



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

Kapitel 04 · Teil 2

REQUIREMENTS ENGINEERING

Was wir in Teil 1 gelernt haben

In Teil 1 haben wir gelernt:

- Was Anforderungen sind (funktional vs. nicht-funktional)
- Wer sich dafür interessiert (Stakeholder)
- Was sie gut macht (INVEST, Testbarkeit)
- Wie man sie dokumentiert (User Stories, GitHub Issues)
- Wie man sie mit Tests verknüpft (Traceability)

Heute: Der *Prozess* des Requirements Engineering.

Die schwierigeren Fragen

Teil 1 hat nicht beantwortet:

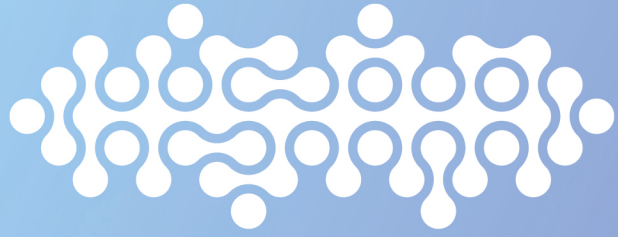
- Wie passen Anforderungen in den Entwicklungsworkflow?
- Wie *entdeckt* man, was Stakeholder wirklich brauchen?
- Was passiert, wenn sich Anforderungen mitten im Projekt ändern?
- Gibt es einen besseren Weg als "alles vorher aufschreiben"?

Heute: Von Erhebung bis Change Management, und warum Agile entstanden ist.

Lernziele

Am Ende dieser Vorlesung werden Sie:

1. **Anforderungen aus verschiedenen Perspektiven sehen** (Entwickler, PM, Product Manager, Gründer)
2. **Erhebungstechniken anwenden** durch Stakeholder-Interviews
3. **Bugs von Change Requests unterscheiden** und warum das wichtig ist
4. **POC-getriebene Ansätze nutzen** um Anforderungen zu entdecken
5. **Traditionelle Modelle verstehen** (Wasserfall, V-Modell, V-Modell XT)
6. **Agile Methodik kennenlernen** als Antwort auf Unsicherheit



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

Geschäftsperspektiven

1

Die Entwickler-Perspektive

Fragen:

- Was genau soll diese Funktion tun?
- Was sind die Randfälle?
- Wie teste ich das?

Fokus: Präzision, Testbarkeit, Implementierungsdetails

Beispiel: "Schnittpunkt anzeigen" → Algorithmus, Datenstrukturen, UI-Komponente

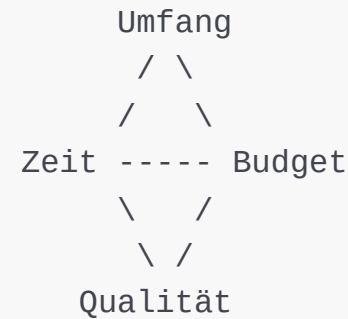
Die Projektmanager-Perspektive

Fragen:

- Wie lange dauert das?
- Was ist die Priorität?
- Was ist der Umfang?

Fokus: Das Magische Dreieck

"Gut, Schnell, Günstig - wähle zwei."



Product Manager & Gründer-Perspektiven

Product Manager:

- Liefert das Mehrwert für Nutzer?
- Was ist das Minimum zum Testen?
- Was sollten wir als nächstes bauen?

Fokus: Wertlieferung, Priorisierung

Startup-Gründer:

- Gibt es einen Markt dafür?
- Werden Leute dafür bezahlen?
- Lösen wir ein echtes Problem?

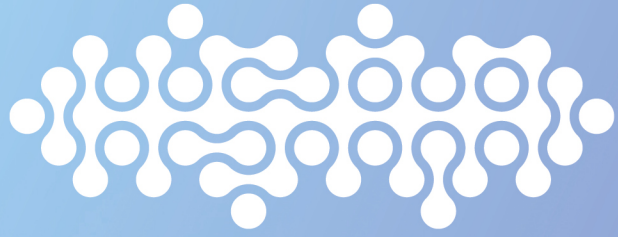
Fokus: Product-Market Fit, Validierung

Gleiche Anforderung, verschiedene Sichtweisen

"Schnittpunkt anzeigen"

Rolle	Sieht es als
Entwickler	Algorithmus, Datenstrukturen, UI-Komponente
Projektmanager	2 Tage Aufwand, abhängig von Chart-Bibliothek
Product Manager	Ermöglicht Kern-Workflow, hohe Priorität
Gründer	Teil des Wertangebots, Differenzierungsmerkmal

Erkenntnis: Kontext prägt die Interpretation!



