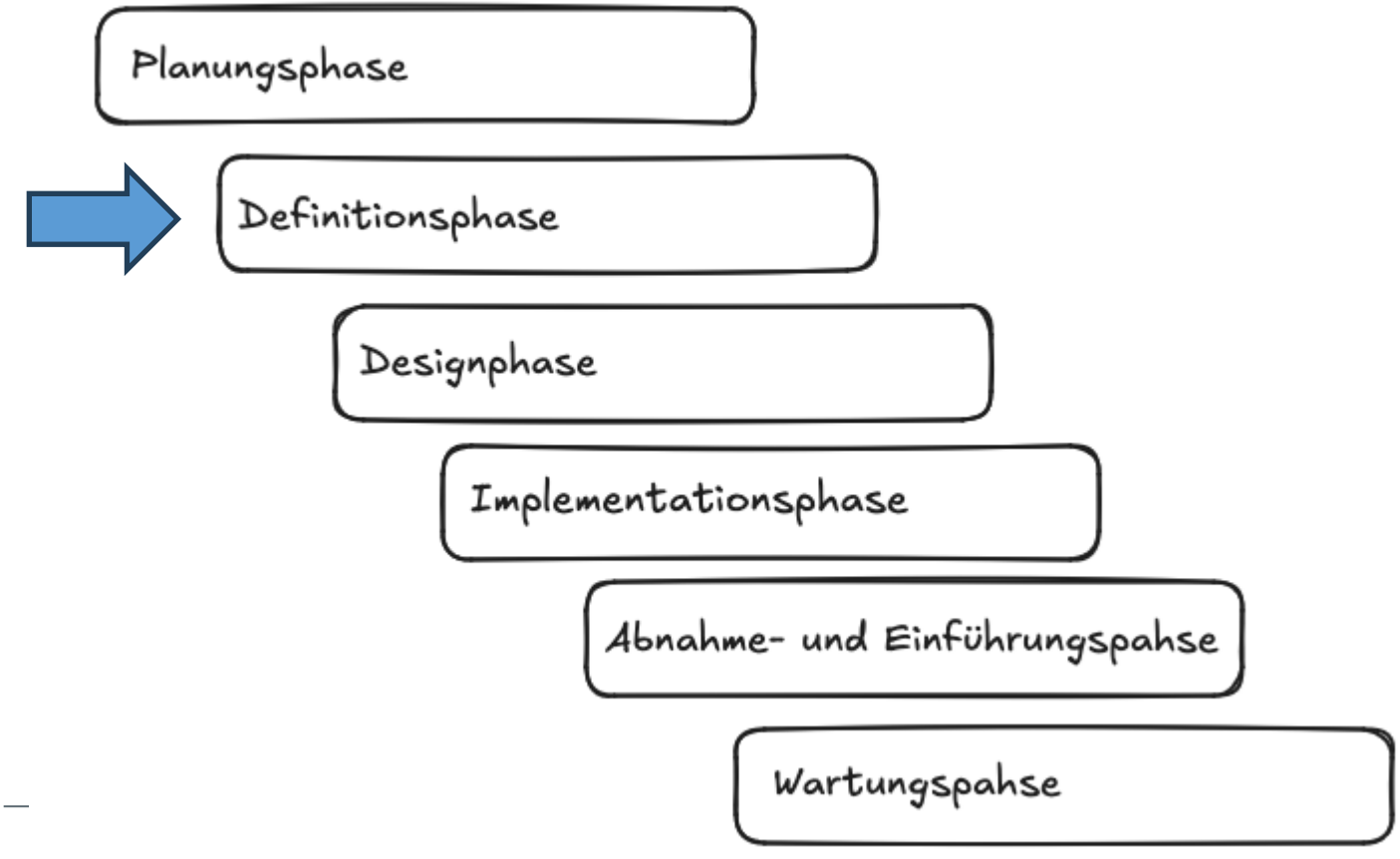
DHBW Stuttgart
Duale Hochschule
Baden-Württemberg

Software Engineering: Requirements Engineering

Dr. Eugenie Giesbrecht

Der Software-Lebenszyklus



Wie geht man grundsätzlich vor?

