



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

Chapter 04

REQUIREMENTS ENGINEERING

Where We Left Off (Chapter 03)

When you tried to improve test coverage for `find_intersection()`, you kept hitting questions like:

- "What *should* happen when the ray is vertical?"
- "Is returning `None` for segments behind the camera correct, or a bug?"
- "Who decided that parallel lines should use `t=0`?"

The uncomfortable truth: Coverage gaps reveal requirements gaps!

The Problem

Throughout this course, you've been writing tests against:

- Docstrings and code comments
- Type hints
- Your own assumptions

But real software projects need *explicit* requirements that everyone agrees on.

Today's question: How do we systematically capture, document, and validate what we're building - *before* we write the tests?

Learning Objectives

By the end of this lecture, you will:

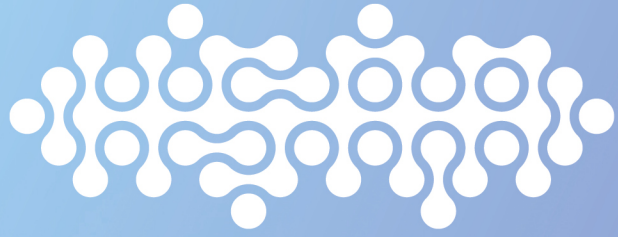
1. **Understand** what requirements are and why they matter
2. **Distinguish** functional vs non-functional requirements
3. **Identify stakeholders** and their different perspectives
4. **Apply quality criteria** to requirements (testable, measurable)
5. **Use modern tools**: User Stories, GitHub Issues, Acceptance Criteria
6. **Link requirements to tests** for traceability

What You Won't Learn Today

Not covered (yet):

- ✗ Formal specification languages (Z, VDM)
- ✗ Model-based requirements (SysML, UML use cases)
- ✗ Regulatory requirements (safety-critical systems)
- ✗ Full Agile methodology (coming in Chapter 04)

Today's Focus: Practical requirements engineering for your projects.



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

Requirements at Code Level

1

Implicit Requirements in Code

```
def find_intersection(  
    x_road: NDArray[np.float64], # Expects numpy array of floats  
    y_road: NDArray[np.float64], # Must be same length as x_road  
    angle_degrees: float,        # Probably 0-360, but not enforced  
    camera_x: float = 0,  
    camera_y: float = 1.5,  
    ) -> tuple[float | None, float | None, float | None]:
```

What's missing?

- What if `x_road` and `y_road` have different lengths?
- What's the valid range for `angle_degrees` ?
- What if `x_road` is empty or unsorted?

Quick Discussion

Think about your own code:

What implicit assumptions have you made that aren't documented anywhere?

Take 1 minute to think of an example from your Road Profile Viewer.

Design by Contract

Design by Contract (DbC) explicitly states:

- **Preconditions:** What must be true BEFORE calling the function
- **Postconditions:** What will be true AFTER the function returns
- **Invariants:** What remains true throughout execution

This is requirements engineering at the function level!

The Comment Approach (Not Recommended!)

```
def find_intersection(...):  
    """  
    Preconditions:  
        - len(x_road) == len(y_road) >= 2  
        - x_road values are monotonically increasing  
        - angle_degrees is in range [0, 360)  
  
    Postconditions:  
        - If intersection exists: returns (x, y, dist) where dist > 0  
        - If no intersection: returns (None, None, None)  
    """
```

Problem: Comments can lie! They get outdated.

Why Comments Are Dangerous

Problem 1: Comments make code harder to read

- Every function starts with 20+ lines of documentation

Problem 2: Comments get outdated

- When you refactor, do you always update the docstring? Be honest.

Problem 3: Comments don't prevent bugs

- The code will crash somewhere with a confusing error

Better Design: One Array, Not Two

Current bug-prone API:

```
find_intersection(  
    x_road=np.array([0, 10, 20]),  
    y_road=np.array([0, 5])  # Oops, different length!  
)
```

Better design: Single array of (x, y) tuples

```
road = np.array([  
    [0.0, 0.0],    # Point 1  
    [10.0, 5.0],   # Point 2  
    [20.0, 10.0],  # Point 3  
)  
find_intersection(road, angle, camera_x, camera_y)
```

Now it's impossible to have mismatched lengths!