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

Anforderungen erheben

2

Erhebungstechniken

Technik	Am besten für	Vorsicht bei
Interviews	Individuelle Perspektiven	Suggestivfragen
Workshops	Konsensbildung	Dominante Stimmen
Beobachtung	Tatsächlicher Workflow	Hawthorne-Effekt
Umfragen	Daten von vielen Nutzern	Oberflächliche Antworten
Prototyping	"Ich weiß es, wenn ich es sehe"	Prototyp ≠ Produkt

Gute vs. schlechte Fragen

Schlecht (suggestiv, geschlossen):

- "Sie wollen Blau, richtig?"
- "Ist das System schnell genug?"
- "Gefällt Ihnen das Design?"

Gut (offen, explorativ):

- "Führen Sie mich durch diese Aufgabe."
- "Was ist der frustrierendste Teil?"
- "Wie würde Erfolg aussehen?"
- "Was passiert, wenn das falsch ist?"

Die 5-Warum-Technik

Nutzer: *"Die Schnittpunktberechnung muss schneller sein."*

Warum? "Weil ich zu lange warte."

Warum ist Warten wichtig? "Ich muss 50 Profile pro Tag verarbeiten."

Warum 50 pro Tag? "So viele Straßenabschnitte vermessen wir."

Warum nicht bündeln? "Ich brauche Ergebnisse vor der nächsten Messung."

Warum sofort? "Um den Kamerawinkel für den nächsten Abschnitt anzupassen."

Echte Anforderung: Nicht "schnellere Berechnung" sondern "Echtzeit-Feedback-Schleife"!

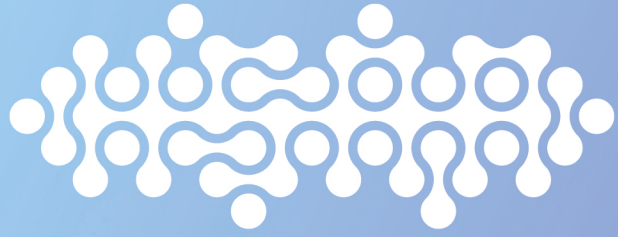
Übung: 5-Warum-Technik

Szenario: Ein Nutzer sagt "Ich muss das Diagramm als PDF exportieren."

In Paaren (3 Minuten):

1. Eine Person spielt den Stakeholder
2. Andere Person fragt 5x "Warum?"
3. Entdeckt das *echte* zugrundeliegende Bedürfnis

Was habt ihr entdeckt?



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

Bug vs. Change Request

3

Definitionen

Bug:

Die Software erfüllt eine bestehende, dokumentierte Anforderung nicht.

Change Request (CR):

Der Stakeholder möchte, dass die Software etwas anderes oder zusätzliches zu bestehenden Anforderungen tut.

Das Schlüsselwort: *Dokumentiert*

Warum diese Unterscheidung wichtig ist

Aspekt	Bug-Fix	Change Request
Abrechnung	Meist kostenlos (Garantie)	Typischerweise kostenpflichtig
Priorität	Oft hoch (gebrochenes Versprechen)	Abhängig vom Geschäftswert
Umfang	Teil des ursprünglichen Umfangs	Erweitert den Umfang
Verantwortung	Entwicklerteam	Keiner (neue Anforderung)
Zeitplan	Verlängert Deadline nicht	Kann Deadline verlängern

Die Grauzone

Kunde sagt: "Der Schnittpunkt ist falsch!"

Entscheidungsbaum:

```
Ist erwartetes Verhalten dokumentiert?  
├─ Ja → Entspricht Software der Dokumentation?  
│   └─ Ja → Change Request (funktioniert wie spezifiziert)  
│       └─ Nein → Bug (entspricht nicht der Spec)  
└─ Nein → Grauzone - Diskussion nötig  
    └─ War es durch Kontext impliziert?
```

Praxisbeispiel

"Die Schnittpunktberechnung liefert falsche Ergebnisse für Winkel $> 180^\circ$."

