Softwarearchitektur[1]

Was ist eine Software-Architektur

Die Komplexität der notwendigen Systeme nimmt rasant zu. Daher ist auch eine Struktur in Software-Systemen von großer Bedeutung. Software-Architektur ist ein *Spezialgebiet des Software-Engineering*, spielt dabei aber eine wesentliche Rolle. Durch die Strukturierung werden *klare Grenzen in eine Anwendung* gebracht.

Spezialgebiet des Software-Engineering

klare Grenzen in eine Anwendung

Moderne Software-Systeme sind während der Einsatzzeit meist auch *kontinuierlichen Änderungen* unterworfen. Die Software-Architektur beschreibt daher nicht nur die statische Struktur einer Anwendung, sondern es fließen auch *nicht funktionale Anforderungen*, wie Skalierbarkeit, Performanz oder Verfügbarkeit in die Architektur ein. Eine gute Architektur erleichtert zudem auch *zukünftige Änderungen und Weiterentwicklungen der Software*.[2]

kontinuierlichen Änderungen

nicht funktionale Anforderungen

zukünftige Änderungen und Weiterentwicklungen der Software

Schichtenarchitektur (Multitier Architecture)[3]

Die Schichtenarchitektur zählt zu den *beliebtesten Architektur-Stilen* in der Software-Entwicklung. Jede Schicht (*Layer*) nimmt dabei eine klar definierte Rolle ein und bietet darüberliegenden Schichten eine Menge an Diensten an. Wie bei den Architektur-Stilen erwähnt, werden durch den Stil auch Regeln beschrieben, die eingehalten werden müssen. Bei der Schichtenarchitektur sind folgende Punkte zu beachten:

beliebtesten Architektur-Stilen

Layer

- Eine Schicht *verbirgt* sowohl *darunterliegende Schichten* als auch die *interne Komplexität*. *Untere* Schichten konzentrieren sich meistens auf die *Technik*, während *obere* Schichten eher den Fokus auf die *Benutzerschnittstelle* legen.

verbirgt

interne Komplexität

Untere

Technik

obere

Benutzerschnittstelle

- *Top-down-Kommunikation*, d. h., Komponenten der höheren Schicht verwenden Dienste der unteren Schicht und nicht umgekehrt.

Top-down-Kommunikation

- Komponenten *innerhalb* einer Schicht sind von ähnlichem *Abstraktionsgrad* (z. B. Komponenten in der Daten-Schicht konzentrieren sich

innerhalb

Abstraktionsgrad

[1] *Softwaresysteme: Präsenztag 4: vSoftwarearchitektur*, Seite 9.

[2] Schatten, *Best Practice Software-Engineering*, Seite 200.

[3] *Softwaresysteme: Präsenztag 4: vSoftwarearchitektur*, Seite 10.

um Persistenz).

- Das Design einer Schicht soll eine *lose Kopplung* zu anderen Schichten ermöglichen.

lose Kopplung

- Die *Kommunikation* zwischen den Schichten erfolgt über klar *definierte Schnittstellen und Protokolle*.[4]

Kommunikation

definierte Schnittstellen und Protokolle

Schichtenbildung[5]

Eine Schicht wird oft als Subsystem entworfen, das sich wiederum aus einer Menge von Teilsystemen (z. B. Services, Geschäftsprozesse etc.) zusammensetzt. Die Schichtenarchitektur ist in ihrer Verwendung und Gestaltung sehr flexibel. Man kann unterschiedliche Formen und Strategien der Schichtenbildung unterscheiden.

Horizontale Schichtenbildung:

Die einfachste Form ist die horizontale Schichtenbildung. Dabei werden die Schichten horizontal gestapelt und jede horizontale Schicht hat eine Aufgabe, wie z. B. den Datenzugriff. Bei komplexeren Systemen kann eine Schicht in weitere Teile geteilt (partitioniert) werden, wobei jede Partition einen bestimmten Business-Fokus hat. Diese Partitionen werden oft in Form von Subsystemen umgesetzt.

Vertikale Schichtenbildung:

Querschnitts-Funktionalitäten, die von allen Schichten in einem System benötigt werden, können in vertikalen Schichten abgebildet werden. Dabei hat eine vertikale Schicht vollen Zugriff auf alle angrenzenden horizontalen und umgekehrt.

Vorteile Schichtenarchitektur[6]

Zusammengefasst bietet die Schichtenarchitektur folgende Vorteile:

- *Einfaches*, effizientes und verständliches *Architekturmuster*.

Einfaches Architekturmuster

[4] Schatten, *Best Practice Software-Engineering*, Seite 211.
[5] *Softwaresysteme: Präsenztag 4: vSoftwarearchitektur*, Seite 11.
[6] *Softwaresysteme: Präsenztag 4: vSoftwarearchitektur*, Seite 12.

