



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

Chapter 04 · Part 2

REQUIREMENTS ENGINEERING

Where We Left Off (Part 1)

In Part 1, you learned:

- What requirements are (functional vs non-functional)
- Who cares about them (stakeholders)
- What makes them good (INVEST, testability)
- How to document them (User Stories, GitHub Issues)
- How to link them to tests (traceability)

Today: The *process* of requirements engineering.

The Harder Questions

Part 1 didn't answer:

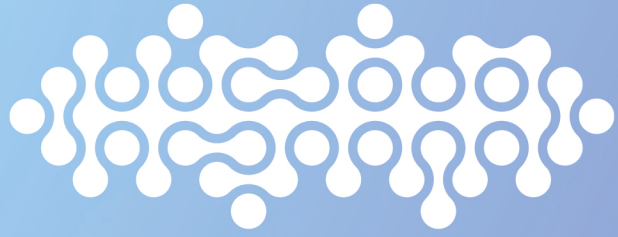
- How do requirements fit into the development workflow?
- How do you actually *discover* what stakeholders need?
- What happens when requirements change mid-project?
- Is there a better way than "write everything upfront"?

Today's focus: From elicitation to change management, and why Agile emerged.

Learning Objectives

By the end of this lecture, you will:

1. **See requirements from different perspectives** (developer, PM, product manager, founder)
2. **Apply elicitation techniques** through stakeholder interviews
3. **Distinguish bugs from change requests** and why this matters
4. **Use POC-driven approaches** to discover requirements
5. **Understand traditional models** (Waterfall, V-Model, V-Modell XT)
6. **Preview Agile methodology** as a response to uncertainty



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

Business Perspectives

1

The Developer's View

Questions:

- What exactly should this function do?
- What are the edge cases?
- How do I test this?

Focus: Precision, testability, implementation details

Example: "Display intersection point" → Algorithm, data structures, UI component

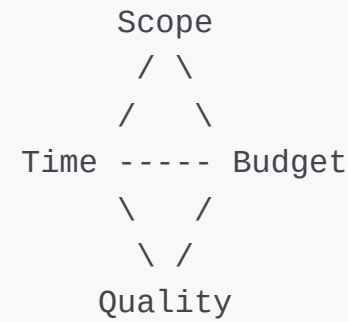
The Project Manager's View

Questions:

- How long will this take?
- What's the priority?
- What's the scope?

Focus: The Iron Triangle

"Good, Fast, Cheap - pick two."



Product Manager & Founder Views

Product Manager:

- Does this deliver value to users?
- What's the minimum to test?
- What should we build next?

Focus: Value delivery, prioritization

Startup Founder:

- Is there a market for this?
- Will people pay for it?
- Are we solving a real problem?

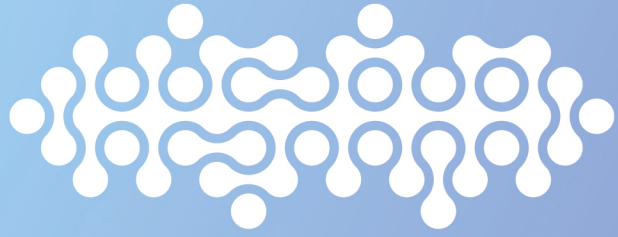
Focus: Product-market fit, validation

Same Requirement, Different Views

"Display intersection point"

Role	Sees It As
Developer	Algorithm, data structures, UI component
Project Manager	2 days effort, depends on chart library
Product Manager	Enables core workflow, high priority
Founder	Part of value proposition, differentiator

Key insight: Context shapes interpretation!



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

Eliciting Requirements

2

Elicitation Techniques

Technique	Best For	Watch Out For
Interviews	Individual perspectives	Leading questions
Workshops	Building consensus	Dominant voices
Observation	Actual workflow	Hawthorne effect
Surveys	Data from many users	Shallow answers
Prototyping	"I'll know it when I see it"	Prototype ≠ product

Good Questions vs Bad Questions

Bad (leading, closed):

- "You want blue, right?"
- "Is the system fast enough?"
- "Do you like the design?"