- Eine Anforderung spezifiziert die qualitativen und quantitativen Eigenschaften eines Produkts
- In der Definitionsphase werden alle Anforderungen gesammelt und beschrieben
- Anforderungen sind die Grundlage für die Systemarchitektur
- Formen:
  - Schriftlich / Tabellarisch: z.B. *FR-01: „Der Nutzer kann eine Aufgabe anlegen.“*
  - Grafisch (z.B. UML-Use-Case Diagramme)
  - User Stories: z.B. *„Als Nutzer möchte ich Aufgaben anlegen können, damit ich meine Arbeit planen kann.“*

Probleme bei der Anforderungsanalyse

- Kunden wissen nicht, was sie wirklich wollen.
- Kunden benutzen ihre eigene Fachsprache.
- Verschiedene Stakeholder können widersprüchliche Anforderungen haben.
- Politische und organisatorische Faktoren können Anforderungen beeinflussen.
- Anforderungen ändern sich während der Analyse und Entwicklung.
- Neue Stakeholder mischen sich ein.

Stakeholder

- Einzelpersonen und Organisationen
- die aktiv an einem Projekt beteiligt sind.
- deren Interessen als Folge der Projektdurchführung oder des Projektabschlusses positiv oder negativ beeinflusst werden können.
- die das Projekt und seine Ergebnisse beeinflussen.

Mini-Quiz: Wer ist ein Stakeholder?

- **Frage 1:** Die neue TODO-App wird an einer Hochschule entwickelt. Wer ist ein Stakeholder?
- A) Studierende
  B) Lehrkräfte
  C) Die IT-Abteilung
  D) Alle oben genannten
- **Frage 2:** Was beschreibt Stakeholder am besten?
- A) Personen, die nur die App testen
  B) Personen, die vom Projekt betroffen sind oder es beeinflussen
  C) Nur die Projektleitung
  D) Nur externe Firmen

- **Frage 3:** Welche der folgenden Personen ist *kein* typischer Stakeholder?
- A) Ein Sicherheitsbeauftragter
  B) Eine interne Reinigungskraft
  C) Eine Nutzerin der App
  D) Eine externe Datenschutzstelle

- **Frage 4:** Warum sind Stakeholder wichtig?
- A) Sie haben alle dieselben Interessen
  B) Sie definieren technische Standards
  C) Sie haben Erwartungen, Wünsche oder Vorgaben, die das Projekt beeinflussen
  D) Sie übernehmen die Codierung

- **Frage 5:** Zwei Stakeholder haben widersprüchliche Wünsche. Was ist typisch?
- A) Das ist normal und kommt häufig vor
  B) Das Projekt muss sofort abgebrochen werden
  C) Einer der Stakeholder wird ignoriert
  D) Das Team entscheidet ohne Rücksprache
- **Frage 6:** Welche Aussage ist richtig?
- A) Stakeholder können ein Projekt unterstützen oder blockieren
  B) Stakeholder dürfen im Projekt nie mitreden
  C) Stakeholder müssen immer dieselbe Meinung haben
  D) Stakeholder dürfen keine Anforderungen schreiben

Anforderungen

Funktionale Anforderungen

WAS und WOFÜR

- Was soll das System leisten?
- Welche Dienste soll es anbieten?
- Eingaben, Verarbeitungen, Ausgaben
- Verhalten in bestimmten Situationen, ggf. was soll es explizit nicht tun

Nicht-funktionale Anforderungen

WIE und WOMIT

- Wie soll das System/einzelne Funktionen arbeiten?
- Qualitätsanforderungen wie Performanz und Zuverlässigkeit
- Anforderungen an die Benutzbarkeit des Systems

Aufgaben in der Anforderungsanalyse (1/2)

Anforderungsermittlung:

Sammeln einer vollständige Menge von Anforderungen von Stakeholdern.

Anforderungsformulierung:

Ermittelte Anforderungen so beschreiben, dass sie eindeutig, testbar und verständlich sind.

Anforderungsanalyse:

Klassifizierung, Bewertung, Vergleich und Prüfung der ermittelten Anforderungen

Aufgaben in der Anforderungsanalyse (2/2)

Anforderungsvalidierung:
Die Kunst, zu richtigen und widerspruchsfrei formulierten Anforderungen zu gelangen.

