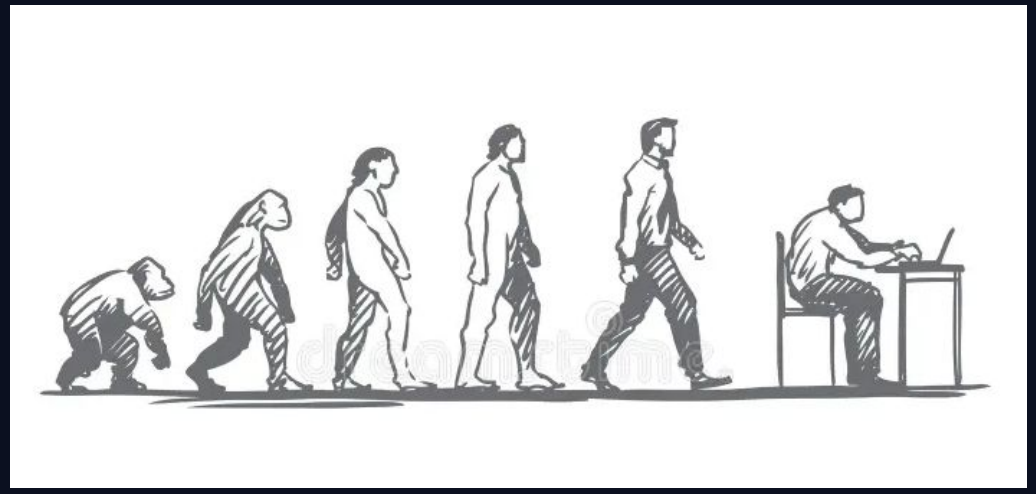
- **Assembly**: tell the machine exactly what to do
- **Compilers:** use an high level language to be translated in assembly
- **Languages and frameworks**: express patterns at higher levels
- **Cloud and DevOps:** define systems, environments, and pipelines
- **AI:** describe outcomes and constraints, then supervise execution

Software development has always been an abstraction story

The developer role has kept moving upward

The work shifts as tooling improves

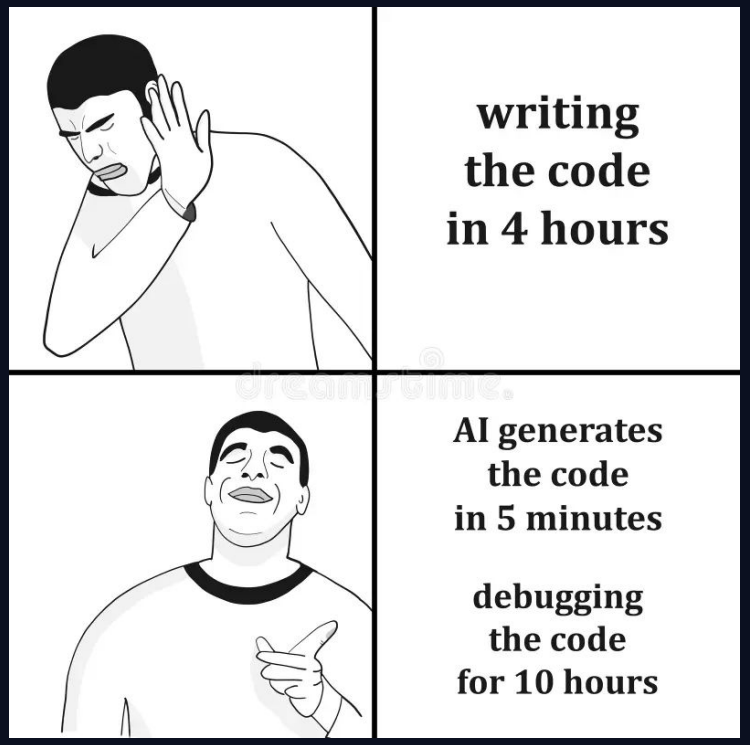
- From handcrafting every instruction
- To composing libraries and services
- To orchestrating delivery systems
- To designing verifiable intent for AI agents



AI-assisted coding: the first wave

The assistant lives inside the implementation loop

- Suggests code, tests, snippets, refactors, and explanations
- Works best when the developer owns the plan and validation
- Optimizes local productivity rather than system-level outcomes



The evolution of AI code generation

In less than 2 years we moved from 33% to 88% of accuracy

SWE-bench scores from Aug 2024 to June 2026 (source)

What AI-assisted coding still leaves unresolved

Fast code is not automatically correct code

- Ambiguous intent can generate plausible but wrong implementations
- Local suggestions can miss architecture, policy, or domain constraints
- Validation often remains manual, late, or implicit

Speed without specification can amplify uncertainty

02

What changes with Agentic AI

Agentic AI changes the loop: planning, acting, testing, and revising

What makes AI “agentic”?