Good (open, exploratory):

- "Walk me through how you do this task."
- "What's the most frustrating part?"
- "What would success look like?"
- "What happens if this is wrong?"

The 5 Whys Technique

User: *"I need the intersection calculation to be faster."*

Why? "Because I'm waiting too long."

Why does waiting matter? "I have to process 50 profiles per day."

Why 50 per day? "That's how many road segments we survey."

Why not batch them? "I need results before the next measurement."

Why immediately? "To adjust the camera angle for the next segment."

Real requirement: Not "faster calculation" but "real-time feedback loop"!

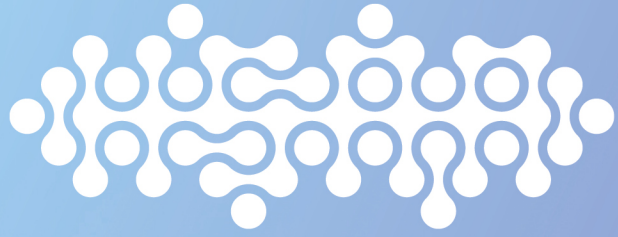
Practice: 5 Whys Exercise

Scenario: A user says "I need to export the chart as PDF."

In pairs (3 minutes):

1. One person plays the stakeholder
2. Other person asks "Why?" 5 times
3. Discover the *real* underlying need

What did you discover?



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

Bug vs Change Request

3

Definitions

Bug:

The software does not meet an existing, documented requirement.

Change Request (CR):

The stakeholder wants the software to do something different or additional to existing requirements.

The key word: *Documented*

Why This Distinction Matters

Aspect	Bug Fix	Change Request
Billing	Usually free (warranty)	Typically billable
Priority	Often high (broken promise)	Depends on business value
Scope	Part of original scope	Expands scope
Blame	Developer team	Neither (new requirement)
Timeline	Doesn't extend deadline	May extend deadline

The Gray Area

Customer says: "The intersection point is wrong!"

Decision tree:

```
Is expected behavior documented?  
├─ Yes → Does software match documentation?  
│   └─ Yes → Change Request (works as specified)  
│   └─ No → Bug (doesn't match spec)  
└─ No → Gray area - discussion needed  
    └─ Was it implied by context?
```

Real Example

"The intersection calculation returns wrong results for angles $> 180^\circ$."

- **If** requirement says "angle in range 0-360°": **Bug** (should handle 180+)
- **If** requirement says "angle in range 0-180°": **Change Request** (new capability)
- **If** requirement doesn't specify: **Discussion needed**

Key insight: Good requirements documentation prevents these disputes.

Beyond Software Development

This distinction has legal and financial consequences!

Perspective	Risk	Protection
As Developer	Customers claim "bugs" for free features	Detailed specs, signed docs
As Customer	Vendors charge CR for genuine bugs	Clear bug definitions, acceptance testing

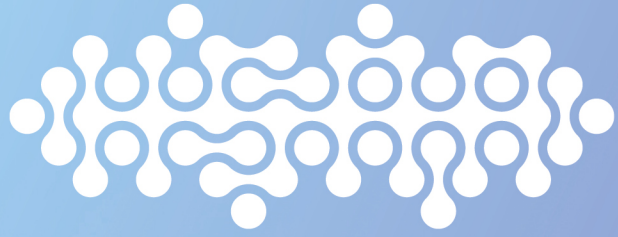
Real-world: Bug vs CR disputes → contract negotiations → legal cases

Quick Discussion

Scenario: The system crashes when loading a file with 1 million points. Documentation says nothing about file size limits.

Bug or Change Request?

How would you handle this?



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

POC-Driven Requirements

4

The Uncomfortable Truth

Customers don't know what they want until they see it.

This isn't a failure - it's human nature.

Traditional RE assumes:

Interview stakeholders → Write complete requirements → Build system → Deliver

Reality:

Stakeholders describe what they *think* they want → You build it → They say "Actually, I meant..."

Lean Startup: Build-Measure-Learn

The Lean Startup approach (Eric Ries) flips the process:

Idea → Build MVP → Measure → Learn → Idea → ...

