Prinzipien und Anwendung des Software Designs anhand von Schichten- und Hexagonaler-Architektur

Simon Thalmaier
Technische Hochschule Ingolstadt
Informatik
Matrikelnummer: 00108692

***Zusammenfassung*——TODO**
***Index Terms*—architecture, ports and adapters, layer, software**

I. Einleitung

Seit Jahrzehnten haben sich Softwaresysteme und ihre Architektur weiterentwickelt. Von den damals weitverbreiteten monolithischen Anwendungen sehen wir aktuell einen Aufschwung an kleinen, modulierten Microservices. Immer mehr Geräte haben eingebaute Software und die Größe bzw. Komplexität von Sourcecode steigt mit jedem Jahr, dadurch auch die verbundenen Entwicklungskosten. Immer mehr wandert der Fokus bei Softwareentwicklung auf Flexibilität, Übersichtlichkeit und Wartbarkeit. Es haben sich daraus verschiedenste Software Architekturen bewährt.
Im Folgenden wird auf bekannte Design-Prinzipien und zwei konkrete Architekturen eingegangen und analysiert.

II. Prinzipien des Software Designs

Prinzipien eines 'guten' Software Designs unterscheiden sich je nach Programmierparadigmen und unterliegender Architektur. Hierbei wird auf die Bekanntesten eingegangen.

A. Die SOLID Prinzipien

Das Akronym, welches von Michael Feathers und Robert C. Martin geprägt wurde, zählt zu dem Fundament eines stabilen Designs. Es beschreibt wünschenswerte Eigenschaften von Komponenten und ihre Beziehungen zueinander. Eine Softwarearchitektur sollte somit den Entwickler dabei unterstützen, diese Prinzipien anzuwenden.

1) Single-Responsibility-Prinzip: Durch diese Richtlinie soll sichergestellt werden, dass die Verantwortlichkeit eines Moduls zu maximal einem Akteur gehört. Konkret darf jedes Modul nur einmal abgeändert werden müssen, unabhängig davon wie viele Anforderungen sich ändern. Bei Verletzung kann eine Anpassung des Codes zu unerwarteten Nebenwirkungen führen. Als Beispiel kann eine Funktion gesehen werden, welche überprüft, ob ein Passwort alle Anforderungen erfüllt. Diese Funktion wird für sowohl Adminaccounts als auch für normale Benutzeraccounts verwendet. Eine neue Anforderung sieht vor, dass Adminaccounts zukünftig eine höhere Mindestlänge besitzt. Eine unaufmerksame Änderung der Funktion hat somit auch eine Auswirkung auf Benutzeraccounts. Dies ist ein Widerspruch des Single-Responsibility-Prinzips. Eine allgemeinere Variante besagt, dass jede Variable, Methode, Klasse usw. genau eine Aufgabe besitzt.

2) Open-Closed-Prinzip: Hierdurch werden zwei gewünschte Aspekte eines Moduls beschrieben. Einerseits sollte ein Modul offen sein für Erweiterungen, andererseits geschlossen gegenüber Veränderung. Dies soll es Entwicklern ermöglichen bereits bestehende Funktionalitäten auszubauen ohne dass der Code, welcher auf diese Funktion basiert, abändern zu müssen. Eine Möglichkeit dieses Prinzip anzuwenden ist die Verwendung eines Interfaces, um Module mit unterschiedlichen Implementierungen

III. Schichtenarchitektur

A. SOLID-Prinzipien in der Schichtenarchitektur

Durch die horizontale Einteilung der Software in Schichten wird eine grobe, natürliche Trennung von Funktionalitäten erzwungen. Somit wird verhindert, dass eine Komponente beispielsweise gleichzeitig Businesslogik und Zugriff auf die Datenbank regelt. Jedoch ist eine vertikale Trennung nicht gegeben und Module können weiterhin verschieden Aufgaben erfüllen, wodurch das *Single-Responsibility-Prinzip* nicht eingehalten ist, ohne die Schichteneinteilung zu verletzten.

Die klare Aufteilung unterstützt den Entwickler abstrakte Schnittstellen zwischen den Schichten zu definieren. Diese Abstraktionen sollten laut dem *Open-Closed-Prinzip* als geschlossen und die konkrete Implementierung der Schichtenfunktion als offen behandelt werden. Weiterhin besagt das *Interface-Segregation-Prinzip*, dass diese Schnittstellen so präzise und klein wie möglich gehalten werden sollen. Bei korrekter Anwendung des Prinzips müssen bei steigender Softwarekomplexität weitere Interfaces beziehungsweise Schichten definiert, wodurch die Übersichtlichkeit des Quelltextes gemindert wird.

Das *Liskovsches Substitutionsprinzip* ist von der unterliegenden Architektur unabhängig und bezieht sich auf die Komposition von Klassen und ihre Relationen zueinander. Dadurch beeinflusst eine Schichtenarchitektur diese Richtlinie nicht.

Literatur

[1] Vaughn Vernon. *Implementing domain-driven design*. Fourth printing. Upper Saddle River, NJ: Addison-Wesley, 2015. ISBN: 9780321834577.