DHBW Stuttgart

Duale Hochschule
Baden-Württemberg

Software Engineering

Dr.-Ing. Eugenie Giesbrecht

Über mich

- M.Sc. Cognitive Science, Osnabrück
- Dr-Ing. am Fakultät für Angewandte Informatik und Formale Beschreibungsverfahren am KIT
- Head of AI and DS, Arnulf Betzold GmbH
- >15 Jahre Berufserfahrung in AI & Data Science (u.a. Forschung, SAP, Siemens, Deutsche Bahn, Daimler, Bosch, 1&1, BMWi, Deutsche Rente, IBM)

- Kontakt:
  Dr. Eugenie Giesbrecht
  eugenie.giesbrecht@lehre.dhbw-stuttgart.de

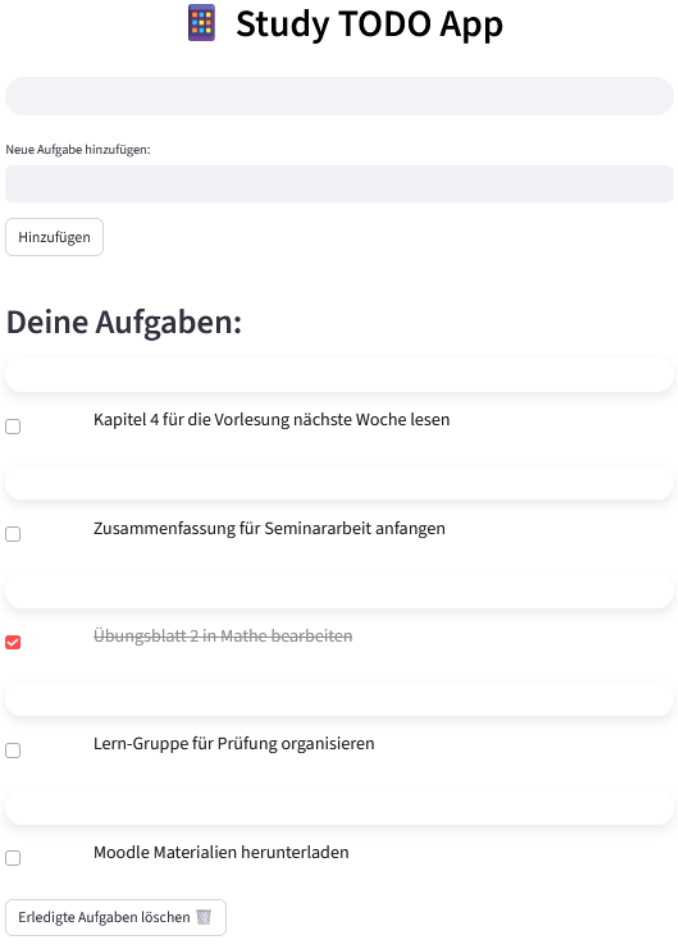
Let‘s get to know each other better!

https://excalidraw.com/#room=35bda51986f693b8e519,aLuUMTZxIrrG_ZNd-K_FUw

~~Agenda~~ Themen-Backlog

- Was ist Software Engineering
- Vorgehensmodelle
- Software-Architekturen
- Software-Design
- Software-Tests & Verifikation
- Software-Wartung
- …
- Schrittweise Entwicklung einer eigenen App

Vorlesungsziel ☺



Organistorisches: Aufbau

- Sehr praktische Vorlesung: Grundlagen des SE + Fortgeschrittenes SE
- Theorie & Übungen
- 13.15-16.30 Block: 10 min Pausen nach jeder Stunde
- Moodle Kurs

Termine

- 21.11: 13.15-16.30
- 26.11: 13.15-16.30
- 28.11: 13.15-16.30
- 01.12: 13.15-16.30
- 05.12: 13.15-16.30
- 08.12: 13.15-16.30
- 12.12 : 13.15-16.30
- 15.12 ganztägig

- 09.01.26 Zoom : 13.15-16.30
- 16.01.26 Zoom : 13.15-17.30
- 30.01.26 Zoom : 13.15-16.30

- Gruppeneinteilung für das abschließende Assignment zwischen dem 15.12.25 und 09.01.26
- Abgabe des Portfolios bis Ende des Semesters

Bewertungskriterien: Software Engineering (60 Punkte)

