Sprachkonzepte

Teil 1: Motivation

Worum geht es im Software-Engineering?

Das Fachgebiet **Software-Engineering** fand beim großen Informatik-Pionier Edsger W. Dijkstra wenig Gnade. Er nannte es *"The Doomed Discipline"*.
Das Kernanliegen sei *"How to program if you cannot."*, ein Widerspruch in sich.
(siehe https://en.wikipedia.org/wiki/Software_engineering#Criticism)

Aber das Software-Engineering wird nicht untergehen, solange es ungelöste Probleme bei der Softwareentwicklung gibt, mit denen es sich befasst, etwa diese:

- Dekompositionsproblem
- Abstraktionsproblem
- Redundanzproblem
- Abhängigkeitsproblem
- Formalisierungsproblem
- problematische Rahmenbedingungen

Software-Engineering: Dekompositionsproblem

Dekomposition (hierarchische Zerlegung von Systemen in Teilsysteme)
ist Grundlage für arbeitsteilige Entwicklung und gute Wartbarkeit von Software

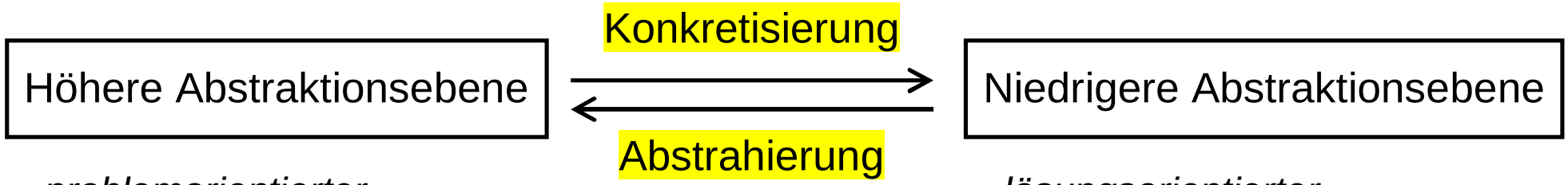
Aspekte für eine Dekomposition:

- Struktur der fachlichen Domäne
  jedes Fachgebiet hat etablierte Zerlegung in Teilgebiete
- Struktur der technischen Plattform
  z.B. Multitier-Architektur bei Web-Anwendungen
- Struktur der Abläufe und Daten, softwaretechnische Prinzipien
  komplexe Funktionalitäten lassen sich in Arbeitsschritte zerlegen
  komplexe Daten lassen sich in Typen bzw. Klassen einteilen
  Information Hiding, Separation of Concerns, Wiederverwendung, ...
- Organisation der beteiligten Entwickler
  Entwickler sind in Firmen, Abteilungen, Teams, Netzwerken, Communities, ... organisiert
  jede Organisationseinheit hat Interessen, Fähigkeiten, Zuständigkeiten, ...

Problem: in jeder Dekomposition dominiert ein Aspekt auf Kosten der anderen

Software-Engineering: Abstraktionsproblem

Softwareentwicklung findet auf unterschiedlichen **Abstraktionsebenen** statt:



- *problemorientierter (mehr "Was", Modellierung)*
- *domänenspezifischer*
- *mehr generalisierend ("unwichtige" Details weglassend, Spezialfälle zusammenfassend)*

ein UML-Klassendiagramm ist abstrakter als Quellcode

Quellcode ist abstrakter als das ausführbare Programm

- *lösungsorientierter (mehr "Wie", Programmierung)*
- *plattformspezifischer*
- *detaillierter, spezialisierter*
- *mehr Redundanz*

eine Klasse ist konkreter als ein Interface

niedrigste Abstraktionsebene ist das ausführbare Programm

Problem: Beschreibungen auf höheren Abstraktionsebenen veralten, weil in späten Projektphasen nur noch auf niedrigen Abstraktionsebenen gearbeitet wird

Software-Engineering: Redundanzproblem

Redundanz bezeichnet allgemein das mehrfache Vorhandensein funktions-, inhalts- oder wesensgleicher Objekte

"Objekte" in diesem Sinne sind bei der Software-Entwicklung z.B. Bezeichner, Klassen, Entwurfsmuster, Codesequenzen, ...

- **Vertikale Redundanz**
  Informationen aus höheren Abstraktionsebenen sind zwangsläufig explizit oder implizit auch in den niedrigeren Abstraktionsebenen enthalten.
  Klassen aus UML-Diagrammen werden explizit im Quellcode wiederholt.
  Anwendungsfälle aus Usecase-Diagrammen sind implizit im Quellcode enthalten.

- **Horizontale Redundanz**
  Informationen wiederholen sich innerhalb einer Abstraktionsebene.
  Klassen müssen programmiert und in Konfigurationsdateien eingetragen werden.
  Informationssysteme: für jeden Datentyp "Anlegen", "Abfragen", "Ändern", "Löschen", ...

Problem: Erhöhter Entwicklungsaufwand und Inkonsistenzen durch Redundanz

Software-Engineering: Abhängigkeitsproblem

Jede Software hat interne und externe Abhängigkeiten:

- **interne Abhängigkeiten** entstehen, wenn Lösungen für Teilprobleme aufeinander aufbauen oder miteinander vermischt sind
  Vermischung von fachlichen und technischen Aspekten, ...
- **externe Abhängigkeiten** entstehen durch Annahmen über den Einsatzkontext und die Entscheidung für eine Ablaufplattform
  Datenaufkommen, Benutzerzahlen, Hardware, Betriebssystem, Standardsoftware, ...

