Software Engineering
1. Einführung und Begriffe

Jonathan Brachthäuser

Agenda

- Organisatorisches
- Begriffsklärung: Software Engineering
- Aufbau der Vorlesung

Organisatorisches

Das Teamprojekt (INF2110) im Überblick

- Vorlesung (mit Klausur)
- Teamprojekt
  - Universitäres Teamprojekt
  - Industrielles Teamprojekt – Tübinger Softwareprojekt (TSP)

Universitäres Teamprojekt

- Beginn: Projektabhängig SS 17 oder WS 17/18
- Dauer: 1 Semester
- 9 ECTS
- Aufgabenstellung und Betreuung durch einen Lehrstuhl der Universität Tübingen
- Liste aller angebotenen Projekte:
  - https://atreus.informatik.uni-tuebingen.de/swprakt
- Bitte nochmals am Freitag auf Vollständigkeit prüfen! Es gab leider bereits Datenverluste in den Textfeldern.

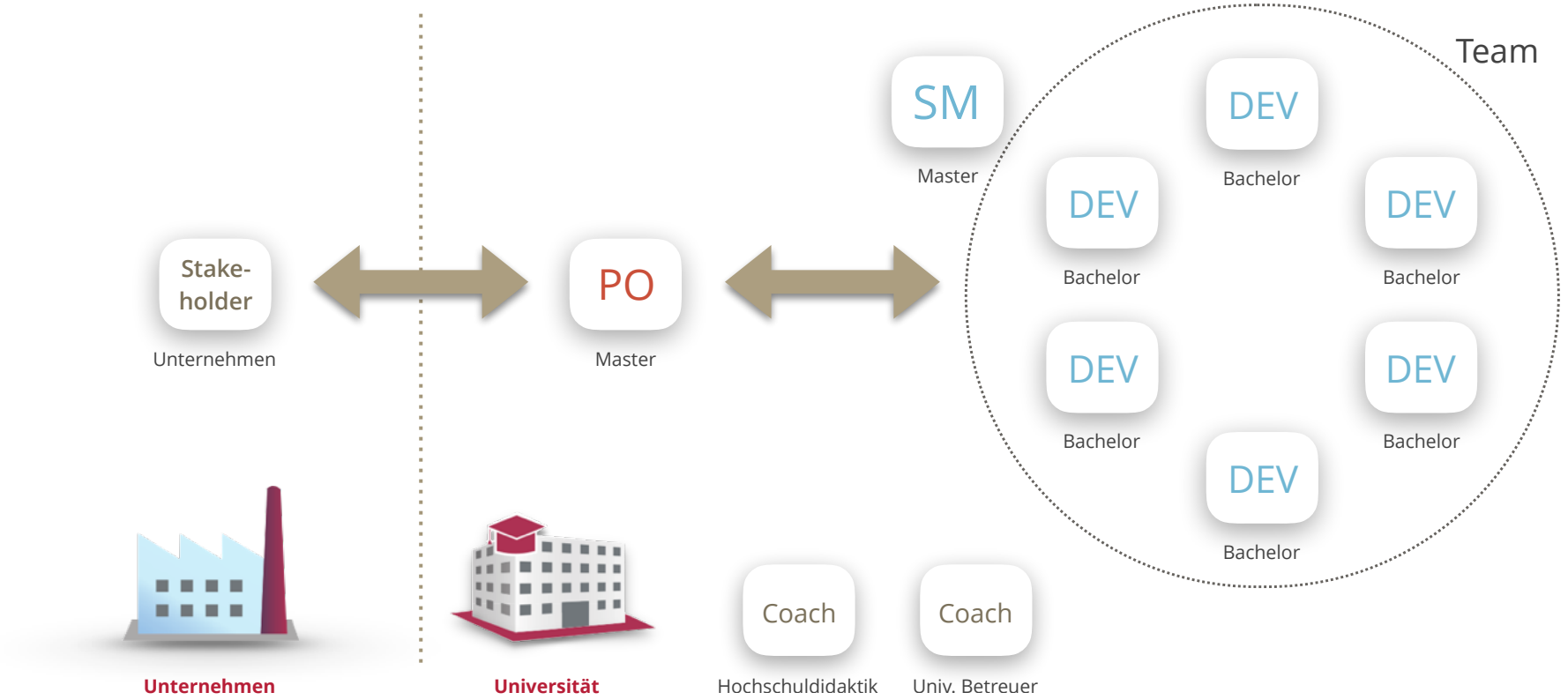
Änderungsankündigung

- Projekte von Prof. Franz
  - **"Bewegungsmessung in Matlab/GNU–Octave und C/C++"**
    erst im WS17/18
  - **"Consciousness, behavior & EEG"**
    findet wie geplant im SS17 statt.

Industrielles Teamprojekt (TSP)

- Beginn: SS 17
- Dauer: 1 Jahr
- 9 ECTS (Teamprojekt) + 3 ECTS (Vorpraktikum)
- Teamgröße: ca. 6 BSc und 2 MSc Studierende
- Aufgabenstellung durch kooperierende Unternehmen
- Betreuung:
  - Lehrstuhl von Prof. Ostermann & industrielle Kooperationspartner
- Website:
  - http://ps.informatik.uni-tuebingen.de/industryproject
- Projektvorstellungsmesse:
  - Freitag, 21.04, 13:30 Hörsaal N3, Morgenstelle

Industrielles Teamprojekt (TSP)



Ablauf

- Heute: Sie können bereits jetzt ihre Projekte auswählen
  - https://atreus.informatik.uni-tuebingen.de/swprakt
  - Es gilt **nicht** "First-come, first-served", sondern es wird automatisch nach angegebenen Prioritäten optimiert.
  - Bitte für **alle 3** Prioritäten ein Projekt auswählen!
- Freitag (21.4): Projektmesse für Industrieprojekte
  - Vorstellung durch Kooperationspartner
  - Forum mit abgeschlossenen Projekten, Kaffee und Kuchen
- Freitag (23:59 Uhr): Registrierungsschluss
  - Nochmals am auf Vollständigkeit prüfen! Es gab leider bereits Datenverluste in den Textfeldern.
- Montag (24.4): Bekanntgabe der Verteilung auf die Projekte

Organisation der Vorlesung

- Umfang: 2 SWS mit insgesamt 9 Vorlesungen
- Begleitend zum Programmierprojekt
- Veranstalter: Jonathan Brachthäuser, Prof. Ostermann
  - (Programmiersprachen und Softwaretechnik, Prof. Ostermann)
- Kein Übungsbetrieb, aber ...
  - Versuchen Sie das Erlernte auf Ihr Projekt zu beziehen!
- Termine:
  - VL: Mi 14:15 – 16:00 in Raum F 119
  - Klausur: 21.06.2017, 14:00 in Raum F 119

Organisation der Vorlesung

- Scheinkriterien:
  - Abschlussklausur
  - Muss zum Abschluss des Programmierprojekts bestanden werden und fließt als 20% Bonus in die Note ein
- Homepage der VL:
  - http://ps.informatik.uni-tuebingen.de/teaching/ss17/se
  - Kopien der Folien, Literaturhinweise, etc.

Lehrveranstaltungsstil

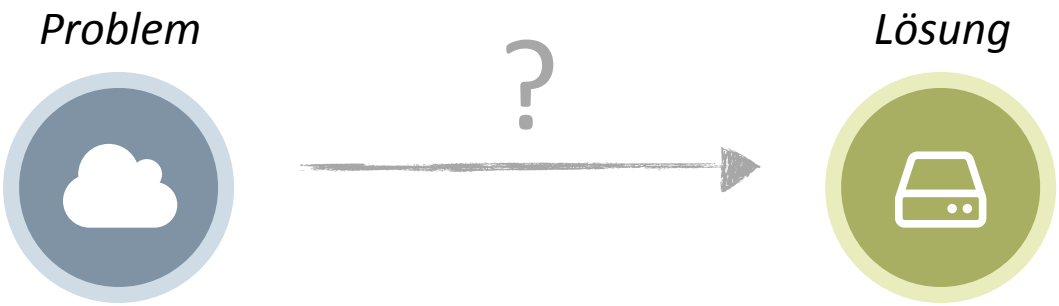
- Konzeptvermittlung durch Folien
- Folienkopien sind auf der Homepage verfügbar, Abweichungen (insb. Korrekturen) sind möglich
- Beispiele häufig an der Tafel
- Zwischenfragen und Kommentare während der Vorlesung sind grundsätzlich erwünscht.
- Einübung des Erlernten erfolgt in den jeweiligen Projekten

Gastvorlesungen

- Amra Avdic – NovaTec GmbH (3. Mai)
  - Agiles Requirements Engineering und Scrum
- Heiko Hütter – DAASI International (10. Mai)
  - Multi-Project & Multi-Product Setups using Scrum
- Mikolaj Wawrzyniak – Trivago GmbH / github (14. Juni)
  - Social coding with git and github

