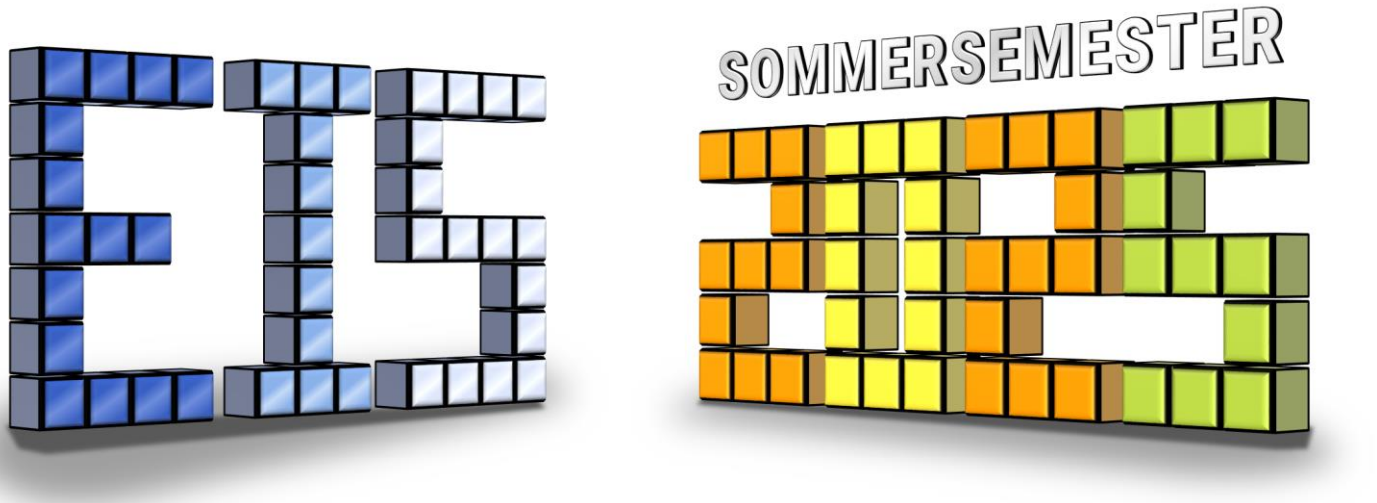
EINFÜHRUNG IN DIE SOFTWAREENTWICKLUNG

Sommersemester 2025



Foliensatz #3

Der Softwareentwurfsprozess

Michael Wand
Institut für Informatik
Michael.Wand@uni-mainz.de

Übersicht

Inhalt heute

- Organisatorisches
- Programmiersprachen
  - Python + MyPy
  - C/C++
  - Java/Scala
- Der Softwareentwicklungsprozess
  - Probleme & grobe Ansätze
- Versionsverwaltung

Einordnung vorab…

Klausurfragen?

Nicht alles ist „Wissenschaft“

- Viele der besprochenen Themen basieren auf
  - Erfahrung
  - Geschmack
  - Soziologischer Evolution (what’s hip, what worked)
- Auswendiglernen macht keine Sinn
  - Keine Wissensfragen in der Klausur
  - Wenn Fragen, dann „warum macht man das“?
- Selbst weiter recherchieren!
  - Gesunde Skepsis ist, wie gesagt, gesund!

Literaturhinweis

Kurze Übersicht zu „Softwareentwurf“

- Bjarn Stroustrup
  „Die C++ Programmiersprache“
  Addison Wesley,
  2. Auflage 1992 (Kapitel fehlen in neuster Auflage)
  - Kapitel 11-13: (insbesondere 11)
    - „11 Programmentwicklung“
    - „12 Design und C++“
    - „13 Bibliotheksdesign“
- Interessante Meinungen zum Thema vom Erfinder von C++
  - Insbesondere Kapitel 11: unabhängig von C++
- Tlw. Basis dieses Abschnitts (sprachneutral)

Der Entwicklungsprozess

Softwareentwicklung ist schwer

Softwareprojekte

- Hohe Misserfolgsquote
  - Statistiken schwanken
  - Auf Google findet in etwa solche Zahlen:
    - „nur 50% erfolgreich“
    - der Rest scheitert (20%)
    - oder überschreitet Zeit- und Budgerahmen deutlich (30%)
- Projektteams sind nicht unbedingt sehr groß
  - Die meisten unter 5 Entwickler/innen
  - Man braucht kein 100-Personen Team für Chaos
- Kampf gegen Komplexität – jede Hilfe nutzen

Was wollen wir erreichen?