Abhängigkeiten können explizit oder (schlimmer) implizit vorhanden sein:

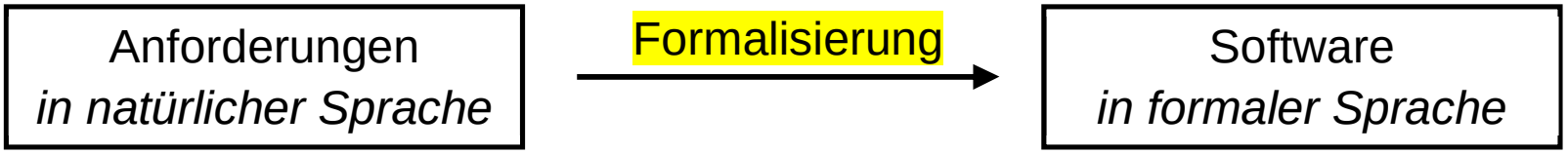
- **explizite Abhängigkeiten** sind programmiersprachlich formuliert und werden automatisch überwacht
  Abhängigkeit zwischen Klassen durch Methodenaufrufe (Überwachung per Compiler), ...
- **implizite Abhängigkeiten** sind stillschweigende Annahmen, deren Einhaltung nicht automatisch überwacht wird
  Magic Numbers, Reihenfolge von Elementen in einer Collection, ...

Problem: Verringerte Flexibilität und Wiederverwendbarkeit durch Abhängigkeiten

Software-Engineering: Formalisierungsproblem

Formalisierung ... bedeutet die Beschreibung eines Phänomens oder die Formulierung einer Theorie in einer formalen Sprache ... *(Wikipedia)*

In der Softwareentwicklung geht es darum, Anforderungen aus der realen Welt mit einer zu erstellenden Software zu erfüllen:



Einige Methoden für eine schrittweise Formalisierung:

- Strukturierung *z.B. standardisierte Gliederung von Texten*
- reduzierte Sprache *z.B. wohl definiertes Fachvokabular*
- Grafik *Skizzen, Diagramme, ...*

Problem: keine 1:1-Übersetzung von natürlicher in formale Sprache möglich, dadurch Verlust oder Verfälschung von Information

Software-Engineering: problematische Rahmenbedingungen

Rahmenbedingungen bei der Softwareentwicklung:

- zeitliche Rahmenbedingungen
  Fertigstellungstermin, vorgesehene Lebensdauer der Software, ...
- finanzielle Rahmenbedingungen
  Entwicklungsbudget, Betriebskosten, ...
- rechtliche Rahmenbedingungen
  Lizenzen, Gewährleistungsansprüche, Datenschutz, ...
- technische Rahmenbedingungen
  einzuhaltende Standards und Normen, vorgegebene Plattformen, ...
- personelle Rahmenbedingungen
  Verfügbarkeit, Qualifikation, ...

Problem: Rahmenbedingungen schränken den Handlungsspielraum ein

Software-Engineering: nichts als Probleme ...

Einige der genannten Probleme bedingen sich gegenseitig:

- jede Dekomposition
  führt zu horizontaler Redundanz bezüglich der nicht dominierenden Aspekte
- zusätzliche Abstraktionsebenen
  führen zu mehr vertikaler Redundanz
- das Vermeiden horizontaler Redundanz
  führt zu mehr Abhängigkeiten und umgekehrt
- horizontale Redundanz mit Konsistenzanforderungen
  führt zu impliziten internen Abhängigkeiten

Es gibt weitere hier nicht genannte Probleme, z.B.:

- beim Projektmanagement
- bei der Anforderungsermittlung
- beim Lösen technischer Detailprobleme

Software-Engineering: Sprachkonzepte

Die genannten Probleme der Softwareentwicklung lassen sich nicht lösen, sondern nur mehr oder weniger gut mit Kompromissen bewältigen.

Bei der Problembewältigung spielen Sprachkonzepte eine wichtige Rolle:

- sprachliche Strukturierungsmittel helfen beim Dekompositions-, Abstraktions-, Redundanz- und Abhängigkeitsproblem

  Module, Klassen, Funktionen, Datentypen, ...

- höhere Programmiersprachen und speziell domänenspezifische Sprachen helfen beim Formalisierungsproblem

  Problemorientierung, Abstraktion von der Plattform, ...

Inhalte der Lehrveranstaltung

Die wichtigsten Sprachkonzepte kennen und beurteilen lernen.

Die Funktionsweise von Compilern und Interpretern verstehen.

Compilerbau-Werkzeuge anwenden können.

Teil 2: Sprachen

Syntax, Semantik, Pragmatik

Teil 3: Programmierparadigmen

imperative / deklarative Programmierung, Anwendungs- / Systemprogrammierung, Scripting, Textgenerierung

Teil 4: Namen

Bindungen, Scopes, Lebensdauern

Teil 5: Typsysteme

Typprüfung, Typinferenz, parametrische Polymorphie

Teil 6: Beispiele