



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

Kapitel 04

REQUIREMENTS ENGINEERING

Wo wir aufgehört haben (Kapitel 03)

Als Sie versuchten, die Testabdeckung für `find_intersection()` zu verbessern, stießen Sie auf Fragen wie:

- "Was *sollte* passieren, wenn der Strahl vertikal ist?"
- "Ist die Rückgabe von `None` für Segmente hinter der Kamera korrekt, oder ein Bug?"
- "Wer hat entschieden, dass parallele Linien `t=0` verwenden sollen?"

Die unbequeme Wahrheit: Abdeckungslücken offenbaren Anforderungslücken!

Das Problem

Während dieses Kurses haben Sie Tests geschrieben gegen:

- Docstrings und Code-Kommentare
- Type Hints
- Ihre eigenen Annahmen

Aber echte Softwareprojekte brauchen *explizite Anforderungen*, auf die sich alle einigen.

Heutige Frage: Wie erfassen, dokumentieren und validieren wir systematisch, was wir bauen
- *bevor* wir die Tests schreiben?

Lernziele

Am Ende dieser Vorlesung werden Sie:

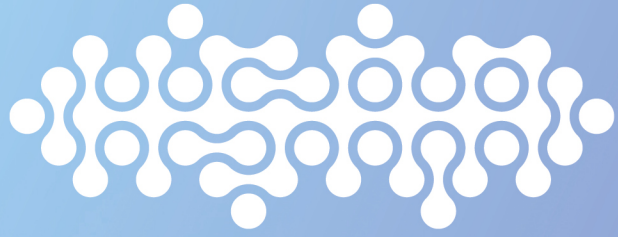
1. **Verstehen**, was Anforderungen sind und warum sie wichtig sind
2. **Unterscheiden** zwischen funktionalen und nicht-funktionalen Anforderungen
3. **Stakeholder identifizieren** und ihre verschiedenen Perspektiven
4. **Qualitätskriterien anwenden** auf Anforderungen (testbar, messbar)
5. **Moderne Tools nutzen**: User Stories, GitHub Issues, Akzeptanzkriterien
6. **Anforderungen mit Tests verknüpfen** für Rückverfolgbarkeit

Was Sie heute nicht lernen

Noch nicht behandelt:

- ✗ Formale Spezifikationssprachen (Z, VDM)
- ✗ Modellbasierte Anforderungen (SysML, UML Use Cases)
- ✗ Regulatorische Anforderungen (sicherheitskritische Systeme)
- ✗ Vollständige Agile-Methodik (kommt in Vorlesung 9)

Heutiger Fokus: Praktisches Requirements Engineering für Ihre Projekte.



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

Anforderungen auf Code-Ebene

1

Implizite Anforderungen im Code

```
def find_intersection(  
    x_road: NDArray[np.float64], # Erwartet numpy Array von Floats  
    y_road: NDArray[np.float64], # Muss gleiche Länge wie x_road haben  
    angle_degrees: float,        # Wahrscheinlich 0-360, aber nicht erzwungen  
    camera_x: float = 0,  
    camera_y: float = 1.5,  
    ) -> tuple[float | None, float | None, float | None]:
```

Was fehlt?

- Was wenn `x_road` und `y_road` unterschiedliche Längen haben?
- Was ist der gültige Bereich für `angle_degrees` ?
- Was wenn `x_road` leer oder unsortiert ist?

Kurze Diskussion

Denken Sie an Ihren eigenen Code:

Welche impliziten Annahmen haben Sie gemacht, die nirgendwo dokumentiert sind?

Nehmen Sie sich 1 Minute, um ein Beispiel aus Ihrem Road Profile Viewer zu finden.

Design by Contract

Design by Contract (DbC) definiert explizit:

- **Vorbedingungen:** Was muss VOR dem Funktionsaufruf wahr sein
- **Nachbedingungen:** Was wird NACH der Rückgabe wahr sein
- **Invarianten:** Was bleibt während der Ausführung wahr

Das ist Requirements Engineering auf Funktionsebene!