- **Wenn** Anforderung sagt "Winkel im Bereich 0-360°": **Bug** (sollte 180+ handhaben)
- **Wenn** Anforderung sagt "Winkel im Bereich 0-180°": **Change Request** (neue Fähigkeit)
- **Wenn** Anforderung nichts spezifiziert: **Diskussion nötig**

Erkenntnis: Gute Anforderungsdokumentation verhindert diese Streitigkeiten.

Über Softwareentwicklung hinaus

Diese Unterscheidung hat rechtliche und finanzielle Konsequenzen!

Perspektive	Risiko	Schutz
Als Entwickler	Kunden behaupten "Bugs" für kostenlose Features	Detaillierte Specs, signierte Dokumente
Als Kunde	Anbieter verlangen CR für echte Bugs	Klare Bug-Definitionen, Abnahmetests

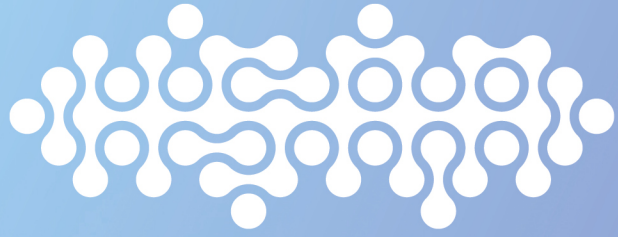
Praxis: Bug vs. CR Streitigkeiten → Vertragsverhandlungen → Rechtsstreitigkeiten

Kurze Diskussion

Szenario: Das System stürzt ab beim Laden einer Datei mit 1 Million Punkten. Die Dokumentation sagt nichts über Dateigrößenbeschränkungen.

Bug oder Change Request?

Wie würdet ihr das handhaben?



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

POC-getriebene Anforderungen

4

Die unbequeme Wahrheit

Kunden wissen nicht, was sie wollen, bis sie es sehen.

Das ist kein Versagen - es ist menschliche Natur.

Traditionelles RE nimmt an:

Stakeholder befragen → Vollständige Anforderungen schreiben → System bauen → Liefern

Realität:

Stakeholder beschreiben was sie *glauben* zu wollen → Du baust es → Sie sagen "Eigentlich meinte ich..."

Lean Startup: Build-Measure-Learn

Der Lean Startup Ansatz (Eric Ries) dreht den Prozess um:

Idee → MVP bauen → Messen → Lernen → Idee → ...

MVP = Minimum Viable Product: Das Kleinste, das du bauen kannst, um deine Hypothese zu testen.

Beispiel für Road Profile Viewer:

- MVP 1: Kommandozeilen-Skript → Nutzer zeigen
- MVP 2: Einfacher Plot mit festen Daten → "Ist das was Sie brauchen?"
- MVP 3: Interaktiver Winkelregler → "Funktioniert dieser Workflow?"

Design Thinking

Design Thinking (IDEO, Stanford d.school):

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

Kernaktivitäten:

- **Empathize:** Nutzer beobachten, ihren Schmerz erleben
- **Define:** In Problemstellungen zusammenfassen
- **Ideate:** Viele Lösungen generieren
- **Prototype:** Schnell, günstig, wegwerfbar bauen
- **Test:** Feedback einholen, Verständnis verfeinern

Anforderungen entstehen aus diesem Prozess!

Gen AI für Rapid Prototyping

Traditionelles Prototyping: Prototyp bauen: 2-4 Wochen → Kunde zeigen → Iterieren

Gen AI-gestützt ("Vibe Coding"): Beschreiben was du willst → Funktionierender Prototyp in Stunden → Kunde *am gleichen Tag* zeigen

Die Erkenntnis: KI ermöglicht so schnelle POC-Erstellung, dass **Anforderungserhebung** interaktiv wird.

Beispiel-Workflow mit Gen AI

Entwickler: "Erstelle eine Python GUI die CSV-Straßenpunkte lädt und als Liniendiagramm mit Winkelregler anzeigt."

[KI generiert Code in 10 Minuten]

Entwickler: *Zeigt dem Straßenbauingenieur*

Ingenieur: "Ich muss auch die Kameraposition sehen."

Entwickler: "Füge Kamerapositions-Marker zum Code hinzu."