Ziele

- Funktion, Robustheit, Testbarkeit
- Flexibilität, Erweiterbarkeit
- Wiederverwendbarkeit
- Verständlichkeit
- Portabilität

(Leicht übersehen)

- (Soziologische Stabilität des Entwicklungsteams)
- (Mentale Stabilität der Anwender des Produktes)

Kernprinzip

Stroustrup

„*Programm-Design und Programmieren sind menschliche Aktivitäten.*

Wer dies vergisst, hat bereits verloren.“

Erinnerung:
Divide and Conquer

Fighting Complexity

Menschliche Limitierungen

- Wir können nur „kleine“ Probleme verstehen
- Zerlege das große Problem in kleine Teile
- Jedes Teil sehr einfach
- Schnittstellen einfach

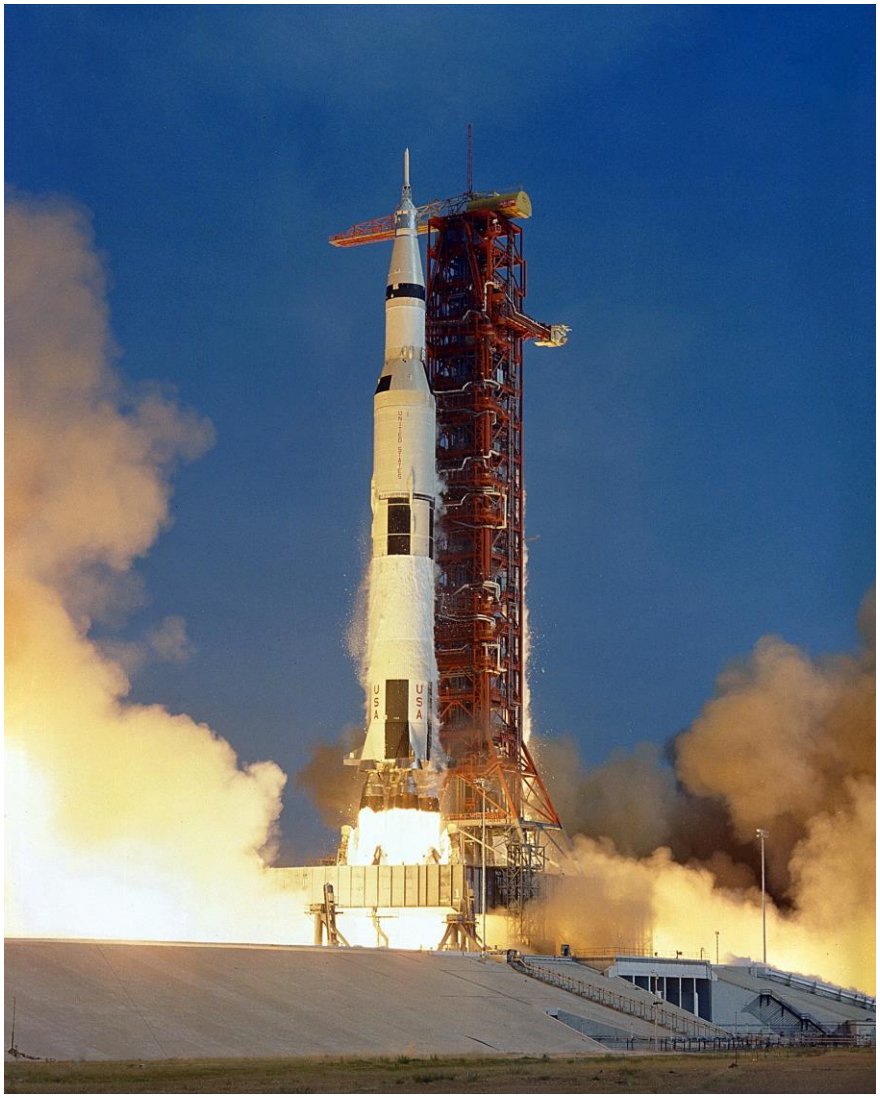
} im Vergleich zur Gesamtaufgabe

„Der Trick“: Kleine Schritte

Menschen können nur kleine Schritte verstehen

- Lösung von Problemen durch Aufteilen
  - „Verschiedene“ Teilprobleme
    - Isolieren
    - Lösungen über einfache Schnittstellen zusammenfügen
  - „Gleiche“ Teilprobleme
    - Iteration / Rekursion
    - Problem schrittweise in Komplexität reduzieren
- Programmieren als Aktivität
  - In kleinen Schritten vorgehen
  - Immer wieder Testen
  - „Riesen Ding zusammenbauen und los“ geht meist schief

