Block 3: Entwurf, Persistenz und Usability (wie wird es technisch gebaut?) - Arbeitsblatt und Skriptum (Entwurf)

Website: Moodle-Service der Universität Wien
Kurs: 2020W Software Engineering 1
Buch: Block 3: Entwurf, Persistenz und Usability (wie wird es technisch gebaut?) - Arbeitsblatt und Skriptum (Entwurf)

Gedruckt von: Srdanovic Nemanja
Datum: Dienstag, 17. November 2020, 20:48

Inhaltsverzeichnis

1. Allgemeine Hinweise

Unterstützung: Es wird für jedes Arbeitsblatt ein individuelles **Skriptum** (Skriptum Block 3: Entwurf) angeboten. Dieses liefert Ihnen die Hintergründe um das Arbeitsblatt zu bearbeiten. Zusätzlich helfen kann ihnen die jeweils empfohlene Literatur sowie die Streams aus Übung und Vorlesung.

Überlegen Sie sich bitte, nach Bearbeitung dieses Arbeitsblattes, je eine kurze Antwort auf die folgenden Fragen. Ihre Antworten werden herangezogen, um die Arbeitsblätter und Vorlesungsinhalte für folgende Semester zu adaptieren.

- Sind bei Ihnen Fragen aufgekommen, welche den Foliensatz des zugehörigen Vorlesungsblocks betreffen?
- Welche Fragen und Bereiche sind Ihnen schwergefallen oder haben Sie nicht verstanden?
- Welche Inhalte fanden Sie am interessantesten oder nützlichsten? Gibt es eine Verbindung zu Inhalten die Sie bereits früher im Studium (und/oder Ihrem Beruf) kennen gelernt haben?
- Wie viel Zeit haben Sie zum Einlesen in den Stoff und zur Bearbeitung der Aufgaben aufgewendet?

2. Einstieg

Im auf den Entwurf fokussierten Foliensatz für Block 3 wird kurz ein historischer Überblick über das Thema Softwareentwurf gegeben. Ein für diesen Überblick relevantes Sprichwort ist, dass Geschichte die Angewohnheit hat sich zu wiederholen. Denken Sie dies trifft hier auch zu? Wenn ja, in welchen Bereichen?

3. Software Entwurf und der Mensch

Was sind die Ziele eines Softwareentwurfs? Warum könnte es dazu kommen, dass absichtlich oder unabsichtlich während des Entwurfs einer Software von diesen Zielen abgewichen wird?

4. Grundlagen: Entwurf

Im Rahmen der Vorlesung aber auch des zugehörigen Übungstutorials wurden generell Wege und Vorgehensmodelle vorgestellt und besprochen, um Softwarearchitekturen zu entwerfen. Diese werden in den folgenden Unterkapiteln noch einmal aufgegriffen und mit praxisnahen Beispielen näher betrachtet.

4.1. Granularitätsebenen

Der Entwurf jeder Software spielt sich auf verschiedenen Granularitätsebenen ab. Überlegen Sie sich welche Unterteilungen Sie durchführen würden? Wie spiegeln sich diese Ebenen im Übungsprojekt von Software Engineering 1 wieder? Welche Ebenen existieren dort? Welche dieser Ebenen werden Sie in einem Softwareprojekt am ehesten selbst verantworten müssen?

4.2. Klassische Vorgehensmodelle

Ein typisches Vorgehensmodell eines Studierenden zur Umsetzung eines Softwareprojektes ist es in dem Moment über die Implementierung nachzudenken in dem die IDE startet und die ersten Klassen und Methoden erstellt werden. Überlegen Sie was passieren wird, wenn Sie solch ein Vorgehen in einem großen Projekt umsetzen.

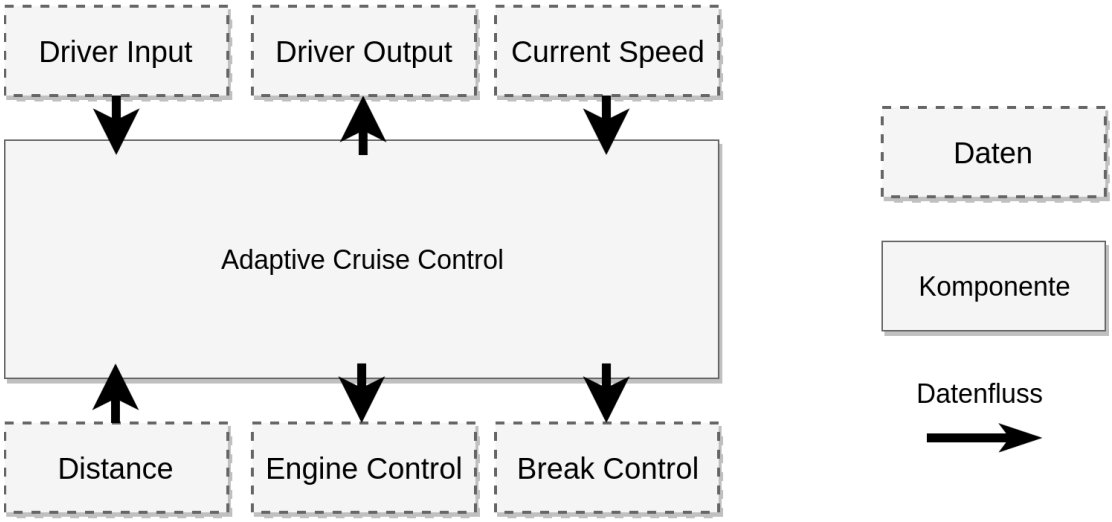
Exkurs: Im Folgenden werden typische Schritte angeführt, die während der Entwicklung einer Software durchlaufen werden sollten bzw. müssen. In welcher Reihenfolge sollten diese durchlaufen werden? Und in welcher Reihenfolge durchlaufen Sie selbst diese normalerweise? Welche dieser Schritte sind direkt von den an ein Projektergebnis gestellten Anforderungen abhängig?