Der Kommentar-Ansatz (Nicht empfohlen!)

```
def find_intersection(...):  
    """  
    Vorbedingungen:  
        - len(x_road) == len(y_road) >= 2  
        - x_road Werte sind monoton steigend  
        - angle_degrees ist im Bereich [0, 360)  
  
    Nachbedingungen:  
        - Falls Schnittpunkt existiert: gibt (x, y, dist) zurück mit dist > 0  
        - Falls kein Schnittpunkt: gibt (None, None, None) zurück  
    """
```

Problem: Kommentare können lügen! Sie werden veraltet.

Warum Kommentare gefährlich sind

Problem 1: Kommentare machen Code schwerer lesbar

- Jede Funktion beginnt mit 20+ Zeilen Dokumentation

Problem 2: Kommentare werden veraltet

- Aktualisieren Sie beim Refactoring immer den Docstring? Seien Sie ehrlich.

Problem 3: Kommentare verhindern keine Bugs

- Der Code wird irgendwo mit einer verwirrenden Fehlermeldung abstürzen

Besseres Design: Ein Array, nicht zwei

Aktuelle fehleranfällige API:

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

Besseres Design: Einzelnes Array von (x, y) Tupeln

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

Jetzt ist es unmöglich, nicht übereinstimmende Längen zu haben!

Besseres Design: Eine Angle-Klasse

Aktuelle fehleranfällige API:

```
find_intersection(x_road, y_road, angle_degrees=500) # Was bedeutet 500°?
```

Besser: Eine Angle-Klasse erstellen

```
@dataclass
class Angle:
    _degrees: float

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

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

Jetzt wird `Angle(500)` automatisch zu `Angle(140)` !

Laufzeitprüfungen mit Asserts (Selten richtig!)

```
def find_intersection(...):  
    assert len(x_road) == len(y_road), "Muss gleiche Länge haben"  
    assert len(x_road) >= 2, "Braucht mindestens 2 Punkte"  
    assert 0 <= angle_degrees < 360, f"Ungültiger Winkel: {angle_degrees}"
```

Probleme:

- Performance-Overhead (läuft bei jedem Aufruf)
- `python -O` entfernt ALLE Asserts!
- Abstürze in Produktion
- Falsches Sicherheitsgefühl

Asserts sind für interne Invarianten, NICHT für Input-Validierung!

Tests als ausführbare Anforderungen

Statt Kommentaren, die veralten können:

```
class TestFindIntersectionRequirements:
    """Tests dokumentieren die Anforderungen für find_intersection()."""