Riesen Ding zusammenbauen und los…



1966:
NASA gets 4.41% of Federal US Budget
[Wikipedia]

Struktur eines Entwicklungsprozesses

Aspekte der Softwareentwicklung

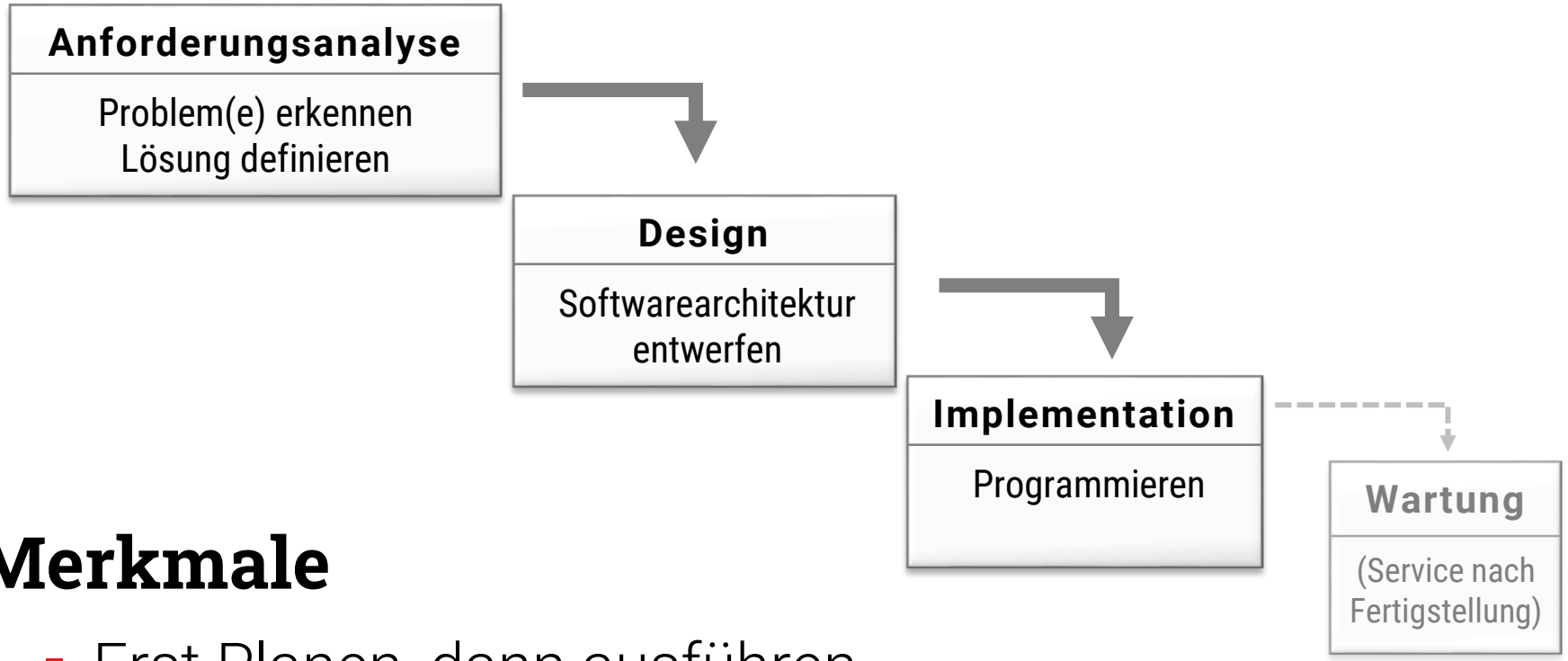
Bestandteile / Schritte

- Analyse (Anforderungsanalyse)
- Design (Softwarearchitektur)
- Implementation („Programmieren“)

Weitere Aspekte

- Management (Meetings, Kommunikation, Entscheidungsprozesse, etc.)
- Dokumentation
- Testen
- Experimentieren / Prototypen

Klassisch: Wasserfallmodel



Merkmale

- Erst Planen, dann ausführen
- Kommunikation (hauptsächlich) abwärts
- Oft an soziologische Hierarchie (auch €€€) gekoppelt
  - Mehr „Macht“ weiter oben
    → (hoffentlich) erfahrenere Entwickler/Innen