- Es besteht eine *saubere Trennung* der einzelnen Schichten. Dadurch kann auch ein verteiltes Arbeiten in Team durchgeführt werden. saubere Trennung

- Schichten sind *austauschbar* und können auch auf verschiedenen Infrastrukturen betrieben werden. Beispielsweise können die Datenbank, die Services und die Komponenten für den Client auf unterschiedlichen Rechnern betrieben werden. austauschbar

- *Minimale Abhängigkeit* zwischen den Schichten. Die Kommunikationswege der Schichten sind durch Schnittstellen klar definiert. Minimale Abhängigkeit

- Die *Wartung und Erweiterbarkeit* ist durch die klare Trennung der Schichten einfach.[7] Wartung und Erweiterbarkeit

Nachteile Schichtenarchitektur[8]

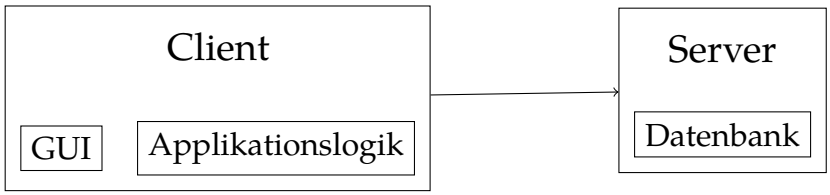
- Wird die Top-down-Kommunikation bei einer Schichtenarchitektur streng eingehalten, kann eine triviale Operationen in einer Schicht oft nur aus einem „Durchreichen“ in die nächste Schicht bestehen. Das kann dazu führen, dass *aus relativ einfachen Operationen komplexe Funktionen* werden. aus relativ einfachen Operationen komplexe Funktionen

- Mehr Schichten bedeutet *mehr Komplexität und Implementierungsaufwand*. Weniger Schichten bedeutet stärkere Kopplung und weniger Flexibilität. Hier muss abgewogen werden, wie viele Schichten wirklich notwendig sind, um die Anforderungen des Systems zu erfüllen. mehr Komplexität und Implementierungsaufwand

- Änderungen an den Schnittstellen der unteren Schichten können Änderungen in den oberen Schichten und damit einen *erhöhten Aufwand bei der Anpassung* zur Folge haben.[9] erhöhten Aufwand bei der Anpassung

[7] Schatten, *Best Practice Software-Engineering*, Seite 212.
[8] *Softwaresysteme: Präsenztag 4: vSoftwarearchitektur*, Seite 13.
[9] Schatten, *Best Practice Software-Engineering*, Seite 213.

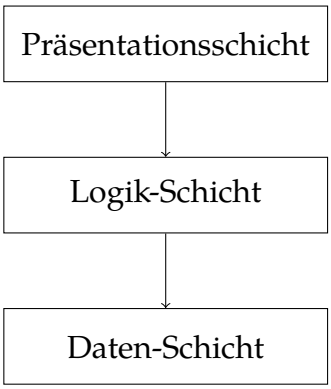
2-Schichtenarchitektur (2-Tier Architecture)[10]



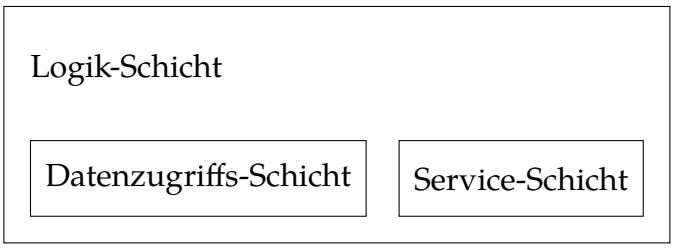
Die 2-Schichtenarchitektur ist bei verteilten Systemen auch als *Client/Server-Architektur* bekannt. Der *Client* ist dann in der Regel ein Programm, das sowohl die *GUI* als auch die gesamte *Applikationslogik* beinhaltet. Der *Server* besteht meist aus einer *relationalen Datenbank*. Für eine *Stammdatenverwaltung* oder ähnliche Systeme war *diese Architektur unter Umständen ausreichend* und auch effizient. Durch die *wachsende Komplexität* in den einzelnen Domänen und die Anforderungen an verteilte Systeme ist es jedoch meist notwendig und auch sinnvoll, Teile des Applikationscodes *auf mehrere Schichten aufzuteilen*. Einzelne Schichten können auch auf einen Server ausgelagert werden.[11]

Client/Server-Architektur
Client
GUI
Applikationslogik
Server
relationalen Datenbank
Stammdatenverwaltung
diese Architektur unter Umständen ausreichend
wachsende Komplexität
auf mehrere Schichten aufzuteilen

3-Schichtenarchitektur (3-Tier Architecture)[12]



[10] *Softwaresysteme: Präsenztag 4: vSoftwarearchitektur*, Seite 15.
[11] Schatten, *Best Practice Software-Engineering*, Seite 213.
[12] Schatten, *Best Practice Software-Engineering*, Seite 214.