MVP = Minimum Viable Product: The smallest thing you can build to test your hypothesis.

Example for Road Profile Viewer:

- MVP 1: Command-line script → Show to user
- MVP 2: Simple plot with hardcoded data → "Is this what you need?"
- MVP 3: Interactive angle selector → "Does this workflow work?"

Design Thinking

Design Thinking (IDEO, Stanford d.school):

Empathize → Define → Ideate → Prototype → Test
↑ _____ |

Key activities:

- **Empathize:** Observe users, experience their pain
- **Define:** Synthesize into problem statements
- **Ideate:** Generate many solutions
- **Prototype:** Build quick, cheap, disposable
- **Test:** Get feedback, refine understanding

Requirements emerge from this process!

Gen AI for Rapid Prototyping

Traditional prototyping: Build a prototype: 2-4 weeks → Show to customer → Iterate

Gen AI-assisted ("vibe coding"): Describe what you want → Working prototype in hours → Show to customer *same day*

The insight: AI lets you create POCs so fast that **requirements discovery becomes interactive.**

Example Workflow with Gen AI

Developer: "Create a Python GUI that loads CSV road points and displays them as a line chart with angle slider."

[AI generates code in 10 minutes]

Developer: *Shows to road engineer*

Engineer: "I need to see the camera position too."

Developer: "Add camera position marker to the code."

[AI updates code in 5 minutes]

Engineer: "Actually, I need to compare two profiles side by side..."

Requirements become conversations with working software!

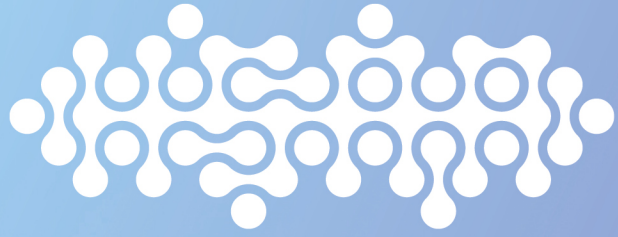
When POC-Driven Works Best

POC-driven is best for:

- High uncertainty / novel domains
- "I'll know it when I see it"
- Fast-moving markets
- Innovation contexts

Traditional RE still works for:

- Regulatory environments
- Well-understood domains
- Large integration systems
- Fixed-scope contracts



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

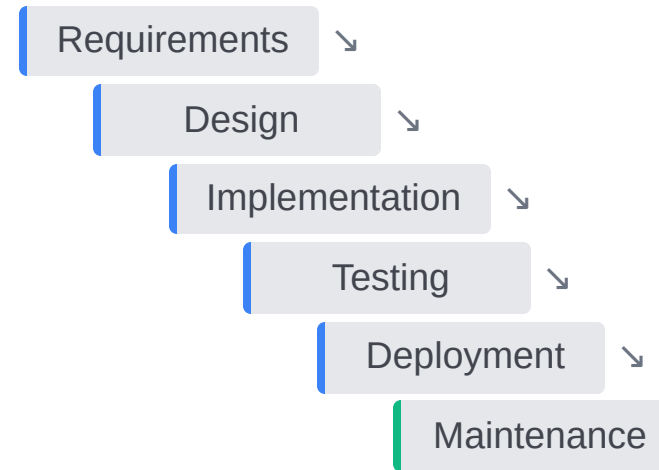
Traditional Development Models

5

The Waterfall Model

History:

- 1956: Benington's sequential phases
- 1970: Winston Royce's paper
- 1976: Bell coined "waterfall"



The Royce Irony

Fun fact: Winston Royce, whose 1970 paper is cited as the origin of Waterfall, actually *warned against* the pure sequential approach!

He called the basic waterfall "risky and inviting failure."

The rest of his paper described iterative improvements and feedback loops.

The industry adopted his diagram but ignored his warnings!