1. Komponentenentwurf
2. Schnittstellenentwurf
3. Algorithmenentwurf
4. Entwurf der Datenstrukturen
5. Abstrakte Softwarespezifikation
6. Systemarchitekturentwurf

4.3. Divide and Conquer & Dekomposition und Modularisierung

Unter "Divide and Conquer" (oder auch "Dekomposition und Modularisierung") wird typischerweise ein Vorgehensmodell verstanden bei dem komplexe und schwer zu überblickende Probleme schrittweise in Teilprobleme zerlegt werden. Dieser Zerlegungsprozess wird so lange durchgeführt bis eine Vielzahl kleiner handlicher Probleme entstehen, die jedes für sich leicht zu verstehen, lösen und abzuschätzen sind. Vergleichbares finden Sie auch immer wieder in Softwareprojekten, indem komplexe Projekte schrittweise in Systeme, Module, Klassen und Methoden unterteilt werden.

Aufgabe: Im Folgenden ist eine schematische Darstellung einer Komponente zur adaptiven Fahrgeschwindigkeitskontrolle gegeben. Wenden Sie auf diese Darstellung die "Divide and Conquer" Strategie an und unterteilen Sie das Problem in mehrere miteinander kommunizierende Komponenten. Welchen Aufbau erhalten Sie hierbei?

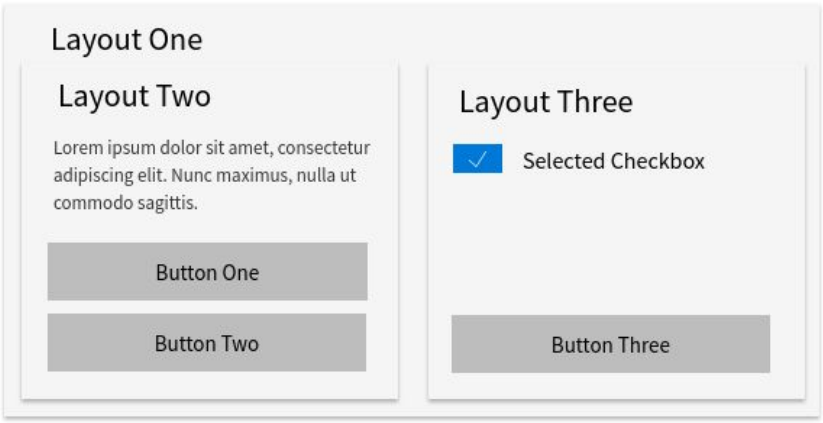


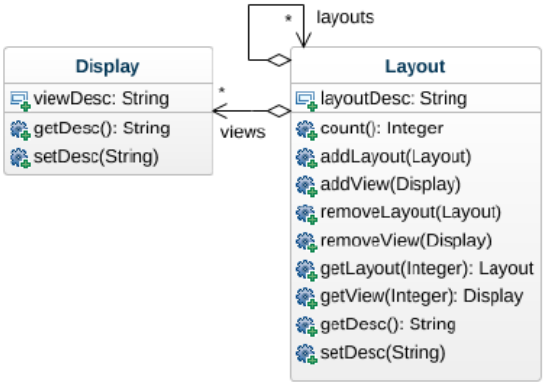
5. Objektorientiertes Design

In den Vorlesungsunterlagen werden eine Reihe von Designprinzipen für den objektorientierten Entwurf vorgestellt. Sich diesen Prinzipien bewusst zu werden bzw. diese zu berücksichtigen führt typischerweise zu besseren Softwareentwürfen die einfacher (schneller, leichter verständlich & weniger Zeilen) umzusetzen aber auch zu erweitern sind. Im Folgenden wird daher jeder Aspekt mit einem Praxisbeispiel näher betrachtet. Versuchen Sie jeweils einen Lösungsvorschlag zu erarbeiten, welcher besonders das jeweils relevante Designprinzip berücksichtigt.

5.1. Entwurf: Abstraktion

Abstraktion wird nicht nur während der Softwareentwicklung verwendet, sondern auch von Ihnen (unbewusst) im täglichen Leben. Überlegen Sie sich Beispiele dafür wo dies der Fall ist. Versuchen wir nun Abstraktion auch im Rahmen der Softwareentwicklung anzuwenden. Analysieren Sie hierzu folgendes Klassendiagramm. Dieses soll es erlauben, grafische Oberflächen während der Laufzeit dynamisch zusammenzufügen und zu erstellen. Hierbei wird einerseits zwischen Klassen unterschieden, welche die eigentliche grafische Visualisierung erzeugen (Knöpfen, Checkboxen, Textfeldern etc. - kurz DisplayView), und Klassen, welche Visualisierungen für verschiedene Inhalte anordnen (Vertikal Layout, Horizontal Layout - kurz LayoutView). Beispielsweise, um folgendes Layout abzubilden:





Aufgabe: Analysieren Sie das gegebene Klassendiagramm, welches einen suboptimalen Lösungsvorschlag zeigt. Welche Probleme treten hier auf und wie könnten diese durch Abstraktion gelöst werden?

5.2. Entwurf: Kapselung

Kapselung wird als eines der fundamentalsten Konzepte, um komplexe Systeme zu entwerfen und umzusetzen, angesehen.

Fragen: Warum ist das so? Können Sie mit eigenen Worten beschreiben was Kapselung "ist"? Warum setzen Ihrer Ansicht nach Softwareentwickler auf Kapselung?