    def test_req_vertical_ray_returns_none(self):
        """REQ-GEOM-001: Vertikale Strahlen sollen None zurückgeben."""
        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: Distanz soll > 0 sein."""
        # ...
```

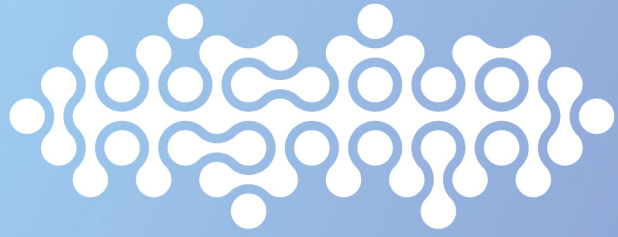
Tests können nicht "out of sync" geraten - wenn sich Code ändert, schlagen Tests fehl!

Die Brücke: Code zu Business

Der gleiche Prozess passiert auf jeder Ebene:

Code-Abdeckungsücke
→ Funktions-Anforderung
→ Modul-Anforderung
→ System-Anforderung
→ Business-Bedarf

Jetzt zoomen wir heraus und betrachten Anforderungen von oben nach unten.



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

Was sind Anforderungen?

2

Definition

Anforderung: *Eine Bedingung oder Fähigkeit, die ein Benutzer benötigt, um ein Problem zu lösen oder ein Ziel zu erreichen.*
— IEEE Standard 610.12

Praktischer ausgedrückt:

*Eine Anforderung beschreibt **was** das System tun soll (oder wie gut es das tun soll), ohne zu spezifizieren **wie** es implementiert werden soll.*

Funktional vs Nicht-Funktional

Typ	Definition	Beispiel
Funktional	Was das System <i>tut</i>	"Schnittpunkt bei Klick anzeigen"
Nicht-Funktional	Wie <i>gut</i> es das tut	"Berechnung in weniger als 100ms"

Nicht-funktionale Kategorien:

- Performance, Sicherheit, Benutzerfreundlichkeit
- Zuverlässigkeit, Skalierbarkeit, Wartbarkeit

Beispiel: Road Profile Viewer Anforderungen

Funktionale Anforderungen:

- FR-1: Straßenprofildaten aus CSV-Dateien laden
- FR-2: Straßenprofil als 2D-Liniendiagramm anzeigen
- FR-3: Kamerastrahl basierend auf benutzerdefinierten Winkel berechnen
- FR-4: Schnittpunkt berechnen und hervorheben
- FR-5: Entfernung von Kamera zum Schnittpunkt anzeigen

Nicht-Funktionale Anforderungen:

- NFR-1: 10.000-Punkte-Profil in unter 2 Sekunden laden
- NFR-2: UI-Aktualisierung innerhalb von 100ms nach Benutzereingabe

Kurze Übung

Klassifizieren Sie diese als FR oder NFR:

1. "Das System soll Benutzern erlauben, das Diagramm zu zoomen"
2. "Das System soll innerhalb von 200ms auf Benutzereingaben reagieren"
3. "Das System soll Ergebnisse als PDF exportieren"
4. "Die Codebasis soll Dokumentation für alle öffentlichen Funktionen haben"

Diskutieren Sie 2 Minuten mit Ihrem Nachbarn!

Anforderungs-IDs: Warum sie wichtig sind

Warum brauchen Anforderungen IDs?

- **Rückverfolgbarkeit:** Anforderungen → Tests → Code verknüpfen
- **Kommunikation:** "Lass uns FR-4 besprechen" ist klarer als "diese Schnittpunkt-Sache"
- **Änderungsverfolgung:** Wenn FR-4 sich ändert, wissen Sie welche Tests betroffen sind