Anforderungsmanagement:
Prüfung und Änderung von Anforderungen.

SMART-Regel

„Eine gute Anforderung ist **SMART**.

- **S – Spezifisch**: klar formuliert, keine Interpretationen
- **M – Messbar**: es muss testbar sein
- **A – Akzeptiert**: Stakeholder stimmen zu
- **R – Realistisch**: technisch machbar
- **T – Terminiert**: zeitlich oder durch Kriterien abschließbar

SMART-Anforderungs-Quiz (10 Beispiele)

- 1. „Das System muss Aufgaben speichern können.“
- 2. „Das Login soll schnell funktionieren.“
- 3. „Das System soll innerhalb von 300 Millisekunden auf Klicks reagieren.“
- 4. „Benutzer sollen jederzeit einen Bericht exportieren können.“
- 5. „Die App soll bis zum 1. Dezember eine Dunkelmodus-Funktion enthalten.“
- 6. „Der Benutzer soll Aufgaben mit einem Klick löschen können.“
- 7. „Die App soll benutzerfreundlich sein.“
- 8. „Das System muss täglich automatisch um 03:00 Uhr Backups erstellen.“
- 9. „Die Anwendung soll ansprechend aussehen.“
- 10. „Die Passwort-Wiederherstellung muss einem Benutzer ermöglichen, innerhalb von 5 Minuten ein neues Passwort zu setzen.“

Constraints

- Feste Grenzen, innerhalb derer wir arbeiten müssen

- Beispiele:
  - Technologie & Hardware: Python, Linux
  - Rechtliche Anforderungen: DSGVO
  - Budget & Zeit

Lastenheft vs. Pflichtenheft

- **Lastenheft – *Was* soll gebaut werden?**
  - kommt vom Kunden
  - beschreibt Anforderungen & Ziele
  - Ergebnisse der Analyse
  - bleibt fachlich, nicht technisch
- **Pflichtenheft – *Wie* wird das umgesetzt?**
  - kommt vom Entwicklungsteam
  - technische Lösung
  - Architektur, Technologien, Schnittstellen
  - dient als verbindliche Arbeitsgrundlage

Nachvollziehbarkeit (Traceability) & IDs

- jede Anforderung bekommt eine **eindeutige ID**

Beispiele:

- FR001 – Aufgabe anlegen
- FR002 – Aufgabe löschen
- NFR005 – Antwortzeit 200ms

Priorisierung von Anforderungen

- Grundlage von technischen und Management-Entscheidungen
- Kompromisse zwischen in Konflikt stehenden Anforderungen finden
- Releases der Software planen (zuerst die wichtigen Anforderungen)

Die MoSCoW-Priorisierungstechnik

MoSCoW

MUST HAVE (MUSS)

Unbedingt erforderlich bzw. muss zwingend im Produkt vorhanden sein.

SHOULD HAVE (SOLLTE)

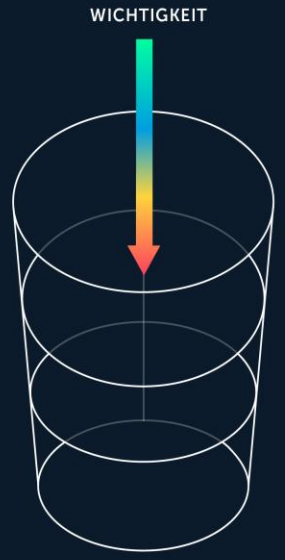
Ist wichtig – das Produkt funktioniert aber auch ohne diese Eigenschaft. Sollte umgesetzt werden, wenn alle MUST-Anforderungen trotzdem erfüllt werden können.

COULD HAVE (KÖNNTE)

Kann bei freien Kapazitäten umgesetzt werden – solange die Erfüllung von höherwertigen Anforderungen nicht beeinträchtigt wird.

WON'T HAVE (NICHT NOTWENDIG)

Wird nicht in dieser Entwicklungsphase umgesetzt, aber eventuell in einer zukünftigen realisiert.