Better Design: An Angle Class

Current bug-prone API:

```
find_intersection(x_road, y_road, angle_degrees=500) # What does 500° mean?
```

Better: Create an Angle class

```
@dataclass
class Angle:
    _degrees: float

    def __init__(self, degrees: float):
        self._degrees = degrees % 360 # Auto-normalize!

    def is_vertical(self) -> bool:
        return abs(math.cos(self.radians)) < 1e-10
```

Now **Angle(500)** automatically becomes **Angle(140)** !

Runtime Checks with Asserts (Rarely Right!)

```
def find_intersection(...):  
    assert len(x_road) == len(y_road), "Must have same length"  
    assert len(x_road) >= 2, "Need at least 2 points"  
    assert 0 <= angle_degrees < 360, f"Invalid angle: {angle_degrees}"
```

Problems:

- Performance overhead (runs every call)
- `python -O` removes ALL asserts!
- Crashes in production
- False sense of security

Asserts are for internal invariants, NOT input validation!

Tests as Executable Requirements

Instead of comments that can get outdated:

```
class TestFindIntersectionRequirements:
    """Tests documenting the requirements for find_intersection()."""

    def test_req_vertical_ray_returns_none(self):
        """REQ-GEOM-001: Vertical rays shall return None."""
        road = np.array([[0, 0], [10, 5]])
        result = find_intersection(road, Angle(90))
        assert result == (None, None, None)

    def test_req_intersection_distance_positive(self):
        """REQ-GEOM-002: Distance shall be > 0."""
        # ...
```

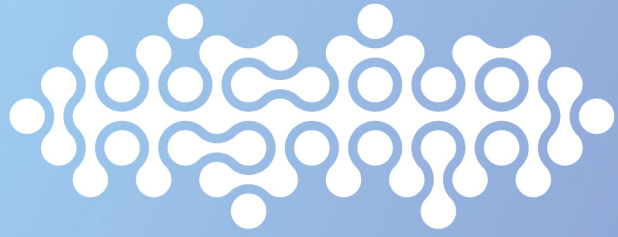
Tests can't get "out of sync" - if code changes, tests fail!

The Bridge: Code to Business

The same process happens at every level:

```
Code Coverage Gap
  → Function Requirement
    → Module Requirement
      → System Requirement
        → Business Need
```

Now let's zoom out and look at requirements from the top down.



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

What Are Requirements?

2

Definition

Requirement: *A condition or capability needed by a user to solve a problem or achieve an objective.*

— IEEE Standard 610.12

More practically:

*A requirement describes **what** the system should do (or how well it should do it), without specifying **how** to implement it.*

Functional vs Non-Functional

Type	Definition	Example
Functional	What the system <i>does</i>	"Display intersection point on click"
Non-Functional	How <i>well</i> it does it	"Calculate in less than 100ms"

Non-functional categories:

- Performance, Security, Usability
- Reliability, Scalability, Maintainability

Example: Road Profile Viewer Requirements

Functional Requirements:

- FR-1: Load road profile data from CSV files
- FR-2: Display road profile as 2D line chart
- FR-3: Calculate camera ray based on user-specified angle
- FR-4: Calculate and highlight intersection point
- FR-5: Display distance from camera to intersection

Non-Functional Requirements:

- NFR-1: Load 10,000-point profile in under 2 seconds
- NFR-2: UI updates within 100ms after user input

Quick Exercise

Classify these as FR or NFR:

1. "The system shall allow users to zoom the chart"
2. "The system shall respond to user input within 200ms"
3. "The system shall export results to PDF"
4. "The codebase shall have documentation for all public functions"

Discuss with your neighbor for 2 minutes!

Requirement IDs: Why They Matter

Why do requirements need IDs?

- **Traceability:** Link requirements → tests → code
- **Communication:** "Let's discuss FR-4" is clearer than "that intersection thing"
- **Change tracking:** When FR-4 changes, you know which tests to update
- **Coverage analysis:** Which requirements have tests?