- **Abdeckungsanalyse:** Welche Anforderungen haben Tests?

Namensschema

Schema	Beispiel	Wann verwenden
Typbasiert	FR-1 , NFR-2	Einfache Projekte
Domänenbasiert	REQ-GEOM-001	Größere Projekte mit Modulen
Hierarchisch	REQ-1.2.3	Zerlegung in Untieranforderungen
Featurebasiert	LOAD-001 , CALC-002	Organisation nach Features

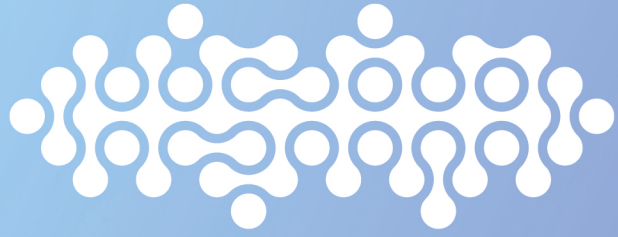
Wichtig: Die Nummer ist nur eine ID, KEINE Priorität!

Wo Anforderungen speichern

Von informell zu formal:

Ebene	Tool	Anwendungsfall
Informell	E-Mails, Klebezettel	Vermeiden für alles Ernsthafte
Semi-formal	Markdown, GitHub Issues, Jira	Die meisten Projekte
Formal	IEEE 830, DOORS	Regulierte Industrien

Für diesen Kurs: GitHub Issues funktionieren super!



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

Stakeholder

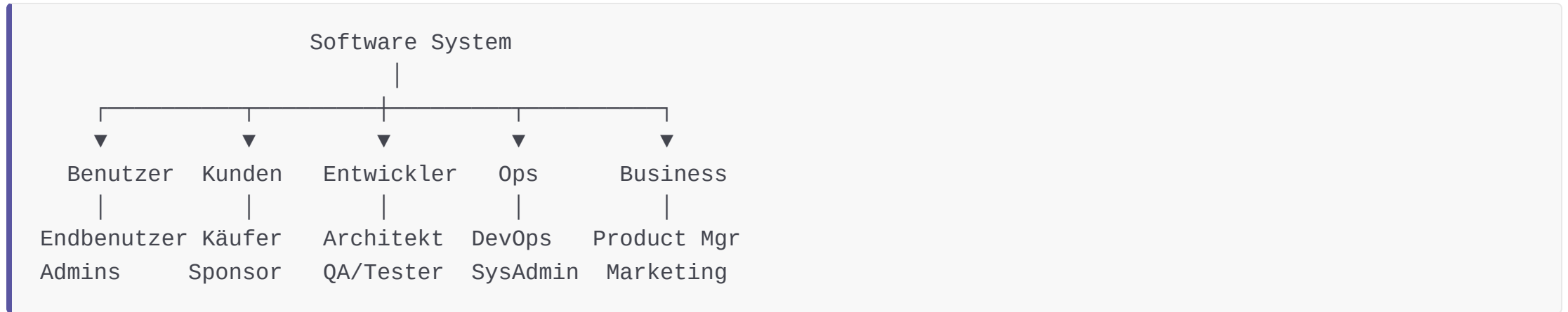
3

Was ist ein Stakeholder?

Stakeholder: *Jede Person oder Organisation, die ein Interesse an dem System hat oder davon betroffen ist.*

Wichtige Erkenntnis: Verschiedene Stakeholder haben verschiedene - manchmal widersprüchliche - Anforderungen.

Arten von Stakeholdern



Ihre erste Aufgabe: Identifizieren Sie WER ein Interesse an Ihrem System hat.
Einen Stakeholder übersehen = dessen Anforderungen übersehen!

Stakeholder-Komplexität variiert

Projekttyp	Stakeholder	Warum?
Medizingerät	Ärzte, Patienten, FDA, Legal, IT, Risiko-Mgmt...	Leben auf dem Spiel, starke Regulierung
Indie-Spiel	Spieler, Entwickler, Plattform	Unterhaltung, minimale Regulierung
Banking-App	Kunden, Regulierer, Sicherheit, Auditoren	Finanzregulierung, Audit-Trails

Stakeholder-Matrix: Road Profile Viewer

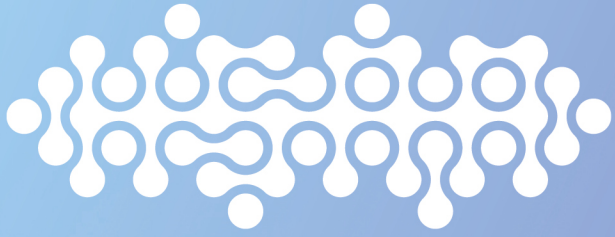
Stakeholder	Rolle	Bedenken	Beispiel-Anforderung
Straßeningenieur	Endbenutzer	Genauigkeit	"Schnittpunkt auf 0.1m genau"
Laborleiter	Kunde	Kosten, Training	"Schulbar in 1 Stunde"
IT-Admin	Operations	Wartung	"Ein-Datei-Deployment"
Entwickler	Intern	Code-Qualität	"Modulare Architektur"
Sicherheitsbeauftragter	Regulierer	Zuverlässigkeit	"Alle Berechnungen protokolliert"

Gruppendiskussion

Die "Schnittpunktberechnung" bedeutet verschiedene Dinge:

- Für den **Ingenieur**: Ein Messwerkzeug
- Für den **Manager**: Kosten-/Zeitersparnis
- Für den **Entwickler**: Ein zu implementierender Algorithmus
- Für **QA**: Ein zu verifizierendes Verhalten

Welche Stakeholder hat IHR Projekt?



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

Was macht eine gute Anforderung aus?

4

Die INVEST-Kriterien

Buchstabe	Kriterium	Schlechtes Beispiel	Gutes Beispiel