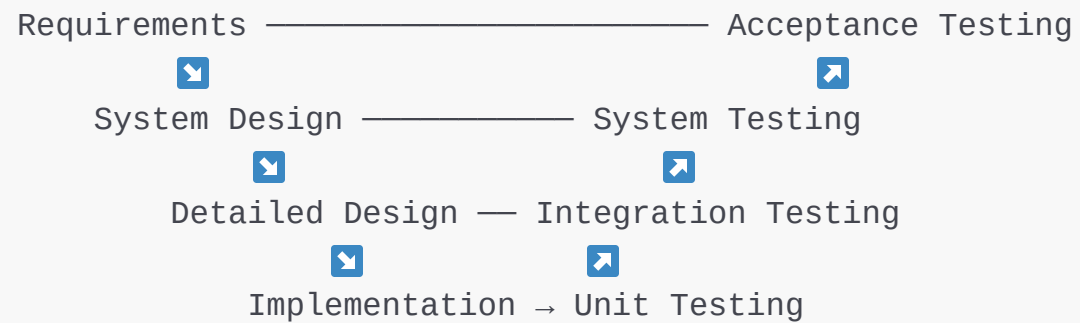
Waterfall Characteristics

- **Sequential phases:** Each phase must complete before next begins
- **Phase gates:** Formal reviews and sign-offs
- **Heavy documentation:** Complete documentation at each phase
- **Big Design Up Front (BDUF):** All requirements/design before coding
- **Change is expensive:** Late changes require revisiting earlier phases

Where it works: Stable requirements, regulated industries, fixed contracts

The V-Model

Key innovation: Each development phase has a corresponding test phase.



Test planning happens in parallel with development, not after!

V-Model: Testing Correspondence

Development Phase	Test Phase	What's Verified
Requirements Analysis	Acceptance Testing	System meets user needs
System Design	System Testing	System works as designed
Detailed Design	Integration Testing	Components work together
Implementation	Unit Testing	Each unit works correctly

Improvement: Tests planned early → defects found earlier

Still sequential: Working software only appears at the end

V-Modell XT: German Government Standard

If you work on German federal IT projects, this is mandatory!

- **1992:** Original V-Model for German Federal Ministry of Defense
- **2004:** V-Modell XT introduced ("eXtreme Tailoring")
- **2019:** Current version 2.3

XT means "Extreme Tailoring" - adaptable to different project types

V-Modell XT Components

Core elements:

- **Decision Points:** Project gates with required approvals
- **~20 Process Building Blocks:** PM, QA, Configuration Management, etc.
- **108 Defined Deliverables:** Each with defined structure
- **35 Defined Roles:** Clear responsibilities
- **Project Types:** Including Agile variants!

Real applications: Federal IT, defense, aerospace, medical devices, automotive

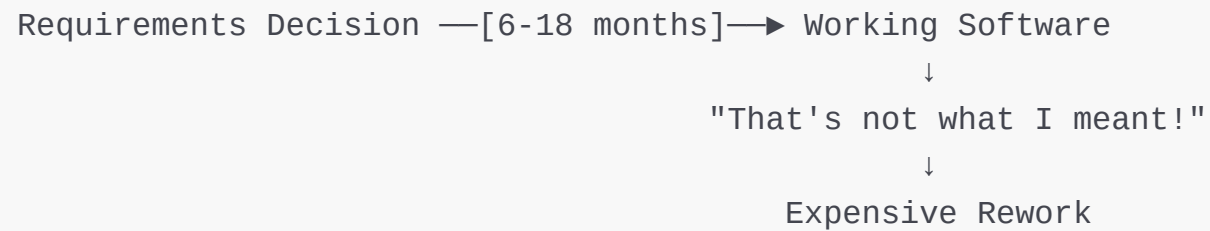
Why V-Modell XT Matters for You

Even if you don't plan to work in government:

- Many large German companies use it (Siemens, Bosch, SAP)
- Shows how regulated industries handle software development
- Makes Agile's contrast clearer
- **Job interviews in German industry often reference it!**

The Problem with Sequential Models

The Late Feedback Problem:



Working software appears only at the end!

