WS 2016/17

Simon Kaiser, 354692
Philipp Hochmann, 356148
Felix Kiunke, 357322
Giacomo Klingen, 356778
Daniel Schleiz, 356092

# Softwaretechnik
Aufgabenblatt 6

---

## Aufgabe 6.1

### a)

**Logische Sicht**: Die logische Sicht beschreibt die Funktionalität des Systems für den Nutzer. Für die Darstellung werden meistens Klassen- und Sequenz-Diagramme verwendet, um sich einerseits das Klassenmoddel klar zu machen und andererseits das Analysemodell weiter zu verfeinern.

**Struktursicht**: In dieser Sicht ist das Augenmerk auf das Softwaremanagement gelegt, das Programm wird von der Sicht des Entwicklers betrachtet. Hier werden meist Komponenten- und Paketdiagramme verwendet, welche sich gut dazu eignen, Subsysteme und dessen Schnittstellen zu verdeutlichen.

**Ablaufsicht**: Diese Sicht beschäftigt sich mit dem Laufzeitverhalten des Programms und ist deswegen insbesondere vom Blickpunkt der Personen, die die Integration des Softwaresystems durchführen, relevant. Hier finden zum beispiel Aktivitätesdiagramme ihre Anwendung zur Darstellung der Prozesse und dessen notwendiger Koordination.

**Physikalische Sicht**: In diesem Abschnitt wird die Systemarchitektur und die Verteilung der Softwarekomponenten in den Fokus gerückt. Diese ist vor allem für Systemarchitekten interessant. In dieser Schicht finden Verteilungsdiagramme Anwendung.

**Szenarien**: in der "+1"-ten Sicht werden alle Sichten vereint und Anwendungsfälle bzw. Use-Cases, sowie mögliche Szenarien betrachtet. Hierbei wird auf die Zusammenarbeit in den Abläufen zwischen den Komponenten und den dazugehörigen Prozessen Wert gelegt. Es bieten sich Use-Case Diagramme zur Darstellung an.

**b)**

- Kein Spaghetti-Code:
  Man sollte, wenn möglich, auf Sprunganweisungen wie "goto" oder "return" in der Mitte von Methoden verzichten, da dies zu einem unübersichtlichen Code führen kann. Dies leigt daran, dass bei solchen Anweisungen oft über weite Distanzen gesprungen wird, wodurch man den Programmfluss nicht sofort ablesen kann. Dies kann unter anderem zur Folge haben, dass man z.B. versehentlich unerreichbaren Code schreibt, was wiederum ungewünschte Nebeneffekte haben kann, falls dieser Code wichtig ist.

- Softwarebibliotheken verwenden
  Um Code für das eigene Problem zu entwickeln hat man meist gewisse Unterprobleme, zum Beispiel das Sortieren eines Arrays. Für sowas existieren meist Bibliotheken, welche diese Unterprobleme schon effizient gelöst haben. Diese zu benutzen spart einerseits Zeit und birgt auch weniger Gefahr für Fehler, die man sonst bei der eigenen Implementierung machen könnte. Natürlich kann es auch sein, dass es genau für den gewünschten Fall keine Bibliotheksmethoden gibt, die das Problem exakt lösen, meist kann man jedoch bestehende Lösungen für das eigene Problem anpassen.

Insgesamt verhelfen Guidelines zu schnellem und gutem Informationsaustausch innerhalb von Teams. Einheitliche Styles machen das Lesen von Code einfacher und erleichtern das Einarbeiten in fremden Code. Außerdem wird der Code so besser wiederverwertbar bzw. ist dadurch besser erweiterbar.

**c)**

Der Begriff der "**hohen Kohäsion**" definiert, dass jede Methode oder Klasse innerhalb einer Komponente eindeutig für eine Aufgabe bestimmt sind, sodass eine klare Trennung innerhalb der Funktionalitäten festzustellen ist. Dies erfordert eine enge Zusammenarbeit der Methoden und Attribute. Dies erleichtert zukünftig die Wartbarkeit in den Komponenten, wobei bestimmte eindeutiger Funktionalitäten ausgetauscht werden können, ohne andere Funktionen zu beeinträchtigen.

Kopplung beschreibt, wie abhängig Komponenten voneinander sind. Bei Softwaresystemen ist meist eine "**lose Kopplung**" erwünscht, das heißt, dass die Komponenten unabhängiger sind und deswegen besser wartbar bze. austauschbar. Beispiele für Kopplungen wären eine Datenkopplung oder Strukturkopplung, was bedeutet, dass Komponenten auf Daten wie zum Beispiel Attribute gegenseitig zugreifen oder es Vererbungsrelationen gibt. Dies

ist bei einer losen Kopplung nicht erwünscht, da unter diesen Abhängigkeiten Erweiterbarkeit leidet. Die Schnittstellenkopplung, also das verwenden von Schnittstellen zwischen Komponenten, ist allerdings in Ordnung, da so eine gewisse Austauschbarkeit gegeben ist.

### Aufgabe 6.2

TODO