Begriffe und Kontext

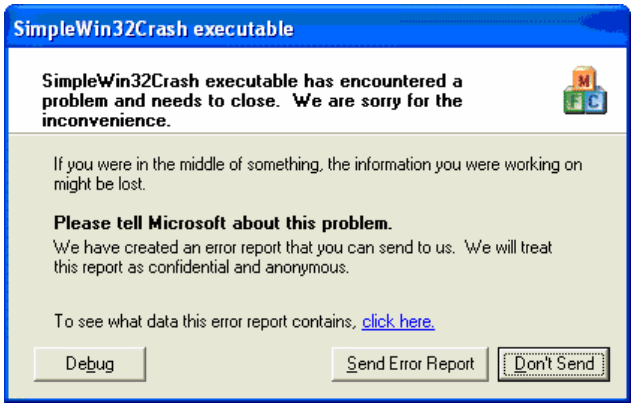
Einordnung: Vom Problem zur Lösung

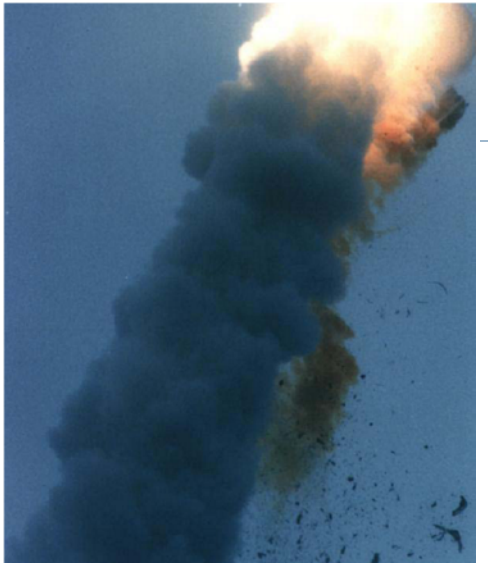


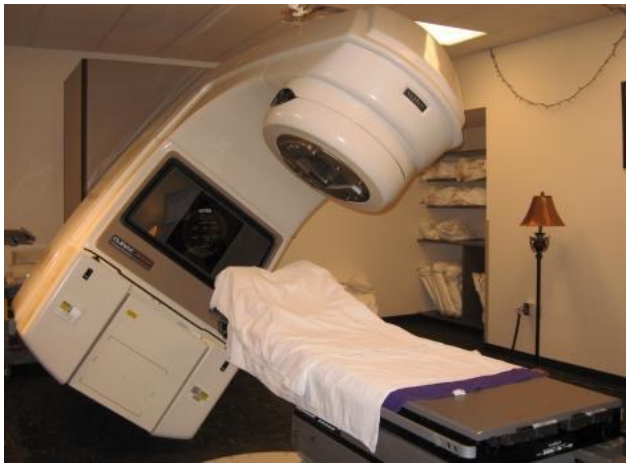
Projekte im Studium
- Klar formulierte Aufgabenstellung
- Aufgabensteller ist selbst Experte und kennt bereits die Lösung
- Durch eine Person zu bearbeiten
- Bearbeitungszeit sehr kurz (ca. 1 Semester)
- "Abgeben und vergessen" - keine weiterführende Entwicklung / Wartung

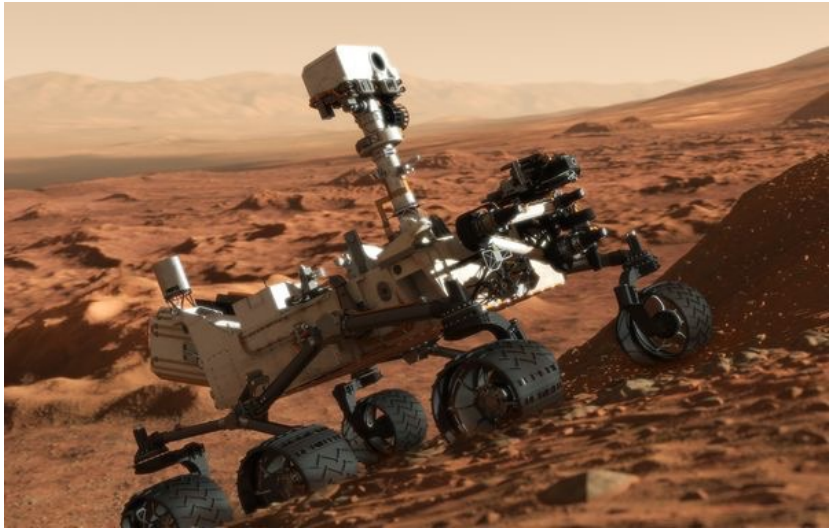
Projekte in der Praxis
- Aufgabenstellung meist zunächst unklar
- Viele Personen beteiligt - heterogene Verteilung von Hintergrundwissen
- Durch viele Personen mit unterschiedlichsten Verantwortungen zu bearbeiten
- Lange Entwicklungszeiten
- Noch längere Weiterentwicklungs- und Wartungszeiten (oft > 10 Jahre)

Fehler in Software







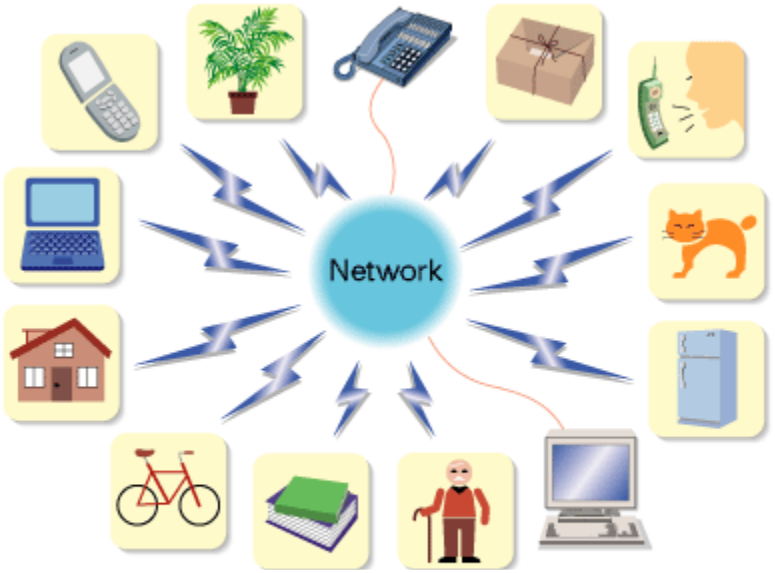


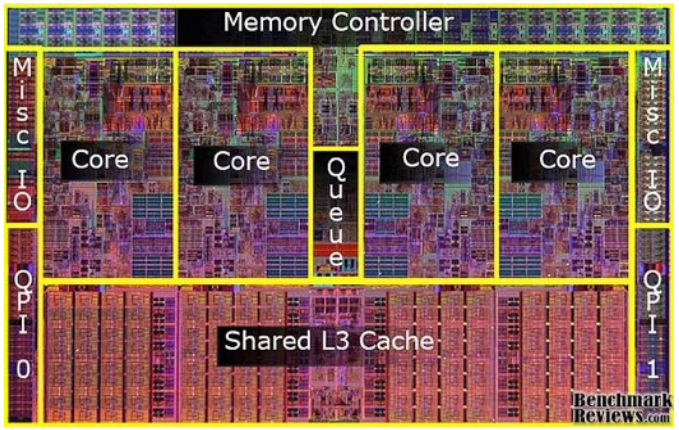
http://iansommerville.com/software-engineering-book/files/2014/07/Bashar-Ariane5.pdf

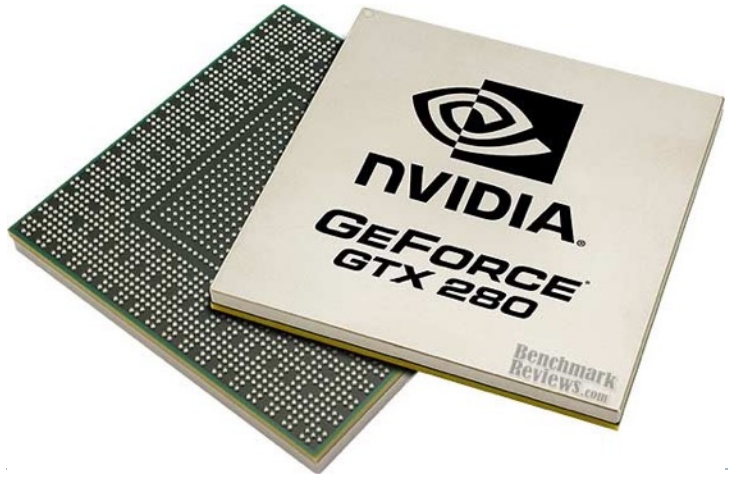
Fehlgeschlagene Softwareprojekte (Beispiele)

- Toll Collect: Geplanter Start 31. Aug 2003, tatsächlicher Start 1. Jan 2006; 3.5 Milliarden EUR Einnahmeausfälle
- Londoner Börse beendet Taurus Projekt 1993 nach 11 Jahren und Budgetüberschreitung um 13200% (800M£)
- Knight Capital verliert im Jahr 2012 rund 400 Mio USD in 45min durch Softwareupdate
- 2017 Eine Fehlkonfiguration führt zu dem vollständigen Löschen der Produktionsdatenbank bei Digital Ocean. Vollständige Wiederherstellung des Systems dauerte 5h.