Vor –und Nachteile der MoSCoW-Methode

- **Vorteile:**
  - Einfachheit und Klarheit in der Priorisierung
  - Flexibilität, um auf Veränderungen zu reagieren
  - Förderung von Teamdiskussionen und Konsensbildung
- **Nachteile:**
  - Risiko der Überfüllung in der Kategorie „Must have“
  - Mögliche Vernachlässigung von „Could have“-Elementen, die langfristig Wert bieten könnten
  - Erfordert Disziplin und Erfahrung, um effektiv zu sein

Alternativen

Die Eisenhower-Matrix: Aufgaben nach Dringlichkeit und Wichtigkeit

1. Aufgaben, die wichtig und dringend sind, werden sofort erledigt.
2. Aufgaben, die wichtig, aber nicht dringend sind, werden geplant.
3. Aufgaben, die dringend, aber nicht wichtig sind, können delegiert werden.
4. Aufgaben, die weder wichtig noch dringend sind, werden am besten gestrichen.

Das Kano-Modell hilft zu erkennen, welche Features Nutzer **begeistern** und welche einfach **erwartet** werden.

1. Basis-Features müssen vorhanden sein, sonst sind Nutzer unzufrieden.
2. Leistungs-Features steigern die Zufriedenheit, je besser sie umgesetzt werden.
3. Begeisterungs-Features bringen zusätzlichen Wow-Effekt, auch wenn sie nicht erwartet werden.

Priorisierung von Anforderungen

- Essentiell (MUSS): die Anforderung muss implementiert werden, sonst ist das System nutzlos
- Notwendig (SOLL): das System ist ohne Umsetzung dieser Anforderung weniger effizient/effektiv
- Wünschenswert (KANN): das System wäre mit dieser Anforderung attraktiver
- Nicht Notwendig

Beispiel Funktionale Anforderungen (1/2)

- **Funktion "Vorlesung in Vorlesungverzeichnis eintragen"**
  - Eingaben: Raum, Zeit und Titel einer Vorlesung.

- **Verarbeitungsschritte**
  - Prüfe, ob der Vorlesungstitel schon vergeben ist
    Prüfe, ob der Raum zur angegebenen Zeit schon vergeben ist
    Wenn nicht, wird die neue Vorlesung eingetragen und die Daten der Vorlesung werden angezeigt
  - Falls vergeben, wird die Vorlesung nicht eingetragen und eine entsprechende Fehlermeldung wird angezeigt.
- **Ausgaben** Die Vorlesung wird angezeigt oder ein Fehler wird gemeldet

Beispiel Funktionale Anforderungen (2/2)

- **Aktionen, die vom System ausgeführt werden sollen**

Z.B.: Das System muss Vorlesungen in die DB aufnehmen können

- **Systeminteraktionen, die dem Nutzer ermöglicht werden**

Z.B.: Das System muss es dem Administrator beim Eintragen einer Vorlesung ermöglichen, Raum, Zeit und Titel einzugeben

- **allg. funkt. Vereinbarungen und Einschränkungen**

Z.B.: Der Client ist für den Kommunikationsaufbau zuständig.

Übung 1a: Funktionale Anforderungen für eine TODO-App (15 min)

- Anforderungen zur TODO-App formulieren
  - 10–15 FRs
  - Priorisierung (MUSS/SOLL/KANN/NICHT NOTWENDIG)
  - Ausgabe als Tabelle

ID	Anforderungstyp	Beschreibung	Priorität
FR-01	Funktional	Das System muss …	MUSS
…			

Übung 1b: Anforderungskatalog pro Gruppe (15 min)

- **1 Reihe = 1 Gruppe**
- **Aufgabe jeder Gruppe:**
- Jedes Gruppenmitglied legt *kurz* seine Tabelle vor.
- Die Gruppe markiert auf einem gemeinsamen Blatt:
  - **FRs, die mindestens 3 Personen haben → „Kandidat FR“**
- Jede Gruppe erstellt aus diesen Kandidaten:
  - **Top 8 FRs**
  - Erste Priorisierung (MUSS/SOLL/KANN/NICHT NOTWENDIG)

https://docs.google.com/spreadsheets/d/1u5fnYFSJQpygpkXaMjAUXIXvPrsEpE9hO7897cXUfOI/edit?usp=sharing

STREAMLIT

Warum Streamlit?