[KI aktualisiert Code in 5 Minuten]

Ingenieur: "Eigentlich muss ich zwei Profile nebeneinander vergleichen..."

Anforderungen werden zu Gesprächen mit funktionierender Software!

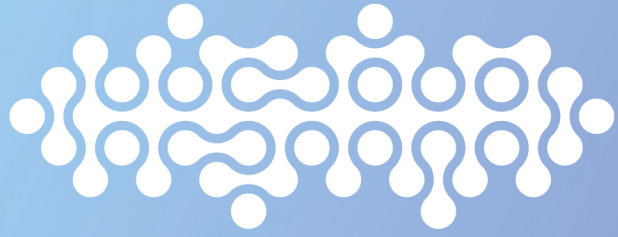
Wann POC-getrieben am besten funktioniert

POC-getrieben ist am besten für:

- Hohe Unsicherheit / neue Domänen
- "Ich weiß es, wenn ich es sehe"
- Schnelllebige Märkte
- Innovationskontexte

Traditionelles RE funktioniert noch für:

- Regulierte Umgebungen
- Gut verstandene Domänen
- Große Integrationssysteme
- Festpreis-Verträge



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

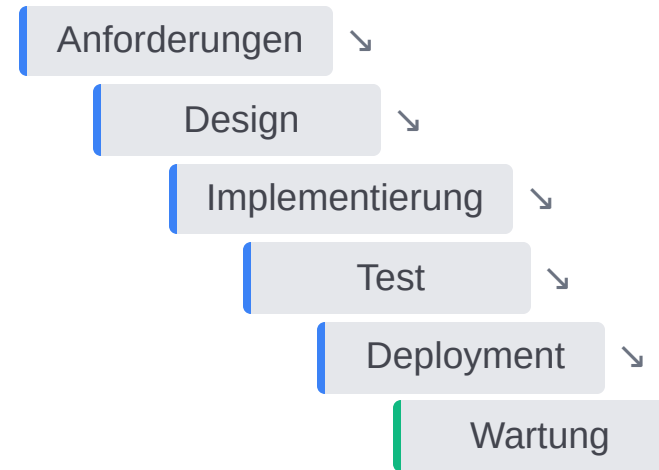
Traditionelle Entwicklungsmodelle

5

Das Wasserfallmodell

Geschichte:

- 1956: Beningtons sequentielle Phasen
- 1970: Winston Royces Paper
- 1976: Bell prägte "Wasserfall"



Die Royce-Ironie

Witziges Faktum: Winston Royce, dessen 1970er Paper als Ursprung des Wasserfalls zitiert wird, hat tatsächlich *davor gewarnt*, den rein sequentiellen Ansatz zu verwenden!

Er nannte den einfachen Wasserfall "riskant und zum Scheitern einladend."

Der Rest seines Papers beschrieb iterative Verbesserungen und Feedback-Schleifen.

Die Industrie übernahm sein Diagramm, aber ignorierte seine Warnungen!

Wasserfall-Eigenschaften

- **Sequentielle Phasen:** Jede Phase muss abgeschlossen sein, bevor die nächste beginnt
- **Phase Gates:** Formale Reviews und Freigaben
- **Umfangreiche Dokumentation:** Vollständige Dokumentation in jeder Phase
- **Big Design Up Front (BDUF):** Alle Anforderungen/Design vor dem Coding
- **Änderungen sind teuer:** Späte Änderungen erfordern Rückkehr zu früheren Phasen

Wo es funktioniert: Stabile Anforderungen, regulierte Branchen, Festpreisverträge

Das V-Modell

Kerninnovation: Jede Entwicklungsphase hat eine entsprechende Testphase.



Testplanung erfolgt parallel zur Entwicklung, nicht danach!

V-Modell: Test-Zuordnung

Entwicklungsphase	Testphase	Was wird verifiziert
Anforderungsanalyse	Abnahmetest	System erfüllt Nutzerbedürfnisse
Systemdesign	Systemtest	System funktioniert wie entworfen
Detailldesign	Integrationstest	Komponenten arbeiten zusammen
Implementierung	Unit-Test	Jede Einheit funktioniert korrekt

Verbesserung: Tests früh geplant → Fehler früher gefunden

Immer noch sequentiell: Funktionierende Software erscheint erst am Ende

V-Modell XT: Deutscher Behördenstandard

Wenn Sie an deutschen Bundes-IT-Projekten arbeiten, ist das Pflicht!