The Cost of Change (Boehm, 1981)

Phase	Relative Cost
Requirements	1x
Design	3-6x
Coding	10x
Testing	15-40x
Production	30-100x

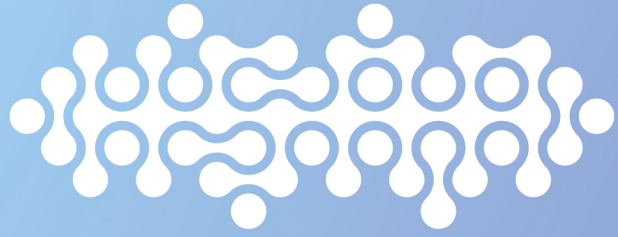
Why? A requirements error in production means: change docs + update design + rewrite code + re-test + redeploy + handle customer impact + data migration...

The Fundamental Tensions

Assumption	Reality
"We can know all requirements upfront"	Users don't know what they want until they see it
"We can design completely before coding"	Design decisions need implementation feedback
"We can build it right the first time"	Software is learned, not manufactured

The Paradox:

- If we plan more → We spend time on plans that will change
- If we plan less → We build the wrong thing



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

Preview of Agile

6

The Agile Manifesto (Preview)

We are uncovering better ways of developing software.

We have come to value:

- **Individuals and interactions** over processes and tools
- **Working software** over comprehensive documentation
- **Customer collaboration** over contract negotiation
- **Responding to change** over following a plan

While there is value in the items on the right, we value the items on the left more.

Agile Requirements

In Agile development:

- Requirements captured as **User Stories** (small, negotiable)
- Stories live in a **Backlog** (prioritized list)
- Work happens in **Sprints** (1-4 weeks)
- Requirements refined **continuously** (backlog grooming)
- Customer feedback happens **frequently** (sprint reviews)

Next lecture: Deep dive into Agile methodology!

Group Discussion

Think about a project you've worked on:

1. Did requirements change during development?
2. How did those changes get handled?
3. Would a more iterative approach have helped?

Share with your neighbor for 3 minutes.

Summary

Concept	Key Point	Testing Connection
Business Perspectives	Same requirement, different views	Different acceptance criteria
Elicitation Techniques	Requirements are discovered	Better questions → better tests
Bug vs CR	Does/doesn't meet spec	Tests define the boundary
POC-Driven RE	Build to discover requirements	Prototype tests hypotheses

Summary (continued)

Concept	Key Point	Testing Connection
Waterfall & V-Model	Sequential phases, late feedback	V-Model pairs dev with test phases
V-Modell XT	German government standard	108 deliverables, decision gates
Cost of Change	Late changes cost more (Boehm)	Tests planned late = defects found late
Agile Preview	Embrace change, iterate quickly	Continuous testing in sprints

The Big Picture (Both Parts)

Stakeholder Needs



Requirements (User Stories, Acceptance Criteria) ← Part 1



Elicitation & Refinement ← Part 2



Tests (verify requirements) ← Part 1



Code (implements requirements, passes tests)



Change Requests & Bug Fixes ← Part 2



Back to stakeholder conversations

Key Takeaways

1. **Different roles see requirements differently** - understand their perspectives
2. **Requirements are discovered, not dictated** - use elicitation techniques
3. **Bug vs CR distinction has business implications** - document clearly
4. **Build to learn** - POCs accelerate requirements discovery
5. **Sequential models have fundamental limitations** - late feedback is expensive
6. **Agile embraces change** - more on this next lecture!

Reflection Questions

1. What requirements are implicit in your current project code?
2. Who are the stakeholders for the Road Profile Viewer?
3. "System crashes with 1 million points, no size limit documented" - Bug or CR?
4. Rewrite "The system should be reliable" to be testable.
5. How would you use Gen AI-assisted POC to discover requirements?

Further Reading

Books:

- Karl Wieggers & Joy Beatty: *Software Requirements* (3rd ed.)
- Eric Ries: *The Lean Startup*
- Mike Cohn: *User Stories Applied*

Standards:

- IEEE 830: Software Requirements Specifications
- IEEE 29148: Requirements Engineering (modern)

Coming Next: Agile methodology - Scrum, sprints, backlogs!



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

Questions?

THANK YOU