I	Independent	"Nach Login, Dashboard zeigen"	"Dashboard für auth. Benutzer"
N	Negotiable	"Blau #0000FF verwenden"	"Buttons visuell hervorgehoben"
V	Valuable	"Datenbank refactoren"	"Profile 50% schneller laden"
E	Estimable	"Besser machen"	"PDF-Export hinzufügen"
S	Small	"Gesamtes Reporting"	"Einzelnen Bericht generieren"
T	Testable	"Soll schnell sein"	"Antwort < 200ms"

Der Testbarkeits-Zusammenhang

Nicht testbare Anforderung:

"Das System soll benutzerfreundlich sein"

Wie testen Sie das? Was bedeutet "benutzerfreundlich"?

Testbare Anforderung:

"Laden und Schnittpunkt berechnen in unter 2 Sekunden"

Jetzt können wir einen konkreten Test schreiben!

Testbare Anforderung → Ausführbarer Test

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

Wichtige Erkenntnis: Wenn Sie keinen Test dafür schreiben können, ist es keine gute Anforderung!

Kurze Übung

Machen Sie diese Anforderungen testbar:

1. ✗ "Das System soll schnell sein"
2. ✗ "Die UI soll intuitiv sein"
3. ✗ "Die Berechnungen sollen genau sein"

Nehmen Sie sich 3 Minuten mit Ihrem Nachbarn, um sie umzuschreiben!

Testpyramide & Anforderungen

Erinnern Sie sich an die Testpyramide?



Erkenntnis: Stakeholder-Anforderungen werden auf Modul/E2E-Ebene getestet.
Unit-Tests verifizieren die *Implementierung* dieser Anforderungen.

Abgeleitete Anforderungen

Stakeholder sagt:

"FR-CALC-001: Schnittpunkt auf Straßenprofil anzeigen"

Entwickler entscheidet:

- Ray-Casting-Algorithmus verwenden
- Vertikale Strahlen speziell behandeln
- None für Randfälle zurückgeben

Abgeleitete Anforderungen (Forts.)

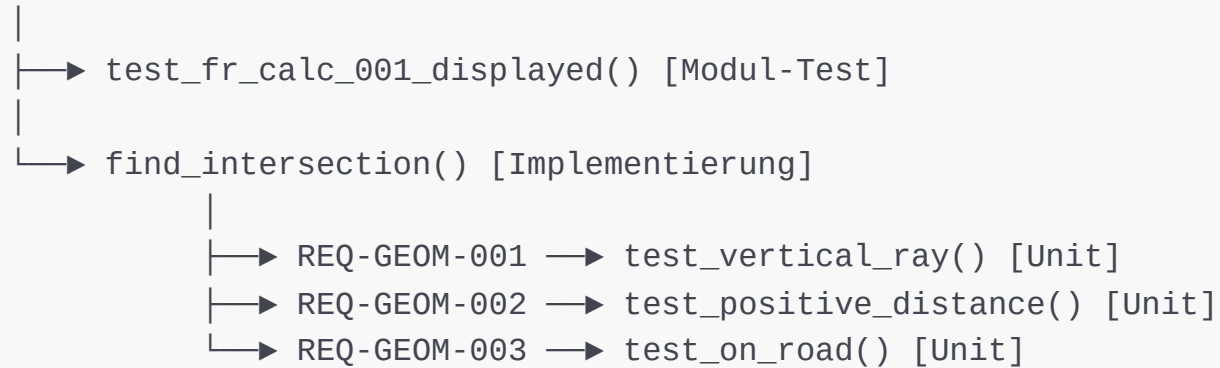
Diese Implementierungsentscheidungen erzeugen abgeleitete Anforderungen:

- REQ-GEOM-001: Vertikale Strahlen geben None zurück
- REQ-GEOM-002: Distanz ist immer positiv
- REQ-GEOM-003: Schnittpunkt liegt innerhalb der Straßengrenzen

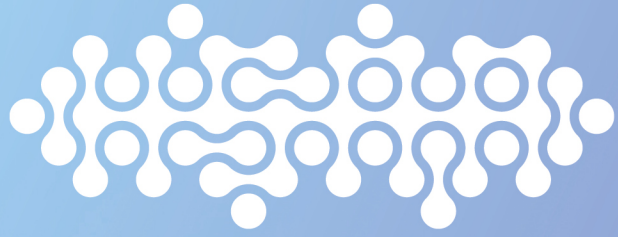
Unit-Tests verifizieren diese abgeleiteten Anforderungen!

Die Rückverfolgbarkeitskette

FR-CALC-001 (Stakeholder)



Die Pyramidenform entsteht natürlich aus dieser Struktur!



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

Dokumentations-Tools

5

User Stories Format

*Als **[Rolle]** möchte ich **[Feature]**, damit **[Nutzen]**.*

Beispiel:

*Als **Straßeningenieur** möchte ich **den Schnittpunkt hervorgehoben sehen**, damit **ich meine Messungen schnell überprüfen kann**.*

Warum User Stories funktionieren

Teil	Zweck
Wer (Rolle)	Identifiziert den Stakeholder
Was (Feature)	Beschreibt den Bedarf, nicht die Lösung