Bereich	Punkte	Kriterien
A) Funktionale App: **Mobile/Web-App mit Python + Streamlit + Figma**	20	Funktionalität, UI-Konsistenz, Fehlerfreiheit, Zusätzliche Features
B) Codequalität	10	Architektur, Dateiorganisation, Lesbarkeit, Nutzung SE-Werkzeuge, Testing
C) (Online) Präsentation am 09. Januar	20	Verständnis der eigenen Lösung, Theoretisches Wissen, Anpassungen live
D) Online-Multiple-Choice-Tests zu Beginn jeder Lektion im November und Dezember	10	Theoriewissen

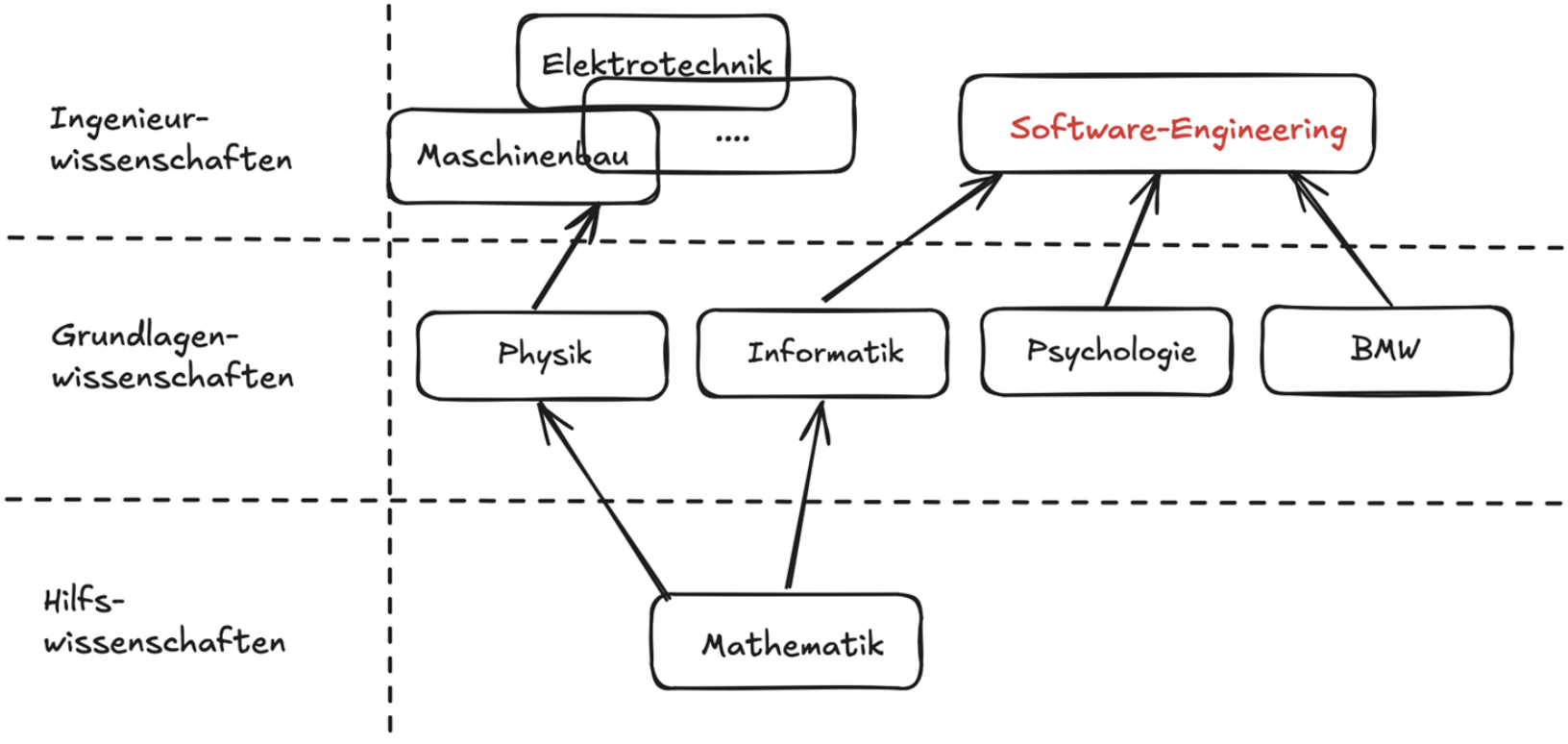
Bewertungskriterien: Fortgeschrittenes Software Engineering (60 Punkte)