Kritik

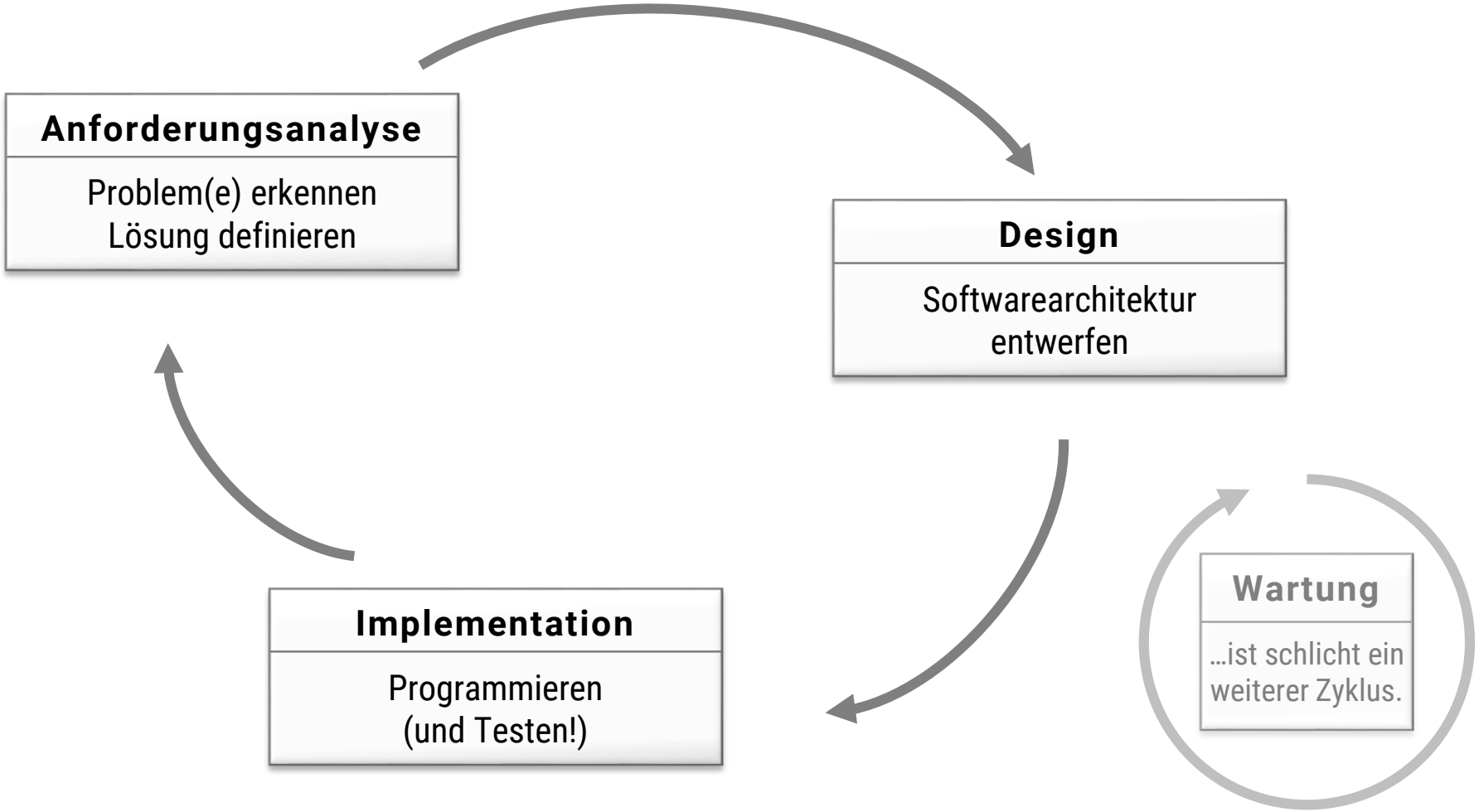
Unbestrittene Vorteile

- Die meisten Fehler werden im ersten Schritte gemacht
  - Die zweitmeisten im Design
  - Explizite Analyse, Design extrem wichtig
- Analyse & Design erfordert mehr Erfahrung

Kritik (Stroustrup)

- Kommunikation hauptsächlich „abwärts“
  - Fehler in Analyse und Design werden spät gefunden
  - Oft „local fixes“ mit „zerstörerischer Struktur“
- Starre Hierarchie hinderlich
  - Behindert Feedback, pers. Weiterentwicklung