Naming Schemes

Scheme	Example	When to Use
Type-based	FR-1 , NFR-2	Simple projects
Domain-based	REQ-GEOM-001	Larger projects with modules
Hierarchical	REQ-1.2.3	Decomposing into sub-requirements
Feature-based	LOAD-001 , CALC-002	Organizing by features

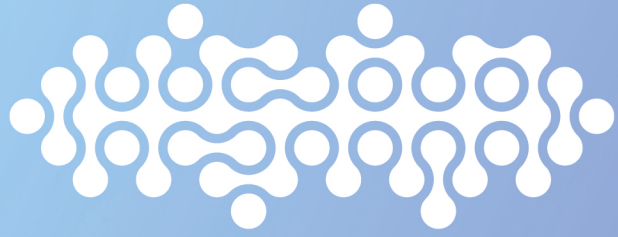
Important: The number is just an ID, NOT a priority!

Where to Store Requirements

From informal to formal:

Level	Tool	Use Case
Informal	Emails, Sticky notes	Avoid for anything serious
Semi-formal	Markdown, GitHub Issues, Jira	Most projects
Formal	IEEE 830, DOORS	Regulated industries

For this course: GitHub Issues work great!



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

Stakeholders

3

What is a Stakeholder?

Stakeholder: *Any person or organization that has an interest in or is affected by the system.*

Key insight: Different stakeholders have different - sometimes conflicting - requirements.

Types of Stakeholders



Your first task: Identify WHO has a stake in your system.
Missing a stakeholder = missing their requirements!

Stakeholder Complexity Varies

Project Type	Stakeholders	Why?
Medical Device	Doctors, Patients, FDA, Legal, IT, Risk Mgmt...	Lives at stake, heavy regulation
Indie Game	Players, Developer, Platform	Entertainment, minimal regulation
Banking App	Customers, Regulators, Security, Auditors	Financial regulation, audit trails

Stakeholder Matrix: Road Profile Viewer

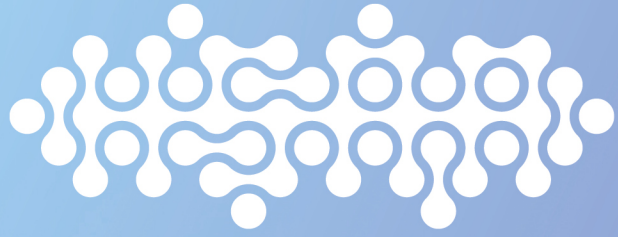
Stakeholder	Role	Concerns	Example Requirement
Road Engineer	End User	Accuracy	"Intersection within 0.1m"
Lab Manager	Customer	Cost, Training	"Trainable in 1 hour"
IT Admin	Operations	Maintenance	"Single-file deployment"
Developer	Internal	Code quality	"Modular architecture"
Safety Officer	Regulator	Reliability	"All calculations logged"

Group Discussion

The "intersection calculation" means different things:

- To the **engineer**: A measurement tool
- To the **manager**: A cost/time saving
- To the **developer**: An algorithm to implement
- To **QA**: A behavior to verify

What stakeholders does YOUR project have?



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

What Makes a Good Requirement?

4

The INVEST Criteria

Letter	Criterion	Bad Example	Good Example
I	Independent	"After login, show dashboard"	"Show dashboard for auth users"
N	Negotiable	"Use blue #0000FF"	"Buttons visually prominent"
V	Valuable	"Refactor database"	"Load profiles 50% faster"
E	Estimable	"Make it better"	"Add PDF export"
S	Small	"Implement all reporting"	"Generate single report"
T	Testable	"Should be fast"	"Response < 200ms"

The Testability Connection

Untestable requirement:

"The system should be user-friendly"

How do you test this? What does "user-friendly" mean?

Testable requirement:

"Load and calculate intersection in under 2 seconds"

Now we can write a concrete test!

Testable Requirement → Executable Test

```
def test_performance():  
    start = time.time()  
    app.load_profile("sample.csv")  
    app.calculate_intersection()  
    assert time.time() - start < 2.0
```

Key insight: If you can't write a test for it, it's not a good requirement!