Bereich	Punkte	Kriterien
A) Funktionale App	20	Funktionalität, UI-Konsistenz, Fehlerfreiheit, Cross-Plattform-Ausführbarkeit
B) Codequalität	20	Architektur, Verwendung von Design Patterns, Anwendung der UI-Prinzipien, Durchgeführte Tests
C) Dokumentation	10	Requirements, UML-Diagramme, Beschreibung der eingesetzten Architektur- & Design-Patterns, Testdoku
D) Online-Test (30.01.26)	10	Theoriewissen: Vorlesungsthemen

DHBW Stuttgart
Duale Hochschule
Baden-Württemberg

Was ist Software-Engineering?

Was ist Software-Engineering?



Gebietseinordnung nach Broy & Rombach, ss.438-451

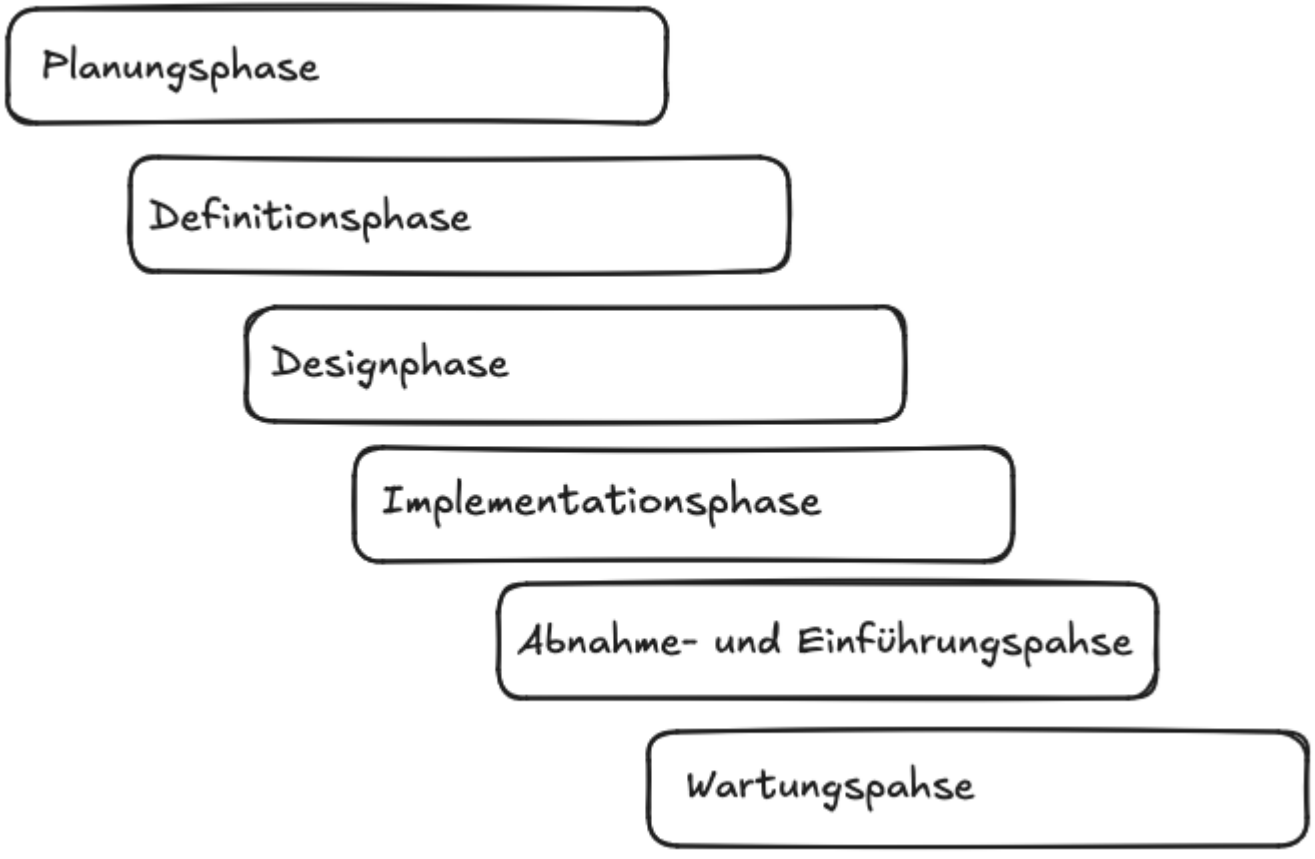
Disziplinen des Software Engineering

- Requirements Engineering
- Software Design
- Programmierung
- Testen
- Deployment
- Wartung