A working definition for engineering contexts

- It can decompose goals into tasks
- It can use tools: code, shell, tests, docs, issue trackers, APIs
- It can evaluate results and iterate
- It can maintain context across a workflow

An agent is not just a model, it is a model embedded in an action loop

AI-assisted coding vs. Agentic Engineering

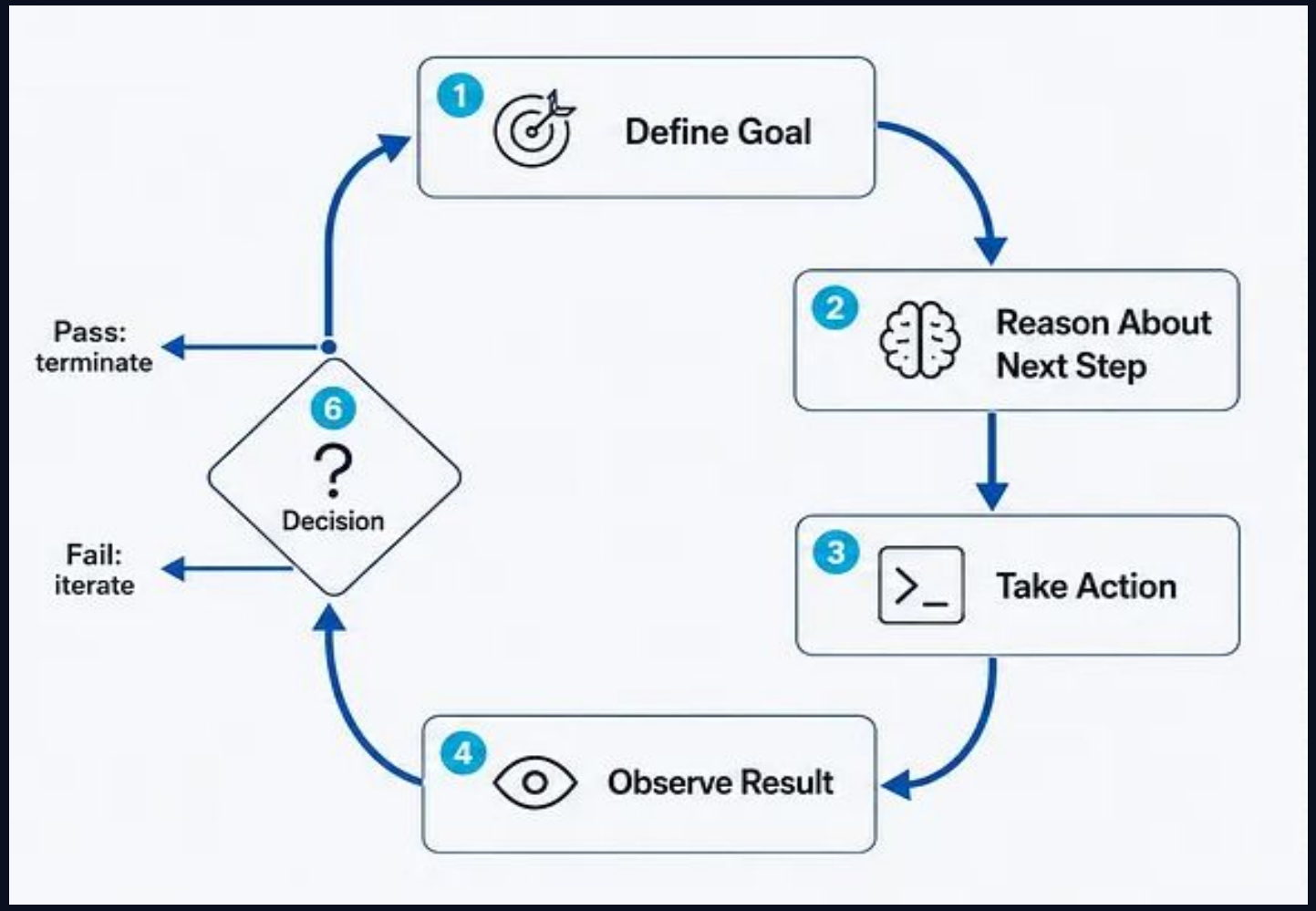
The core shift you wanted to discuss

- **AI-assisted coding**: human drives, AI suggests
- **Agentic Engineering**: human specifies, agent executes, evidence decides
- The deliverable expands from code to code plus tests, traces, decisions, and verification