Quick Exercise

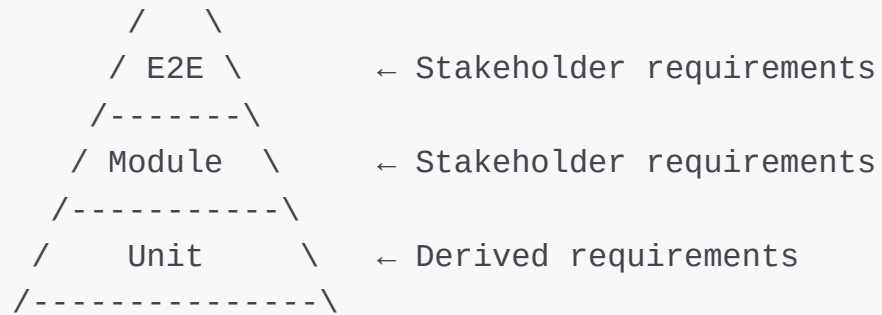
Make these requirements testable:

1. ✗ "The system should be fast"
2. ✗ "The UI should be intuitive"
3. ✗ "The calculations should be accurate"

Take 3 minutes with your neighbor to rewrite them!

Testing Pyramid & Requirements

Remember the Testing Pyramid?



Insight: Stakeholder requirements are tested at module/E2E level.
Unit tests verify *implementation* of those requirements.

Derived Requirements

Stakeholder says:

"FR-CALC-001: Display intersection point on road profile"

Developer decides:

- Use ray-casting algorithm
- Handle vertical rays specially
- Return None for edge cases

Derived Requirements (cont.)

These implementation decisions create derived requirements:

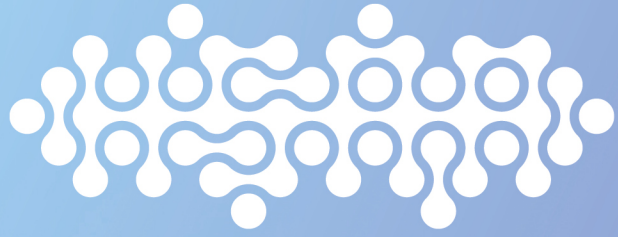
- REQ-GEOM-001: Vertical rays return None
- REQ-GEOM-002: Distance is always positive
- REQ-GEOM-003: Intersection lies within road bounds

Unit tests verify these derived requirements!

The Traceability Chain

```
FR-CALC-001 (Stakeholder)
|
|→ test_fr_calc_001_displayed() [Module Test]
|
|→ find_intersection() [Implementation]
    |
    |→ REQ-GEOM-001 → test_vertical_ray() [Unit]
    |→ REQ-GEOM-002 → test_positive_distance() [Unit]
    |→ REQ-GEOM-003 → test_on_road() [Unit]
```

The pyramid shape emerges naturally from this structure!



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

Documentation Tools

5

User Stories Format

*As a **[role]**, I want **[feature]**, so that **[benefit]**.*

Example:

*As a **road engineer**, I want to **see the intersection point highlighted**, so that **I can quickly verify my measurements**.*

Why User Stories Work

Part	Purpose
Who (role)	Identifies the stakeholder
What (feature)	Describes the need, not the solution
Why (benefit)	Explains value, helps prioritization

Acceptance Criteria: Given-When-Then

Feature: Intersection Calculation

Scenario: Normal intersection with road

Given a road profile is loaded with points [(0,0), (10,5), (20,10)]

And the camera is at position (0, 15)

When I set the viewing angle to 45 degrees

Then the system should display an intersection point

And the distance should be greater than 0

Scenario: Vertical ray (edge case)

Given a road profile is loaded

When I set the viewing angle to 90 degrees

Then the system should display "No intersection"

Tools like pytest-bdd can execute these as tests!

GitHub Issues as Requirements

User Story

As a road engineer, I want to export results to PDF
so that I can include them in reports.