Aktuelle Herausforderungen

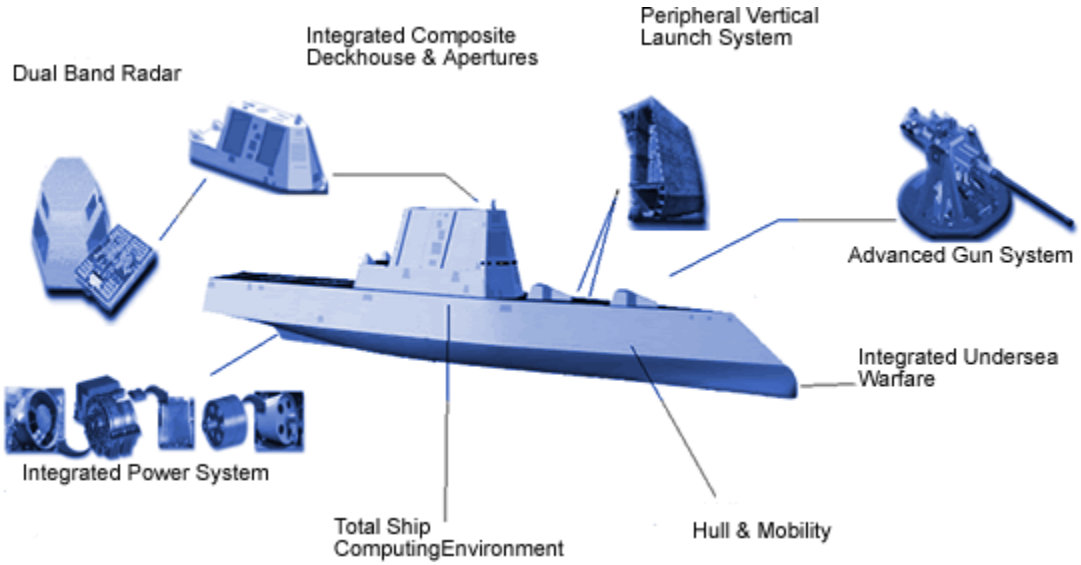






Extreme Komplexität (Beispiel)

- DDX U-Boot
- Viele eingebettete Systeme
- Zusammen 30.000.000.000 Zeilen Code (Schätzung)
- In 142 Programmiersprachen



Woran liegt's?

- unzureichend spezifizierte Anforderungen
- häufiges Ändern der Anforderungen während des Projekts
- Mangel an Ressourcen
- inkompetente Mitarbeiter
- wenig Benutzer-Einbeziehung
- fehlende Unterstützung durch das Management
- zu große Erwartungen
- falsche Schätzung der Zeit/Kosten
- Managementfehler
- Obsolete Projekte (inzwischen bessere Lösungen)

Softwaretechnik als Lösungsidee

*Software Engineering: „The Establishment and use of sound **engineering principles** in order to obtain **economically** software that is **reliable** and works **efficiently** on **real** machines.”*

[Bauer 1975, S. 524]

- Begriff 1968 geprägt
  - Systematisches Herangehen
  - Publikation von bewährtem Vorgehen und Erfahrung
  - Entwurf, Teile-und-Herrsche
  - Wiederverwendung
  - Qualitätssicherung

Softwaretechnik in der Informatik

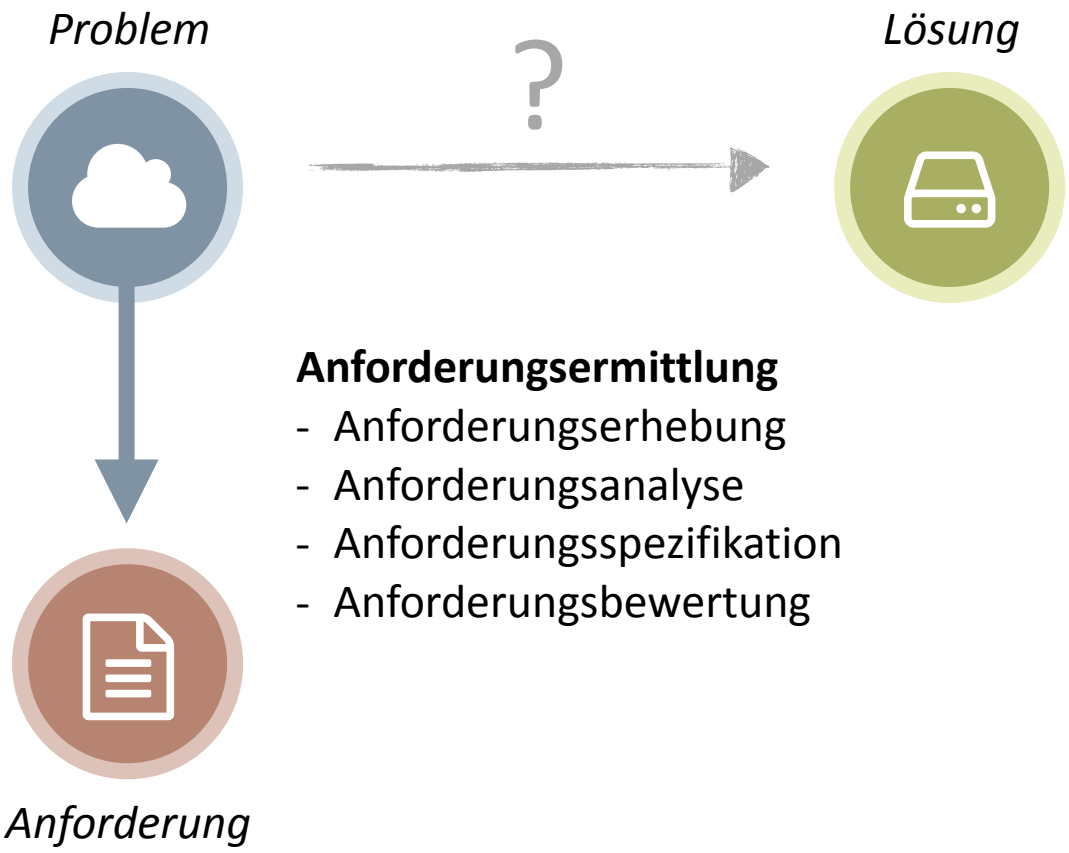
Informatik (computer science)

- Theorien und Methoden für Computer und Softwaresysteme

Softwaretechnik (software engineering)

- Praktische Erstellung von Software

Einordnung: Das Problem verstehen



Anforderungsermittlung

- Anforderungserhebung
- Anforderungsanalyse
- Anforderungsspezifikation
- Anforderungsbewertung

Funktionale vs. Nichtfunktionale Anforderungen

Funktionale Anforderungen

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

Nichtfunktionale Anforderungen

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

Beispiel: Funktionale Anforderungen

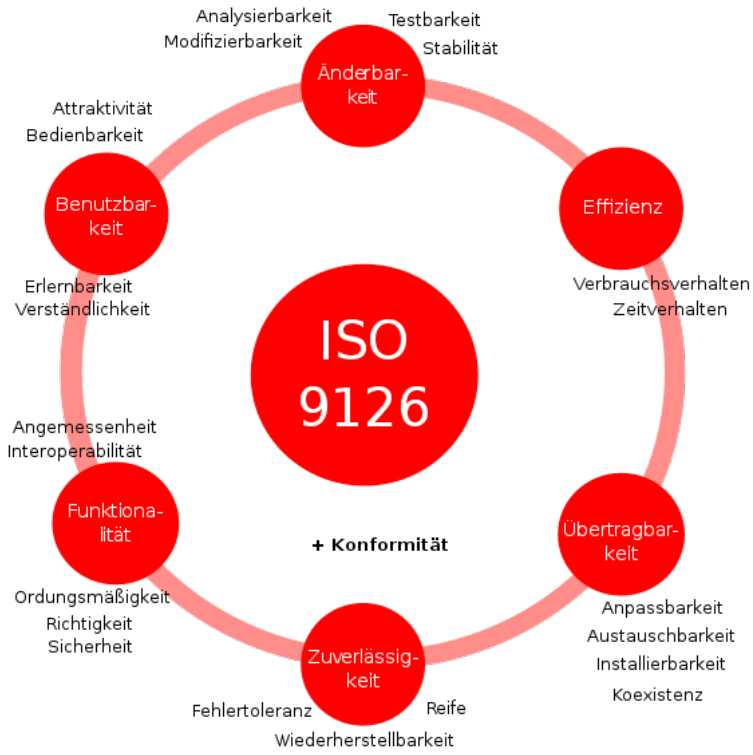
- Funktion: Vorlesung eintragen
- Eingaben: Raum, Zeit und Titel einer Vorlesung.
- Verarbeitungsschritte:
  - Prüfe, ob der Vorlesungstitel schon vergeben ist
  - Prüfe, ob der Raum zur angegebenen Zeit schon vergeben ist
  - Wenn nicht, wird die neue Vorlesung eingetragen und die Daten der Vorlesung 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.