- Keine HTML/CSS/JS-Kenntnisse nötig
- In Minuten lauffähige Web-Apps
- Ideal für Prototypen & Lehre
- Funktioniert direkt mit Python-Bibliotheken
- Wenig Code, schnelle Ergebnisse

Wie man Streamlit benutzt

- Jede Streamlit-App ist eine einzige Python-Datei, z. B. app.py.

- **Die App läuft lokal über:** streamlit run app.py
- **Streamlit installieren:** pip install streamlit
- **Teste, ob die Installation funktioniert hat:** streamlit hello

Wichtige Elemente

- **Texte mit Streamlit anzeigen**
  - st.title(), st.header(), st.subheader(), st.caption(), st.code(), st.write()
- **Media**
  - st.image(), st.audio(), st.video()
- **Eingabe-Widgets**
  - st.text_input
  - st.button
  - st.checkbox
- **Fortschritt und Status:** st.balloons(), st.progress()

Session State

Warum nötig?

- Speichert Aufgabenliste
- Überlebt Interaktionen

Beispiel:

```
if 'todos' not in st.session_state:
    st.session_state.todos = []
```

Übung: Minimaler Aufbau TODO-App mit Streamlit (30 min)

- Aufgabe: Eine funktionale, minimalistische TODO-App erstellen.
- **Grundfunktionen**:
  - Session State initialisieren
  - Möglichkeit Aufgaben hinzufügen (Textfeld, Button, …)
  - Übernehmen Sie die Aufgabe in die Liste im Session State
  - Lassen Sie Aufgaben mit Checkboxen anzeigen
  - Löschfunktion

Bonus

- Filter (offen / erledigt)

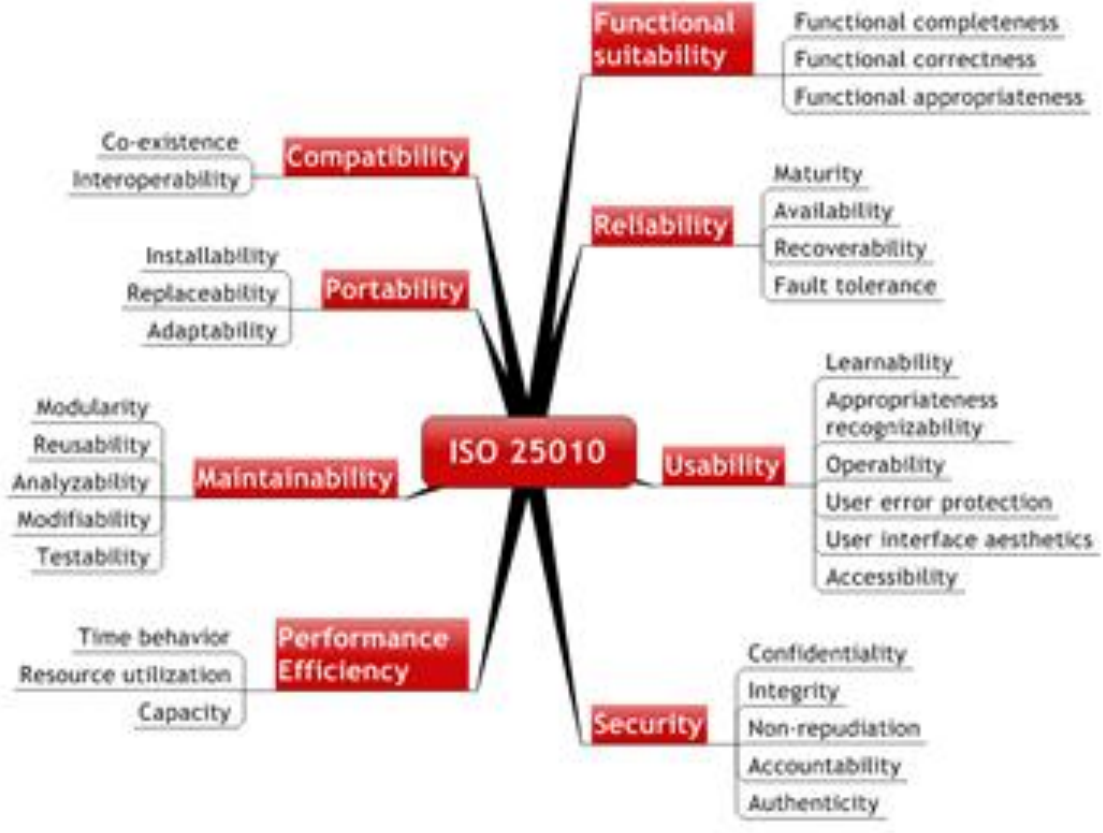
Nicht Funktionale Anforderungen (NFR)