Rollen in Software Engineering

- Requirements Engineer / Business Analyst
- Software Developer (Frontend/Backend)
- Tester / QA Engineer
- UX/UI Designer
- Project Manager / Scrum Master

Der Software-Lebenszyklus



Übung 1: Lebenszyklus eines realen Projekts (To-Do-App)

- **Gruppenarbeit (2–3 Personen)**
- **Aufgabe**
  - Erstellen Sie eine **Skizze des Software-Lebenszyklus** einer To-Do-App. Formulieren Sie pro Lebenszyklusphase **1–2 kurze SätzeAblauf:**
- **Timing**
  - 15 Min – Gruppenarbeit
  - Inhalte je Phase besprechen
  - 1–2 präzise Sätze formulieren
  - 15 Min - Präsentation
  - Jede Gruppe ca. 2 Minuten

Geschichtlicher Überblick

- Mitte 1960er Jahre
- Mit schnellerer Hardware wurde Software wichtiger
- Steigende Anforderungen, aber
  - Qualifiziertes Personal fehlte
  - Softwareentwicklung durch “Bastelei”
  - Software war unzuverlässig, ständige Wartung
  - Bestehende Systeme intransparent und unübersichtlich, kaum änderbar
  - Kosten und Dauer überstiegen Erwartungen
  - Anforderungen oft nicht erfüllt
- Softwarekosten überstiegen Hardwarekosten
- Große Softwareprojekte scheiterten

Software Engineering & “Software-Krise”

- Juni 1965 (Computers and Automation), 1966 in Zeitschrift der Association for Computing Machinery
- 07.10-11.10.1968 - NATO Software Engineering Conference in Garmisch-Partenkirchen
- Initiator - der Münchner Computerpionier Friedrich Bauer
- Sechzig Experten aus elf Ländern
- “Softwarekrise“

Komplexität der Software-Entwicklung

- Vielzahl von Teilnehmern (Kunden, Architekten, Entwicklern, Testpersonal, Wartungsspezialisten, etc)
- Unterschiedliche Erfahrungen, Verfahren, Methoden
- Wirtschaftliche und gesetzliche Faktoren
- Grenzübergreifende Unterschiede
- Unterschiedliche Ziele vieler Projektteilnehmer

Software Engineering - eine Art Baukastensystem, in dem Standards, Verfahren und Methoden angeboten werden, um einen möglichst erfolgreichen Projektverlauf zu gewährleisten

Die Schlussfolgerungen aus der Software-Krise

- Software-Entwicklung entsprach dem Bau von Häusern ohne Architektur, Pläne, Maschinen,…
- Erstellung von Software sollte nicht länger kreative Kunst sondern ingenieurmässige Wissenschaft sein mit wohldefinierten Vorgehensweisen
- Deshalb Einführung von Software-Engineering für die Erstellung von Software-Systemen (… Softwareentwicklung, Softwaretechnik …)
- **Zitat nach Dijkstra [Di72]:**

„Als es noch keine Rechner gab, war auch das Programmieren noch kein Problem, als es dann ein paar leistungsschwache Rechner gab, war das Programmieren ein kleines Problem und nun, wo wir gigantische Rechner haben, ist auch das Programmieren zu einem gigantischen Problem geworden. In diesem Sinne hat die elektronische Industrie kein einziges Problem gelöst, sondern nur neue geschaffen. Sie hat das Problem geschaffen, ihre Produkte zu nutzen.“

Therac 25 (1985-1987): kleine Fehler, große Folgen

Werbefilm

Der teuerste Softwarefehler aller Zeiten: Ariane 5 - Rakete (1996)

- 4. Juni 1996 in Französisch Guyana
- Knapp 40 Sekunden nach ihrem Start ging die Ariane 5 in Flammen auf
- lediglich eine einzige Codezeile im System