Beispiele: Nicht-funktionale Anforderungen

- technische Anforderungen
  - Das System muss mit Java entwickelt werden und muss in der Sun Java VM 1.5 laufen
- ergonomische Anforderungen
  - Das System muss die gespeicherten Objekte formatiert ausgeben können (Formatvorgabe).
  - Die Benutzerführung erfolgt in deutsch
- Anforderungen an die Dienstqualität
  - Das System muss jede Anfrage des Benutzers innerhalb von 30 Sekunden ausführen (auf System XY).
  - Der Speicherbedarf darf 512mb nicht übersteigen

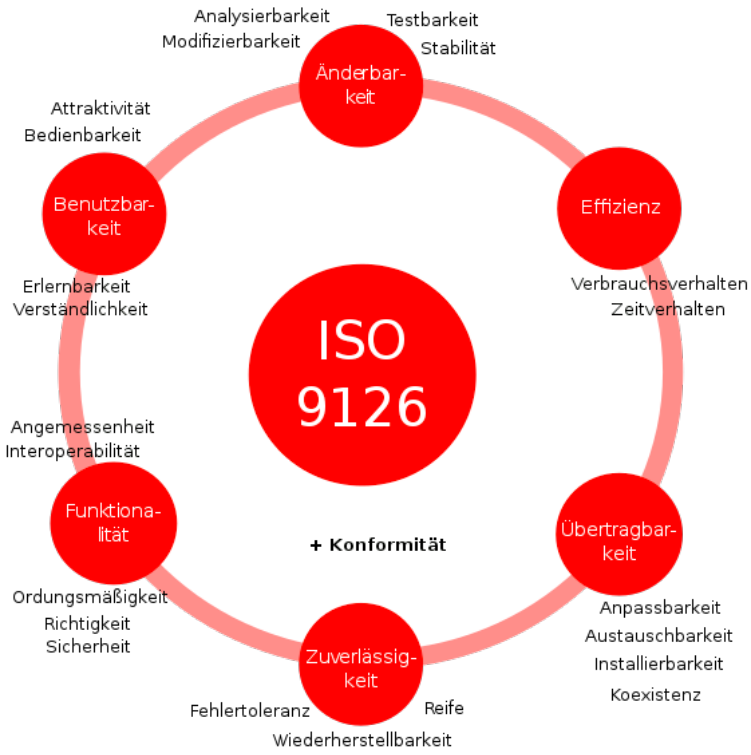
Softwarequalität hat viele Aspekte

- Funktionalität
  - Korrektheit, Angemessenheit, Interoperabilität, Sicherheit
- Zuverlässigkeit
  - Reife, Fehlertoleranz, Wiederherstellbarkeit
- Benutzbarkeit
  - Verständlichkeit, Bedienbarkeit, Erlernbarkeit, Robustheit

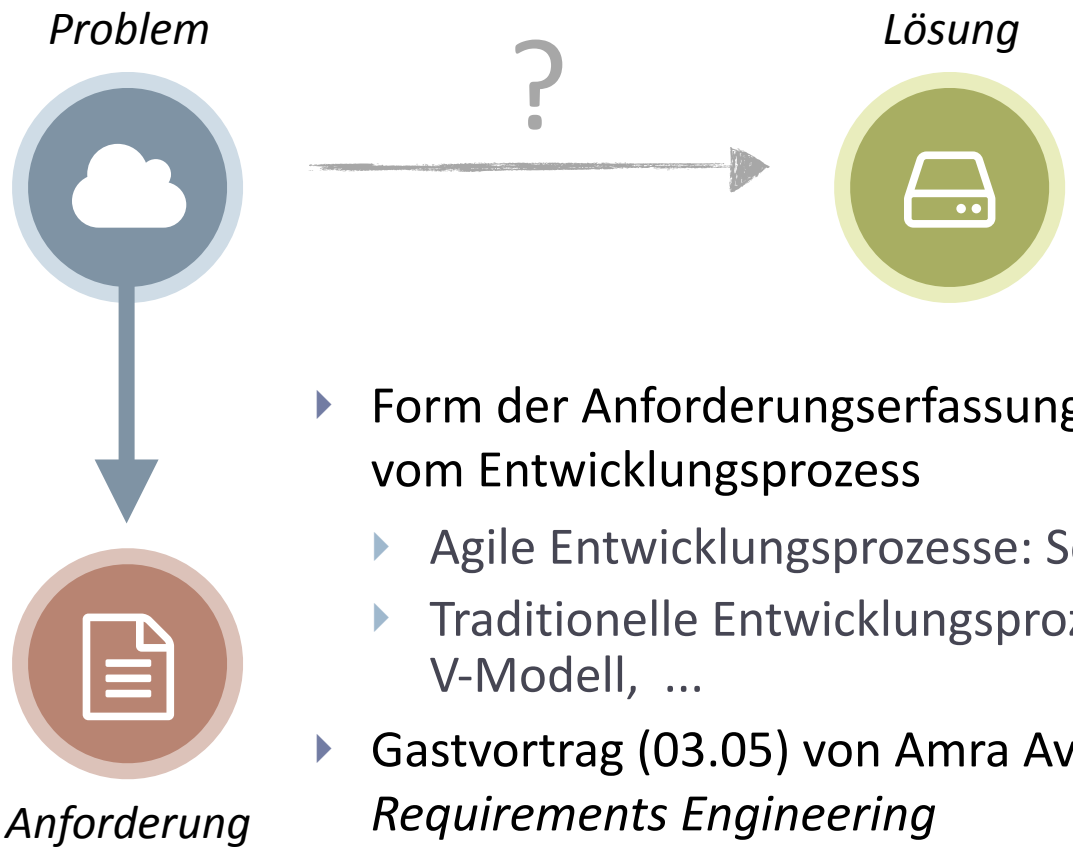


Softwarequalität hat viele Aspekte (2)

- Effizienz
  - Wirtschaftlichkeit, Zeitverhalten, Verbrauchsverhalten
- Änderbarkeit
  - Analysierbarkeit, Wartbarkeit, Testbarkeit, Erweiterbarkeit
- Übertragbarkeit
  - Anpassbarkeit, Installierbarkeit, Konformität, Austauschbarkeit



Einordnung: Das Problem verstehen



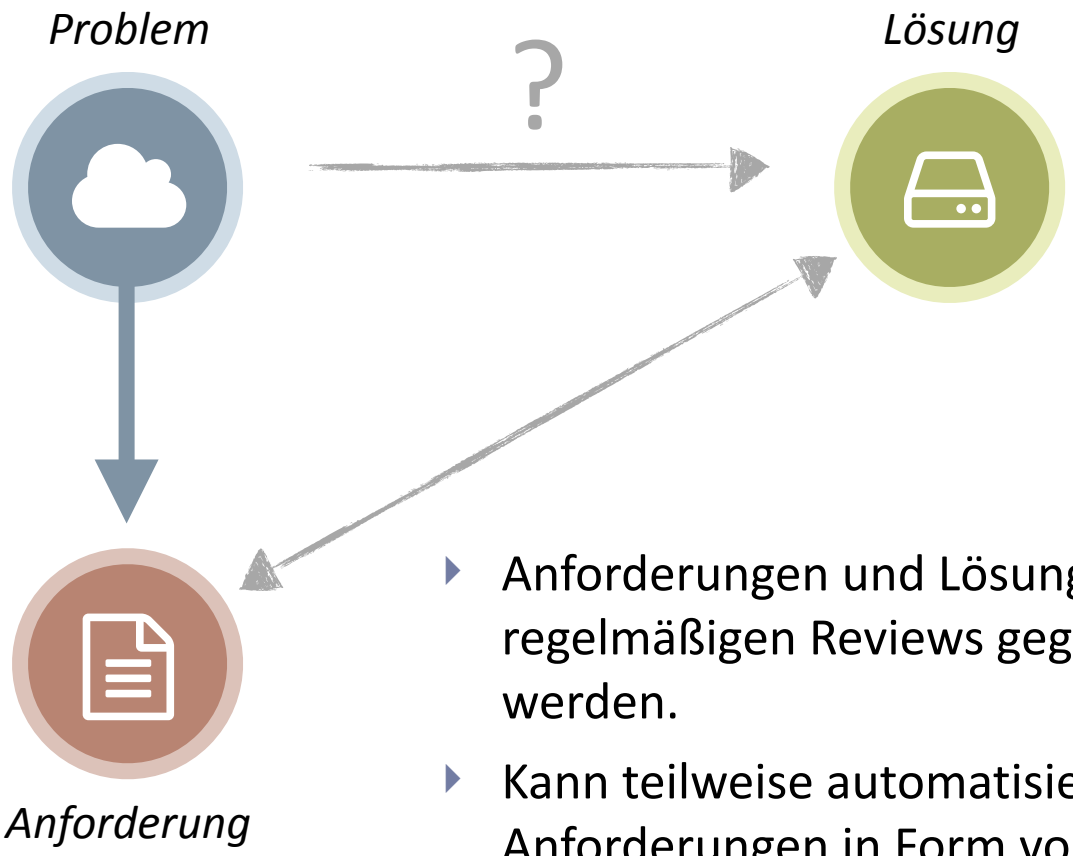
- Form der Anforderungserfassung ist abhängig vom Entwicklungsprozess
  - Agile Entwicklungsprozesse: Scrum, XP, ...
  - Traditionelle Entwicklungsprozesse: Wasserfall, V-Modell, ...
- Gastvortrag (03.05) von Amra Avdic zu *Agile Requirements Engineering*

Prüfung von Anforderungen

- Wird der Bedarf des Kunden **vollständig** abgedeckt?
- **Verständlich** formuliert?
- **Konsistent** mit den anderen Anforderungen?
- **Realistisch** mit Budget und Technologie?
- Anforderung **prüfbar**?
- **Änderbar** ohne Einfluss auf andere Anforderungen?