Warum (Nutzen)	Erklärt den Wert, hilft bei Priorisierung

Akzeptanzkriterien: Given-When-Then

Feature: Schnittpunktberechnung

Scenario: Normaler Schnittpunkt mit Straße

Given ein Straßenprofil ist geladen mit Punkten [(0,0), (10,5), (20,10)]

And die Kamera ist an Position (0, 15)

When ich den Blickwinkel auf 45 Grad setze

Then soll das System einen Schnittpunkt anzeigen

And die Distanz soll größer als 0 sein

Scenario: Vertikaler Strahl (Randfall)

Given ein Straßenprofil ist geladen

When ich den Blickwinkel auf 90 Grad setze

Then soll das System "Kein Schnittpunkt" anzeigen

Tools wie pytest-bdd können diese als Tests ausführen!

GitHub Issues als Anforderungen

User Story

Als Straßeningenieur möchte ich Ergebnisse als PDF exportieren, damit ich sie in Berichte einfügen kann.

Akzeptanzkriterien

- [] Export-Button auf Hauptbildschirm sichtbar
- [] PDF enthält Straßenprofil-Diagramm
- [] PDF enthält Schnittpunkt-Koordinaten
- [] Dateiname standardmäßig "strassenprofil_JJJJ-MM-TT.pdf"

Priorität

Hoch - Von mehreren Benutzern angefragt

Verwandt

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

Warum GitHub Issues hervorragend sind

INVEST-Kriterien sind subjektiv! Was ist "klein genug"?

GitHub Issues ermöglichen kollaborative Verfeinerung:

@pm: Ich denke, das reicht für eine Woche.

@dev-alice: PDF-Generierung allein dauert 3 Tage. Können wir in "basic" und "styled" aufteilen?

@dev-bob: Auch, ist das Datumsformat testbar? Welche Zeitzone?

@pm: Gute Punkte. Akzeptanzkriterien aktualisiert.

Die Konversation wird für spätere Referenz aufbewahrt!

Praktische Übung

Schreiben Sie eine User Story für eines dieser Features:

1. Zoom-Steuerung zum Straßenprofil-Diagramm hinzufügen
2. Kamerapositionen speichern und laden
3. Zwei Straßenprofile nebeneinander vergleichen

Nehmen Sie sich 5 Minuten, dann teilen Sie mit der Klasse!

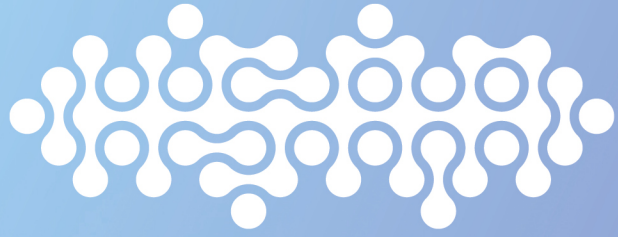
Übungsvorlage

Verwenden Sie dieses Format:

Als [Rolle] möchte ich [Feature], damit [Nutzen].

Akzeptanzkriterien:

- ☐ Kriterium 1
- ☐ Kriterium 2
- ☐ Kriterium 3



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

Anforderungen mit Tests verknüpfen

6

Was ist Rückverfolgbarkeit?

Rückverfolgbarkeit: Die Fähigkeit, Anforderungen → Tests → Code in beide Richtungen zu verknüpfen.

Warum es wichtig ist:

- **Vorwärts:** Hat jede Anforderung einen Test?
- **Rückwärts:** Verifiziert jeder Test eine Anforderung?
- **Auswirkungsanalyse:** Wenn sich eine Anforderung ändert, was ist betroffen?

Rückverfolgbarkeit mit pytest implementieren