Die 3-Schichtenarchitektur ist der bekannteste und beliebteste Stil, um ein System zu strukturieren. Der *Wunsch den Code der Geschäftslogik vom Client zu lösen*, führt dazu, dass eine neue Schicht (*Logik- oder Business-Schicht*) eingeführt wird, die diese Logik beinhaltet. In der Praxis gibt es mehrere Variationen, wie die Logik-Schicht auf den Client und den Server aufgeteilt ist.

Wunsch den Code der Geschäftslogik vom Client zu lösen

Logik- oder Business-Schicht

Auf der untersten Ebene der 3-Schichtenarchitektur befindet sich die *Daten-Schicht*, die eine Menge an Datenquellen zur Verfügung stellt. Typische Objekte in dieser Schicht sind relationale oder objektorientierte Datenbanken. In diesen Objekten werden die Daten von den Applikationen persistiert.

Daten-Schicht

Aufbauend auf der Daten-Schicht befindet sich nun die neue *Logik-Schicht*, welche die Kernfunktionalität des Systems beinhaltet. Hier erfolgt die tatsächliche Abbildung der Problemdomäne in Form von Logik- und Serviceobjekten. Typische Funktionen, die in der Business-Schicht angeboten werden, sind die Verarbeitung und die Aufbereitung von Daten für die Präsentationsschicht.

Logik-Schicht

Die Logik-Schicht wird bei *komplexen Systemen* oft noch in eine *Datenzugriffs-Schicht* (*Data Access Layer*) und eine *Service-Schicht* unterteilt.

komplexen Systemen

Datenzugriffs-Schicht

Data Access Layer

Service-Schicht

Die Datenzugriffs-Schicht besteht aus einer Menge von *Datenzugriffsobjekten*, sogenannten *Data Access Objects* (*DAOs*), die den Zugriff auf die darunterliegende Daten-Schicht abstrahieren. Diese DAOs werden von Servicekomponenten der Service-Schicht in Anspruch genommen, um Daten zu laden und zu speichern.

Datenzugriffsobjekten

Data Access Objects (DAOs)

Jegliche Form der Interaktion mit dem Benutzer erfolgt in der *Präsentationsschicht*. Komponenten in dieser Schicht beinhalten auch eine gewisse Logik für die Verarbeitung von Ereignissen, etwa die Logik, die auf das Betätigen eines Buttons oder eine Tastenkombination reagiert. Die Logik sollte jedoch hauptsächlich auf GUI-Komponenten bezogen sein und nicht darauf, Geschäftslogik zu implementieren.

Präsentationsschicht

5-Schichtenarchitektur (5-Tier Architecture)[13]

Durch die Trennung der Logik-Schicht in die Datenzugriffs-Schicht und Service-Schicht wurde eine weitere Entkoppelung von Services zu den Daten erzielt. Jedoch sind die Implementierungen der DAOs sehr stark mit der Struktur der zu verwendeten Datenquelle verknüpft. Eine Änderung der Struktur in der Datenquelle (etwa das Hinzufügen einer neuen Spalte in einer Tabelle) bewirkt auch eine Änderung der DAO-Implementierung. Um Domänenobjekte mit externen Datenquellen in Verbindung zu setzen, die anschließend über DAOs geladen und gespeichert werden können, wird ein Framework, das Datenmapping durchführt, verwendet. Dieses Datenmapping löst Struktur-Information weitgehend aus der DAO-Implementierung. In Abschnitt 9.4.8 wird beschrieben, wie ein objektrelationales Mapping (O/R Mapping) funktioniert.[14]

Die nächste neue Schicht ist die sogenannte Prozess-Schicht.[15] Diese liegt zwischen der Logik-Schicht und der Präsentationsschicht. Ziel ist es, die Services so atomar wie möglich und damit wiederverwendbar zu implementieren. Die Prozess-Schicht kümmert sich dann um Abläufe (Prozesse) höherer Granularität und wird daher häufig unter Verwendung einer Prozess-Engine 7 umgesetzt. Die atomaren Services aus der Logik-Schicht werden (häufig mithilfe grafischer Tools) zu einem Prozess kombiniert, der dann in einer eigenen Prozess-Sprache (z. B. BPEL) vorliegt. Da sich die Ablauflogik nicht mehr im Code befindet, können Prozesse leichter verändert und an neue Bedürfnisse angepasst werden.[16]

Literatur

[1] Alexander Schatten. *Best Practice Software-Engineering. Eine praxiserprobte Zusammenstellung von komponentenorientierten Konzepten, Methoden und Werkzeugen*. 2010.

[2] *Softwaresysteme: Präsenztag 4: vSoftwarearchitektur*. `https://www.studon.fau.de/file2785768_download.html`.

[13] *Softwaresysteme: Präsenztag 4: vSoftwarearchitektur*, Seite 16.
[14] Schatten, *Best Practice Software-Engineering*, Seite 214-215.
[15] *Softwaresysteme: Präsenztag 4: vSoftwarearchitektur*, Seite 17.
[16] Schatten, *Best Practice Software-Engineering*, Seite 214-215.