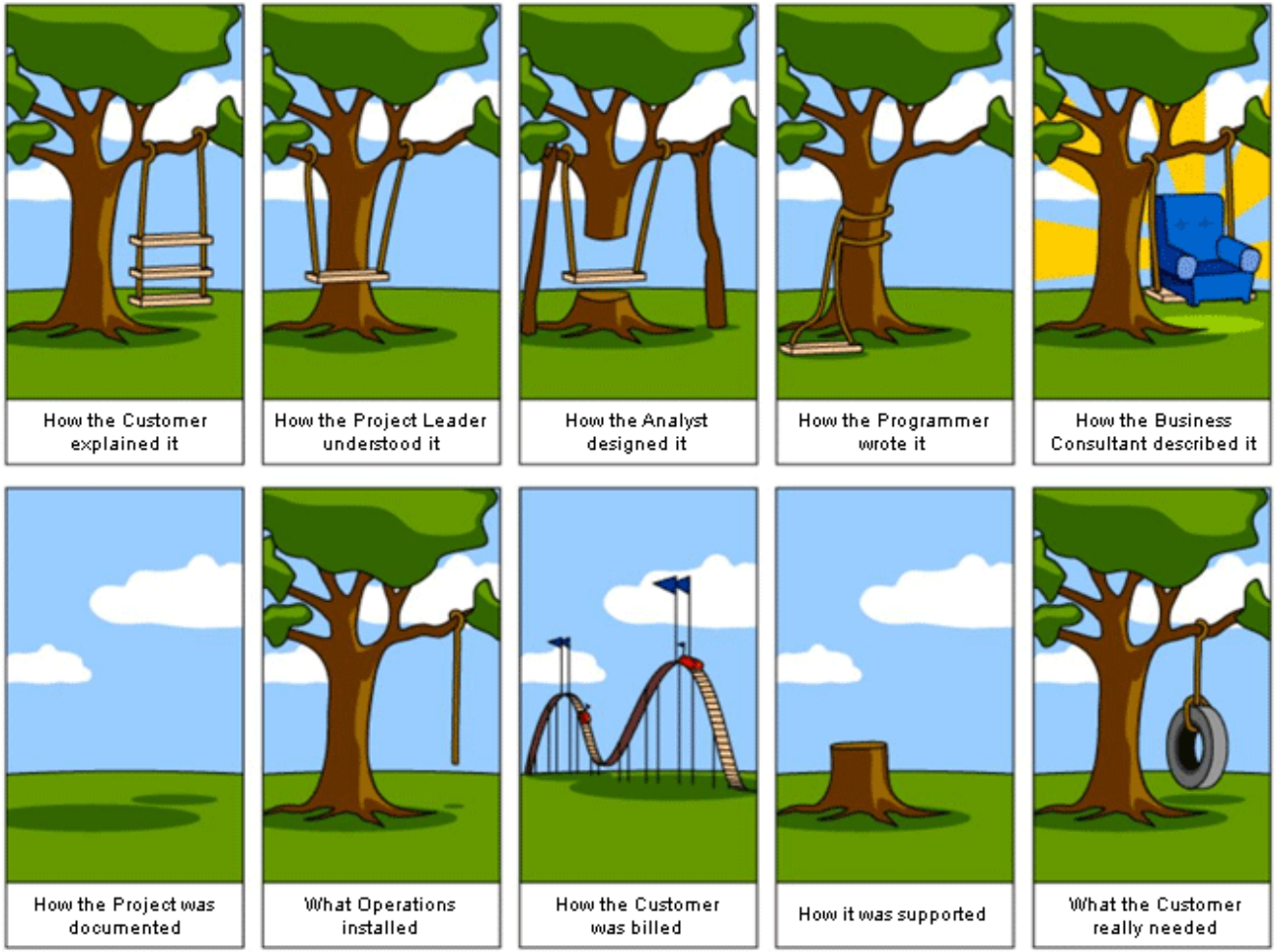
- Regelmäßige Reviews
- Kunden und potentielle Benutzer in Anforderungsanalyse mit einbeziehen

Einordnung: Die Lösung bewerten

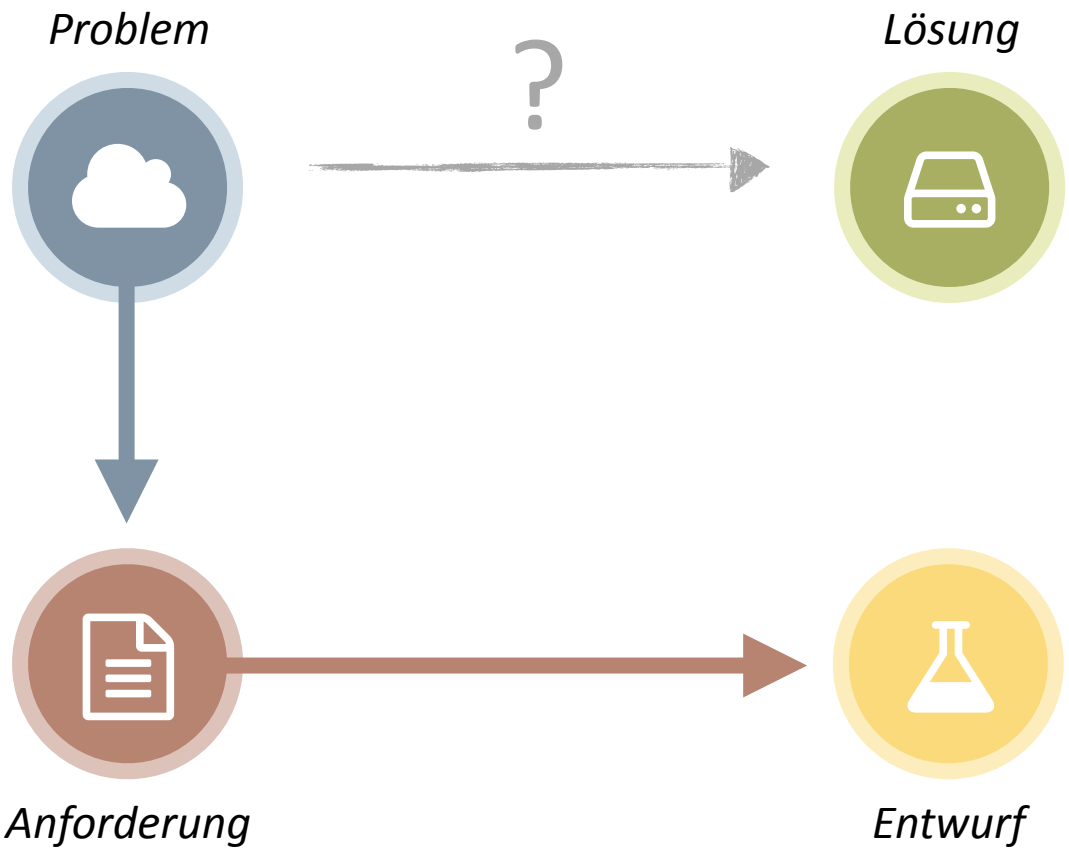


- Anforderungen und Lösung sollten in regelmäßigen Reviews gegenseitig validiert werden.
- Kann teilweise automatisiert werden, wenn Anforderungen in Form von ausführbaren Tests spezifiziert sind.

Kommunikation?



Einordnung: Entwurfsphase



Einordnung: Entwurfsphase

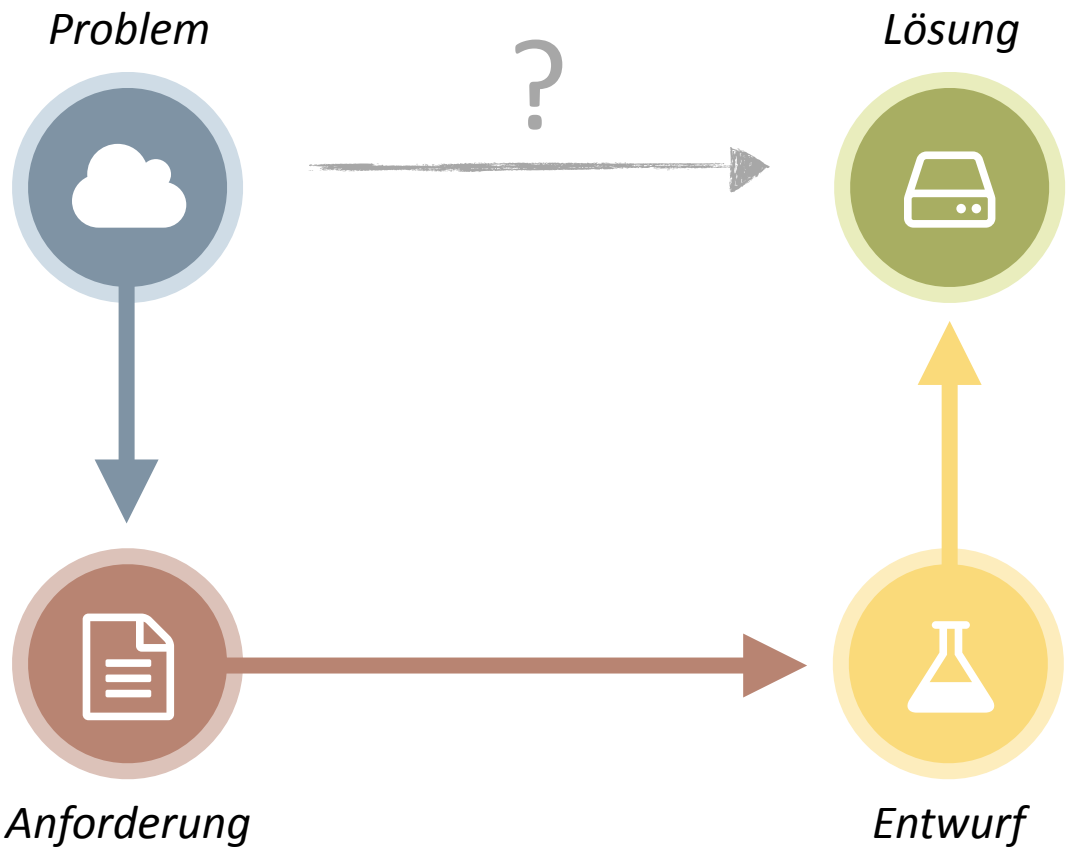
- *"Entwurf"* kann auf verschiedenen Abstraktionsebenen verstanden werden:
  - **Systemebene**. z.B. Aufteilung in Komponenten, Services u. entsprechende Schnittstellen. Auch Deployment und Integration in existierende Systemlandschaft.
  - **Paketebene**. z.B. Aufteilung der Verantwortlichkeiten auf Klassen und Gestaltung ihrer Interaktionen
  - **Klassenebene**. z.B. Entwurf einzelner Methoden und von Verträgen, die diese einhalten müssen (Vor- und Nachbedingungen, Invarianten).