„Moderne“ Sicht: Iterativ



Schritte wiederholen, inkrementelle Verbesserung

Weitere Aspekte

Management

- Verschiedene Vorgehensmodelle
  - z.B. „XP (extreme programming)“, „Agile“, „Scrum“, etc.
  - Wichtiges Thema in Vorl. „Software Engineering“
- Vorgehen abhängig von Projektgröße und Komplexität
  - 2 Entwickler, „triviales“ Tool mit 10.000 LOC*)
    → Reinhacken nach 2h Meeting möglich
  - 2 Entwickler, 10.000 LOC, Autopilot für Passagierflugzeug
    → Starke Formalisierung, Zertifizierung, mehrjährig
  - 7 Entwickler, 1M LOC, 1 Jahr Laufzeit
    → Komplexeres Vorgehensmodell
  - Großprojekt, >10 Personen (sehr groß: Linuxkernel, Officepaket): komplexe, oft vielschichtige Struktur

**) LOC = „lines of code“*

Weitere Aspekte

Dokumentation

- Offensichtlich wichtig
  - Oft vernachlässigt
- Zuviel kann auch schädlich sein
  - Unflexibel wg. Änderungsaufwand
    - „dann müssen wir das ganze Handbuch neu schreiben…“
    - Argument für iterative Entwicklung
  - Auch problematisch: „out-of-sync“
    - Doku entspr. nicht „ist-Zustand“
    - Abhilfe: Integration von Dokumentation und Entwurf/Progr.
    - In der Implementationsphase z.B.:
      JAVA-Doc, Python-Doc-Comments, Doxygen (C++)
    - „CASE“-Tools für Analyse, Entwurf (computer aided SE)

Weitere Aspekte

Testen

- Sehr wichtig
- Gleiche Aufmerksamkeit wie programmieren selbst

Andiskutiert in EiP

- Unit-Tests, Assertions, Test-Suites

Populäre Vorgehensmodelle

- „Test-Driven-Development“ (TDD)
- Tests parallel zum Code schreiben kontinuierlich testen

Experimentieren / Prototypen

Erfahrung ist wichtig

- Gutes Design für unbekannte Problem kaum möglich (meine Erfahrung)
  - Der zweite oder dritte Anlauf funktioniert (vielleicht)

Erfahrungen gewinnen

- Experimente sind wichtig!
  - Einfache Lösung „reinhacken“ und schauen, wo es kracht
  - Prototypen bauen
- Gefahr von Prototypen
  - Wenn die Deadline naht, wird daraus schnell das Produkt

Anregungen & Prinzipien

Leitlinien

Stroustrup (Auszug)

- Wisse, was Du erreichen willst
- Stecke Dir spezifische und erreichbare Ziele
- Suche nicht nach technischen Lösungen für soziologische Probleme
- Denke langfristig
  - im Design
  - im Umgang mit Menschen
- Verwende [gute*)] Systeme als … Inspiration
- Entwerfe in Hinsicht auf Änderung
  - Flexibilität, Erweiterbarkeit, Portabilität, Wiederverwendung

*) eigene Ergänzung

(Die C++ Programmiersprache, 2. Auflage, Kap.11)

Leitlinien

Stroustrup (Auszug)

- Verwende die besten Werkzeuge…
  - im Design und
  - in der Implementation
- Experimentiere, analysiere und teste so früh und oft wie möglich
- Halte ein der Projektgröße angemessenes Niveau von Formalisierung
- Einfachheit: so einfach wie möglich, aber nicht einfacher

(Die C++ Programmiersprache, 2. Auflage, Kap.11)

Meine Erfahrung

Analogie zu Statistical Learning Theory

- „Occams Razor“
- Einfachstes Modell (Systemstruktur), das (die) gut genug ist, ist die beste

Aber: Qualifikationen

- Einfach: In our heads (für Menschen!)
  - Nicht 100% equivalent zu kurzer Code, Eleganz, etc.
- Abstraktionen, die gut zum Problem passen
  - „Alles passt magisch zusammen“
  - Struktur passt zum Problem
- Lösung: Erfahrung oder „Standard-Rezepte“ (→ EIS)

Be Careful, but Relax…

Softwareentwurf

- Menschliche Aktivität
- Erfahrung und „guter Geschmack“ unersetzlich
- „There is no silver bullet“ (Fred Brooks)
  - Skeptisch bei „absoluten“ Versprechen sein