- **1992:** Original V-Modell für das Bundesministerium der Verteidigung
- **2004:** V-Modell XT eingeführt ("eXtreme Tailoring")
- **2019:** Aktuelle Version 2.3

XT bedeutet "Extreme Tailoring" - anpassbar an verschiedene Projekttypen

V-Modell XT Komponenten

Kernelemente:

- **Entscheidungspunkte:** Projektmeilensteine mit erforderlichen Genehmigungen
- **~20 Vorgehensbausteine:** PM, QS, Konfigurationsmanagement, etc.
- **108 definierte Produkte:** Jedes mit definierter Struktur
- **35 definierte Rollen:** Klare Verantwortlichkeiten
- **Projekttypen:** Inklusive agiler Varianten!

Echte Anwendungen: Bundes-IT, Verteidigung, Luft- und Raumfahrt, Medizinprodukte, Automotive

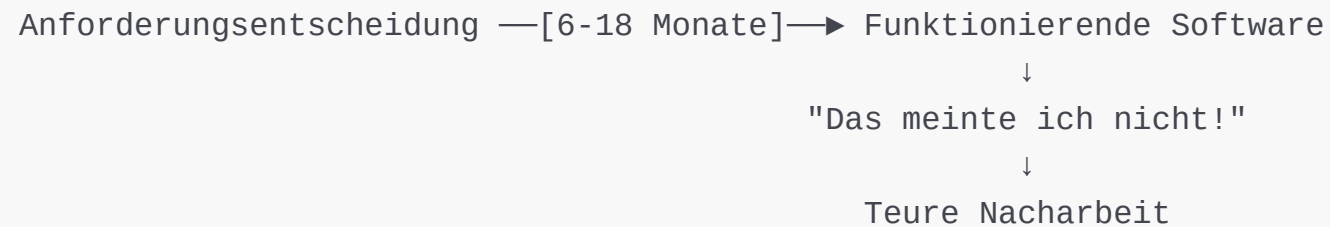
Warum V-Modell XT für Sie wichtig ist

Auch wenn Sie nicht in der Verwaltung arbeiten wollen:

- Viele große deutsche Unternehmen nutzen es (Siemens, Bosch, SAP)
- Zeigt, wie regulierte Branchen Softwareentwicklung handhaben
- Macht den Kontrast zu Agile klarer
- **Vorstellungsgespräche in der deutschen Industrie referenzieren es oft!**

Das Problem mit sequentiellen Modellen

Das Problem des späten Feedbacks:



Funktionierende Software erscheint erst am Ende!

Die Kosten der Änderung (Boehm, 1981)

Phase	Relative Kosten
Anforderungen	1x
Design	3-6x
Coding	10x
Test	15-40x
Produktion	30-100x

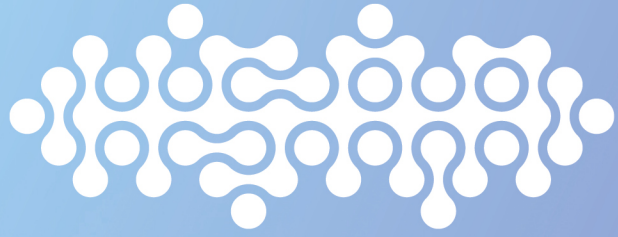
Warum? Ein Anforderungsfehler in Produktion bedeutet: Docs ändern + Design aktualisieren + Code neu schreiben + neu testen + neu deployen + Kundenauswirkungen + Datenmigration...

Die fundamentalen Spannungen

Annahme	Realität
"Wir können alle Anforderungen vorher kennen"	Nutzer wissen nicht, was sie wollen, bis sie es sehen
"Wir können vollständig vor dem Coding designen"	Designentscheidungen brauchen Implementierungs-Feedback
"Wir können es beim ersten Mal richtig bauen"	Software wird gelernt, nicht gefertigt

Das Paradoxon:

- Wenn wir mehr planen → Wir verbringen Zeit mit Plänen, die sich ändern werden
- Wenn wir weniger planen → Wir bauen das Falsche



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

Vorschau auf Agile

6

Das Agile Manifest (Vorschau)

Wir erschließen bessere Wege, Software zu entwickeln.