Entwurfsphase: Inhalte in der Vorlesung

Entwurf

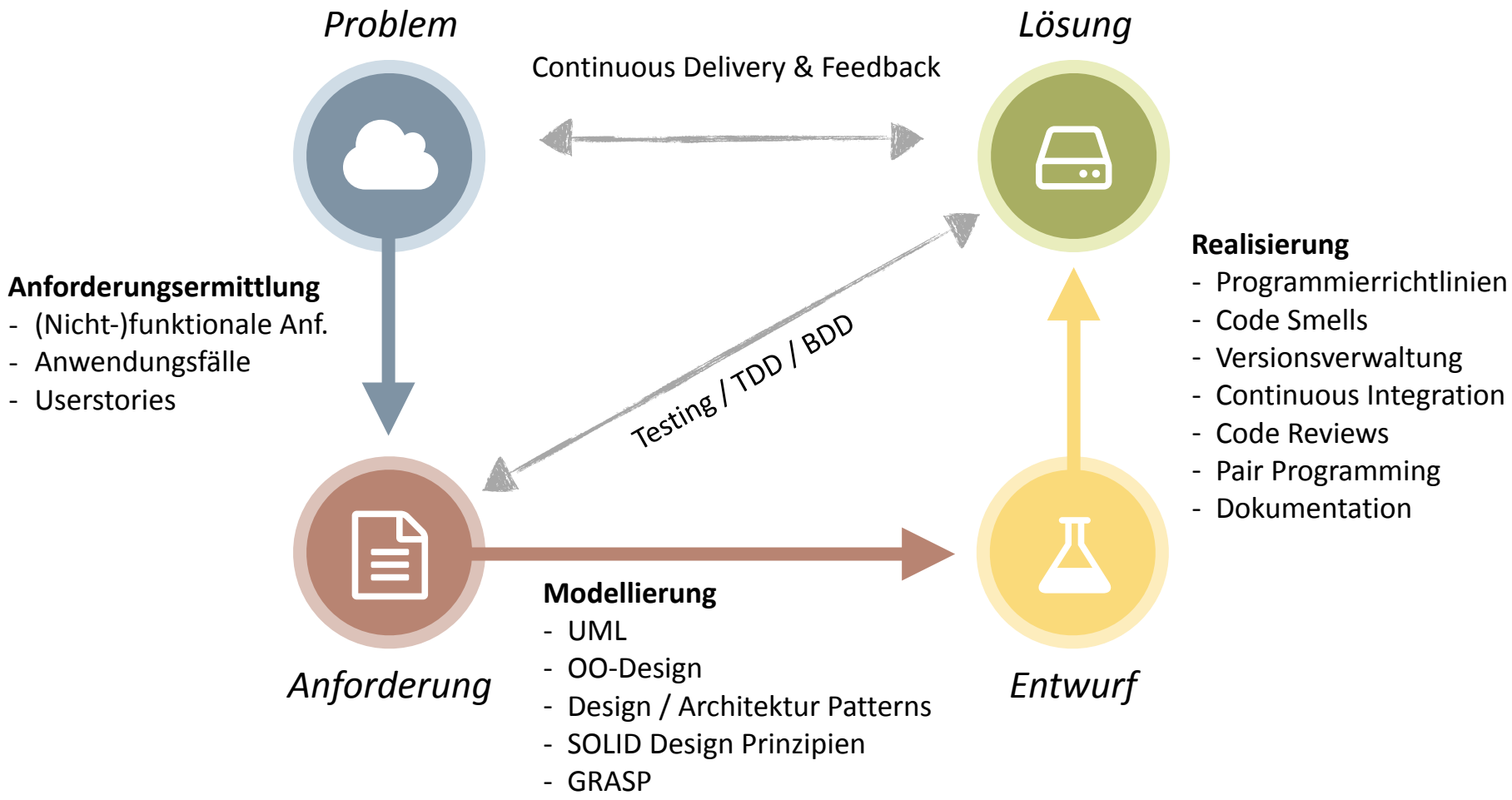
- UML
  - Entwürfe dokumentieren und kommunizieren
  - Beispiele für Diagrammtypen: Klassendiagramme, Sequenzdiagramme, Statecharts
- Design Patterns
  - Lösungsmuster wiederverwenden und kommunizieren
  - Auch auf Software-Architekturebene
- SOLID und GRASP Design Prinzipien
  - Strukturiert Vor- und Nachteile von Entwürfen abwägen
  - Erweiterbarkeit, Wiederverwendbarkeit und Wartbarkeit optimieren

Einordnung

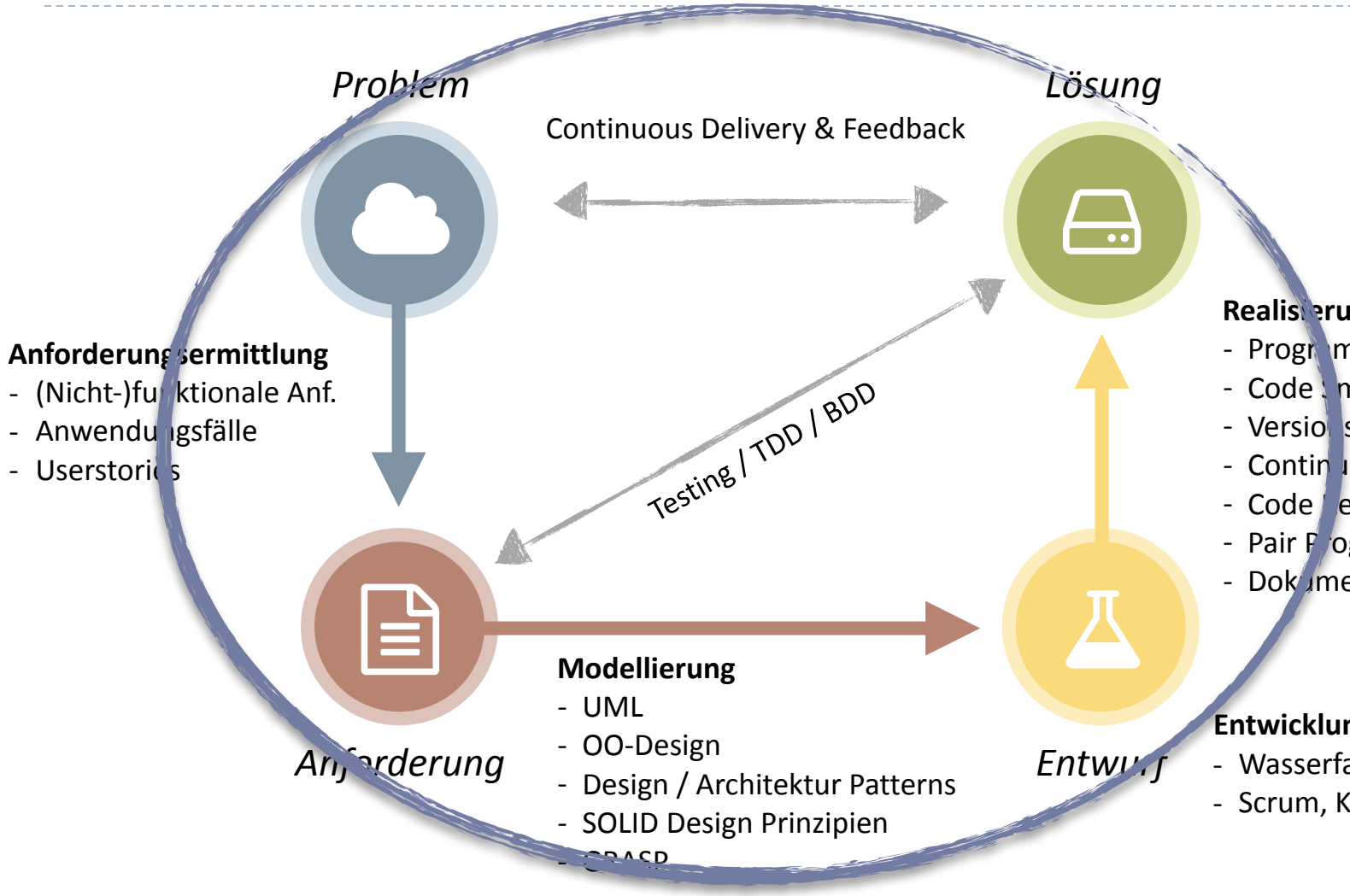


Realisierung
- Programmierrichtlinien
- Code Smells und Refactorings
- Versionsverwaltung
- Continuous Integration
- Code Reviews
- Pair Programming
- Code Dokumentation