Video

Rakete Ariane-5

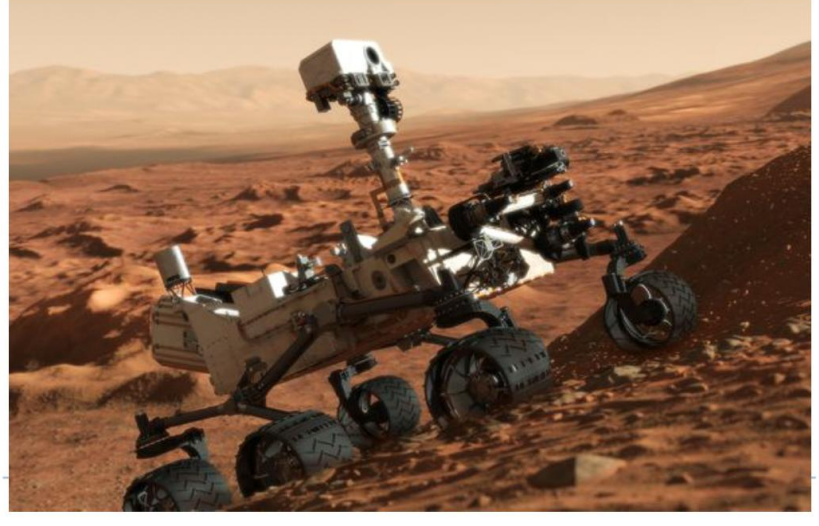
Falsche Software war schuld am Debakel

Am 4. Juni 1996 hob erstmals eine europäische Ariane-5-Rakete ab. Der Flug dauerte allerdings nur gut 30 Sekunden. Die Rakete kam vom Kurs ab, zerbrach und explodierte. Am Weltraumbahnhof Kourou regneten glühende Trümmer vom Himmel.

Lorenzen, Dirk | 11. Juni 2024, 02:57 Uhr

Scheitern der Mars-Spirit-Mission (1999) durch einen Einheitenfehler im Navigationssystem

- kritischer Fehler im Navigationssystem: imperiale versus metrische Einheiten
- etwa 57 Kilometer statt der vorgesehenen 150-170 km



Knight Capital Crash (2012): ein Albtraum in nur 45 Minuten

- Knight Capital: Große Wall-Street-Handelsfirma, spezialisiert auf Hochfrequenzhandel – täglich Milliardenbewegungen
- August 2012: Neue Software für spezielles Börsenprogramm
- Ein alter Test-Code wurde versehentlich aktiviert:
  - 4 Mio. Trades in 45 Minuten
  - 150 Aktien betroffen, fast 397 Mio. Aktien bewegt
  - Verlust: 440 Mio. USD

USA, Kanada, Weltraum, …

“Since 1989, some 120 Million Deutschmarks have been wasted trying to design and install a new computer system for the police in Hamburg, Germany. The system, designed to ease the load of paperwork on the police never worked. … What strikes me are the three conclusions of the whole affair that appeared in Hamburger Abendblatt, 16 April 1998 … ”[ACM SIGSOFT Software Engineering Notes, vol. 23, no. 4 (1998), S. 22]

- 😟 365 Jobs wurden gestrichen und es gab keine Pläne für Neueinstellungen. Polizeioffiziere mussten die Papierarbeit übernehmen
- 😟 Die für das Design des Systems verantwortliche Beratungsfirma führt als einen Grund des Desasters an: ‘police people were in charge of the development team. Software design was not one of their core competences.’
- 😟 Der Innenminister Hartmuth Wrocklage führte die Probleme darauf zurück, dass die Komplexität des Projekts unterschätzt worden war

Und wo standen wir 10 Jahre später?