The center of gravity moves from prompt-to-code to spec-to-system

The new engineering loop

Agentic workflows are iterative and evidence-based



The loop is only trustworthy when the success condition is explicit

Source of the image

Where agents create leverage

Tasks that benefit from autonomy

- Feature scaffolding across files and services
- Migration and refactoring work
- Test generation and failure triage
- Documentation, API clients, and repetitive integration work

Agents are strongest when goals are bounded and verification is available

Where agents create new risk

Autonomy increases the need for guardrails

- They can confidently satisfy the wrong objective
- They can change too much surface area
- They can pass shallow tests while violating hidden requirements
- They can make undocumented architectural decisions

Autonomy without specification becomes uncontrolled delegation

03

Why specs become critical

The specification becomes the source of intent, constraint, and verification

What is a “spec” in this context?

Not a document for a shelf; a machine-usable contract

- **Intent**: what outcome should exist
- **Scope**: what is included and excluded
- **Constraints**: architecture, security, performance, compliance
- **Acceptance criteria**: how success is proven

A spec is executable intent, even before it becomes executable code

Why specs matter more with AI agents

Agents need a stable target and a verifiable finish line

- They reduce ambiguity in planning and implementation
- They support automated test creation and review
- They preserve human intent across long workflows
- They provide auditability for decisions and changes

The better the spec, the smaller the trust gap

Example: spec-kit

https://github.com/github/spec-kit

- An open source toolkit from Github that allows you to focus on product scenarios and predictable outcomes instead of vibe coding every piece from scratch
- **Spec-Driven Development** flips the script on traditional software development
- For decades, code has been king — specifications were just scaffolding we built and discarded once the "real work" of coding began
- **Driven Development** changes this: specifications become executable, directly generating working implementations rather than just guiding them

A typical spec-kit workflow

1. **/speckit.constitution**
   Define project principles, constraints, quality rules, coding standards.

2. **/speckit.specify**
   Describe the feature in terms of intent, users, behavior, and outcomes.

3. **/speckit.clarify**
   Let the agent find ambiguities and ask questions.

4. **/speckit.checklist**
   Generate checks to validate the quality and completeness of the spec.

5. **/speckit.plan**
   Define architecture, technology choices, data model, integration strategy.

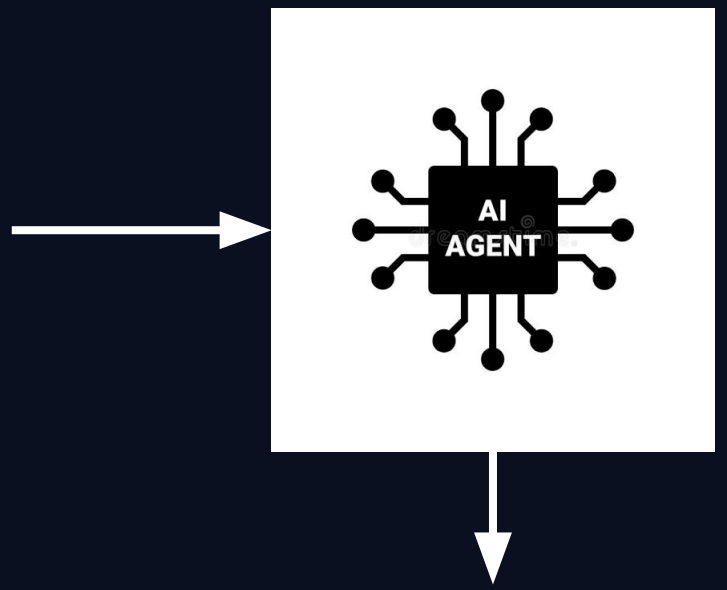
6. **/speckit.tasks**
   Break the plan into actionable implementation tasks.

7. **/speckit.analyze**
   Check consistency across constitution, spec, plan, and tasks.

8. **/speckit.implement**
   Let the agent implement the tasks.