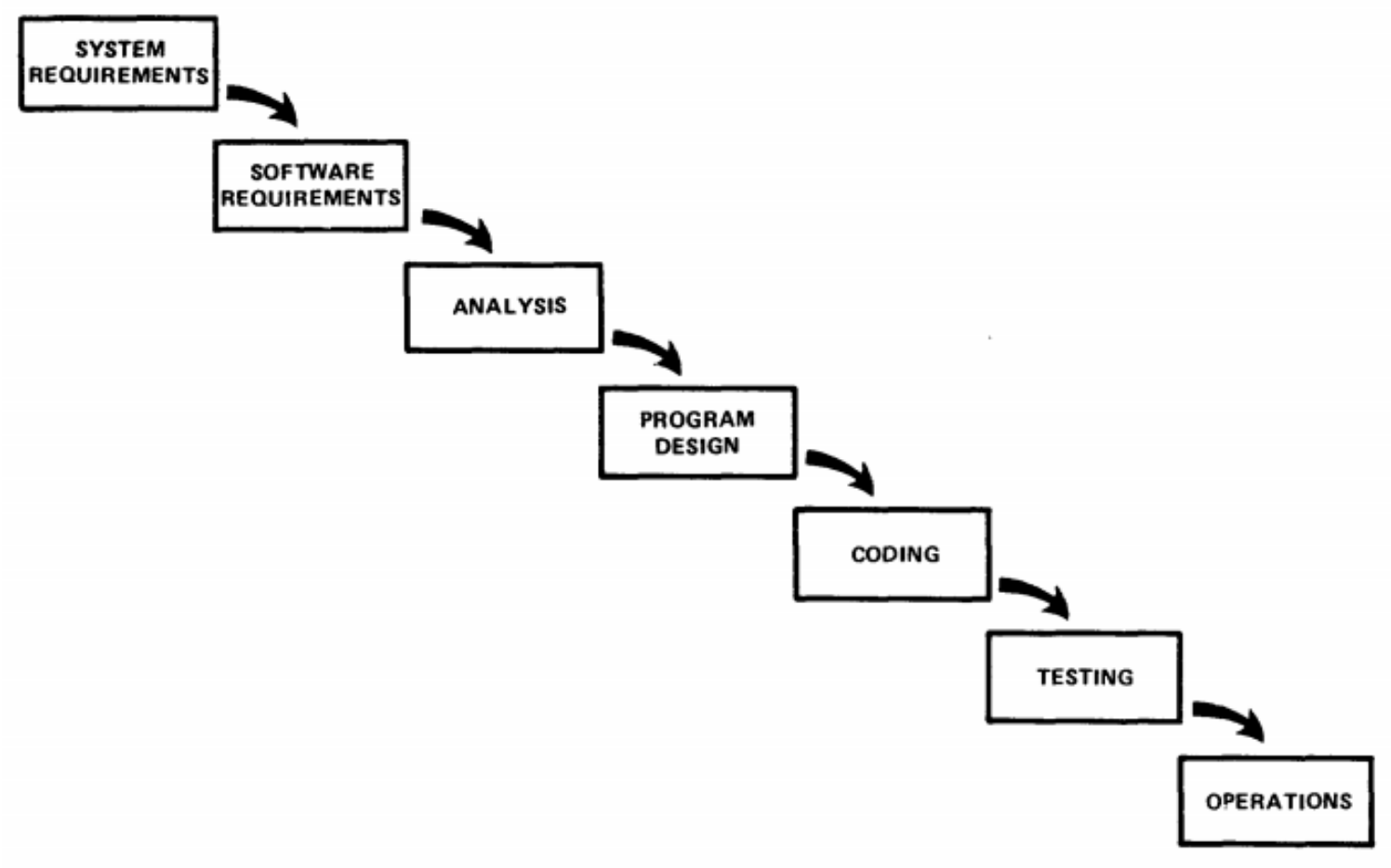
Einordnung: Zusammenfassung



Einordnung: Zusammenfassung

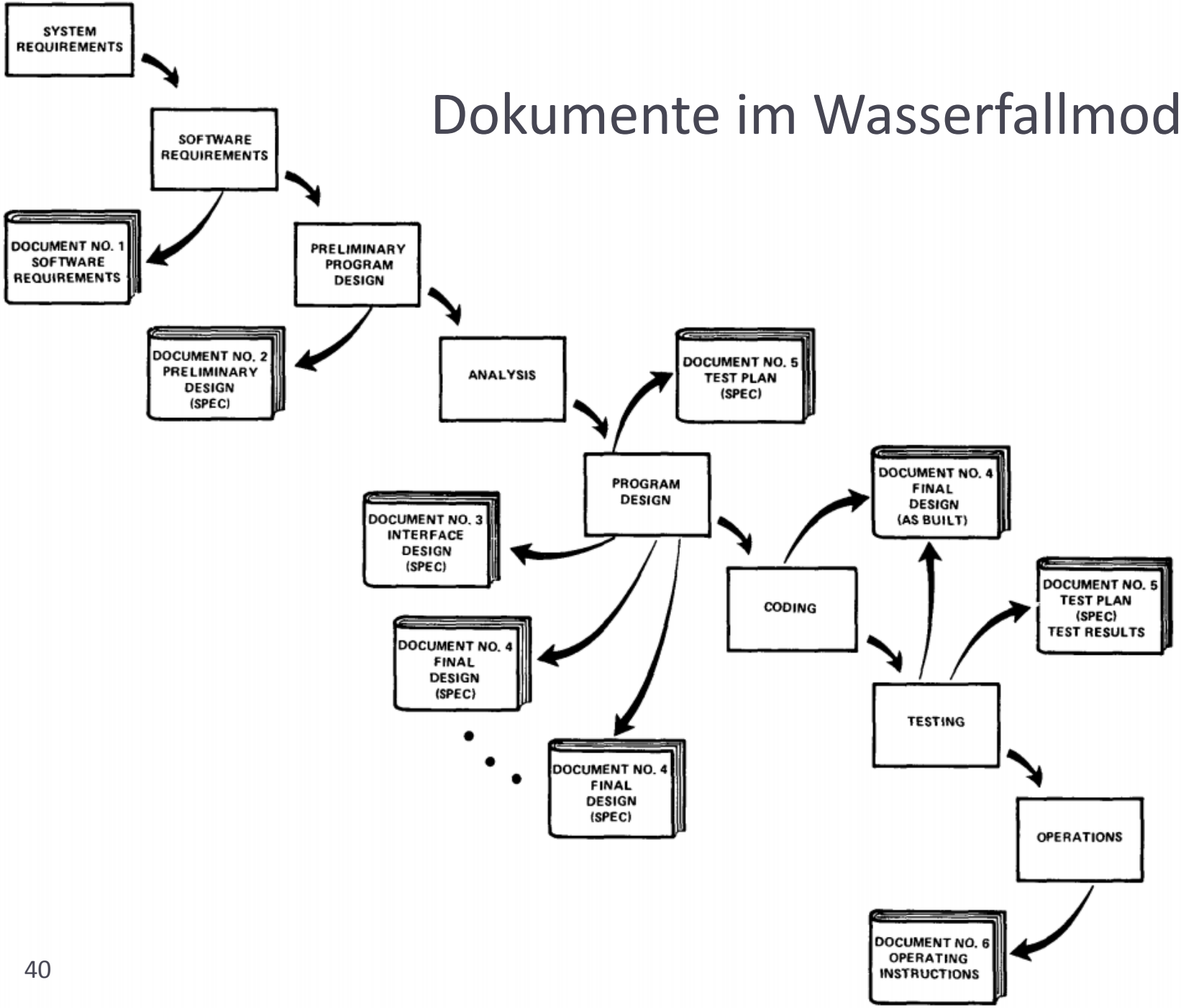


Entwicklungsprozess: Wasserfall



Bilderquelle: Managing the Development of Large Software Systems, Winston Royce (1970)