Relax

- Nicht übertreiben
  - Einfaches Design, mit Mitteln, die man vollständig versteht, ist viel besser als „fancy“ aber nicht völlig verstanden
- Auch Perfektionismus kann Projekte töten

Warnung vor „Cargo Cult“

Kein „wie“ ohne „warum“

- (Geschichte von Feynman)

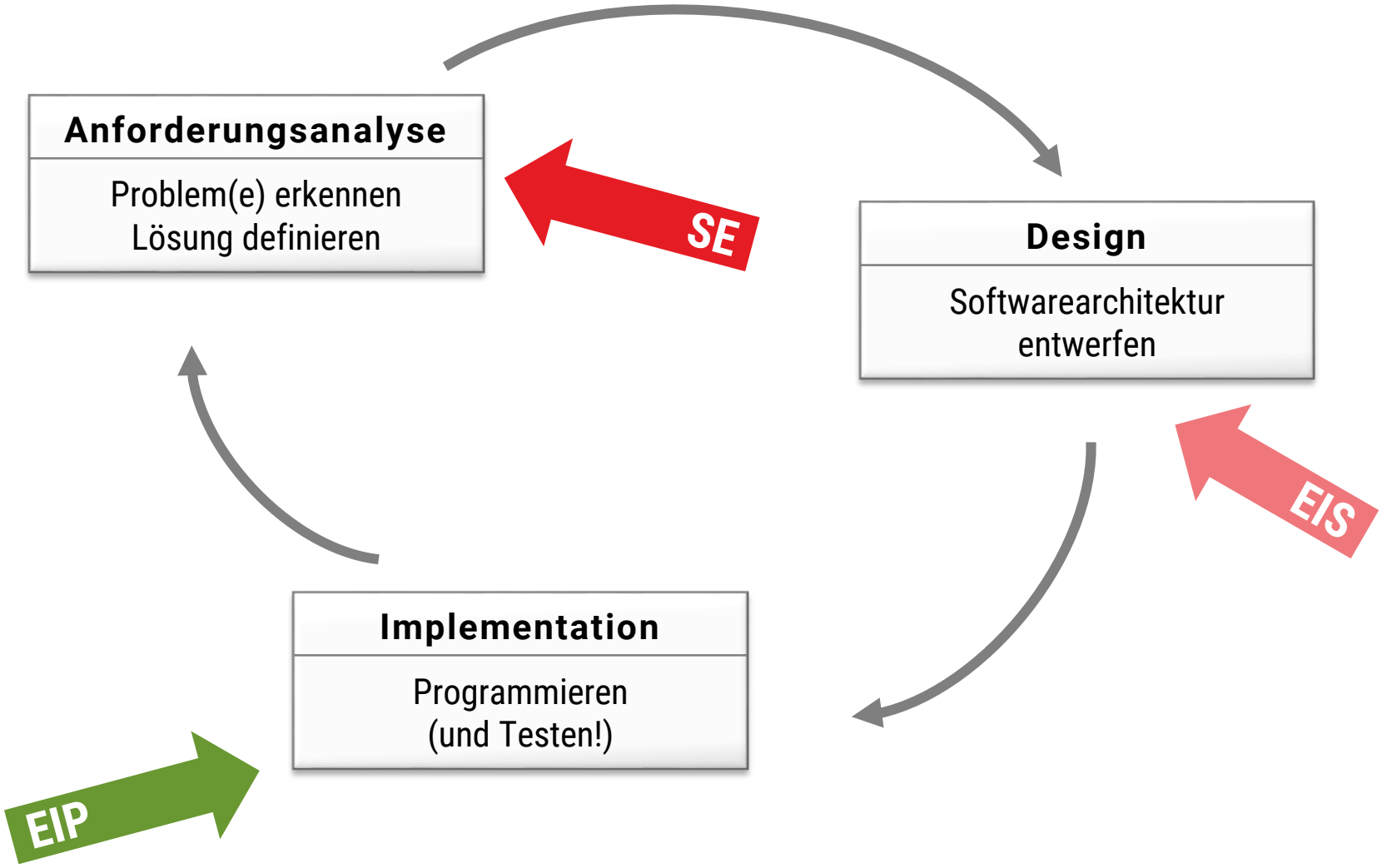
Problematisch

- Unittest schreiben weil „man muss halt“
- Code-Reviews „damit man es gemacht hat“
- Übungszettel abschreiben weil „ich brauch die Punkte“