Acceptance Criteria

- [] Export button visible on main screen
- [] PDF includes road profile chart
- [] PDF includes intersection coordinates
- [] File name defaults to "road_profile_YYYY-MM-DD.pdf"

Priority

High - Requested by multiple users

Related

- Implements: FR-EXPORT-001
- Tests: test_export.py

Why GitHub Issues Excel

INVEST criteria are subjective! What's "small enough"?

GitHub Issues enable collaborative refinement:

@pm: I think this is small enough for one week.

@dev-alice: PDF generation alone is 3 days. Can we split into "basic" and "styled"?

@dev-bob: Also, is the date format testable? Which timezone?

@pm: Good points. Updated acceptance criteria.

The conversation is preserved for future reference!

Hands-On Exercise

Write a User Story for one of these features:

1. Adding a zoom control to the road profile chart
2. Saving and loading camera positions
3. Comparing two road profiles side-by-side

Take 5 minutes, then share with the class!

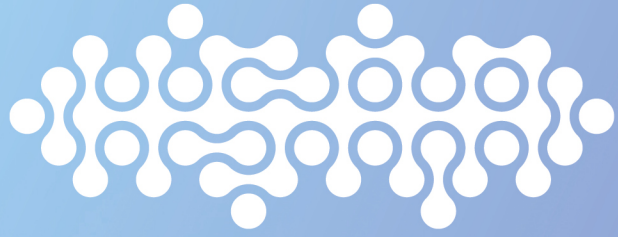
Exercise Template

Use this format:

As a [role], I want [feature], so that [benefit].

Acceptance Criteria:

- ☐ Criterion 1
- ☐ Criterion 2
- ☐ Criterion 3



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

Linking Requirements to Tests

6

What is Traceability?

Traceability: *The ability to link requirements → tests → code in both directions.*

Why it matters:

- **Forward:** Does every requirement have a test?
- **Backward:** Does every test verify a requirement?
- **Impact analysis:** If requirement changes, what's affected?

Implementing Traceability with pytest

```
import pytest

@pytest.mark.requirement("FR-4")
def test_intersection_found():
    """FR-4: Calculate and highlight intersection point."""
    result = find_intersection(road, angle=45.0)
    assert result[0] is not None

@pytest.mark.requirement("FR-4")
@pytest.mark.requirement("NFR-2")
def test_intersection_performance():
    """FR-4 + NFR-2: Calculate within 100ms."""
    start = time.time()
    result = find_intersection(road, angle=45.0)
    assert (time.time() - start) * 1000 < 100
```

Code Coverage vs Requirements Coverage

Metric	What it measures	Limitation
Code Coverage	% of lines/branches executed	100% ≠ all requirements tested
Requirements Coverage	% of requirements with tests	100% ≠ requirements correct

You need both:

- High code coverage → tests exercise implementation
- High requirements coverage → tests verify specification

Avoiding "Orphan" Requirements

Orphans: Code or tests that can't trace to any requirement.

Signs of orphans:

- Unit tests with no clear purpose (`test_thing_works`)
- Code that "might be useful someday"
- Features nobody asked for

Prevention: Every derived requirement should trace back to a stakeholder need.

Summary

Concept	Key Point	Testing Connection
Requirements	What the system should do	Tests verify requirements
FR vs NFR	What vs how well	Different test types
Stakeholders	Different perspectives	Different acceptance criteria
INVEST	Good requirements are testable	Testability enables test design
User Stories	As/I want/So that	Acceptance criteria = test cases
Traceability	Links Req-Tests-Code	Requirements coverage metric

Key Takeaways

1. **Coverage gaps reveal requirements gaps**
2. **Better types beat better comments**
3. **Tests are executable requirements**
4. **Stakeholders have different perspectives**
5. **If you can't test it, it's not a good requirement**
6. **Derived requirements explain why we have many unit tests**
7. **Traceability links requirements to code to tests**

What's Next: Part 2

Continue to Part 2:

- Requirements in the development workflow
- Waterfall vs iterative approaches
- Eliciting requirements through interviews
- Bug vs Change Request distinction
- POC-driven requirements discovery
- Preview of Agile methodology



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

Questions?

THANK YOU