Software-Qualitätseigenschaften gemäß ISO 25010

- Was ist ISO?
- Die ISO 25010 ist

die Nachfolge-Norm

der ISO 9126



https://www.johner-institut.de/blog/iec-62304-%20medizinische-software/iso-9126-und-iso-25010/#iso25010/

Beispiele der NFR

- **Usability** → Wir formulieren z. B. *„Ein Nutzer soll eine Aufgabe mit maximal drei Klicks anlegen können.“*
- **Reliability** → Daraus wird z. B. *„Eine Aktion soll unter identischen Bedingungen stets das gleiche Ergebnis liefern.“*
- **Security** → *„Nur angemeldete Nutzer dürfen Aufgaben löschen.“*
- **Performance** → *„Die App muss innerhalb von zwei Sekunden startbereit sein.“*

Quiz: NFR – SMART oder nicht?

1. Die App soll benutzerfreundlich sein.
2. Das System muss innerhalb von 200 Millisekunden auf eine Eingabe reagieren.
3. Die Anwendung soll zuverlässig laufen.
4. Alle Daten müssen verschlüsselt gespeichert werden.
5. Die App soll modern aussehen.
6. Das System muss mindestens 99,5% Verfügbarkeit pro Monat gewährleisten.
7. Die Ladezeit der Startseite soll unter 1 Sekunde liegen.
8. Die App soll intuitiv bedienbar sein.
9. Passwörter müssen nach dem aktuellen Sicherheitsstandard gehasht gespeichert werden.
10. Die Anwendung soll mit allen aktuellen Browsern kompatibel sein.

Übung 3a: Nicht Funktionale Anforderungen für eine TODO-App

- Anforderungen zur TODO-App formulieren
  - Mindestens 10 NFRs
  - Priorisierung (MUSS/SOLL/KANN/NICHT NOTWENDIG)
  - Ausgabe als Tabelle

ID	Anforderungstyp	Beschreibung	Priorität
…			
NFR-01	Nicht-Funktional	Die App soll …	SOLL

Übung 3b: Gemeinsamer Anforderungskatalog für NFR

- **1 Schritt (10 Minuten): 1 Reihe = 1 Gruppe**
- **Aufgabe jeder Gruppe:**
- Jedes Gruppenmitglied legt *kurz* seine Tabelle vor.
- Die Gruppe markiert auf einem gemeinsamen Blatt:
  - **NFRs, die mindestens 3 Personen haben → „Kandidat NFR“**
- Jede Gruppe erstellt aus diesen Kandidaten:
  - **Top 5 NFRs**
  - Erste Priorisierung (MUSS/SOLL/KANN/NICHT NOTWENDIG)

Übung 3c: Gruppenabgleich (10 Minuten)

- Jede Gruppe überträft alle Anforderungen (FR und NFR) in eine gemeinsame Tabelle
- Jeder bekommt 10 digitale Votes („x“):
  - **6 Punkte für FRs**
  - **4 Punkte für NFRs**
- Bitte verteilen Sie die Kreuzchen so, wie Sie es für sinnvoll halten.
- Regel: **Pro Person max. 2 Punkte pro Anforderung**, damit nicht alles auf eine einzige fällt

Übung 3d: Der gemeinsame Nenner

- **FR-Übernahme-Regeln**
- **FRs mit ≥ 20 Punkten → MUSS**
- **FRs mit 10–19 Punkten → SOLL**
- **FRs mit 5–9 Punkten → KANN**
- **FRs mit < 5 Punkten → nicht im MVP**
- **NFR-Übernahme-Regeln**
- **Top 3 NFRs werden MUSS**
- Alle anderen NFRs → SOLL oder KANN je nach Punktzahl

What‘s next?

Bitte Figma Educational beantragen

- https://help.figma.com/hc/de/articles/360041061214-Figma-for-Education