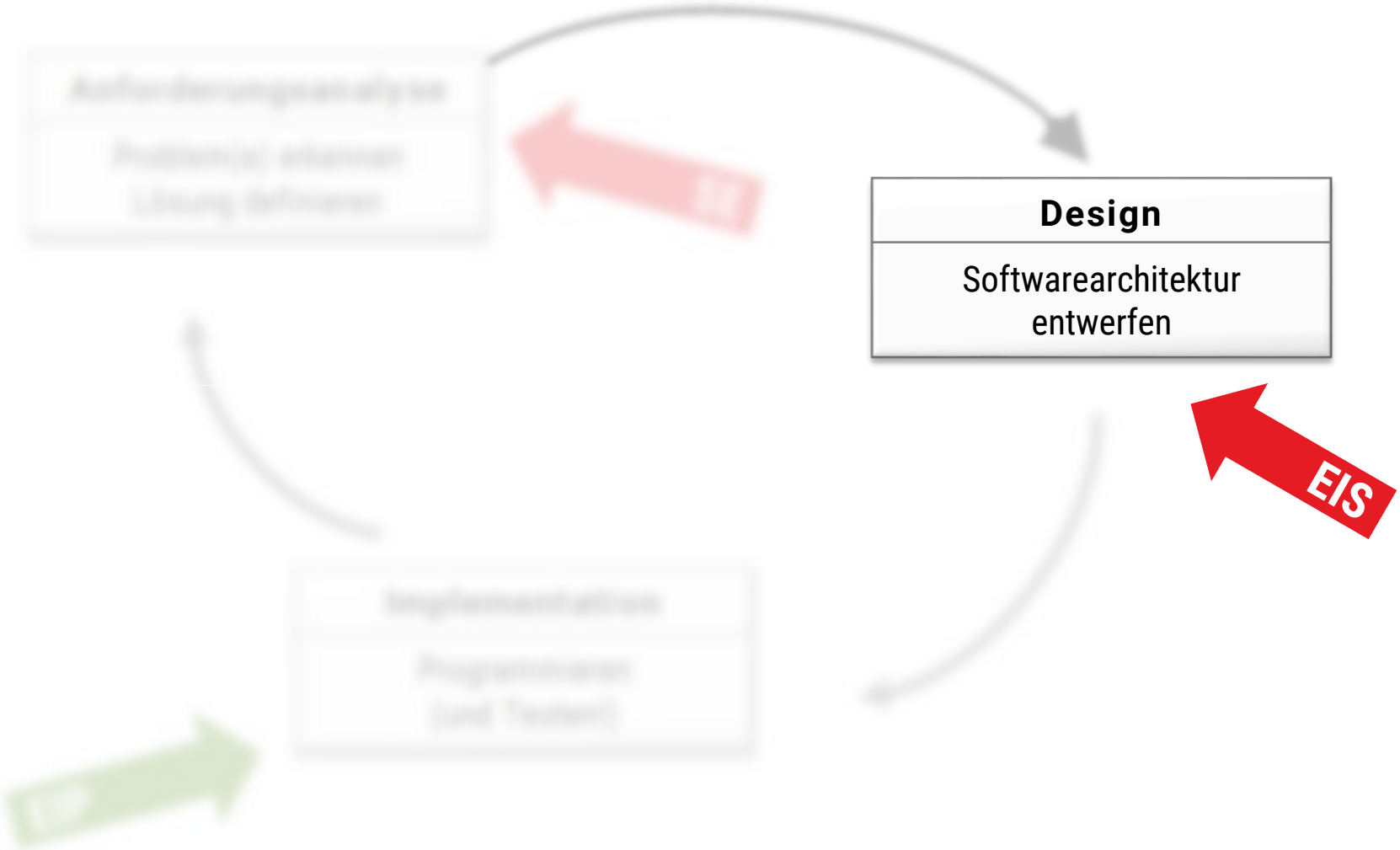
Beim Einsatz von Konzepten/Methoden

- Warum hilft das? Wie erreiche ich das Ziel sinnvoll?
- Bürokratie ist manchmal nötig, aber nicht das Ziel

Nun: Softwareentwurf (Design)



Nun: Softwareentwurf (Design)



Prozedural, OOP, Funktional

Ziel der Programmiertechniken

Wiederverwendung!