Aufgabe: Im Folgenden werden zwei Praxisbeispiele in Java und C gegeben bei welchen leider nicht auf korrekte Kapselung geachtet wurde. Können Sie erkennen wo das Problem liegt? Wie würden Sie dieses beheben?

Erstes Beispiel:

```
1. public class Person {
2.   public Mouth mouth;
3. }
4.
5. //Exemplary access on the mouth variable
6. person.mouth.contents.add(lotsOfAlcohol);
```

Zweites Beispiel, dieses verwendet einen abstrakten Typ namens List:

```
1. int sum(List* list) {
2.   int s = 0;
3.   int length = length(list);
4.   for(int i = 0; i < length; ++i) {
5.     s = s + ((int*)list)[i];
6.   }
7.   return s;
8. }
```

Zusatzfrage: Hängen Abstraktion und Kapselung zusammen? Wenn ja wie? Denken Sie vor allem daran welche Stakeholder (eines Codestücks) jeweils von dem einen bzw. anderen Aspekt profitiern.

5.3. Entwurf: Kopplung und Kohäsion

Software wird heute typischerweise nicht mehr in der Form eines großen monolithischen Blocks umgesetzt (negatives Extrembeispiel: alle Logik, Methoden, Datenstrukturen, befinden sich in einer Klasse). Stattdessen erfolgt eine feingranulare Unterteilung in Klassen, Methoden, Module, Algorithmen, etc. Manche Unterteilungen sind jedoch objektiv "besser" wie andere und hierbei sind die Aspekte Kopplung und Kohäsion eine der wichtigsten Designprinzipien, um die Qualität eines Softwaredesigns zu messen. Hierbei kann man dieses Prinzip auf Klassen, Methoden, Module aber auch ganze Softwarelandschaften anwenden.
Können Sie bezüglich Kopplung und Kohäsion folgende Fragestellungen mit eigenen Worten beantworten?

- **Was:** Beschreiben Sie mit eigenen Worten was Kopplung und Kohäsion ist? Was will man hinsichtlich dieser beiden Begriffe erreichen? Sind diese beiden Aspekte immer voneinander abhängig?
- **Messbarkeit:** Warum kann man mittels Kopplung und Kohäsion die Qualität eines Designs beurteilen? Kann man diese Qualität denn auch mathematisch berechnen? Können unterschiedliche Lösungen eine unterschiedliche "gute" bzw. bzw. "schlechte" Kopplung aufweisen? Welche Unterschiede würden Ihnen einfallen?
- **Messbarkeit, Beispiel für Kohäsion:** Welches der beiden folgenden Codebeispiele weist eine höhere Kohäsion auf und warum? Wie würden Sie die Kohäsion berechnen (vgl., LCOM41)? Welche Hinweise gibt Ihnen als Entwickler die Information, dass eine niedrige Kohäsion vorliegt?

Beispiel 1:

```
1. public class NumberManipulator {
2.   private int number = 0;
3.
4.   public int getNumberValue() {
5.     return number;
6.   }
7.
8.   public void addOne() {
9.     number ++;
10.  }
11.
12.  public void subtractOne() {
13.    number --;
14.  }
15. }
```

Beispiel 2:

```
1. public class NumberManipulator {
2.   private int firstNumber = 0;
3.   private int secondNumber = 0;
4.   private int thirdNumber = 0;
5.
6.   public void incrementFirst() {
7.     firstNumber++;
8.   }
9.
10.  public void incrementSecond() {
11.    secondNumber ++;
12.  }
13.
14.  public void incrementThird() {
15.    thirdNumber++;
16.  }
17. }
```

Vorteile: Was sind die Vorteile von hoher (starker) Kohäsion und niedriger (schwacher) Kopplung? Nehmen wir an, dass Sie wider besseren Wissens eine Lösung erstellen, welche eine hohe Kopplung aufweist. Wann und welche Probleme entstehen Ihnen dadurch?

Optimieren: Wie kann man Kopplung und Kohäsion optimieren?

Anwendbarkeit: Kopplung kann auf verschiedenen Granularitätsebenen auftreten. Analysieren Sie folgenden Source Code. Welche Ebenen können Sie dabei entdecken? Überlegen Sie weiter wie man erkennen könnte, welche Methoden in den verschiedenen Klassen eine niedrige oder hohe Kohäsion aufweisen? Wann während dem Entwurf einer Software lässt sich dieses Prinzip anwenden?