Dokumente im Wasserfallmodell



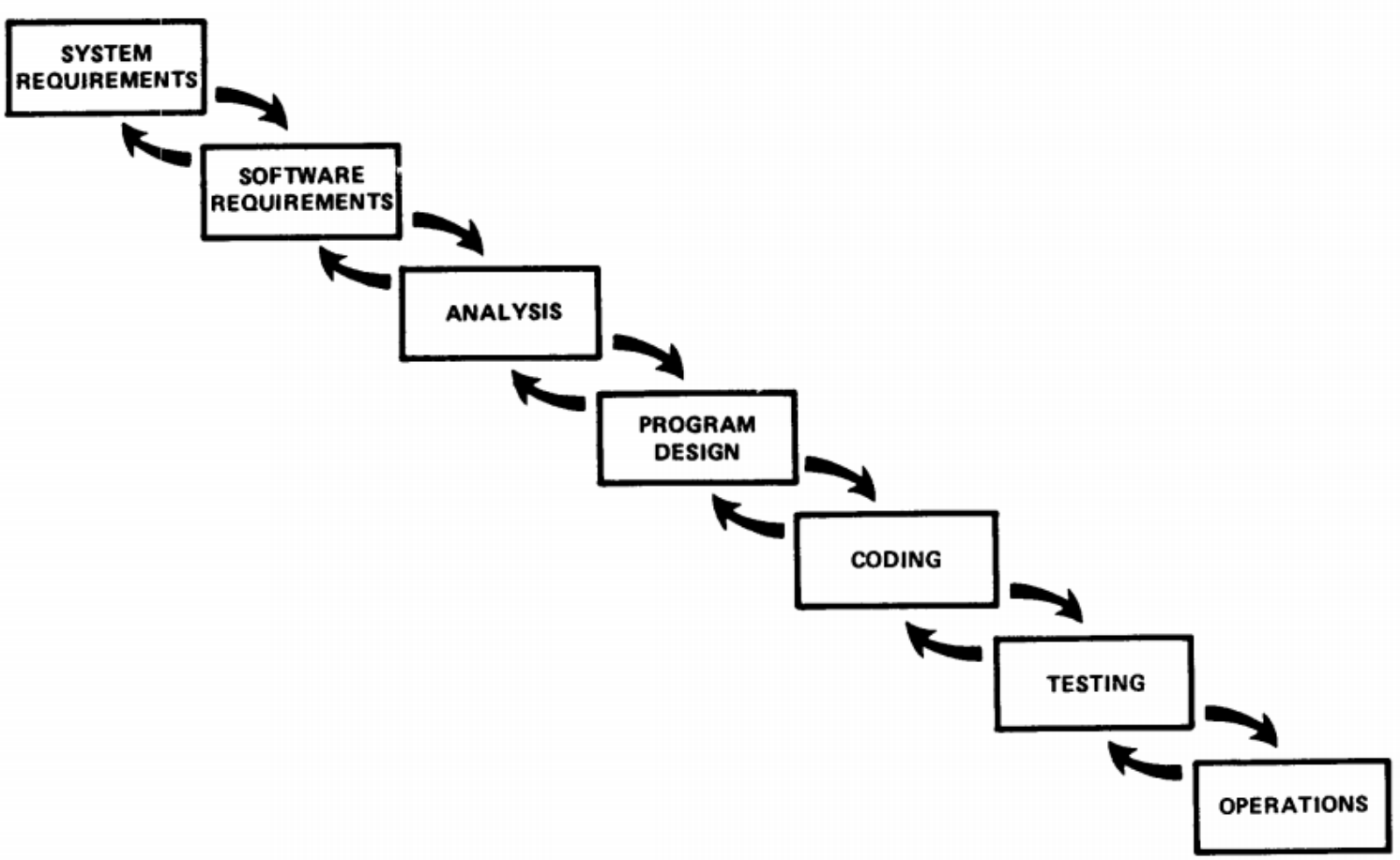
Vorteile des Wasserfall-Modells

- Klar definierte Grenzen zwischen Phasen
  - ermöglicht z.B. Erledigung von Phasen durch Drittparteien
- Kein Rücksprung zu vorherigen Phasen
  - lässt den Fortschritt im Projekt einschätzen
- Stabile Anforderungen erlauben stabile zeitliche und finanzielle Planung

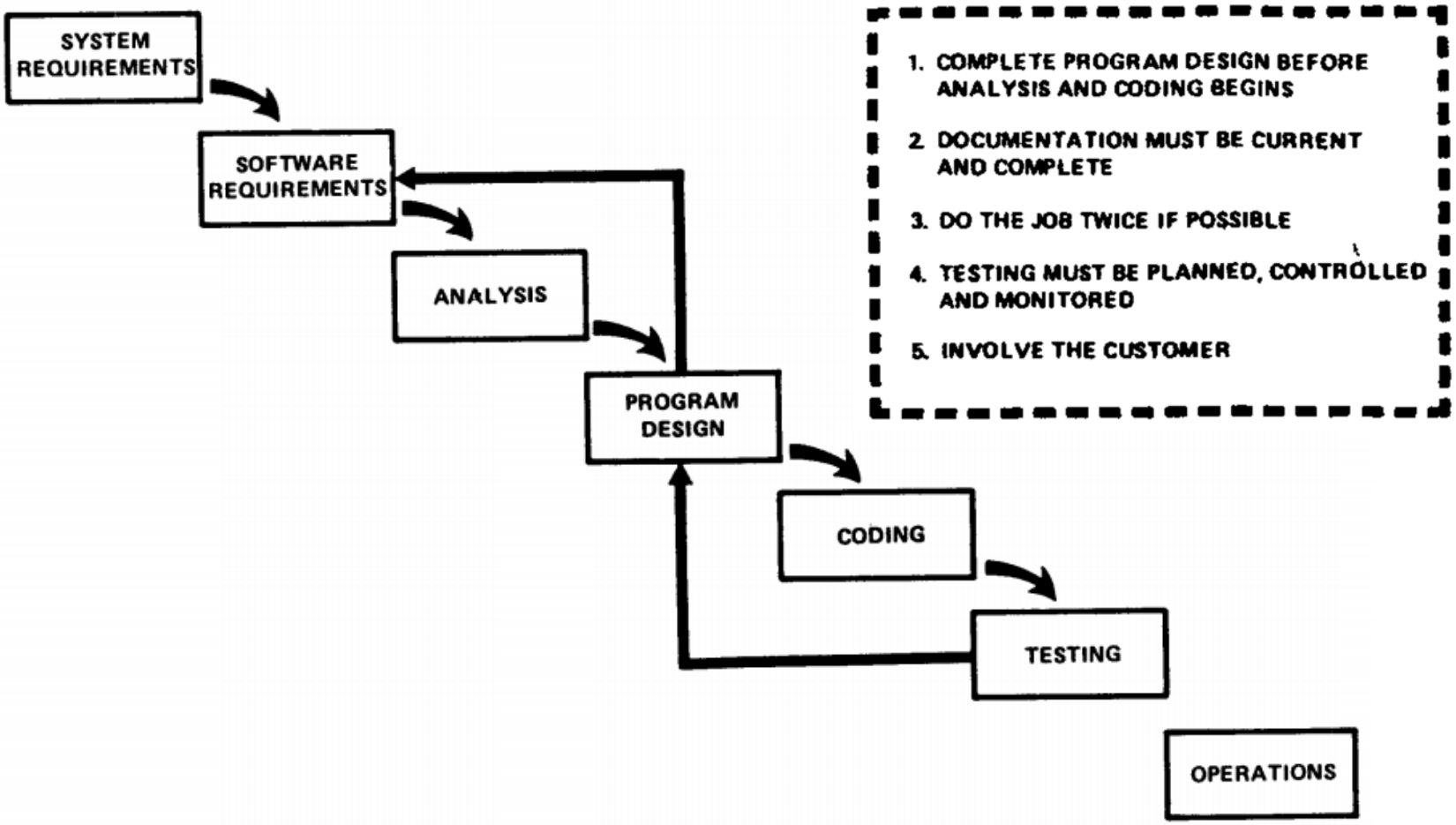
Nachteile des Wasserfall-Modells

- Probleme in der Anforderungsbeschreibung (allg. in früheren Phasen) werden evtl. erst zu spät festgestellt
- Änderungen in den Anforderungen können nur schwer berücksichtigt werden
- Anforderungen sind i.d. Praxis selten stabil / vollständig bekannt
- Phasen können schlecht parallelisiert werden, da sie aufeinander aufbauen

Wasserfall mit Rücksprung



Wasserfall mit Rücksprung (2)



Entwicklungsprozesse: Extreme Programming (XP)

- Evolutionäre Vorgehensweise (inkrementell)
  - keine umfassende Anforderungsermittlung
  - Entwicklung in kleinen Schritten
  - Ergebnis jedes Schrittes ist ein lauffähiges Release
  - Releases werden vom Auftraggeber abgenommen

Extreme Programming (XP) - Praktiken (1)

- Kleine Releases
- Planspiel
  - Auftraggeber formuliert User Stories mit Prioritäten
  - Im Planspiel wird festgelegt, welche Stories in das nächste Release kommen
- Tests
  - Werden vor der Implementierung festgelegt
  - Gelten als Spezifikation
- Systemmetaphern
  - Gemeinsamer Wortschatz von Anwendern, Stakeholder und Entwickler

Extreme Programming (XP) - Praktiken (2)

- Einfacher Entwurf
  - "to do the simplest thing that could possibly work"
- Refactoring ist erlaubt und erwünscht
- Collective Code Ownership
- Programmierrichtlinien
- Continuous Integration
- Dokumentation primär durch selbsterklärenden Code
- Geregelte Arbeitszeiten
- Anwenderverteter im Team

Ablauf der Vorlesung (vorläufig)

- 26.4 – Realisierung
  - Code Smells, Refactorings, Documentation, ...
- 03.5 – Anforderungsermittlung
  - Gastvortrag: Agile Requirements Engineering & Scrum
- 10.5 – Softwareprozesse & Softwareentwurf
  - Gastvortrag: Scrum im Multiprojekt und Multiproduktumfeld
  - Einführung in UML
- 17.5 – Softwareentwurf 2
  - Designpatterns, Design Prinzipien, Objekt Orientiertes Design
- 24.5 – Software Testen
- 31.5 – Realisierung: Versionsverwaltung und Continuous Integration
- 14.6 – Realisierung: Gemeinsam an Code arbeiten
  - Gastvortrag: Social coding and Open source development using git and github
- 21.6 – Klausur

Klausurfragen?

- Was denken Sie wären gute Klausurfragen?
- ...
- ...
- ...