- Über 10000 IT-Projekte im Zeitraum zwischen 2011-2015
- Im Mittel sind nur 11-39% der IT-Projekte gelungen.
- Schon 52-60% werden zumindest als „herausfordernd" beschrieben und
- 9-29% gelten gar als gescheitert

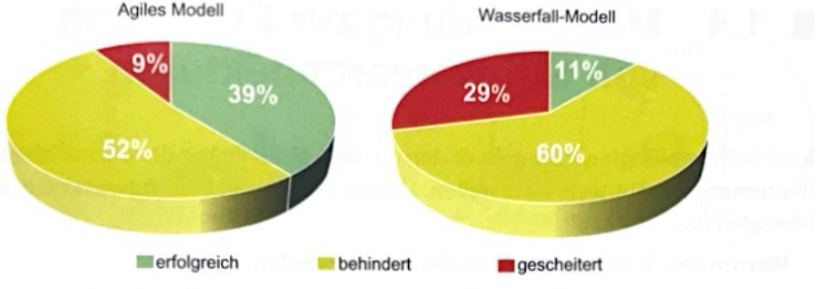


Abbildung 1.4 Standish Group CHAOS-Studie 2011–2015 (>10.000 Software-Projekte jeder Größe)

Übung 2a: Analyse realer Softwarefehler (10 Minuten)

Gruppen: 2–3 Personen

Aufgabe: Analyse eines realen historischen Softwarefehlers

Beantworten Sie gemeinsam die folgenden Fragen:

- Was ist passiert?
- Was war der technische Fehler?
- Was war der organisatorische Fehler?
  (z. B. Prozesse, Tests, Kommunikation, Dokumentation)
- Welche Maßnahme hätte den Fehler verhindern können?

=> Tragen Sie die Ergebnisse stichpunktartig zusammen.

Vorschläge

- **Ariane 5 Explosion** (1996)
  Integer-Overflow konvertiert 64-bit float → 16-bit int → Raketensteuerung stürzt ab → 370 Mio $ Schaden.
- **Mars Climate Orbiter** (1999)
  Ein Team nutzte metrisches System, das andere imperial → Raumsonde stürzt ab → 193 Mio $.
- **Knight Capital Crash** (2012)
  Fehlerhafte Deployment-Prozesse aktivierten alten Testcode → Börsenalgorithmus kaufte zu Milliardenpreisen → 440 Mio $ Verlust in 45 Minuten.
- **Therac-25** (1985–87)
  Race conditions + fehlende Safety-Checks → 6 Patienten erhalten tödliche Strahlendosis.

Übung 2b: Wie konnte es zu einem Fehler in Milliardenhöhe in unserer TODO-App kommen?

Gruppen: 2–3 Personen

- Welcher Fehler könnte in unserer TODO-App ähnlich passieren?
- Wie hoch wäre der Schaden bei einer echten Firma, die diese App verkauft?
- Welche Maßnahmen sollten wir in unserem Projekt einführen, um es zu verhindern?

=> Tragen Sie die Ergebnisse stichpunktartig zusammen.

Was ist Softwarequalität?

- „Software quality refers to the degree to which software conforms to its requirements and meets the needs of its users. It is formally defined as “the capability of a software product to satisfy stated and implied needs when used under specified conditions.” Another definition states that software quality depends on “the degree to which those established requirements accurately represent stakeholder needs, wants, and expectations.” High quality software meets its requirements, which in turn should accurately reflect stakeholder needs. Quality is about aligning the software with both its formal requirements as well as true user needs.“

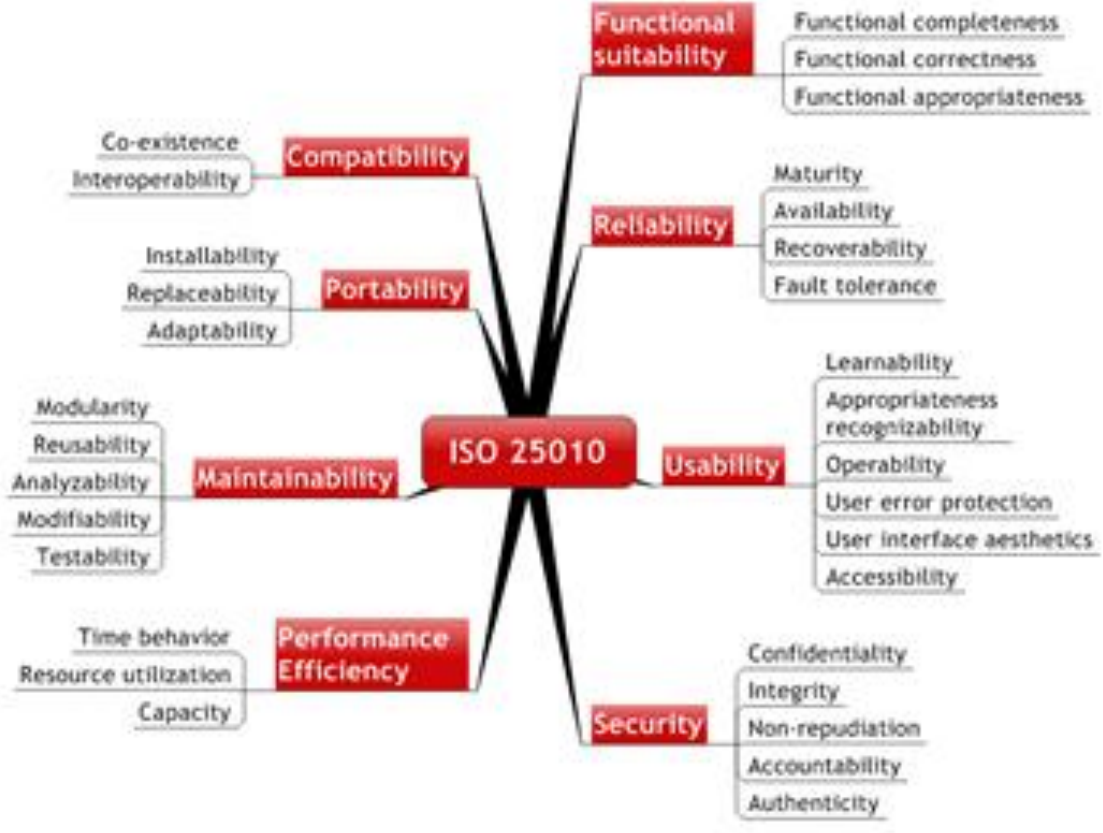
https://www.computer.org/resources/what-is-software-quality#software-quality

Software-Qualitätseigenschaften gemäß ISO 25010

- Was ist ISO?
- Die ISO 25010 ist

die Nachfolge-Norm

der ISO 9126

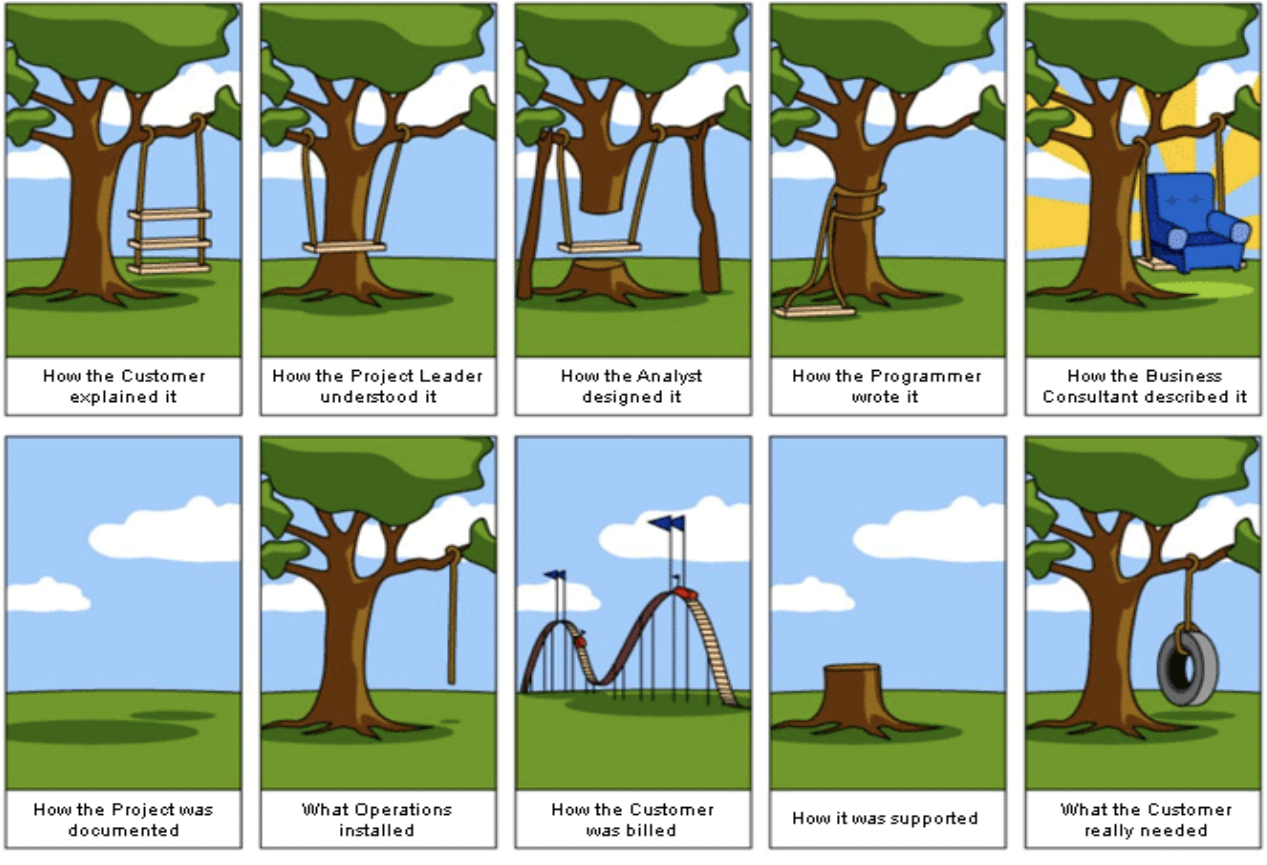


https://www.johner-institut.de/blog/iec-62304-%20medizinische-software/iso-9126-und-iso-25010/#iso25010/

Normen als Checklisten

ISO 25010-Aspekt	(G)UI: Nutzer-Schnittstelle	System-Schnittstellen	Laufzeit-umgebung
Funktionalität (Functionality)	x	x	x
Zuverlässigkeit (Reliability)	x	x	x
Gebrauchstauglichkeit (Usability)		x	
(IT-)Sicherheit (Security)	x	x	x
Effizienz (Efficiency)	x	x	x
Wartbarkeit (Maintainability)			
Portabilität (Portability)			x
Kompatibilität (Compatibility)		x	

WORAN scheitert's?



Fehlerarten: Error, Defect, Failure

- Error – „menschlicher“ Fehler
- Defect – Fehler im Produkt
- Failure - Ausfall

Einfluss der Organisationskultur: Positive kulturelle Einflüsse

- Exzellenz wird über schnelle Veröffentlichung gestellt
- Management unterstützt „Dinge richtig machen“ → technische Schulden minimieren
- Ethik und Verantwortung haben Vorrang vor Gewinn oder Zeitplan
- Hochwertige Arbeit wird belohnt und anerkannt
- Lern- und Verbesserungs-Kultur statt Schuldzuweisung
- Zusammenarbeit intern und mit Kunden wird gefördert
- Vielfalt bringt unterschiedliche Perspektiven auf Qualität
- Kundenzufriedenheit als zentraler Maßstab

Einfluss der Organisationskultur: Negative kulturelle Einflüsse

- Führungskräfte engagieren sich nicht für Qualität
- Unterinvestitionen in Schulungen, Tools und Prozesse
- Konflikte zwischen technischen und geschäftlichen Zielen
- Schlechte Kommunikation, schwache Teamdynamik
- Häufige Überstunden → Termindruck
- Mangelnde Transparenz

Übung 3: Schlechte Softwarequalität erkennen

- Arbeit in Gruppen: 2–3 Personen, 20 Minuten
- **Aufgabe:** Jede Gruppe
  - bearbeitet **ein Szenario schlechter Softwarequalität** (siehe nächste Folie).
  - Ordnet das Problem einem **ISO-25010-Qualitätsmerkmal** zu.
  - Beantwortet: Warum ist das Problem schlecht? (Aus Nutzersicht? Aus Entwicklersicht?) Welche Maßnahmen könnten das Problem beheben?

10 kurze Szenarien schlechter Qualität, je Gruppe ein Szenario:

- Eine App friert beim Scrollen ein
- UI hat vier Buttons mit identischem Icon
- Backend speichert Passwörter im Klartext
- Kein Ladeindikator bei langen Berechnungen
- Feature funktioniert nur auf Chrome, nicht auf Firefox
- Code ist 5000 Zeilen in einer Datei ohne Struktur
- App stürzt ab, wenn das Netzwerk langsam ist
- Fehlerhafte Rechtschreibung und unklare Beschriftungen in Menüs
- Keine Zugangsbeschränkung – jeder kann alles bearbeiten
- App lässt sich nur auf Android installieren, nicht auf iOS