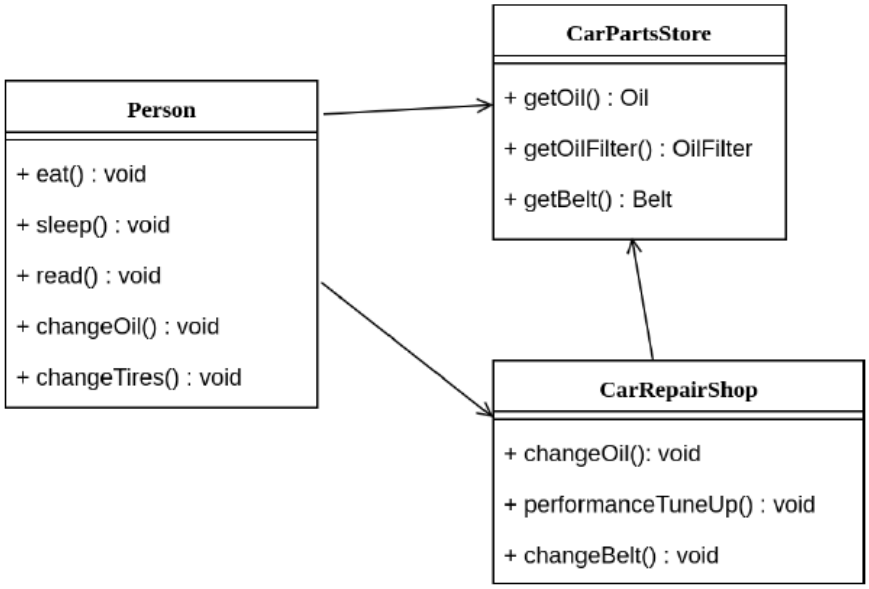
Zusatzaufgabe: Ziehen wir ein weiteres Mal exemplarisch das unten angeführte Codebeispiel heran. Wie könne man auf Ebene der Klassen die Kohäsion messen bzw. beurteilen? Welche Hinweise liefert mir niedrige Kohäsion in diesem Fall?

```
1. package P1;
2.
3. public class C2 {
4.
5.   public static void m4() {
6.   }
7.
8.   public static void m5() {
9.     P2.C3.m6();
10.  }
11. }
12.

13. public class C1 {
14.
15.   public static void m1() {
16.     m2();
17.   }
18.
19.   public static void m2() {
20.   }
21.
22.   public static void m3() {
23.     C2.m4();
24.   }
25. }
26.
27. package P2;
28.
29. public class C3 {
30.
31.   public static void m6() {
32.   }
33. }
```

Weitere Zusatzaufgabe: Im Folgenden ist ein kleines UML Klassendiagramm gegeben. Dieses konzentriert sich auf drei Klassen, die abbilden, dass manche Personen Ihr Auto selbst warten und es nur dann in eine Werkstatt bringen, wenn Sie selbst eine Reparatur nicht durchführen können oder wollen. Wie sieht es hier bezüglich der Aspekte Kopplung

und Kohäsion aus? Welche Vorteile/Nachteile hat diese Lösung und wie könnte man die Nachteile beheben?



5.4. Entwurf: Open-Close-Prinzip

Das Open-Close-Prinzip ist vor allem für längerfristige Projekte relevant die sich über Jahre weiterentwickeln und immer wieder mit neuen Funktionen erweitert werden. Ursprünglich wurde dieses Designprinzip von Bertrang Meyer beschrieben als *"software entities (classes, modules, functions, etc.) should be open for extension, but closed for modification"*. Diese Beschreibung ist sehr abstrakt, können Sie diese mit eigenen Worten ausformulieren, sodass klarer wird was mit dieser Aussage gemeint ist? Welche Aspekte typisierter Sprachen, denken Sie, können verwendet werden, um das Open-Close-Prinzip umzusetzen?

Zusatzaufgabe: Im Folgenden ist ein kleines Codebeispiel gegeben in dem das Open-Close-Prinzip verletzt wird. Identifizieren Sie wodurch die Verletzung des zuvor genannten Prinzips hervorgerufen wird. Welche Maßnahmen sind notwendig, um diese Prinzipienverletzung zu beheben.

```
1. public double Area (object[] shapes) {
2.   double area = 0;
3.   foreach(var shape in shapes) {
4.     if(shape is Rectangle) {
5.       Rectangle rectangle = (Rectangle)shape;
6.       area += rectangle.Width * rectangle.Height;
7.     }
8.     else {
9.       Circle circle = (Circle)shape;
10.      area += circle.Radius * circle.Radius * Math.PI;
11.    }
12.  }
13.
14.  return area;
15. }
```

5.5. Entwurf: Liskovsches Substitutionsprinzip

Das Liskovsche Substitutionsprinzip ist ein Designprinzip aus der sogenannten SOLID Gruppe. SOLID ist eine Zusammenfassung von fünf oft angewendeten Designprinzipien die, unter anderem, auch in den Übungsfolien zum Architekturtutorial kurz vorgestellt werden. Das Liskovsche Substitutionsprinzip besagt: *"Wenn eine Funktion oder Methode als Aufrufparameter eine gewissen (Super-)Klasse erwartet dann soll immer derselbe Aufruf auch mit einer Subklasse möglich sein."*

Warum: Warum ist dieser Aspekt wichtig und relevant? Was passiert, wenn dagegen verstoßen wird?

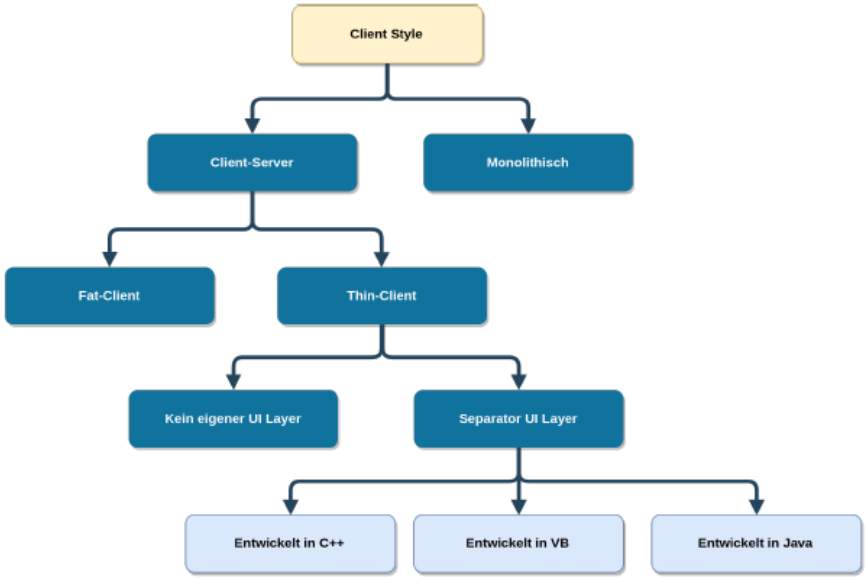
Erstes Praxisbeispiel: Fällt Ihnen ein Beispiel in Java ein bei dem gegen dieses Prinzip verstoßen wird (Tipp: Sehen Sie sich typische Datenstrukturen durch)? Wie ließe sich dieser Verstoß beheben?