```
import pytest

@pytest.mark.requirement("FR-4")
def test_intersection_found():
    """FR-4: Schnittpunkt berechnen und hervorheben."""
    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: Berechnung innerhalb von 100ms."""
    start = time.time()
    result = find_intersection(road, angle=45.0)
    assert (time.time() - start) * 1000 < 100
```

Code-Abdeckung vs Anforderungs-Abdeckung

Metrik	Was sie misst	Einschränkung
Code-Abdeckung	% der Zeilen/Branches ausgeführt	100% ≠ alle Anforderungen getestet
Anforderungs-Abdeckung	% der Anforderungen mit Tests	100% ≠ Anforderungen korrekt

Sie brauchen beides:

- Hohe Code-Abdeckung → Tests prüfen Implementierung
- Hohe Anforderungs-Abdeckung → Tests verifizieren Spezifikation

"Verwaiste" Anforderungen vermeiden

Verwaiste: Code oder Tests, die nicht zu einer Anforderung zurückverfolgt werden können.

Anzeichen für Verwaiste:

- Unit-Tests ohne klaren Zweck (`test_thing_works`)
- Code der "irgendwann nützlich sein könnte"
- Features die niemand angefragt hat

Prävention: Jede abgeleitete Anforderung sollte zu einem Stakeholder-Bedarf zurückführen.

Zusammenfassung

Konzept	Kernpunkt	Test-Verbindung
Anforderungen	Was das System tun soll	Tests verifizieren Anforderungen
FR vs NFR	Was vs wie gut	Verschiedene Testtypen
Stakeholder	Verschiedene Perspektiven	Verschiedene Akzeptanzkriterien
INVEST	Gute Anforderungen sind testbar	Testbarkeit ermöglicht Testdesign
User Stories	Als/Möchte ich/Damit	Akzeptanzkriterien = Testfälle
Rückverfolgbarkeit	Verknüpft Anf-Tests-Code	Anforderungs-Abdeckungs-Metrik

Wichtige Erkenntnisse

1. **Abdeckungslücken offenbaren Anforderungslücken**
2. **Bessere Typen schlagen bessere Kommentare**
3. **Tests sind ausführbare Anforderungen**
4. **Stakeholder haben verschiedene Perspektiven**
5. **Wenn Sie es nicht testen können, ist es keine gute Anforderung**
6. **Abgeleitete Anforderungen erklären, warum wir viele Unit-Tests haben**
7. **Rückverfolgbarkeit verknüpft Anforderungen mit Code und Tests**

Was kommt als Nächstes: Teil 2

Weiter zu Teil 2:

- Anforderungen im Entwicklungs-Workflow
- Wasserfall vs iterative Ansätze
- Anforderungen durch Interviews erheben
- Bug vs Change Request Unterscheidung
- POC-getriebene Anforderungsermittlung
- Vorschau auf Agile-Methodik



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

Fragen?

VIELEN DANK