- Alles „nur einmal programmieren“
  - DRY – don’t repeat yourself
  - Fehler vermeiden
  - Arbeit minimieren
  - Strukturierung des Systems
- Alle strukturellen Programmiertechniken (für „Software-Entwurf“) dienen letztlich diesem Zweck

DRY: Mittel zum Zweck

- Sehr wichtig, aber keine „Religion“, nicht übertreiben

Leitlinie

Softwareentwurf als Bibliotheksentwurf

- Softwarekomponenten als allgemeine Problemlösungen
- Unabhängig von Kontext / Anwendung nützlich
- Macht es einfacher, Kernidee zu identifizieren
- Nicht zu allgemein
  - So allgemein wie möglich ohne (übertriebenen) Zusatzaufwand
  - Bewährte Muster funktionieren oft am besten (wenn verfügbar / bekannt)

Zusammenfügen

Kernproblem: Schnittstellen

- Schnittstellen sollen abstrahieren
  - z.B. 200 LOC → eine Funktionssignatur
  - „Information hiding“ (Geheimnisprinzip)
- Schnittstellen sollen einfach sein
  - Einfaches Grundprinzip, Leicht zu verstehen
  - Klare Annahmen / Invarianten
  - „As simple as possible but not simpler“
- Schnittstellen gut dokumentieren
- Flexibler Einsatz (z.B. via Polymorphie)
  - Polymorphie: Gleicher Code für verschiedene Datentypen

Techniken

Strukturierung von Programmen

- Prozedural
- Funktional
- Objekt-orientiert
- Meta-Programmierung
  - „Statisch“
  - „Dynamisch“

Wir werden dies ausführlich kennenlernen

Abstraktionen

Modularisierung

Grundprinzip

- Schnittstellen für Module schaffen
  - Module können dann leicht hinzugefügt werden
  - „Selbes Prinzip, andere Variante“

Beispiele

- Verschiedene Arten „Monster“ im Computerspiel
- Verschiedene Malwerkzeuge für Malprogramm
- Verschiedene Ereignisse in interaktivem UI
- Verschiedene Formatierungsalgorithmen für Textverarbeitung

Modularisierung

Grundprinzip

- Abstraktion
  - Details leicht zu ändern
  - „Versteckt“ vor Benutzer des Moduls
  - Stabile Schnittstelle für variierende Implementation

Beispiele

- Änderung von Algorithmen und Datenstrukturen
  - Prototyp mit linearer Liste/Array,
    Produktionssystem mit B-Tree auf Remote Server
  - Umstellung des Datenbank Backends
- Portierung auf neues OS/UI/CPU etc.

Wünschenswert

Gute Eigenschaften

- Einfache Schnittstellen
- Natürliche Schnittstellen
- „Orthogonale“ Module
  - Frei kombinierbar
  - Möglichst keine Wechselwirkungen/Einschränkungen
  - z.B. Trennung von Format und Inhalt auf einer Webseite
  - z.B. Aussehen und Verhalten eines Monsters im Spiel
  - z.B. verschiedene Befehle in Python
- Einfach verständliche Struktur

Technische Herausforderungen

Erweiterung

- Neue Datentypen
  - z.B. Rainbow-Pinsel vs. Bleistift in Malprogram
  - z.B. Spielemonstern mit vier Armen, die Hit-Points haben
- Neuer Code
  - z.B. Simulation von Malwerkzeugen
  - z.B. Treffer erkennen, wenn Held auf Monster ballert
- Neue Datentypen brauchen (auch) neuen Code

Weg zur Lösung

- Variablen, die auf Code verweisen
  - „Functional Programming“ im allgemeinen Sinne (auch OOP)

Roadmap

Programmiertechniken

- Prozedural – ohne Funktionsvariablen
- OOP – Funktionsvariablen (nur) in Klassen
- FP – Datenflussarchitekturen
- Allgemeinere Muster

Außerdem

- Spezialitäten und Tricks, auch orthogonal dazu

Wir lösen mehrmals das gleiche Problem

Verwobene Aspekte

Architektur

Konzepte in Programmiersprachen

Muster für Probleme

Muster für Anwendungen

Programmiertechniken:
z.B. Serialisierung
mit Introspection,
Tag-Listen, Unit-Test,
Fehlerbehandlung

motiviert

ermöglicht/
vereinfacht

Funktionale &
Objektorientierte
Programmierung

Typisierung

Polymorphie
(parametrisch, subtyping,
multiple dispatch etc.)

Metaprogrammierung