Spec-kit outcome

- A markdown that contains:
  - **Feature**
  - **User goal**
  - **Functional requirements**
  - **Constraints**
  - **Acceptance criteria**



- The Agent can derive:
  - **Plan**
  - **Tasks**

From prompt engineering to spec engineering

The durable artifact is not the prompt

- Prompts are often conversational, temporary, and under-specified
- Specs are versioned, reviewable, testable, and reusable
- The prompt becomes a delivery mechanism for the spec, not the source of truth

Prompting is interaction; specification is engineering

04

How Agentic Engineering works

How teams structure work when agents become collaborators

Agentic Engineering workflow

A practical end-to-end flow

- Human creates or reviews the spec
- Agent proposes plan, assumptions, and affected files
- Agent implements in small, reviewable increments
- Automated checks produce evidence
- Human approves decisions and merges with traceability

Humans own intent and accountability; agents own execution loops

New artifacts in the workflow

Code is only one part of the output

- Spec files and decision records
- Generated test cases and evaluation reports
- Agent plans, logs, and change summaries
- Policy checks and security findings
- Human review comments linked to acceptance criteria

The engineering record expands beyond commits

Human-in-the-loop control points

Where judgment must remain explicit

- Approve scope and non-goals before implementation
- Review assumptions before large changes
- Require evidence for acceptance criteria
- Escalate ambiguous requirements to humans
- Merge only when traceability is clear

Human oversight must be part of the design, not a fallback when things go wrong

Metrics that matter

How to measure spec-driven agentic work

- Cycle time from accepted spec to verified change
- Defect leakage from ambiguous or missing criteria
- Review effort per change size
- Automated test coverage of acceptance criteria
- Rework caused by incorrect assumptions

Measure the quality of intent, not only the speed of output

Adoption roadmap

A staged path for teams

- Start with bounded tasks: tests, docs, refactors, small features
- Standardize spec templates and acceptance criteria
- Add automated checks and policy gates
- Introduce agent execution with traceability
- Scale to larger workflows after feedback loops are reliable

Before granting autonomy, define how outcomes will be verified

05

Practical adoptions, patterns and risks

From Promise to Practice: Scaling with Control

What changes for developers

The skills that become more valuable

- Requirement decomposition and domain modeling
- Writing precise acceptance criteria
- Designing testable interfaces and constraints
- Reviewing generated plans and evidence
- Maintaining architecture and socio-technical context

The best developers become intent designers and evidence reviewers

Engineers are becoming managers of AI agents

What we'll talk today

- Adoption Patterns: The **Agentic Engineering Loop**
  - Human approval before implementation
  - Small, reversible changes
  - Test-first or acceptance-criteria-first workflows
  - Traceability from requirement → task → code → test
  - Agent output reviewed like a pull request, not accepted as truth

Key Risks: Where Agentic Engineering Break

Agentic AI does not remove engineering risk

1. **Ambiguous specs**
2. **Over-trust**
3. **Security & Compliance**
4. **Loss of human-in-the-loop**

Source: https://www.youtube.com/watch?v=m8l2p3nQtR4

Key Risks: Ambiguous specs

- **Ambiguous specs:**
  - Vague requirements produce plausible but wrong implementations
  - Missing constraints become hidden assumptions
  - Edge cases are often invented or ignored by the agent
- **Mitigation:**
  - Use clarification steps, acceptance criteria, examples, non-goals, and explicit constraints

Key Risks: Over-trust

- **Over-trust:**
  - AI-generated code can appear correct but be subtly wrong
  - Teams may skip review because output is fast and confident
  - Hallucinated APIs, libraries, or architecture choices can enter production
  - Errors scale quickly when agents automate repetitive changes
- **Mitigation:**
  - Require validation loops: tests, code review, static analysis, benchmarks, and explicit confidence checks

Key Risks: Security & Compliance

- **Security & Compliance**
  - Agents may expose secrets, logs, or sensitive data
  - Generated code can introduce insecure dependencies or patterns
  - Compliance requirements may be missed without explicit constraints
  - Automated changes can bypass normal governance controls
- **Mitigation**
  - Embed security policies into the workflow: least-privilege access, secret scanning, dependency checks, audit logs, and compliance gates

Key Risks: Loss of human-in-the-loop

- **Loss of human-in-the-loop**
  - Humans may become reviewers of outcomes only, not decisions
  - Critical trade-offs can be made without product or engineering judgment
  - Accountability becomes unclear when agents act autonomously
  - Team learning may decline if agents hide implementation reasoning
- **Mitigation**
  - Keep humans in control of intent, approval, and exceptions: require checkpoints for architecture, risk, production deploys, and irreversible actions

Thanks!

Contact: enrico@zimuel.it

Follow me on Linkedin: