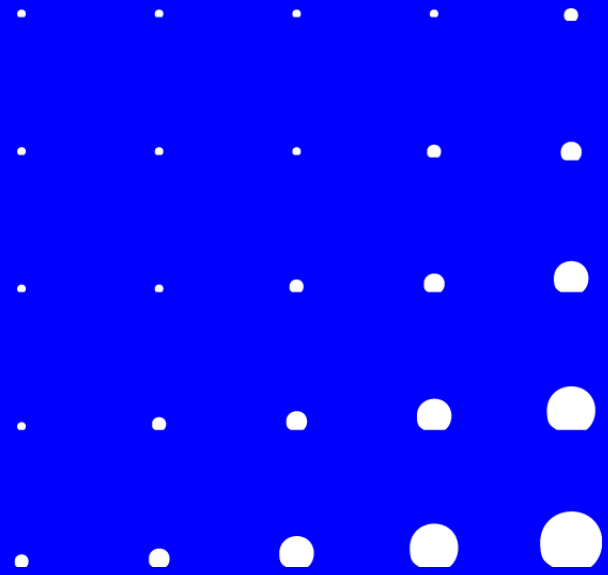
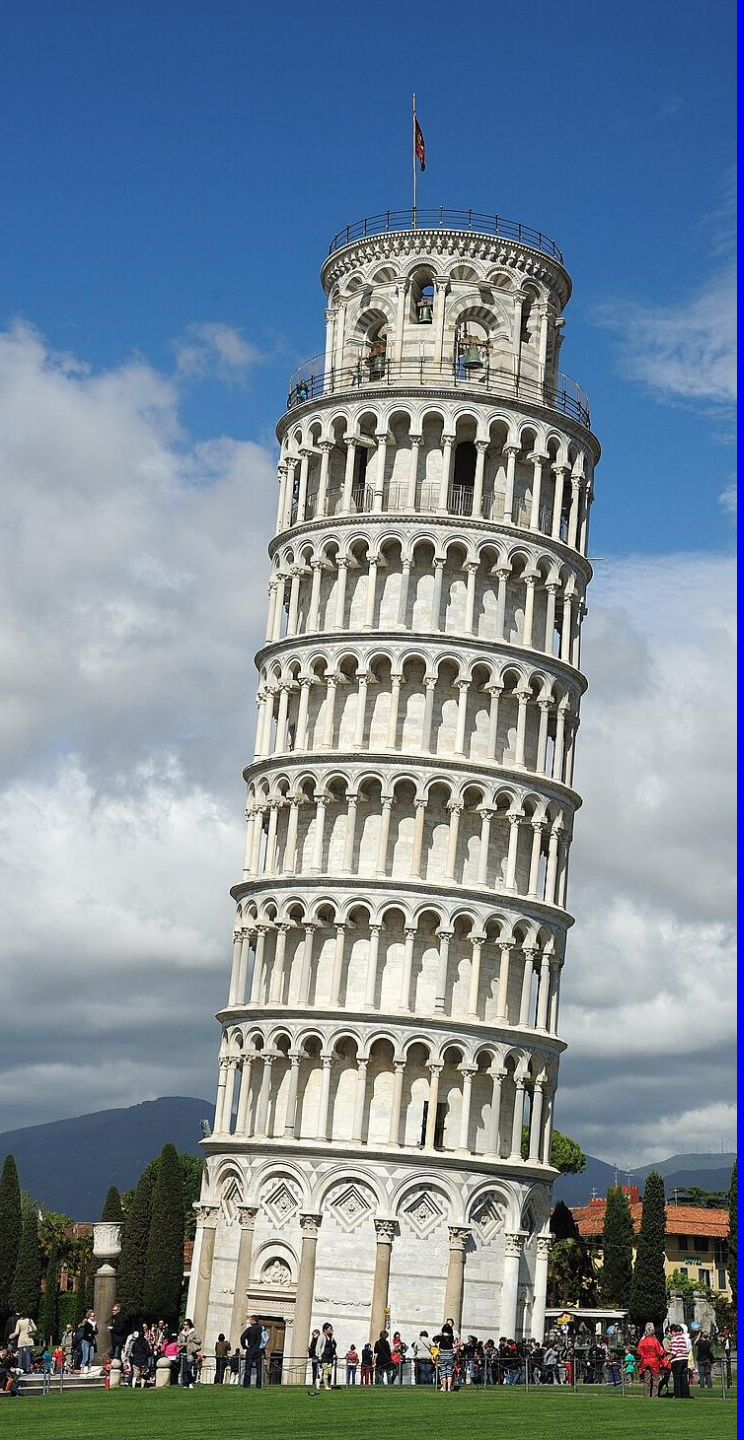
Technische Hochschule
mannheim

Software Engineering 2 – Teil 2

Ingenieurmäßige Entwicklung moderner Software

Stephan Mechler





Technische Hochschule
mannheim

Software Engineering

In der mittelalterlichen Baukunst entstanden Bauwerke aus überliefertem Erfahrungswissen und Trial-and-Error.

Softwareentwicklung funktionierte lange genauso.

Erst die Softwarekrise der 1960er-Jahre machte daraus eine systematische Ingenieurdisziplin.

Abbildung 1: The Leaning Tower of Pisa. Quelle: Saffron Blaze (2006), Wikimedia Commons, CC BY-SA 3.0.

Über diese Vorlesung

Technische Hochschule mannheim

Ziel

Systematische, ingenieurmäßige Entwicklung von Software verstehen und anwenden

Umfang

30 Lehrveranstaltungsstunden, Bachelor (fortgeschritten)

Voraussetzungen

Grundlagen der Programmierung, Objektorientierung, Datenstrukturen

Prüfungsform

Klausur und Projektarbeit

Agenda

Technische Hochschule mannheim

1 Einführung in das Thema
2 Bedarf an ingenieurmäßigem Vorgehen
3 Softwarelebenszyklus & Vorgehensmodelle
4 Zeitgemäße Methodiken (Agile, DevOps)
5 Anforderungen – Requirements Engineering
6 Softwaredesign
7 Softwarearchitektur
8 Testen von Software
9 Wartung & Softwareevolution
10 Versionsverwaltung mit Git
11 Code-Erstellung mit KI-Werkzeugen

1 Einführung in das Thema

Was ist Software Engineering – und warum brauchen wir es?

Was ist Software?

Technische Hochschule mannheim

Definition

Programme samt zugehöriger Daten und Dokumentation, die zur Lösung einer Aufgabe zusammenwirken

Mehr als nur Quellcode

Konfiguration, Daten, Betriebsumgebung und Dokumentation gehören dazu

Besondere Eigenschaften

Immateriell und nahezu kostenlos kopierbar

Kein physischer Verschleiß – aber „Alterung“ durch Änderungen und Umfeld

Hohe Komplexität, schwer sichtbar und schwer messbar

Software vs. klassische Ingenieurprodukte

Technische Hochschule mannheim

Kein Verschleiß

Software „rostet“ nicht – Fehler sind von Beginn an enthalten

Keine Fertigung im klassischen Sinn

Die Herstellung ist reines Kopieren; der Aufwand steckt in der Entwicklung

Hohe Änderbarkeit – Fluch und Segen

Leicht zu ändern, dadurch aber auch leicht zu verschlechtern

Unsichtbarkeit (F. Brooks)

Struktur ist nicht räumlich anschaulich – Modelle schaffen Abhilfe

Was ist Software Engineering?

Technische Hochschule mannheim

IEEE 610.12

Die Anwendung eines systematischen, disziplinierten und quantifizierbaren Ansatzes auf Entwicklung, Betrieb und Wartung von Software

Kernidee

Ingenieurmäßiges, methodisches Vorgehen statt „Programmierkunst“ im Alleingang

Drei Dimensionen

Mensch & Organisation – Prozess & Methoden – Werkzeuge & Technik

Abgrenzung

Programmieren ≠ Software Engineering: Engineering umfasst den gesamten Lebenszyklus

Die Softwarekrise

Technische Hochschule mannheim

NATO-Konferenz 1968 (Garmisch)

Begriff „Software Engineering“ wird geprägt – als Antwort auf eine Krise

Symptome

Projekte überschreiten Zeit und Budget massiv

Geringe Qualität, unzufriedene Anwender, schwer wartbare Systeme

Steigende Komplexität übersteigt die verfügbaren Methoden

Konsequenz

Ruf nach systematischen, wiederholbaren Methoden – Geburt der Disziplin

http://homepages.cs.ncl.ac.uk/brian.randell/NATO/

Technische Hochschule
mannheim

37 Sekunden

Ariane 5, Flug 501, 1996: Ein wiederverwendetes Softwaremodul der Ariane 4 schrieb eine 64-Bit-Zahl in ein 16-Bit-Feld. Der Überlauf riss die Rakete auseinander – 370 Mio. Dollar. Kein einziges Bauteil war defekt.

Abbildung 2: Start der Ariane 5 ECA. Quelle: Thilo Kranz / DLR / Ariane, CC BY 3.0 DE, Ausschnitt

Wenn Softwarefehler teuer werden – Beispiele

1

Therac-25 (1985–87)

Bestrahlungsgerät

Race Condition in der Steuerung

Tödliche Überdosen

→ Sicherheitskritische SW

2

Ariane 5 / Flug 501 (1996)

Überlauf bei 64→16-Bit-Konvertierung

Wiederverwendung ohne Test

Rakete gesprengt

→ ca. 370 Mio. $ Schaden

3

Mars Climate Orbiter (1999)

Einheiten verwechselt

Newton vs. Pound-Force

Sonde verglüht

→ Schnittstellenfehler

Warum scheitern Softwareprojekte?

Technische Hochschule mannheim

Häufig genannte Ursachen (z. B. in Branchenstudien wie den Standish CHAOS-Reports)

- Unklare, unvollständige oder sich ändernde Anforderungen
- Mangelnde Einbindung der Nutzer und Stakeholder
- Unrealistische Planung von Zeit und Budget
- Fehlendes Management der Risiken und Änderungen
- Unzureichende Qualitätssicherung und Tests

Erfolgsfaktoren

- Klare Ziele, Nutzerbeteiligung, kompetentes Team, iteratives Vorgehen

Qualitätsmerkmale von Software – ISO/IEC 25010

Technische Hochschule mannheim

Acht Produktqualitätsmerkmale

Funktionale Eignung – Performanz – Kompatibilität

Benutzbarkeit (Usability) – Zuverlässigkeit

Sicherheit (Security) – Wartbarkeit – Portabilität

Wichtige Unterscheidung

Äußere Qualität (Nutzersicht) vs. innere Qualität (Entwicklersicht)

Konsequenz

Qualität muss explizit geplant, gemessen und konstruiert werden

Teildisziplinen des Software Engineering

Technische Hochschule mannheim

Requirements Engineering – Ermitteln und Festlegen der Anforderungen

Software-Design & -Architektur – Strukturieren der Lösung

Implementierung – Umsetzung in Code

Qualitätssicherung & Testen – Nachweis der Korrektheit

Konfigurations- & Versionsmanagement – Beherrschen der Artefakte

Projektmanagement – Planung, Steuerung, Risiken

Wartung & Evolution – Betrieb und Weiterentwicklung

Technische Hochschule
mannheim

Kahoot!

Technische Hochschule
mannheim

2 Bedarf an ingenieurmäßigem Vorgehen

Warum Disziplin, Methode und Struktur über Erfolg entscheiden

Was heißt „ingenieurmäßig“?

Technische Hochschule mannheim

Systematisch statt zufällig

Nachvollziehbare, wiederholbare Vorgehensweise statt Einzelfall-Improvisation

Modellbasiert

Abstraktion und Modelle, bevor (und während) gebaut wird

Messbar und bewertbar

Qualität und Fortschritt werden quantifiziert

Arbeitsteilig und werkzeuggestützt

Definierte Rollen, Prozesse und Tools im Team

Warum reicht „drauflos programmieren“ nicht?

Technische Hochschule mannheim

Skalierung

Große Systeme übersteigen die Kapazität einzelner Köpfe

Arbeitsteilung

Viele Beteiligte benötigen gemeinsame Strukturen und Schnittstellen

Lebensdauer

Software lebt oft Jahrzehnte – Wartbarkeit ist entscheidend

Risiko & Kosten

Fehler in kritischen Systemen kosten Geld, Vertrauen oder Leben

Komplexität und ihre Beherrschung

Technische Hochschule mannheim

Wesentliche vs. zufällige Komplexität (Brooks)

Wesentlich: dem Problem inhärent – zufällig: durch Werkzeuge/Vorgehen verursacht

Grundstrategien

Abstraktion – Unwesentliches ausblenden

Zerlegung (Divide & Conquer) – in beherrschbare Teile

Hierarchisierung – Strukturierung in Ebenen

Wiederverwendung – Bewährtes nutzen statt neu erfinden

Grundprinzipien des Software Engineering

1

Abstraktion

Wesentliches herausstellen

Details verbergen

Modelle & Schnittstellen

2

Modularisierung

Zerlegung in Module

Hohe Kohäsion

Lose Kopplung

3

Kapselung

Geheimnisprinzip (Parnas)

Implementierung verbergen

Stabile Schnittstellen

Wirtschaftliche Bedeutung

Technische Hochschule mannheim

Lebenszykluskosten

Der Großteil der Kosten entsteht nach der ersten Auslieferung – in der Wartung

Total Cost of Ownership

Entwicklung, Betrieb, Wartung, Schulung und Stilllegung betrachten

Qualität zahlt sich aus

Gute Struktur senkt langfristige Kosten deutlich

Time-to-Market

Schnelligkeit ist wichtig – aber nicht auf Kosten der Wartbarkeit

Fehlerkosten – die „Rule of Ten“

Technische Hochschule mannheim

Grundregel

Je später ein Fehler entdeckt wird, desto teurer ist seine Behebung

Tendenz (Größenordnung)

Anforderungen → Design → Implementierung → Test → Betrieb

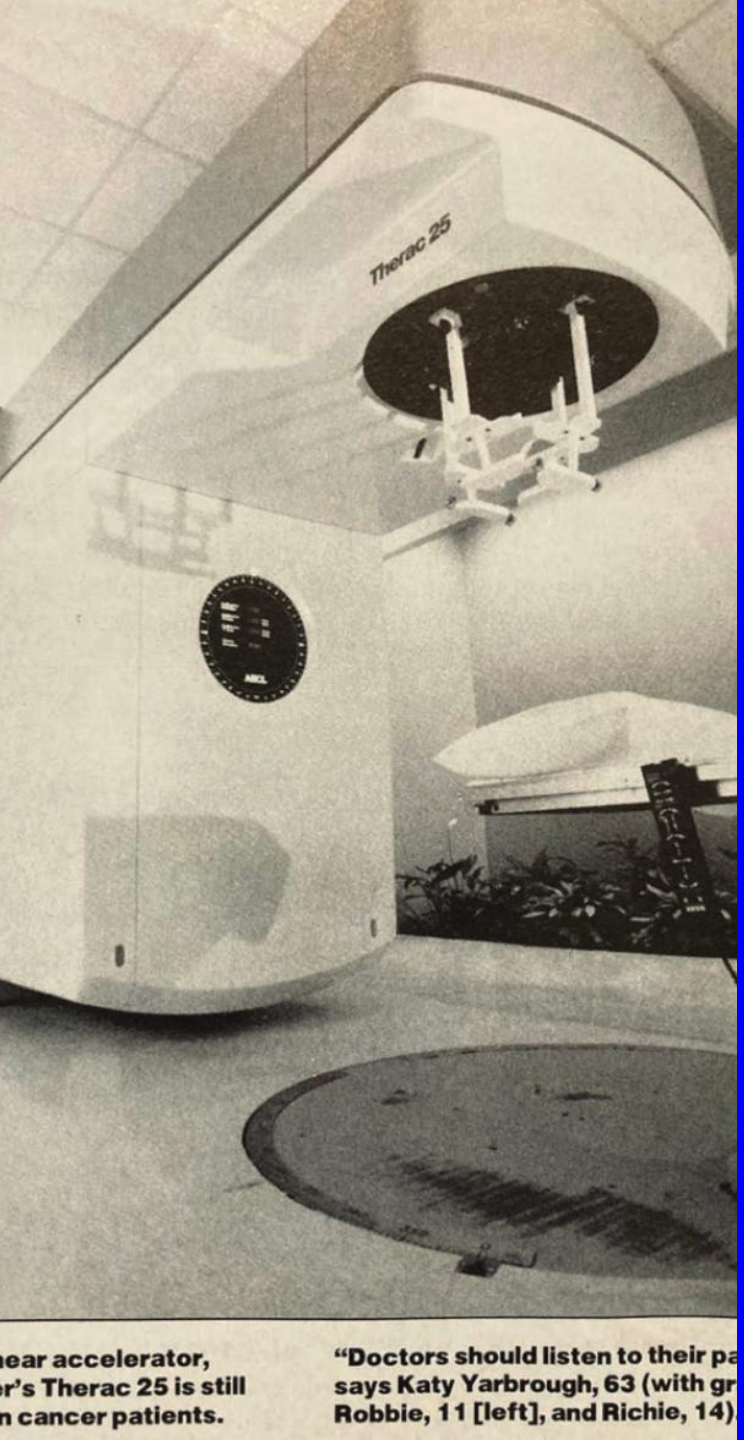
Behebungskosten steigen tendenziell um etwa Faktor 10 je Phase

Einordnung

Heuristik bzw. Größenordnung – keine universelle empirische Regel

Konsequenz

Frühe Qualitätssicherung (Reviews, klare Anforderungen) lohnt sich besonders



Technische Hochschule
mannheim

Sechs Patienten

Therac-25, 1985 bis 1987: Bei schneller Eingabekorrektur gab das Bestrahlungsgerät ein Vielfaches der eingestellten Dosis ab – sechs Menschen starben. Die Steuersoftware war vom Vorgängermodell übernommen. Dort hatte eine mechanische Verriegelung den Fehler abgefangen; beim Therac-25 hatte man sie eingespart.

Abbildung 3: Therac-25. Quelle: Wikimedia Commons (2001), Public Domain (U.S. Government Work).

Rollen im Softwareprojekt

Technische Hochschule mannheim

1	2	3
Fachlich	**Technisch**	**Steuernd**
Auftraggeber / Kunde	Architekt:in	Projektleitung
Product Owner	Entwickler:in	Scrum Master
Fachexperten	Tester:in / QA	Requirements Engineer
Anwender	DevOps / Betrieb	Qualitätsmanagement

3 Softwarelebenszyklus & Vorgehensmodelle

Wie wir Entwicklung über die Zeit strukturieren

Der Softwarelebenszyklus

Technische Hochschule mannheim

Typische Phasen

- Anforderungsanalyse – Was soll das System leisten?
- Entwurf / Design – Wie wird es strukturiert?
- Implementierung – Umsetzung in Code
- Test & Integration – Nachweis der Qualität
- Auslieferung & Betrieb – produktiver Einsatz
- Wartung & Evolution – Pflege über die Lebensdauer
- Stilllegung – geordnete Ablösung

Was ist ein Vorgehensmodell?

Technische Hochschule mannheim

Definition

Ein Rahmen, der Aktivitäten, Reihenfolge, Ergebnisse und Rollen im Projekt festlegt

Zweck

Planbarkeit, Steuerbarkeit und Vergleichbarkeit von Projekten

Zwei große Familien

Plangetrieben (sequenziell) – z. B. Wasserfall, V-Modell

Agil (iterativ-inkrementell) – z. B. Scrum, Kanban

Kein Modell ist universell „richtig“ – Kontext entscheidet

Das Wasserfallmodell

Technische Hochschule mannheim

Herkunft

Häufig auf Royce (1970) zurückgeführt – die spätere Lehrbuch-Interpretation ist jedoch stärker sequenziell als Royces ursprüngliche Argumentation

Idee

Sequenzielle Phasen, jede schließt mit definiertem Ergebnis ab

Ablauf

Analyse → Entwurf → Implementierung → Test → Betrieb → Wartung

Annahme

Anforderungen sind früh vollständig und stabil bekannt

Rückkopplung

Nur eingeschränkt zur jeweils vorherigen Phase vorgesehen

Wasserfallmodell – Bewertung

1

Vorteile

Einfach & verständlich

Klare Meilensteine

Gut dokumentiert

Gut planbar bei Stabilität

2

Nachteile

Unflexibel bei Änderungen

Späte Fehlererkennung

Späte lauffähige Software

Risiko spät sichtbar

3

Geeignet, wenn

Anforderungen stabil

Domäne gut verstanden

Regulatorik/Doku-Pflicht

Festpreisverträge

Technische Hochschule
mannheim

16 Monate zu spät

Denver International Airport, 1995: Das automatische Gepäcksystem wurde als Gesamtsystem geplant und erst am Ende in Betrieb genommen. Im Test zerriss es Koffer und verlor Gepäck. Der Flughafen öffnete 16 Monate später als geplant – die Verzögerung kostete rund eine Million Dollar pro Tag.

Abbildung 4: Denver International Airport. Quelle: Murphpics (2023), Wikimedia Commons, CC BY-SA 4.0.

Das V-Modell

Technische Hochschule mannheim

Grundidee

Erweiterung des Wasserfalls – jeder Konstruktionsphase wird eine Teststufe zugeordnet

Linke Seite (Spezifikation/Konstruktion)

Anforderungen → Grobentwurf → Feinentwurf → Implementierung

Rechte Seite (Integration/Test)

Unit-Test → Integrationstest → Systemtest → Abnahmetest

Kernbotschaft

Verifikation und Validierung werden früh mitgeplant

Das V-Modell XT

Technische Hochschule mannheim

Hintergrund

Standard für öffentliche IT-Projekte in Deutschland; „XT“ = eXtreme Tailoring

Tailoring

Das Modell wird projektspezifisch zugeschnitten – nur relevante Bausteine

Vorgehensbausteine

Projektmanagement, QS, Konfigurationsmanagement, Systemerstellung u. a.

Entscheidungspunkte

Definierte Meilensteine mit prüfbaren Ergebnissen

Inkrementelle Entwicklung

Technische Hochschule mannheim

Idee

Das System wird in mehreren, aufeinander aufbauenden Ausbaustufen geliefert

Vorgehen

Kernfunktion zuerst, dann schrittweise Erweiterung um weitere Funktionen

Vorteile

Früher Nutzen, frühes Feedback, verteiltes Risiko

Voraussetzung

Sinnvolle Zerlegung in eigenständig nutzbare Inkremente

Iterative Entwicklung

Technische Hochschule mannheim

Idee

Wiederholte Durchläufe verfeinern das System schrittweise

Unterschied zu inkrementell

Inkrementell = mehr Funktion; iterativ = bessere/verfeinerte Funktion

In der Praxis

Moderne Verfahren kombinieren beides: iterativ-inkrementell

Nutzen

Unsicherheit wird durch frühe, wiederholte Validierung reduziert

Das Spiralmodell (Boehm, 1986)

Technische Hochschule mannheim

Kernidee

- Risikogetriebenes
- iteratives Modell
- jede Windung adressiert die größten Risiken

Vier Aktivitäten je Zyklus

→ Ziele festlegen
→ Risiken bewerten
→ Entwickeln & testen
→ Nächsten Zyklus planen

Stärke

Explizite Risikoanalyse

geeignet für große, unsichere Projekte

Schwäche

Aufwendig

erfordert Risikomanagement-Kompetenz

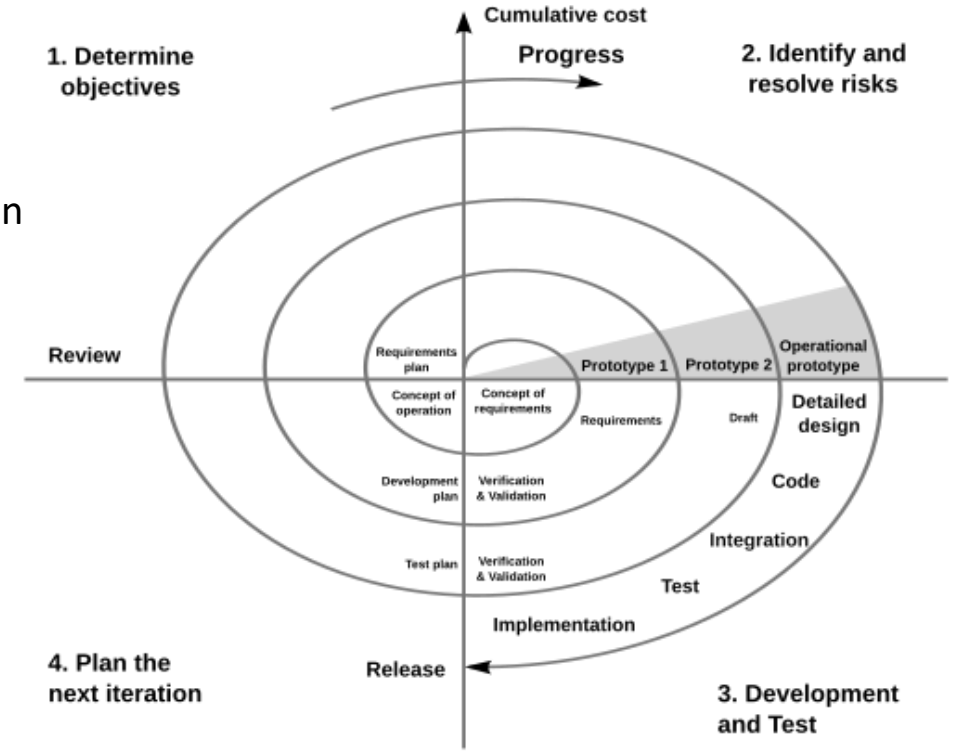


Abbildung 5: Spiralmodell nach Boehm. Quelle: Conny, Wikimedia Commons, Public Domain; nach Boehm (1988).

Rational Unified Process (RUP)

Technische Hochschule mannheim

Charakter

Iterativ, anwendungsfallgetrieben und architekturzentriert

Vier Phasen

Inception → Elaboration → Construction → Transition

Disziplinen

Quer zu den Phasen: Anforderungen, Analyse/Design, Implementierung, Test …

Einordnung

Brücke zwischen schwergewichtig-plangetrieben und iterativ

Prototyping

Technische Hochschule mannheim

Zweck

Unsicherheiten klären, indem früh ein (Teil-)Modell gebaut wird

Arten

Wegwerf-Prototyp – nur zur Klärung, wird verworfen

Evolutionärer Prototyp – wächst zum Produkt

Horizontal (Oberfläche) vs. vertikal (eine Funktion komplett)

Risiko

„Quick & dirty“-Prototypen werden ungewollt zum Produkt

Auswahl eines Vorgehensmodells

Technische Hochschule mannheim

1

Eher plangetrieben

Stabile Anforderungen

Hohe Doku-/Regulatorik

Großes Festpreisprojekt

Sicherheitskritisch

2

Eher agil

Unklare Anforderungen

Schnelles Feedback nötig

Kleines, erfahrenes Team

Hohe Änderungsrate

3

Entscheidungsfaktoren

Risiko & Größe

Domäne & Erfahrung

Kundenverfügbarkeit

Unternehmenskultur

4 Zeitgemäße Methodiken

Agile Entwicklung, Lean und DevOps

Warum agile Methoden?

Technische Hochschule mannheim

Probleme schwergewichtiger Prozesse

Lange Vorlaufzeiten, späte Ergebnisse, hoher Dokumentationsaufwand

Anforderungen ändern sich schneller als die Planung

Beobachtung

Der Kunde weiß oft erst beim Sehen, was er wirklich will

Antwort der Agilität

Kurze Zyklen, kontinuierliches Feedback, Anpassung statt starrer Plan

Funktionierende Software als wichtigstes Fortschrittsmaß

Das Agile Manifest (2001) – vier Werte

Technische Hochschule mannheim

1

Menschen

Individuen und Interaktionen

über

Prozesse und Werkzeuge

2

Software & Kunde

Funktionierende Software

über umfangr. Doku

Zusammenarbeit mit Kunden

über Vertragsverhandlung

3

Wandel

Reagieren auf Veränderung

über

Befolgen eines Plans

Agile Prinzipien (Auszug)

Technische Hochschule mannheim

Kundenzufriedenheit durch frühe und kontinuierliche Lieferung

Änderungen willkommen heißen – auch spät im Projekt

Funktionierende Software in kurzen Abständen liefern

Tägliche Zusammenarbeit von Fach- und Entwicklungsseite

Motivierte Individuen, denen man vertraut, ins Zentrum stellen

Direkte Kommunikation (face-to-face) bevorzugen

Nachhaltiges Tempo, technische Exzellenz, Einfachheit

Selbstorganisierte Teams und regelmäßige Reflexion

Agil vs. plangetrieben

Technische Hochschule mannheim

1

Planung

Plangetrieben: vorab vollständig

Agil: rollierend & adaptiv

2

Lieferung

Plangetrieben: am Ende

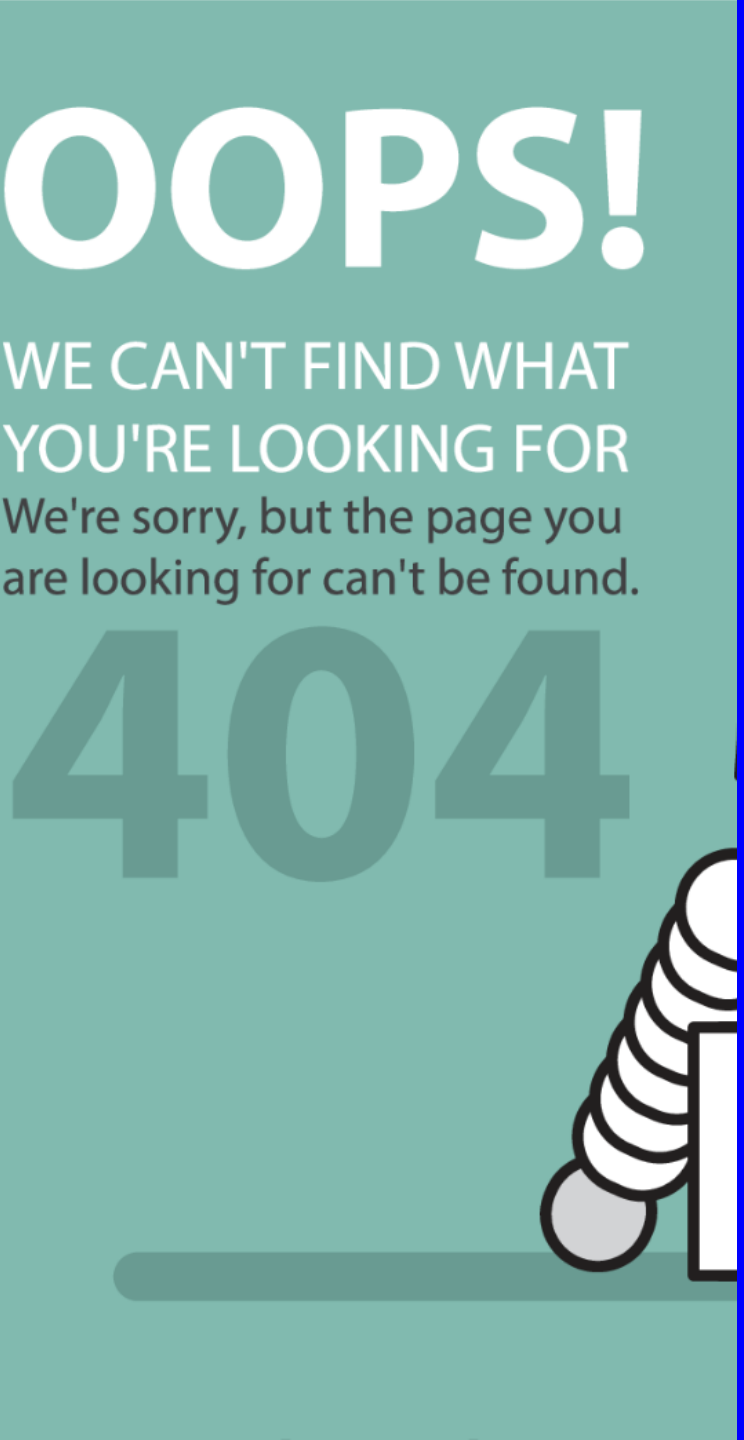
Agil: in kurzen Iterationen

3

Änderungen

Plangetrieben: über Change-Control

Agil: erwünscht & eingeplant



Sechs Anmeldungen

Healthcare.gov ging am 1. Oktober 2013 live – nach Jahren Entwicklung, aber ohne Lasttest unter realen Bedingungen. Am ersten Tag schafften sechs Personen eine vollständige Anmeldung. Ein kleines Team brachte das System danach in Wochen zum Laufen: kurze Zyklen, Monitoring, Priorisierung nach Wirkung.

Abbildung 6:**Vectorportal.com (2018):** *404 Error Page Template, Ausschnitt*, Creative Commons Attribution 4.0. https://vectorportal.com/de/vector/fehler-404-seitenvorlage/28553 [Zugriff: 20.08.2026].

Scrum im Überblick

Technische Hochschule mannheim

Was ist Scrum?

Leichtgewichtiges Rahmenwerk für komplexe Produktentwicklung

Empirische Steuerung

Transparenz – Überprüfung (Inspection) – Anpassung (Adaptation)

Sprints

Feste Zeitboxen (meist 1–4 Wochen) mit lieferbarem Increment

Drei Säulen

Rollen, Artefakte und Events bilden das Gerüst

Scrum – Rollen / Verantwortlichkeiten

Technische Hochschule mannheim

1

Product Owner

Verantwortet Wertmaximierung

Pflegt das Product Backlog

Verantwortet Produktziel

Priorisiert

2

Scrum Master

Dient dem Team

Beseitigt Hindernisse

Coacht zu Scrum

Schützt vor Störungen

3

Developers

Setzen das Increment um

Selbstorganisiert

Cross-funktional

Verantworten Qualität

Scrum – Artefakte
(mit Commitments, Scrum Guide 2020)

Technische Hochschule mannheim

1

Product Backlog

Geordnete Liste aller gewünschten Funktionen

Vom PO gepflegt

Commitment: Product Goal

2

Sprint Backlog

Auswahl für den Sprint

+ Umsetzungsplan

Commitment: Sprint Goal

3

Increment

Lieferbares Ergebnis

Potenziell auslieferbar

Commitment: Definition of Done

Scrum – Events

Technische Hochschule mannheim

Sprint

Container für alle anderen Events; fester Rhythmus

Sprint Planning

Was und wie wird im Sprint umgesetzt? – Sprint-Ziel

Daily Scrum

Tägliche Synchronisation (15 Min.) der Developers

Sprint Review

Increment wird Stakeholdern gezeigt, Feedback eingeholt

Sprint Retrospective

Team verbessert seine Arbeitsweise

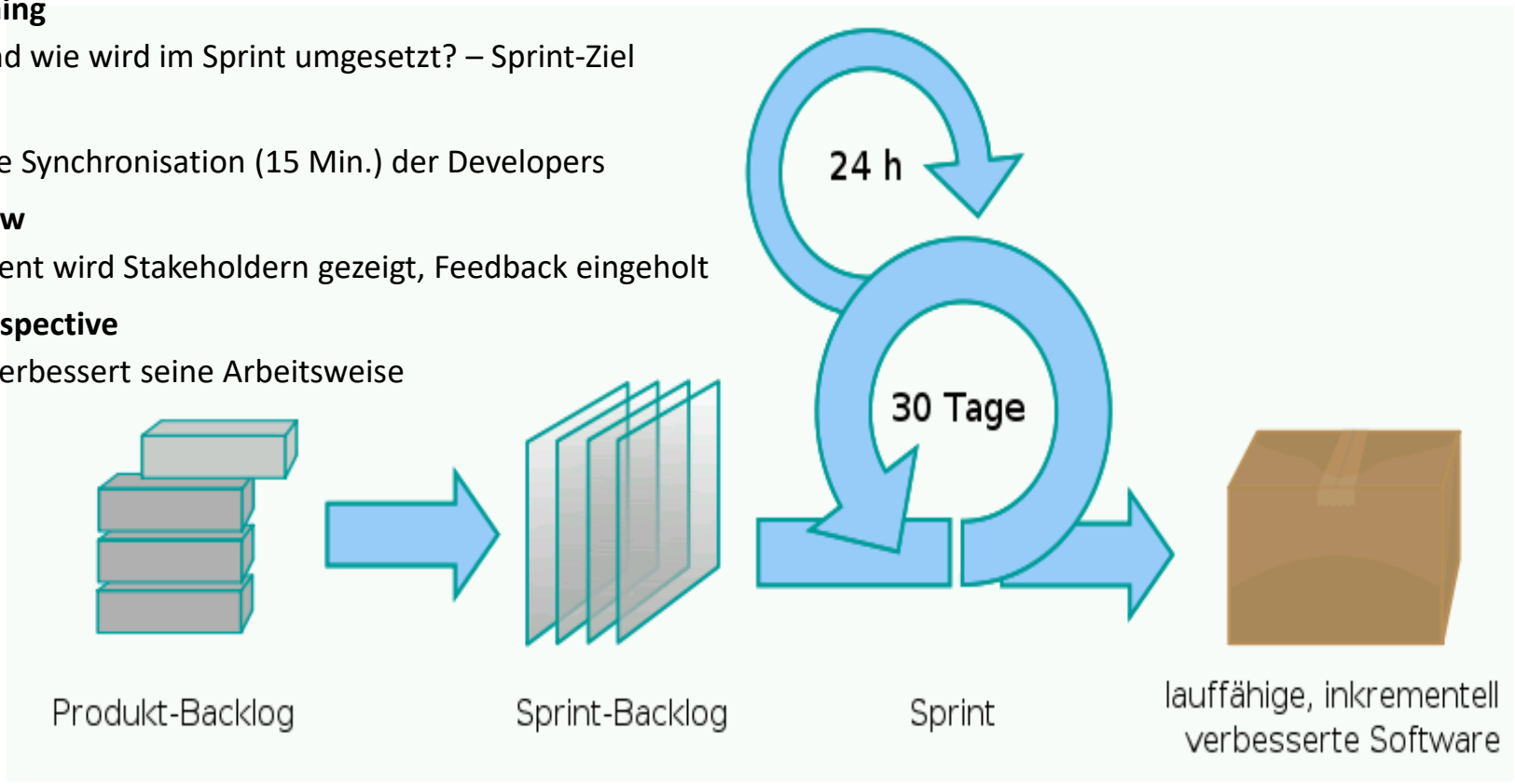


Abbildung 7: Scrum-Prozess. Quelle: Lakeworks (2009), Wikimedia Commons, CC BY-SA 4.0, eigene Bearbeitung.

User Stories und INVEST

Technische Hochschule mannheim

Format

„Als <Rolle> möchte ich <Ziel>, um <Nutzen> zu erreichen.“

INVEST-Kriterien guter Stories

Independent – unabhängig

Negotiable – verhandelbar

Valuable – wertstiftend

Estimable – schätzbar

Small – klein genug

Testable – testbar (Akzeptanzkriterien)

Aufwandsschätzung im agilen Umfeld

Technische Hochschule mannheim

Story Points

Relative Schätzung der Komplexität statt absoluter Zeitangaben

Planning Poker

Team schätzt verdeckt, diskutiert Abweichungen, schätzt erneut

Velocity

Erfahrungswert: erledigte Story Points pro Sprint – zur Prognose

Vorteil

Schwarmintelligenz, weniger Scheingenauigkeit als Stundenschätzung

Kanban

Technische Hochschule mannheim

Ursprung

Aus dem Lean-/Toyota-Produktionssystem übernommen

Kernpraktiken

Arbeit visualisieren (Kanban-Board)

Work in Progress (WIP) begrenzen

Fluss messen und steuern (Lead/Cycle Time)

Kontinuierliche, evolutionäre Verbesserung

Charakter

Fluss-orientiert, ohne feste Iterationen – Pull-Prinzip

Scrum vs. Kanban

Technische Hochschule mannheim

1

Rhythmus

Scrum: feste Sprints

Kanban: kontinuierl. Fluss

2

Rollen

Scrum: PO, SM, Dev

Kanban: keine vorgeschrieben

3

Steuerung

Scrum: Sprint Goal

Kanban: WIP-Limits

Beide: Pull & Transparenz

Lean Software Development

Technische Hochschule mannheim

Herkunft

Übertragung der Lean-Prinzipien (Toyota) auf Softwareentwicklung

Prinzipien (Poppendieck)

Verschwendung eliminieren – Lernen verstärken

Entscheidungen so spät wie möglich – so schnell wie möglich liefern

Team befähigen – Integrität einbauen – das Ganze optimieren

Verschwendung

Unnötige Features, Wartezeiten, Übergaben, Nacharbeit

DevOps

Technische Hochschule mannheim

Idee

Entwicklung (Dev) und Betrieb (Ops) als gemeinsame Verantwortung

Ziel

Schnellere, zuverlässigere und häufigere Auslieferung

CALMS

Culture – Automation – Lean – Measurement – Sharing

Praktiken

Infrastructure as Code, Monitoring, automatisierte Deployments

Continuous Integration & Continuous Delivery

Technische Hochschule mannheim

Continuous Integration (CI)

Häufiges Zusammenführen von Code; jeder Commit wird automatisch gebaut und getestet

Continuous Delivery/Deployment (CD)

Jederzeit auslieferbarer Stand; Auslieferung automatisiert (auf Knopfdruck oder automatisch)

Typische Pipeline

Build → Unit-Tests → Integrationstests → Paketierung → Deployment → Monitoring

Nutzen

Schnelles Feedback, weniger Integrationsschmerz, kleine Releases

Skalierung agiler Methoden

Technische Hochschule mannheim

Herausforderung

Mehrere Teams an einem Produkt koordinieren

Rahmenwerke

SAFe – Scaled Agile Framework (umfassend, unternehmensweit)

LeSS – Large-Scale Scrum (minimalistisch, nah am Scrum-Kern)

Nexus, Scrum@Scale als weitere Ansätze

Warnung

Skalierung erhöht Komplexität – nur so viel Prozess wie nötig

5 Anforderungen

Requirements Engineering – das richtige System bauen

Was sind Anforderungen?

Technische Hochschule mannheim

Definition (IREB/IEEE)

Eine Bedingung oder Fähigkeit, die ein System erfüllen muss, um einen Bedarf zu decken oder einen Vertrag/Standard zu erfüllen

Requirements Engineering

Systematisches Ermitteln, Dokumentieren, Prüfen und Verwalten von Anforderungen

Zentrale Frage

Bauen wir das System richtig – und bauen wir das richtige System?

Warum ist RE so wichtig?

Technische Hochschule mannheim

Größte Fehlerquelle

Viele und besonders teure Fehler entstehen aus mangelhaften Anforderungen

Hebelwirkung

Frühe Klärung verhindert teure Nacharbeit (Rule of Ten)

Kommunikation

Anforderungen sind die Brücke zwischen Fach- und Entwicklungswelt

Grundlage für alles Weitere

Design, Test, Abnahme und Vertrag bauen auf den Anforderungen auf

Technische Hochschule mannheim

Pfund gegen Newton

1999 verglühte der Mars Climate Orbiter in der Marsatmosphäre. Ein Team lieferte Schubwerte in Pound-force, das andere erwartete Newton. Beide Programme rechneten korrekt – nur stand in keiner Anforderung, welche Einheit gilt. Verloren: eine Mission für rund 328 Millionen Dollar.

Abbildung 8: Mars Climate Orbiter während Tests. Quelle: NASA (1998), Wikimedia Commons, Public Domain.

Arten von Anforderungen

Technische Hochschule mannheim

1	2	3
Funktional	**Nicht-funktional**	**Randbedingungen**
Was das System tut	Wie gut es etwas tut	Vorgaben & Grenzen
Funktionen & Verhalten	Performanz, Sicherheit	Technologie, Recht
Ein-/Ausgaben	Usability, Verfügbarkeit	Budget, Termine
Geschäftsregeln	Qualitätsanforderungen	Standards

Nicht-funktionale Anforderungen

Technische Hochschule mannheim

Oft erfolgskritisch – und oft vernachlässigt

- Beispiele entlang ISO/IEC 25010
- Performanz – Antwortzeiten, Durchsatz, Ressourcen
- Zuverlässigkeit & Verfügbarkeit
- Sicherheit (Security) & Datenschutz
- Benutzbarkeit (Usability) & Barrierefreiheit
- Wartbarkeit & Portabilität

Wichtig

- Messbar formulieren („< 200 ms“ statt „schnell“)

Der RE-Prozess

Technische Hochschule mannheim

Ermittlung (Elicitation)

Anforderungen von Stakeholdern und aus Quellen gewinnen

Analyse & Verhandlung

Konflikte klären, priorisieren, Machbarkeit prüfen

Spezifikation / Dokumentation

Anforderungen verständlich und prüfbar festhalten

Validierung & Prüfung

Sind es die richtigen Anforderungen? Reviews, Prototypen

Verwaltung (Management)

Änderungen, Versionen und Nachverfolgbarkeit über die Zeit

Stakeholder und Ermittlungstechniken

Technische Hochschule mannheim

Stakeholder identifizieren

Alle, die Interesse am oder Einfluss auf das System haben

Befragungstechniken

Interviews, Fragebögen, Workshops

Beobachtungstechniken

Feldbeobachtung, Apprenticing (Mitarbeiten)

Unterstützende Techniken

Prototyping, Personas, Szenarien, Mock-ups

Dokumentenanalyse

Vorhandene Systeme, Normen und Altdokumente auswerten

Lasten- vs. Pflichtenheft

Technische Hochschule mannheim

1

Lastenheft

Vom Auftraggeber

Das „Was“ und „Wofür“

Anforderungen aus Kundensicht

Grundlage der Ausschreibung

2

Pflichtenheft

Vom Auftragnehmer

Das „Wie“

Umsetzung der Lasten

Basis des Vertrags

3

Modern

Oft Product Backlog

User Stories statt Heft

Lebendige Doku

Werkzeuggestützt

Anforderungen sprachlich formulieren

Technische Hochschule mannheim

Problem

Natürliche Sprache ist mehrdeutig, unvollständig, interpretierbar

Satzschablonen (z. B. MASTeR/Rupp)

„Das System MUSS dem <Nutzer> die Möglichkeit bieten, <Funktion> …“

Verbindlichkeit (RFC 2119)

MUSS / SOLL / KANN klar unterscheiden

Vermeiden

Schwammige Wörter wie „benutzerfreundlich“, „schnell“, „etc.“

Use Cases (Anwendungsfälle)

Technische Hochschule mannheim

Zweck

Beschreiben das Zusammenspiel von Akteuren und System aus Nutzersicht

Bestandteile

Akteure, Vorbedingungen, Hauptszenario, Alternativ-/Fehlerszenarien, Nachbedingungen

Use-Case-Diagramm (UML)

Überblick über Akteure und ihre Anwendungsfälle

Stärke

Fokus auf Ziele und Abläufe, gut für Diskussion mit Stakeholdern

User Stories vs. klassische Anforderungen

Technische Hochschule mannheim

User Story

Kurz, gesprächsorientiert, mit Akzeptanzkriterien; agil und iterativ

Klassische Spezifikation

Detailliert, vollständig vorab; plangetrieben und vertragsnah

Gemeinsamkeit

Beide drücken einen Bedarf mit Nutzen aus

Wahl

Hängt von Vorgehensmodell, Domäne und Regulatorik ab

Qualitätskriterien guter Anforderungen

Technische Hochschule mannheim

Eindeutig – nur eine Interpretation möglich

Vollständig – kein wesentlicher Aspekt fehlt

Widerspruchsfrei – keine Konflikte untereinander

Verständlich – für alle Beteiligten klar

Prüfbar / testbar – Erfüllung ist nachweisbar

Notwendig – trägt zum Ziel bei

Verfolgbar – eindeutig identifizierbar (ID, Traceability)

Realisierbar – technisch und wirtschaftlich machbar

Priorisierung von Anforderungen

Technische Hochschule mannheim

1	2	3
MoSCoW	**Kano-Modell**	**Weitere**
Must have	Basismerkmale	Wert vs. Aufwand
Should have	Leistungsmerkmale	Risiko-basiert
Could have	Begeisterungsmerkmale	Stakeholder-Voting
Won’t have (now)		

Anforderungsmanagement & Traceability

Technische Hochschule mannheim

Anforderungen ändern sich

Geordneter Änderungsprozess statt unkontrollierter „Scope Creep“

Nachvollziehbarkeit (Traceability)

Verknüpfung: Anforderung → Design → Code → Testfall

Versionierung & Baselines

Definierte, abgestimmte Stände als Bezugspunkt

Werkzeuge

Von Issue-Trackern bis zu spezialisierten RM-Tools

Technische Hochschule
mannheim

6 Softwaredesign

Vom „Was“ zum „Wie“ – die Lösung strukturieren

Was ist Softwaredesign?

Technische Hochschule mannheim

Definition

Festlegung der internen Struktur eines Systems: Komponenten, ihre Verantwortlichkeiten und Zusammenspiel

Zwei Ebenen

Architektur (Grobentwurf) – Gesamtstruktur, große Bausteine

Feinentwurf (Detailentwurf) – Klassen, Methoden, Datenstrukturen

Ziel

Verständliche, änderbare und wiederverwendbare Lösung

```
module.exports = leftpad;

function leftpad (str, len, ch) {
  str = String(str);

  var i = -1;

  ch || (ch = ' ');
  len = len - str.length;


  while (++i < len) {
    str = ch + str;
  }

  return str;
}
```

Elf Zeilen Code

Im März 2016 zog ein Entwickler sein npm-Paket left-pad zurück – elf Zeilen, die Text linksseitig auffüllen. Weltweit brachen Builds ab, weil große Projekte es indirekt einbanden. Niemand hatte diese Abhängigkeit bewusst gewählt; sie steckte tief in der Kette.

Abbildung 9: [Motiv und Quelle ergänzen]

Design-Qualität: Kopplung & Kohäsion

Technische Hochschule mannheim

Kohäsion (cohesion) – hoch ist gut

Wie stark gehören die Bestandteile eines Moduls zusammen?

Ein Modul = eine klar umrissene Aufgabe

Kopplung (coupling) – niedrig ist gut

Wie stark hängen Module voneinander ab?

Lose Kopplung → leichter änderbar und testbar

Leitsatz

High Cohesion, Low Coupling

Modularität & Geheimnisprinzip

Technische Hochschule mannheim

Modularisierung

Zerlegung in überschaubare, austauschbare Einheiten mit klaren Schnittstellen

Geheimnisprinzip (Information Hiding, Parnas)

Entwurfsentscheidungen, die sich ändern könnten, hinter Schnittstellen verbergen

Schnittstelle vs. Implementierung

Stabile Schnittstelle nach außen, frei änderbare Implementierung innen

Nutzen

Änderungen bleiben lokal, parallele Entwicklung wird möglich

SOLID-Prinzipien

Technische Hochschule mannheim

1

S – SRP

Single Responsibility

Eine Klasse,

ein Änderungsgrund

2

O – OCP

Open/Closed

Offen für Erweiterung,

geschlossen für Änderung

3

L – LSP

Liskov Substitution

Subtypen müssen

Basistypen ersetzen können

4

I – ISP

Interface Segregation

Viele schlanke

statt einer fetten Schnittstelle

5

D – DIP

Dependency Inversion

Von Abstraktionen abhängen,
nicht von Details

Nutzen

Wartbar & testbar

Erweiterbar

Geringe Kopplung

Weitere Entwurfsprinzipien

1

DRY / KISS

Don’t Repeat Yourself

Keep It Simple

Redundanz vermeiden

Einfachheit zuerst

2

YAGNI

You Aren’t Gonna

Need It

Nichts auf Vorrat bauen

3

SoC

Separation of Concerns

Belange trennen

Klare Zuständigkeiten

Entwurfsmuster (Design Patterns)

Technische Hochschule mannheim

Idee (Gang of Four)

Bewährte, wiederverwendbare Lösungsschablonen für wiederkehrende Entwurfsprobleme

Drei Kategorien

Erzeugungsmuster – Objekterzeugung (z. B. Factory, Singleton, Builder)

Strukturmuster – Komposition von Klassen/Objekten (z. B. Adapter, Decorator, Facade)

Verhaltensmuster – Interaktion & Verantwortung (z. B. Observer, Strategy, Command)

Nutzen

Gemeinsame Sprache, erprobte Lösungen, bessere Struktur

Beispiele klassischer Muster

Technische Hochschule mannheim

1

Strategy

Algorithmus austauschbar

Verhalten kapseln

Statt if/else-Ketten

2

Observer

1:n-Benachrichtigung

Lose Kopplung

Event-/UI-Systeme

3

Factory

Erzeugung kapseln

Gegen Interface programmieren

UML im Entwurf

Technische Hochschule mannheim

Unified Modeling Language

Standardisierte grafische Notation zur Modellierung von Software

Strukturdiagramme

Klassendiagramm, Komponentendiagramm, Verteilungsdiagramm

Verhaltensdiagramme

Sequenzdiagramm, Aktivitätsdiagramm, Zustandsdiagramm, Use-Case-Diagramm

Pragmatik

So viel Modell wie nötig – UML als Kommunikationsmittel, nicht Selbstzweck

Anti-Patterns & Code Smells

Technische Hochschule mannheim

Anti-Patterns

- Verbreitete, aber schädliche Lösungsmuster
- Big Ball of Mud – keine erkennbare Struktur
- God Class – eine Klasse macht alles
- Spaghetti Code – unübersichtlicher Kontrollfluss

Code Smells

- Warnzeichen: lange Methoden, Duplikate, zu viele Parameter, enge Kopplung

Gegenmittel

- Refactoring, Reviews, Prinzipientreue

7 Softwarearchitektur

Die grundlegende Struktur eines Systems

Was ist Softwarearchitektur?

Definition (ISO/IEC/IEEE 42010)

Grundlegende Konzepte und Eigenschaften eines Systems – verkörpert in seinen Elementen, Beziehungen und Entwurfsprinzipien

Bauwerk-Analogie

Die „tragenden Wände“ – schwer und teuer im Nachhinein zu ändern

Architektur vs. Entwurf

Architektur: weitreichende, strukturbestimmende Entscheidungen

Entwurf: lokale Detailentscheidungen innerhalb der Architektur

Warum Architektur wichtig ist

Technische Hochschule mannheim

Qualität entsteht in der Architektur

Performanz, Skalierbarkeit, Sicherheit und Wartbarkeit werden hier festgelegt

Kommunikation

Gemeinsames Verständnis für Team, Management und Stakeholder

Frühe Entscheidungen, große Wirkung

Architekturfehler sind später nur mit hohem Aufwand korrigierbar

Conway's Law

Systemstruktur spiegelt die Kommunikationsstruktur der Organisation

Architektursichten – das 4+1-Modell

Technische Hochschule mannheim

Idee (Kruchten)

Verschiedene Stakeholder brauchen verschiedene Sichten auf das System

Die Sichten

Logische Sicht – Funktionalität, zentrale Abstraktionen

Prozesssicht – Nebenläufigkeit, Performanz

Entwicklungssicht – Module, Organisation des Codes

Physische Sicht – Verteilung auf Hardware

+1: Szenarien (Use Cases) verbinden die Sichten

Qualitätsmerkmale & Architekturbewertung

Technische Hochschule mannheim

Architektur dient den Qualitätszielen

Sie ist immer ein Kompromiss zwischen konkurrierenden Zielen

Beispiele für Trade-offs

Performanz vs. Wartbarkeit, Sicherheit vs. Benutzbarkeit

Bewertungsmethode ATAM

Architecture Tradeoff Analysis Method – Szenarien-basiert

Szenarien

Qualitätsanforderungen konkret und prüfbar machen

Architekturstile im Überblick

Technische Hochschule mannheim

Schichtenarchitektur (Layered)

Client-Server / N-Tier

Model-View-Controller (MVC) und Varianten

Pipes & Filter

Microservices und Service-orientierte Architektur (SOA)

Ereignisgetriebene Architektur (Event-Driven)

Hexagonal / Ports & Adapters, Clean Architecture

Schichtenarchitektur

Technische Hochschule mannheim

Prinzip

Aufteilung in horizontale Schichten mit klar definierten Aufgaben

Typische Schichten

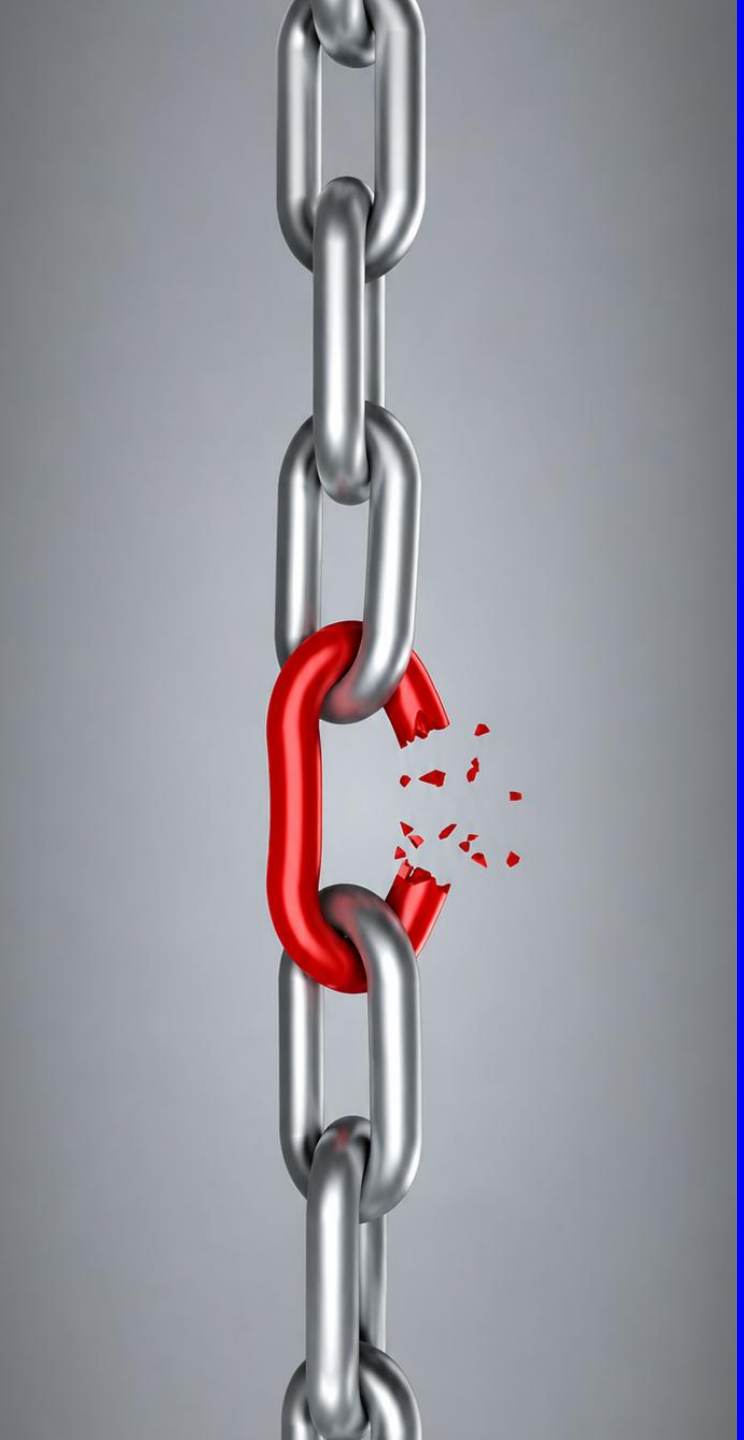
Präsentation → Anwendungslogik → Domäne → Datenzugriff

Regel

Schichten nutzen nur tiefer liegende Schichten (Abhängigkeitsrichtung)

Bewertung

Einfach und verbreitet, aber kann zu starren „Durchstichen“ führen



Technische Hochschule
mannheim

Drei Tage Stillstand

Im August 2008 beschädigte ein Datenbankfehler bei Netflix die zentrale Datenhaltung. Drei Tage lang konnte kein DVD-Versand stattfinden. Ein Fehler legte alles lahm, weil alles am selben Kern hing. Die Antwort war der Umbau zu Diensten, die einzeln ausfallen dürfen.

Abbildung 10: KI-generierte Darstellung.

Monolith vs. Microservices

Technische Hochschule mannheim

1

Monolith

Eine Deployment-Einheit

Einfacher Start

Schwerer zu skalieren

Enge Kopplung droht

2

Microservices

Unabhängige Dienste

Eigene Skalierung

Hohe Betriebskomplexität

Verteilte Systeme

3

Wahl

Team- & Systemgröße

Oft: Modulith zuerst

Komplexität abwägen

Kein Selbstzweck

Weitere wichtige Architekturansätze

Technische Hochschule mannheim

Ereignisgetrieben (Event-Driven)

Komponenten kommunizieren lose über Ereignisse/Nachrichten

Hexagonal / Ports & Adapters

Fachlogik im Kern, Technik (DB, UI) über austauschbare Adapter angebunden

Clean Architecture

Abhängigkeiten zeigen nach innen zur Domäne; Technik bleibt austauschbar

Gemeinsames Ziel

Fachlogik von technischen Details entkoppeln

Architektur dokumentieren & entscheiden

Technische Hochschule mannheim

arc42

Bewährte, gegliederte Vorlage für Architekturdokumentation

Architecture Decision Records (ADR)

Kurze Dokumente: Kontext, Entscheidung, Konsequenzen – nachvollziehbar festgehalten

C4-Modell

Context, Container, Component, Code – Architektur in vier Zoomstufen

Grundsatz

Dokumentieren, was beim Verstehen und Entscheiden hilft – nicht mehr

8 Testen von Software

Qualität nachweisen und Fehler frühzeitig finden

Grundbegriffe des Testens

Technische Hochschule mannheim

Fehlhandlung (Error)

Menschliche Handlung, die zu einem Defekt führt

Defekt / Fehlerzustand (Defect/Bug)

Fehlerhafte Stelle im Code oder Dokument

Fehlerwirkung (Failure)

Nach außen sichtbares Fehlverhalten zur Laufzeit

Testen

Geplante Ausführung mit dem Ziel, Fehlerwirkungen aufzudecken

Verifikation vs. Validierung

Technische Hochschule mannheim

Verifikation

Bauen wir das System richtig? – Übereinstimmung mit der Spezifikation

Validierung

Bauen wir das richtige System? – Erfüllung der tatsächlichen Bedürfnisse

Dijkstra

„Testen kann die Anwesenheit von Fehlern zeigen, nie deren Abwesenheit.“

Konsequenz

Testen schafft Vertrauen, aber keine Beweise für Korrektheit

Teststufen (V-Modell)

Technische Hochschule mannheim

Komponententest (Unit-Test)

Einzelne Bausteine isoliert prüfen

Integrationstest

Zusammenspiel der Komponenten prüfen

Systemtest

Das Gesamtsystem gegen die Anforderungen prüfen

Abnahmetest (Acceptance)

Validierung durch den Kunden in realitätsnaher Umgebung

Testarten

Technische Hochschule mannheim

Funktionale Tests

Prüfen, ob das System die geforderten Funktionen erfüllt

Nicht-funktionale Tests

Last-/Performanztest, Sicherheitstest, Usability-Test

Regressionstest

Nach Änderungen sicherstellen, dass nichts „kaputtgeht“

Smoke-/Sanity-Test

Schneller Grundfunktionstest nach einem Build

Black-Box-Testverfahren

Technische Hochschule mannheim

1

Äquivalenzklassen

Eingaben in Klassen gleichen Verhaltens

Ein Vertreter je Klasse

Reduziert Testfälle

2

Grenzwertanalyse

Fehler an Grenzen

Werte an/um Grenze

min, min±1, max …

Sehr wirksam

3

Weitere

Entscheidungstabellen

Zustandsbasiert

Use-Case-basiert

White-Box-Testverfahren

Idee

Testfälle aus der inneren Struktur (Kontrollfluss) ableiten

Überdeckungsmaße (Coverage)

Anweisungsüberdeckung (C0) – jede Anweisung mindestens einmal

Zweigüberdeckung (C1) – jeder Entscheidungszweig

Pfad-/Bedingungsüberdeckung – anspruchsvoller

Hinweis

Hohe Überdeckung ≠ Fehlerfreiheit – Qualität der Testfälle zählt

Testpyramide & Automatisierung

Technische Hochschule mannheim

Testpyramide (Cohn)

Viele schnelle Unit-Tests, weniger Integrationstests, wenige UI-/E2E-Tests

Warum so?

Untere Ebenen sind schneller, stabiler und billiger

Anti-Pattern „Ice Cream Cone“

Zu viele langsame, brüchige UI-Tests, zu wenige Unit-Tests

Automatisierung

Voraussetzung für CI/CD und schnelle Rückmeldung

Test-Driven Development (TDD)

Technische Hochschule mannheim

Zyklus: Red – Green – Refactor

Red: einen fehlschlagenden Test schreiben

Green: minimalen Code schreiben, der den Test erfüllt

Refactor: Code verbessern, Tests bleiben grün

Effekt

Testbares Design, hohe Überdeckung, lebende Spezifikation

Verwandt

BDD/ATDD – verhaltens- bzw. abnahmegetriebene Entwicklung

Unit-Test-Frameworks & Mocking

Technische Hochschule mannheim

xUnit-Familie

JUnit (Java), pytest/unittest (Python), NUnit (.NET), Jest (JS) …

Aufbau eines Tests: AAA

Arrange – Act – Assert

Test-Doubles

Dummy, Stub, Mock, Fake, Spy – Abhängigkeiten ersetzen

Nutzen von Mocks

Isolation der Unit, deterministische und schnelle Tests

Statische Qualitätssicherung

Reviews

Inspektion, Walkthrough, Pair Programming – Fehler ohne Ausführung finden

Statische Analyse

Linter und Tools (z. B. SonarQube) prüfen Code automatisch

Metriken

Komplexität, Duplikate, Coverage als Indikatoren

Clean Code

Lesbarkeit und Wartbarkeit als bewusstes Qualitätsziel

Technische Hochschule
mannheim

45 Minuten

2012 verteilte Knight Capital neue Handelssoftware auf acht Server – auf einem blieb alter Code liegen. Der handelte 45 Minuten lang unkontrolliert. Verlust: 440 Mio. Dollar. Die Firma existierte danach nicht mehr.

Abbildung 11: Börsentafel auf den Philippinen. Quelle: Katrina Tuliao (2008), Wikimedia Commons, CC BY 2.0.

Qualitätssicherung als Gesamtbild

Technische Hochschule mannheim

Konstruktiv vs. analytisch

Fehler vermeiden (Standards, Reviews) vs. Fehler finden (Tests)

Shift Left

Qualitätsmaßnahmen so früh wie möglich im Prozess

Definition of Done

Klare Kriterien, wann eine Aufgabe wirklich fertig ist

Kontinuierlich

Qualität ist Aufgabe des ganzen Teams, nicht nur einer Testphase

9 Wartung & Softwareevolution

Software lebt – und muss gepflegt werden

Was ist Softwarewartung?

Definition

Alle Änderungen nach der Auslieferung, um Software nutzbar zu halten und zu verbessern

Bedeutung

Häufig der größte Kostenblock im gesamten Lebenszyklus

Realität

Erfolgreiche Software wird über viele Jahre weiterentwickelt

Anspruch

Wartbarkeit muss bereits beim Entwurf mitgedacht werden

Arten der Wartung

Technische Hochschule mannheim

1

Korrektiv / Adaptiv

Korrektiv: Fehler beheben

Adaptiv: an neues Umfeld anpassen

2

Perfektiv

Funktionen verbessern

Neue Anforderungen

Performanz optimieren

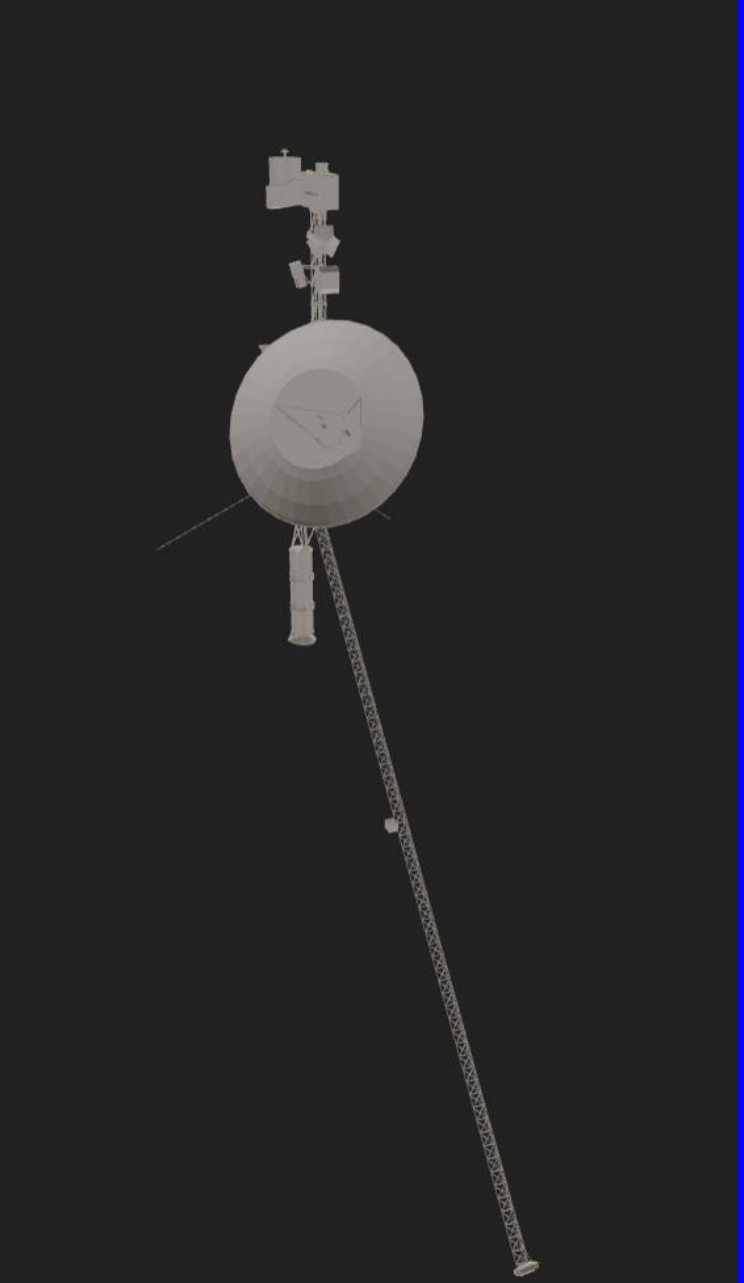
Größter Anteil

3

Präventiv

Künftigen Problemen vorbeugen

Wartbarkeit erhöhen



Technische Hochschule
mannheim

Fernwartung

Am 14. November 2023 sendete Voyager 1 unsinnige Daten. Ursache: ein einzelner Speicherchip mit 256 Wörtern, in dem auch Programmcode lag. Reparieren war unmöglich, ein freier Speicherblock gab es ebenfalls nicht. Das Team zerlegte den Code und verteilte ihn auf Speicherreste. Patch am 18. April 2024, Antwort zwei Tage später. Signallaufzeit: 22,5 Stunden pro Richtung.

Abbildung 12: Voyager-Sonde. Quelle: Fac-tory-o (2018), Wikimedia Commons, CC BY-SA 4.0.

Lehman’sche Gesetze der Softwareevolution

Technische Hochschule mannheim

Kontinuierlicher Wandel

Genutzte Software muss sich ständig anpassen, sonst wird sie unbrauchbar

Zunehmende Komplexität

Ohne Gegenmaßnahmen steigt die Komplexität mit jeder Änderung

Abnehmende Qualität

Software verschlechtert sich, wenn sie nicht aktiv gepflegt wird

Konsequenz

Bewusste Investition in Struktur und Refactoring nötig

Legacy-Systeme & technische Schulden

Technische Hochschule mannheim

Legacy-System

Alt, geschäftskritisch, schwer wartbar – oft schlecht dokumentiert

Technische Schuld (Technical Debt)

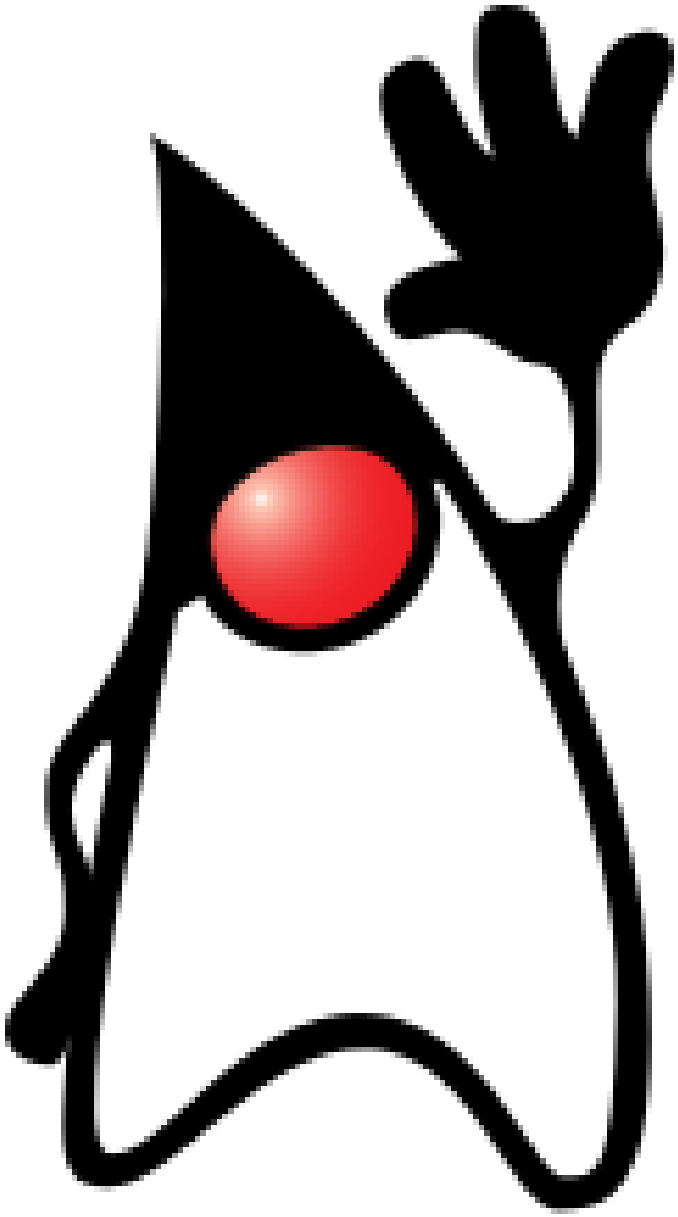
Schnelle Lösungen heute verursachen „Zinsen“ in Form späterer Mehrarbeit

Umgang

Schulden sichtbar machen, priorisieren und gezielt abbauen

Gefahr

Unbeherrschte Schulden bremsen jede Weiterentwicklung



Technische Hochschule
mannheim

Eine Zeile, überall

Im Dezember 2021 wurde in der Java-Bibliothek Log4j eine Lücke bekannt: Eine präparierte Zeichenkette in einer Logmeldung genügte, um fremden Code auszuführen. Betroffen waren unzählige Systeme weltweit. Gepflegt wurde die Bibliothek von einer Handvoll Freiwilliger – unbezahlt.

Abbildung 13: Duke, das Java-Maskottchen. Quelle: Sun Microsystems Inc. / sbmehta (2007), Wikimedia Commons, BSD-3-Clause.

Refactoring & Reengineering

Technische Hochschule mannheim

Refactoring (Fowler)

Interne Struktur verbessern, ohne das äußere Verhalten zu ändern

Voraussetzung

Automatisierte Tests als Sicherheitsnetz

Reverse Engineering

Aus dem Code Modelle/Verständnis zurückgewinnen

Reengineering

System analysieren und in verbesserter Form neu aufbauen

Wartbarkeit aktiv gestalten

Technische Hochschule mannheim

Lesbarer Code

Klare Namen, kleine Einheiten, konsistenter Stil

Tests & Dokumentation

Sicherheitsnetz und Wissensbewahrung

Modularität

Lose Kopplung hält Änderungen lokal

Kontinuierliche Pflege

Aktualisierungen, Boy-Scout-Rule: Code besser hinterlassen als vorgefunden

Technische Hochschule
mannheim

10 Versionsverwaltung mit Git

Änderungen beherrschen und im Team zusammenarbeiten

Warum Versionsverwaltung?

Technische Hochschule mannheim

Probleme ohne VCS

- Datei-Chaos: „final_v2_wirklich_final.zip“
- Kein nachvollziehbarer Verlauf, kein Wer-Was-Wann
- Parallele Arbeit führt zu überschriebenen Änderungen

Ein Versionskontrollsystem bietet

- Vollständige Historie und Rückkehr zu früheren Ständen
- Parallele Entwicklung über Branches
- Nachvollziehbarkeit und Zusammenarbeit im Team

Fünf Backups, keins nutzbar

Am 31. Januar 2017 löschte ein Administrator bei GitLab versehentlich ein Produktionsverzeichnis. Danach zeigte sich: von fünf vorgesehenen Backup- und Replikationsverfahren funktionierte keines wie gedacht. Wiederhergestellt wurde aus einer Momentaufnahme, die Stunden alt war.

Abbildung 14: Backup. Quelle: John (2005), „Backup Backup Backup – And Test Restores“, Wikimedia Commons, CC BY 2.0, eigener Ausschnitt.

Zentral vs. verteilt

Technische Hochschule mannheim

1

Zentral (z. B. SVN)

Ein zentrales Repo

Online nötig

Single Point of Failure

2

Verteilt (Git)

Jeder hat volles Repo

Offline arbeiten

Schnell & robust

De-facto-Standard

3

Plattformen

GitHub, GitLab

Bitbucket

Azure DevOps

Git – Grundkonzepte

Technische Hochschule mannheim

Repository

Projekt samt vollständiger Versionshistorie (Ordner .git)

Commit

Ein Schnappschuss des Projektstands mit eindeutiger ID (Hash)

Snapshots statt Diffs

Git speichert Zustände des gesamten Baums, nicht nur Zeilenänderungen

HEAD

Zeiger auf den aktuell ausgecheckten Commit/Branch

Die drei Bereiche

Technische Hochschule mannheim

1

Working Directory

Deine Arbeitskopie

Hier wird editiert

git status zeigt Änderungen

2

Staging Area

Vormerkbereich (Index)

git add

Auswahl für Commit

3

Repository

Lokale Historie

git commit

Dauerhaft gespeichert

Grundlegende Git-Befehle

Technische Hochschule mannheim

git init – neues Repository anlegen

git clone <url> – bestehendes Repo kopieren

git status – Zustand der Arbeitskopie ansehen

git add <datei> – Änderungen vormerken (stagen)

git commit -m "Nachricht" – Schnappschuss speichern

git log – Versionshistorie anzeigen

git diff – Unterschiede ansehen

Branching & Merging

Technische Hochschule mannheim

Branch

Eigenständiger Entwicklungszweig – leichtgewichtig und schnell in Git

Wozu?

Features, Bugfixes oder Experimente isoliert entwickeln

Befehle

git branch / git switch (checkout) / git merge

Merge

Zusammenführen zweier Zweige; ggf. Fast-Forward oder Merge-Commit

Mit Remote-Repositories arbeiten

Technische Hochschule mannheim

Remote

Gemeinsames Repository auf einem Server (z. B. „origin“)

git fetch – Änderungen vom Remote holen (ohne mergen)

git pull – holen und integrieren (fetch + merge)

git push – eigene Commits hochladen

Tracking-Branches

Lokale Branches sind mit Remote-Branches verknüpft

Merge vs. Rebase

Technische Hochschule mannheim

1

Merge

Erhält die Historie

Erzeugt Merge-Commit

Nicht-linear

Sicher für geteilte Branches

2

Rebase

Setzt Commits neu auf

Lineare Historie

Saubere Optik

3

Goldene Regel

Niemals geteilte/ öffentliche History rebasen

Konflikte lösen

Technische Hochschule mannheim

Wann entstehen Konflikte?

Wenn dieselben Stellen parallel unterschiedlich geändert wurden

Vorgehen

Git markiert Konfliktstellen (<<<<, ====, >>>>)

Manuell die gewünschte Lösung herstellen

git add und Merge/Rebase fortsetzen

Vorbeugen

Häufig integrieren, kleine Commits, gute Absprachen

Git-Workflows

Technische Hochschule mannheim

1

Git Flow

main + develop

feature/release/hotfix

Strukturiert

Für Releasezyklen

2

GitHub Flow

main + Feature-Branches

Pull Requests

Einfach

Für Continuous Delivery

3

Trunk-Based

Ein Hauptzweig

Kurze Branches

Feature Flags

Stark CI-getrieben

Pull Requests & Code Review

Technische Hochschule mannheim

Pull / Merge Request

Antrag, einen Branch in den Zielzweig zu übernehmen

Code Review

Andere prüfen Änderungen vor dem Merge – Qualität und Wissenstransfer

Automatisierte Checks

CI-Pipeline (Build, Tests, Linting) als Voraussetzung

Nutzen

Weniger Fehler, gemeinsames Verständnis, bessere Codequalität

Git – Best Practices

Technische Hochschule mannheim

Aussagekräftige Commit-Messages

Knappe Betreffzeile, bei Bedarf erklärender Text (das „Warum“)

Kleine, fokussierte Commits

Eine logische Änderung pro Commit

.gitignore – generierte Dateien und Secrets nicht einchecken

Tags für Releases (z. B. v1.2.0)

Häufig pushen und pullen – Integration nicht aufschieben

Technische Hochschule
mannheim

11 Code-Erstellung mit KI-Werkzeugen

KI-Assistenten sinnvoll und verantwortungsvoll einsetzen

KI in der Softwareentwicklung

Technische Hochschule mannheim

Ein Paradigmenwechsel

KI-Assistenten sind in wenigen Jahren fester Bestandteil der Entwicklung geworden

Was sie können

Code vorschlagen, erklären, übersetzen, Tests und Doku erzeugen

Was sie (noch) nicht ersetzen

Verständnis, Entwurfsentscheidungen, Verantwortung und Urteilsvermögen

Haltung dieser Vorlesung

Werkzeug zur Produktivität – kein Ersatz für Software-Engineering-Kompetenz

Wie funktionieren KI-Code-Assistenten?

Technische Hochschule mannheim

Große Sprachmodelle (LLM)

Trainiert auf riesigen Mengen Text und Quellcode

Prinzip

Vorhersage des wahrscheinlich nächsten Tokens auf Basis des Kontexts

Wichtige Folgerung

Sie „verstehen“ nicht – sie erzeugen plausible Muster

Kontext zählt

Qualität hängt stark vom mitgegebenen Kontext (Prompt, Code) ab

Werkzeuglandschaft (Auswahl)

Technische Hochschule mannheim

Im Editor / Autovervollständigung

GitHub Copilot, Tabnine, Amazon Q / CodeWhisperer

KI-zentrierte Editoren / Agenten

Cursor, Claude Code, JetBrains AI Assistant

Chat-basierte Assistenten

ChatGPT, Claude, Gemini – für Erklärung, Entwurf, Review

Hinweis

Markt entwickelt sich rasant – Prinzipien sind wichtiger als Produkte

Typische Einsatzfelder

Technische Hochschule mannheim

1	2	3
Erstellen	**Verstehen**	**Verbessern**
Code-Vervollständigung	Code erklären	Refactoring
Boilerplate	Doku erzeugen	Übersetzen (Sprachen)
Tests generieren	Fremden Code einarbeiten	Optimierung
Datenmodelle	Fehlersuche	Review-Hinweise

Prompt Engineering für Code

Technische Hochschule mannheim

Kontext geben

Sprache, Framework, Zielsetzung und Randbedingungen nennen

Präzise sein

Erwartete Ein-/Ausgaben, Beispiele und Einschränkungen angeben

Iterativ arbeiten

Ergebnis prüfen, Feedback geben, schrittweise verfeinern

Zerlegen

Große Aufgaben in kleine, prüfbare Teilschritte aufteilen

Chancen & Produktivität

Technische Hochschule mannheim

Geschwindigkeit

Schnelleres Schreiben von Routinecode und Boilerplate

Einstieg & Lernen

Unbekannte APIs und Sprachen schneller erschließen

Fokus

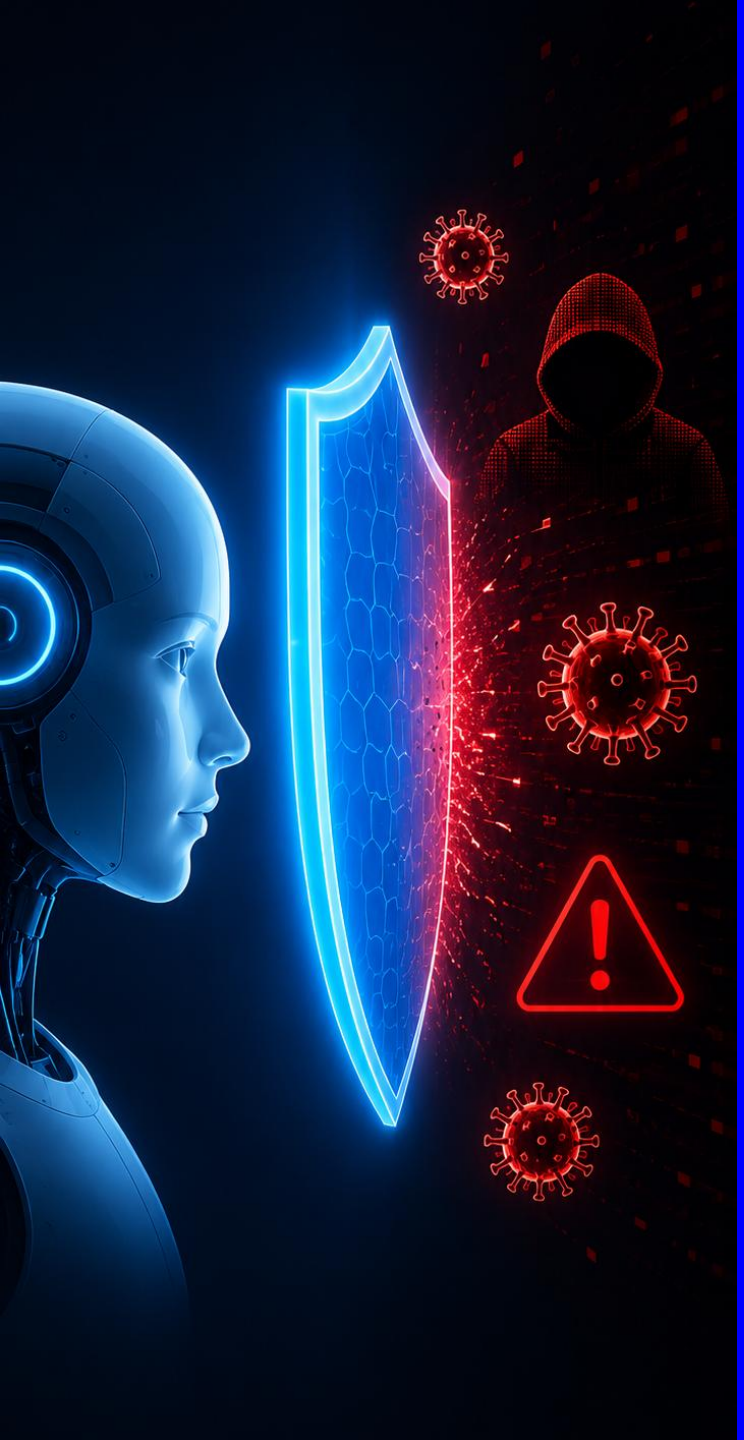
Mehr Zeit für Entwurf und schwierige Probleme

Belege

Studien zeigen messbare Beschleunigung bei klar umrissenen Aufgaben

Aber

Produktivitätsgewinn variiert stark mit Aufgabe und Erfahrung



Technische Hochschule mannheim

Das Paket, das nie existierte

```
> pip install huggingface-cli
```

Sicherheitsforscher Lanyado registrierte den Namen, den KI-Assistenten immer wieder vorschlugen – und zählte 30.000 Downloads in drei Monaten. Selbst ein Konzern hatte den erfundenen Befehl in eine eigene README kopiert. Vertrauen ist keine Verifikation.

Abbildung 15: KI-generierte Darstellung.

Risiken & Grenzen

Technische Hochschule mannheim

Halluzinationen

Plausibel aussehender, aber falscher Code oder erfundene APIs

Sicherheitslücken

Generierter Code kann verwundbar oder unsicher sein

Lizenz- & Urheberrecht

Unklare Herkunft von Trainingsdaten und Vorschlägen

Bias & Veralten

Verzerrungen und überholte Praktiken aus Trainingsdaten

Automation Bias

Gefahr, Vorschlägen unkritisch zu vertrauen

Qualität & Verantwortung

Technische Hochschule mannheim

Review-Pflicht

Generierter Code ist ein Vorschlag – er muss verstanden und geprüft werden

Tests

Automatisierte Tests bleiben unverzichtbar – gerade bei KI-Code

Verantwortung bleibt menschlich

Die Entwickler:innen haften für das Ergebnis, nicht das Werkzeug

Grundsatz

Nur einchecken, was man selbst erklären und vertreten kann

KI über den gesamten SE-Lebenszyklus

Technische Hochschule mannheim

Anforderungen

User Stories formulieren, Akzeptanzkriterien vorschlagen

Design & Architektur

Varianten skizzieren, Muster vorschlagen – Entscheidung bleibt beim Menschen

Implementierung & Test

Code, Testfälle und Mock-Daten generieren

Wartung

Legacy-Code erklären, Refactorings und Migrationen unterstützen

Datenschutz, IP & rechtliche Aspekte

Technische Hochschule mannheim

Vertraulichkeit

Keine Geheimnisse oder personenbezogenen Daten unbedacht an Dienste senden

Unternehmensrichtlinien

Freigegebene Werkzeuge, On-Premise- bzw. Enterprise-Optionen beachten

Geistiges Eigentum

Lizenzkonformität und Herkunft von Code prüfen

Regulatorik

Z. B. EU AI Act und branchenspezifische Vorgaben berücksichtigen

Auswirkungen & Ausblick

Technische Hochschule mannheim

Wandel der Rolle

Vom reinen Code-Schreiben hin zu Spezifizieren, Prüfen und Integrieren

Wichtiger denn je

SE-Grundlagen: Anforderungen, Design, Tests, Architektur, Reviews

Agentische Werkzeuge

Assistenten übernehmen zunehmend mehrstufige Aufgaben – unter Aufsicht

Fazit

KI verstärkt gute Entwickler:innen – ersetzt aber kein solides Fundament

KI-Tools – Best Practices

Technische Hochschule mannheim

1

Einsetzen

Für Routine & Einstieg

Als Sparringspartner

Kontext mitgeben

Klein iterieren

2

Prüfen

Immer reviewen

Tests schreiben

Sicherheit beachten

Nichts blind vertrauen

3

Schützen

Keine Geheimnisse teilen

Richtlinien beachten

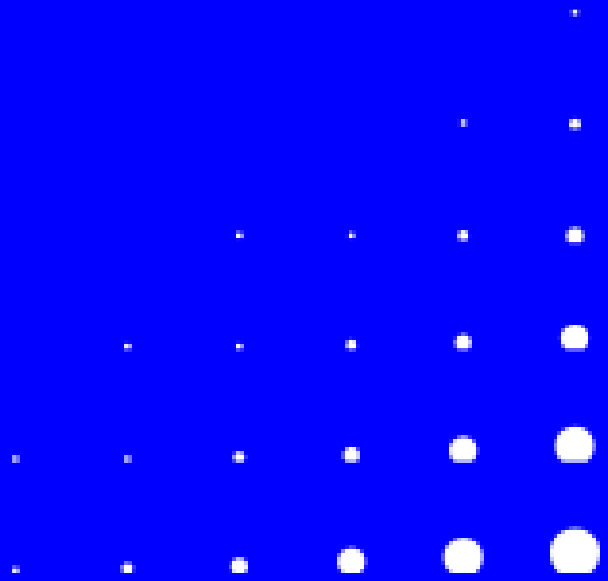
Lizenzen prüfen

Verantwortung tragen

Technische Hochschule
mannheim

Vielen Dank für Ihre Aufmerksamkeit!

Fragen, Anregungen und Diskussion willkommen



Literatur & Quellen (Auswahl)

Technische Hochschule mannheim

Balzert, H. (2009): *Lehrbuch der Softwaretechnik – Basiskonzepte und Requirements Engineering.* 3. Auflage. Heidelberg: Spektrum Akademischer Verlag.

Chacon, S.; Straub, B. (2014): *Pro Git.* 2. Auflage. Berkeley, CA: Apress. https://git-scm.com/book/de/v2 [Zugriff: 20.08.2026].

Fowler, M. (2018): *Refactoring – Improving the Design of Existing Code.* 2. Auflage. Boston: Addison-Wesley.

Gamma, E.; Helm, R.; Johnson, R.; Vlissides, J. (1994): *Design Patterns – Elements of Reusable Object-Oriented Software.* Reading, MA: Addison-Wesley.

International Requirements Engineering Board (IREB) (2026): *CPRE Foundation Level – Lehrplan.* Version 3.3.0, 01.04.2026. https://cpre.ireb.org/de/downloads-and-resources/downloads [Zugriff: 20.08.2026].

International Software Testing Qualifications Board (ISTQB) (2025): *Certified Tester Foundation Level – Lehrplan.* Deutschsprachige Ausgabe, Version 4.0.2. Herausgegeben durch Austrian Testing Board, German Testing Board e. V. und Swiss Testing Board. https://www.gtb.de/wp-content/uploads/2025/02/ISTQB_CTFL_Lehrplan-2025_V4.0.2_DE.pdf [Zugriff: 20.08.2026].

Ludewig, J.; Lichter, H. (2023): *Software Engineering – Grundlagen, Menschen, Prozesse, Techniken.* 4., überarbeitete und erweiterte Auflage. Heidelberg: dpunkt.verlag.

Martin, R. C. (2008): *Clean Code – A Handbook of Agile Software Craftsmanship.* Upper Saddle River, NJ: Prentice Hall.

Martin, R. C. (2018): *Clean Architecture – A Craftsman's Guide to Software Structure and Design.* Boston: Prentice Hall.

Schwaber, K.; Sutherland, J. (2020): *Der Scrum Guide – Der gültige Leitfaden für Scrum: Die Spielregeln.* November 2020. https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-German.pdf [Zugriff: 20.08.2026].

Sommerville, I. (2018): *Software Engineering.* 10., aktualisierte Auflage. Hallbergmoos: Pearson. https://elibrary.pearson.de/book/99.150005/9783863268350 [Zugriff: 20.08.2026].