Wir haben gelernt zu schätzen:

- **Individuen und Interaktionen** *mehr als Prozesse und Werkzeuge*
- **Funktionierende Software** *mehr als umfassende Dokumentation*
- **Zusammenarbeit mit dem Kunden** *mehr als Vertragsverhandlung*
- **Reagieren auf Veränderung** *mehr als das Befolgen eines Plans*

Obwohl die Werte auf der rechten Seite ihren Wert haben, schätzen wir die auf der linken Seite höher.

Agile Anforderungen

In der agilen Entwicklung:

- Anforderungen erfasst als **User Stories** (klein, verhandelbar)
- Stories leben in einem **Backlog** (priorisierte Liste)
- Arbeit erfolgt in **Sprints** (1-4 Wochen)
- Anforderungen werden **kontinuierlich** verfeinert (Backlog Grooming)
- Kundenfeedback erfolgt **häufig** (Sprint Reviews)

Nächste Vorlesung: Tiefere Einführung in agile Methodik!

Gruppendiskussion

Denken Sie an ein Projekt, an dem Sie gearbeitet haben:

1. Haben sich Anforderungen während der Entwicklung geändert?
2. Wie wurden diese Änderungen gehandhabt?
3. Hätte ein iterativerer Ansatz geholfen?

Teilen Sie mit Ihrem Nachbarn für 3 Minuten.

Zusammenfassung

Konzept	Kernpunkt	Test-Verbindung
Geschäftsperspektiven	Gleiche Anforderung, verschiedene Sichten	Verschiedene Abnahmekriterien
Erhebungstechniken	Anforderungen werden entdeckt	Bessere Fragen → bessere Tests
Bug vs. CR	Erfüllt/erfüllt nicht die Spec	Tests definieren die Grenze
POC-getriebenes RE	Bauen um Anforderungen zu entdecken	Prototyp testet Hypothesen

Zusammenfassung (Fortsetzung)

Konzept	Kernpunkt	Test-Verbindung
Wasserfall & V-Modell	Sequentielle Phasen, spätes Feedback	V-Modell paart Entw. mit Testphasen
V-Modell XT	Deutscher Behördenstandard	108 Produkte, Entscheidungspunkte
Kosten der Änderung	Späte Änderungen kosten mehr (Boehm)	Tests spät geplant = Fehler spät gefunden
Agile Vorschau	Änderung annehmen, schnell iterieren	Kontinuierliches Testen in Sprints

Das große Bild (Beide Teile)

Stakeholder-Bedürfnisse



Anforderungen (User Stories, Abnahmekriterien) ← Teil 1



Erhebung & Verfeinerung ← Teil 2



Tests (verifizieren Anforderungen) ← Teil 1



Code (implementiert Anforderungen, besteht Tests)



Change Requests & Bug Fixes ← Teil 2



Zurück zu Stakeholder-Gesprächen

Wichtigste Erkenntnisse

1. **Verschiedene Rollen sehen Anforderungen unterschiedlich** - verstehe ihre Perspektiven
2. **Anforderungen werden entdeckt, nicht diktiert** - nutze Erhebungstechniken
3. **Bug vs. CR Unterscheidung hat geschäftliche Auswirkungen** - dokumentiere klar
4. **Bauen um zu lernen** - POCs beschleunigen die Anforderungsentdeckung
5. **Sequentielle Modelle haben fundamentale Grenzen** - spätes Feedback ist teuer
6. **Agile nimmt Änderungen an** - mehr dazu in der nächsten Vorlesung!

Reflexionsfragen

1. Welche Anforderungen sind implizit in Ihrem aktuellen Projektcode?
2. Wer sind die Stakeholder für den Road Profile Viewer?
3. "System stürzt ab bei 1 Million Punkten, kein Größenlimit dokumentiert" - Bug oder CR?
4. Schreiben Sie "Das System soll zuverlässig sein" um, sodass es testbar ist.
5. Wie würden Sie KI-gestützten POC nutzen, um Anforderungen zu entdecken?

Weiterführende Literatur

Bücher:

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

Standards:

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

Nächstes Mal: Agile Methodik - Scrum, Sprints, Backlogs!



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

Fragen?

VIELEN DANK