Zweites Praxisbeispiel: Sehen Sie sich außerdem die beiden Java Klassen `java.sql.Date` und `java.util.Date` an. Was fällt Ihnen auf?

Drittes Praxisbeispiel: Ein Standardbeispiel zur Erläuterung dieses Prinzips ist die Ableitung einer Kreisklasse von einer Ellipsenklasse. Warum verletzt diese Ableitung das Liskovsche Substitutionsprinzip und welchen Effekt hat dies?

6. Exkurs: Softwareentwurf, Entscheidungsräume

Oft stellt sich die Frage welche Architektur nun genau gewählt werden soll da es hier ja zahlreiche Möglichkeiten gibt und auch eine Vielzahl von Anforderungen berücksichtigt werden müssen wie: Flexibilität, Technologie, Budget, Leistungsfähigkeit, Netzwerkfähigkeit, etc.
Nehmen Sie an, dass sich basierend auf dem Ihnen bekannten Wissen über Architekturen folgender beispielhafter Entscheidungsraum ergibt. Der Hauptfokus der Architektur wäre hierbei der Aspekt "Flexibilität". Welche Entscheidungen wären dadurch nicht mehr länger valide?



7. Layered-Architecture

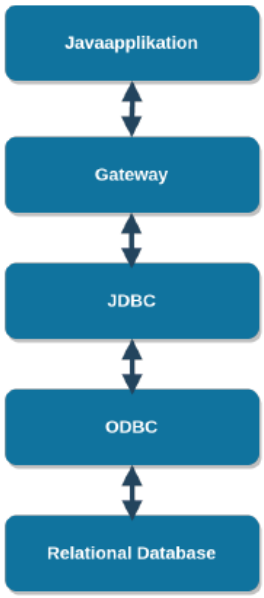
Das Layered-Architecture Pattern ist eines der wichtigsten Architekturpattern und vereint auch einige der zuvor angesprochenen Designprinzipien in sich. Entsprechend behandelt dieses Arbeitsblatt das Layered-Architecture Pattern auf verschiedene Arten. Versuchen Sie die folgenden Fragen und Aufgabenstellungen zu lösen.

Was: Beschreiben Sie mit eigenen Worten was das Layered-Architecture Pattern ist und wie es "funktioniert". In welchen Szenarien und Granularitäten lässt es sich anwenden?

Designprinzipien: Dieses Pattern kombiniert eine Vielzahl von Designprinzipien die zuvor vorgestellt worden sind. Welche sind das? Warum haben die Erfinder des Layered-Architecture Pattern die jeweiligen Prinzipien eingesetzt?

Beispiele: Zuvor wurde erwähnt, dass das Layered-Architecture Pattern sehr häufig eingesetzt wird. Welche weit verbreitete Software oder welches weit verbreitete System ist Ihnen bekannt, dass dieses Pattern einsetzt?

Verstehen: Im Folgenden ist exemplarisch ein kleines Programm dargestellt, dass eine Layered-Architecture verwendet. Nehmen Sie an, dass die Klassen vor allem dazu dienen über abstrahierte SQL Statements auf eine relationale Datenbank zuzugreifen. Beispielsweise umfasst der Gateway Layer Klassen um Tabellen und Zeilen in diesen Tabellen abzubilden. Diskutieren Sie warum diese Architektur die Flexibilität erhöht? Auf welche Art von erwarteten Änderungen ist diese Architektur ausgelegt? Welche Schwächen hat diese Architektur?



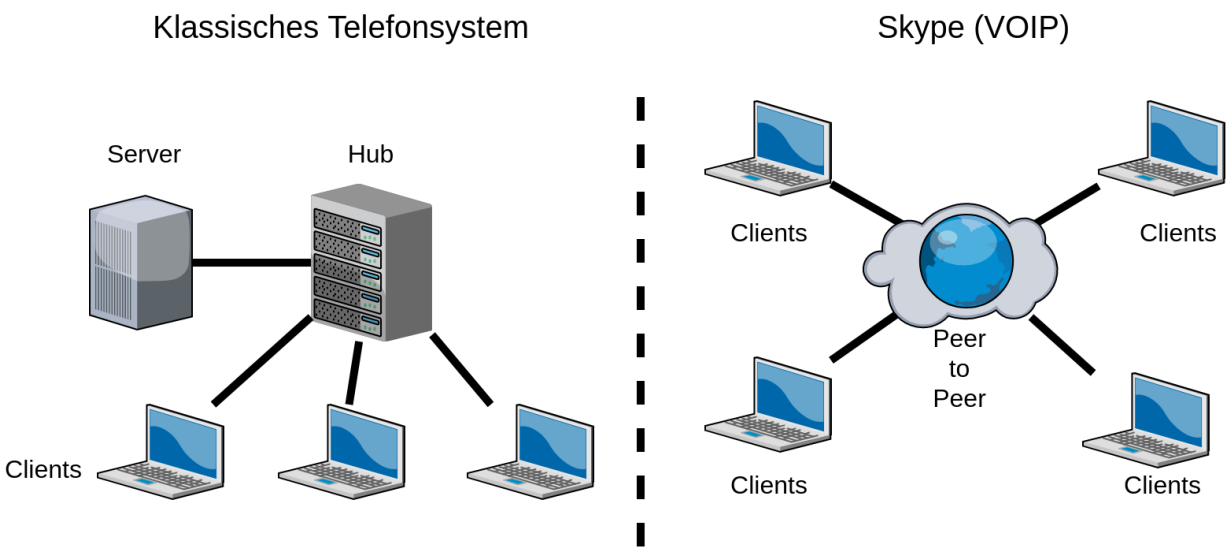
Entwerfen: Versuchen Sie nun das Layered-Architecture Pattern einzusetzen, um eine einfache Aufgabenstellung zu lösen. Nehmen Sie an, dass Sie eine Software entwickeln sollen die es erlaubt über eine grafische Oberfläche auf die Umsätze und Lagerbestände verschiedener Warenhäuser zuzugreifen. Die notwendigen Daten werden in unterschiedlichen Systemen vorgehalten und müssen im Rahmen dieser Aufgabenstellung aggregiert und visualisiert werden. Entwerfen Sie grob eine Layered-Architecture für diese Aufgabenstellung. Welche Layer würden Sie definieren? Wie würden diese miteinander kommunizieren und welche Funktionen wären den jeweiligen Layern von Ihnen zugedacht worden?

8. Anforderungen und Architekturen: Teil 1

Unterschiedliche Architekturen können, obwohl die selben funktionalen Anforderungen unterstützt werden, deutliche Unterschiede bei den nichtfunktionalen Anforderungen aufweisen.

Aufgabe: Analysieren Sie die beiden folgenden Architekturen, die das klassische kabelgebundene Telefonsystem sowie das von Skype (VOIP) realisierte Telefonsystem abbilden. Welche Schwächen und Stärken haben beide Architekturen? Wie Verhalten sich beide Systeme hinsichtlich der Aspekte: Skalierbarkeit, Zuverlässigkeit, Erweiterbarkeit (neue Funktionen), und der Toleranz gegenüber Stromausfall.

Zusatzaufgabe: Für welchen Ansatz würden Sie sich entscheiden, wenn Sie ein Telefonsystem für einen Rettungsdienst erstellen müssen? Begründen Sie Ihre Entscheidung.



9. Exkurs: Anforderungen und Architekturen: Teil 2

Nehmen Sie folgendes real aufgetretene Szenario an: Sie arbeiten bei einem Hosting Provider der unter anderem E-Mail Server an zahlreiche Kunden vermietet. Sobald bei einem der Kunden Probleme auftreten wird der Kundendienst von dem betroffenen Kunden kontaktiert. Der Kundendienst analysiert (durchsucht) anschließend die Logfiles aller relevanter E-Mailserver, um die Ursache der eingemeldeten Probleme zu identifizieren. Ihre Aufgabe ist es nun herauszufinden wie die Grobarchitektur einer Softwarelösung, welche den Kundendienst bei dieser Tätigkeit unterstützt, optimalerweise gestaltet sein sollte.

Aufgabe: Identifizieren Sie verschiedene Architekturen, welche die gegebene Aufgabe auf unterschiedliche Arten lösen. Welche Vor- und Nachteile weisen die unterschiedlichen Architekturen hinsichtlich der Aspekte: Aktualität der Log Daten, Skalierbarkeit und Usability auf.

Beachten Sie hierbei folgende Hinweise zur Komplexität des Problems:
- Sie müssen hunderte von Servern verwalten.
- Es fallen jeden Tag mehrere Gigabyte an Logdateien an.
- Das Durchsuchen und Zusammentragen dieser Logfiles belastet das Netzwerk derzeit stark und dauert lange.

Hinweise zur Lösungsfindung:
- Wie werden die Logfiles gespeichert? Zentral vs. dezentral?
- Wie werden die Logfiles durchsucht? Kann dieser Prozess verteilt werden?
- Können Berechnungen vorab durchgeführt werden, um bei neuen Suchanfragen Zeit zu sparen?

10. Service-Orientierte-Architekturen (SOA)

Moderne Lösungen werden oft als Service-Orientierte-Architekturen entworfen, da diese Architekturform die Flexibilität und Modularisierbarkeit der entstehenden Lösung erhöht. Darüber hinaus lassen sich Service-Orientierte-Architekturen auch hervorragend in modernen Cloud Infrastrukturen integrieren.

Was: Können Sie mit eigenen Worten beschreiben was eine SOA ist bzw. ausmacht? Aus welchen wesentlichen Bestandteilen besteht diese? Wie wird das Konzept der SOA in den Folien theoretisch beschrieben? Wie wird es im Vergleich dazu in der Praxis gelebt?

Vorteile und Nachteile: Überlegen Sie sich welche Vorteile durch SOA entstehen. Wie spiegeln sich diese in den SOA-Paradigmen wieder (siehe Vorlesungsfolien)? Während die Vorteile in den Vorlesungsfolien ausführlich behandelt werden sind Nachteile deutlich unterrepräsentiert. Ist dies so weil SOA einfach keine Nachteile aufweist?

Case Study: Im zugehörigen Vorlesungsfoliensatz wird immer wieder hervorgehoben, dass all die vorgestellten Maßnahmen (z.B. die Designprinzipien oder Architekturpattern) vor allem darauf abzielen gewisse positive Eigenschaften zu fördern. Beispielsweise, dass Änderungen einfach durchgeführt werden können oder neue Funktionen leicht hinzugefügt werden können ohne bestehenden Source Code anpassen zu müssen.
Warum ist speziell zur Optimierung dieser beiden Aspekte SOA gut geeignet? Kommt es in echten Projekten überhaupt so oft zu Änderungen, dass diese beiden Vorteile relevant werden? Wir werden uns dieser Thematik, während der Vorlesung, anhand einer Case Study über das Flugunternehmen "Norwegian Air Shuttle" nähern.

11. Fortgeschritten: Entwürfe selbst erstellen

Im Folgenden werden die Ziele und vom Auftraggeber genannte Anforderungen für eine Bibliotheksverwaltungssoftware genannt. Versuchen Sie die in den Vorlesungsfolien genannten Entwurfsschritte und Designprinzipien auf diese Inhalte anzuwenden, um einen vollständigen Softwarentwurf durchzuführen. Daher strukturierte oder grafische Anforderungen ableiten, Datenstrukturen definieren, UML Klassendiagramme erstellen und die Kommunikation zwischen den Klassen mit UML Sequenzdiagrammen prüfen.

Ziel: Die neue Bibliotheksverwaltungssoftware soll es erlauben die Bibliothek und deren Ressourcen effizienter einzusetzen. Dies soll die Effizienz der Mitarbeiter erhöhen und den Kundenservice verbessern. Hierzu soll es beispielsweise möglich sein, Bücher schneller im Lager wiederfinden zu können.

Anforderungen: Das Bibliothekssystem soll Bücher während ihrer ganzen Lebensspanne begleiten; während diese ausgeborgt, reserviert, returniert oder verlängert werden. Es soll möglich sein, unterschiedliche Berechtigungen für unterschiedliche Kundengruppen abzubilden.
Es gibt derzeit zwei disjunkte Kundengruppen: "Normale Kunden" und "Förderer der Bibliothek". Während erstere maximal fünf Bücher für drei Wochen ausborgen dürfen sind die Regelungen für Förderer flexibler gestaltet: Förderer dürfen maximal zehn Bücher für sechs Wochen ausborgen. Kunden beider Gruppen dürfen jedes ausgeborgte Buch einmal für drei (Normale Kunden) bzw. sechs (Förderer der Bibliothek) Wochen verlängern, wenn dieses nicht bereits reserviert wurden. Kunden beider Gruppen dürfen Bücher reservieren.
Verwaltende Funktionen wie das ausborgen, returnieren, verlängern oder reservieren eines Buches darf nur von Bibliothekaren durchgeführt werden. Das durchsuchen der Buchbestände soll jedoch sowohl Bibliothekaren als auch Kunden möglich sein.

12. Software Design Pattern

Einer der vier in den Vorlesungsfolien (und dem Architekturtutorial aus der Übung vorgestellten Schritte zur Durchführung eines Softwareentwurfes ist die Identifikation und Anwendung von passenden Software Design Pattern. Hierbei wird ein bestehender Rohentwurf analysiert und geprüft ob sich Teile des Rohentwurfs mittels bekannter und erprobter Design Pattern optimieren oder umsetzen lassen. Dieser Schritt führt typischerweise zu besser wartbarer und verständlicherer Software. Wir werden deshalb hier nun die sogenannten Design Pattern behandeln.

Hinweis: Pattern stellen nur einen kleinen Teil des Bereiches Software Engineering dar und sind auch kein Patentrezept, sondern ein Aspekt unter vielen, welche beachtet werden sollten, um zu guten Softwareentwürfen zu gelangen. Wir bieten deshalb hier absichtlich nur eine Einführung in die für Pattern wichtigsten Aspekte und Ausprägungen. Vertiefend behandelt wird dieses Thema, unter anderem, in den LVs:

- 051050 VU Software Engineering 2
- 052500 VU Distributed Systems Engineering

Was: Was sind Software Pattern? Welche Eigenschaften muss eine Lösung aufweisen, um ein Software Pattern zu werden? Zieht man diese Eigenschaften in Betracht, können dann Datenstrukturen (z.B. ArrayList‹?› oder HashSet‹?›) Software Pattern sein?
Wie verhält es sich mit den initial vorgestellten Designprinzipien? Handelt es sich bei diesen um Pattern?

Frameworks & Pattern: Wie ist das Verhältnis zwischen Frameworks und Design Pattern? Sind Frameworks immer auch Design Pattern?

Idiome & Pattern: Wie sieht es mit Programmieridiomen aus? Können Sie ein beispielhaftes Programmieridiom nennen? Handelt es sich bei Programmieridiomen um Design Pattern?

Aufgabe: Welches Programmieridiom könnten Sie in folgenden Source Code anwenden, um diesen zu verbessern?

```
1. interface Colors {
2.   public static final int RED = 0;
3.   public static final int GREEN = 1;
4.   public static final int BLUE = 2;
5. }
6.
7. public class Main implements Colors {
8.   public static void main(String[] args) {
9.     int color = RED;
10.  }
11. }
```

Zusatzaufgabe: Könnte es sich bei dem folgenden Programmieridiom um ein Pattern handeln?

```
1. addWindowListener(
2.   new WindowAdapter() {
3.     public void windowClosing(WindowEvent we) {
4.       System.exit(0);
5.     }
6.   }
7. );
```

12.1. Observer Pattern & Alternativen

Sehen Sie sich die in den Vorlesungsfolien gegebene beispielhafte Implementierung (Klassendiagramm) des Observer Patterns an. Vergleichen Sie dieses mit der Implementierung des Observer Patterns im Java 8 Framework (siehe Observable und Observer). Welche Unterschiede fallen Ihnen auf? Wie wirken sich diese Unterschiede auf die Kopplung zwischen den Klassen aus? Welche Lehre für den Umgang mit Design Pattern können Sie aus diesen Abweichungen ziehen?

Zusatzaufgabe: In Java 9 wurde das Pattern als deprecated markiert (engl. für veraltet bzw. etwas dass es zu vermeiden gilt). Warum könnte es hierzu gekommen sein? Hängt diese Entscheidung mit dem Pattern selbst überhaupt zusammen? Inwiefern wäre das Konzept der PropertyChangeListener eine Alternative und welche Unterschiede weist es auf?

Hinweis: Eine flexiblere und praktikablere Weiterführung dieser Weiterentwicklung/Nachfolger des Observer Patterns und dessen Alternativen werden Sie im "professional Tutorial" der Übung kennenlernen.

12.2. Factory Pattern

Ist folgender Code eine korrekte Implementierung des Factory Patterns?

- Klären Sie hierzu zuerst: Wann wird es eingesetzt und welche Vorteile hat es?
- Danach: Vergleichen Sie es mit den Implementierungsvorgaben aus der Literatur (z.B. aus den bereitgestellten Unterlagen).
- Was fällt Ihnen dabei auf?

```
1. public class XMLReaderFactory {
2.   // Returns an instance of a class
3.   // that implements the XMLReader interface.
4.   // The class is based on a system property.
5.   public static XMLReader createXMLReader();
6. }
7.
8. public interface XMLReader {
9.   public void setContentHandler(ContentHandler hndlr);
10.  public void parse (InputStream is);
11. }
```

12.3. Singleton Pattern

In PR1 und PR2 wurden globale Variablen noch "verteufelt". Mit dem Singleton Pattern scheint ein sehr ähnlicher Ansatz jetzt auf einmal als breit akzeptiertes Design Pattern angewandt zu werden. Woran liegt das und wo liegen die Unterschiede zwischen beiden?

Zusatzaufgabe 1: Wie würde ein minimales Beispiel für ein Singleton Pattern in Java aussehen?

Zusatzaufgabe 2: Können Sie sich Einsatzszenarien für das Singleton Pattern vorstellen? Insbesondere, wann sollte dieses verwendet werden und wann nicht?

12.4. Strategy Pattern

Nehmen Sie an Sie sollen eine Software entwickeln die abhängig vom Ort des Empfängers ein anderes Logistikunternehmen für den Versand auswählt und nützt. Wie können Sie dieses Verhalten mittels Strategy Pattern implementieren? Können Sie sich ein alternatives Konzept für die Implementierung vorstellen? Wenn ja, welche Nachteile/Vorteile hätte dieses gegenüber dem Einsatz des Strategy Patterns?

Zusatzaufgabe: Wie würde ein minimales Beispiel für ein Strategy Pattern in Java aussehen?

12.5. Welches Pattern eignet sich?

Nehmen Sie an es existiert ein Klasse Server, welche die folgenden Methoden aufweist. Welches Pattern eignet sich dabei besonders, um den Umgang mit dieser Klasse zu optimieren?

```
1. public class Server {
2.
3.   public void startBooting() {
4.     System.out.println("Starts booting...");
5.   }
6.
7.   public void readSystemConfigFile() {
8.     System.out.println("Reading system config files...");
9.   }
10.
11.  public void init() {
12.    System.out.println("Initializing...");
13.  }
14.
15.  public void initializeContext() {
16.    System.out.println("Initializing context...");
17.  }
18.
19.  public void initializeListeners() {
20.    System.out.println("Initializing listeners...");
21.  }
22.
23.  public void createSystemObjects() {
24.    System.out.println("Creating system objects...");
25.  }
26.
27.  public void releaseProcess() {
28.    System.out.println("Releasing process...");
29.  }
30.
31.  public void destory() {
32.    System.out.println("Destorying...");
33.  }
34.
35.  public void destroySystemObjects() {
36.    System.out.println("Destroying system objects...");
37.  }
38.
39.  public void destoryListeners() {
40.    System.out.println("Destroying listeners...");
41.  }
42.
43.  public void destoryContext() {
44.    System.out.println("Destroying context...");
45.  }
46.
47.  public void shutdown() {
48.    System.out.println("Shutting down...");
49.  }
50. }
```

12.6. Welches Pattern eignet sich? (Fortgeschritten)

Aufgabe: Sie Arbeiten an einer Software, welche direkt mit einer vorgegebenen Hardware interagieren soll. Der Hersteller der Hardware stellt eine große Anzahl an Klassen und Softwarekomponenten zur Verfügung um verschiedene Aufgabenstellung mit der Hardware feingranular abbilden zu können. Diese bereitgestellten Klassen und Komponenten werden an einer Vielzahl von Stellen in der Software verwendet. Sie befürchten, dass die hierdurch entstehende Lösung zu kompliziert werden könnte. Beschreiben Sie welche Risiken auftreten können und welches Design Pattern sich eignet um diese Risiken zu negieren?

Zusatzaufgabe: Nehmen Sie an die zur Verfügung gestellt Hardware ist noch nicht final und es ist sehr wahrscheinlich, dass sich die Schnittstellen auf die Hardware noch ändern werden. Welche Probleme könnten hierdurch auftreten? Wie adaptieren Sie die zuvor vorgeschlagene Lösung um auf diese Probleme zu reagieren? Sehen Sie eine Parallele zur Aufgabenstellung der Übung?

12.7. Designprinzipien und Nachteile von Pattern

Designprinzipien: Welche Designprinzipien muss eine Lösung inkludieren, um ein Pattern zu werden?

Nachteile von Pattern: Ein Kollege hat sich Ihren Architekturvorschlag angesehen und schlägt vor an über 20 Stellen Änderungen vorzunehmen, um zahlreiche Design Pattern zu berücksichtigen. Mit welchen Nachteilen könnte dieser Vorschlag verbunden sein?

13. Menschliche Faktoren

Warum sollte man Softwarearchitekturen für große Projekte im Team, also nicht alleine, entwerfen? Warum lassen wir Sie trotzdem in SE1 Ihr Projekt alleine planen und umsetzen?