Übung zur Vorlesung Software Engineering

Klausurvorbereitung—Teil 1

TECHNISCHE UNIVERSITÄT DARMSTADT

Sammlung von Übungsaufgaben—Teil 1

Dieses Übungsblatt ist Teil 1 einer Sammlung von Übungsaufgaben zur zusätzlichen Vertiefung. Die Zusammenstellung erlaubt keinerlei Aussagen über die Klausur. Insbesondere werden nicht alle klausurrelevanten Themen abgedeckt.

Aufgabe 1: Stakeholder und deren Ziele identifizieren

Beschreibung: Eine hessische Universität beauftragt Sie mit der Entwicklung eines Hochschulinformationssystems namens PeLICAN. Studierende sollen sich in dem System zu Kursen und Prüfungen anmelden und diese Anmeldungen verwalten (bspw., sich vor Ablauf von Fristen von Prüfungen anmelden) können sowie Zugriff zu in Kursen veröffentlichten Lernmaterialien haben. Gesetzliche Vorgaben, wie zum Beispiel der Datenschutz, müssen von dem System eingehalten werden; insbesondere sollen persönliche Daten von Studierenden nur Akteuren zugänglich sein, welche diese zur Erfüllung ihrer Aufgaben benötigen. Lernmaterialien können von Dozent_innen hochgeladen werden, welche auch Prüfungsergebnisse für die Studierenden eintragen. Dozent_innen haben über das System außerdem die Möglichkeit, allen oder einzelnen Studierenden ihrer Kurse Nachrichten zu senden. Mitarbeiter_innen der Prüfungsbüros haben Zugriff auf alle Kurse der Fachbereiche, für die sie zuständig sind; ihnen soll es ermöglicht werden, effizient Prüfungsergebnisse zu einzusehen und zu veröffentlichen, sowie bspw. Leistungsspiegel auf Anfrage von Studierenden zu erstellen. Sowohl Dozent_innen als auch Mitarbeiter_innen der Prüfungsbüros verwenden Software von Drittanbietern, um z.B. Noten zu berechnen. Die Kompatibilität mit solchen Anwendungen soll durch flexible Schnittstellen des Systems nach außen sichergestellt werden.

Aufgabe: Nennen Sie drei Stakeholder mit direktem (Interactor viewpoints) sowie zwei mit indirektem (Indirect viewpoints) Interesse an dem System. Beschreiben Sie jeweils kurz das Interesse eines jeden Stakeholders; beachten Sie dabei insbesondere die folgenden Aspekte: Was möchten **direkte** Stakeholder mit dem System tun, und welche Rahmenbedingungen muss das System für **indirekte** Stakeholder erfüllen?

Aufgabe 2: Bewertung der Aussagekraft von Anforderungen

Aufgabe: Betrachten Sie die gegebenen Anforderungen. Bewerten Sie, ob die Anforderungen geeignet sind, um eine fundierte Aussage darüber abzugeben, wann die spezifizierte Anforderung erfüllt ist. Begründen Sie Ihre Einschätzung kurz. Formulieren Sie gegebenenfalls nicht prüfbare Anforderungen unter Einbeziehung von zusätzlichen Rahmenbedingungen und Angaben, welche sie prüfbar machen, um.

Hinweis: Bei Ihrer Einschätzung der Anforderungen ist es hilfreich, wenn Sie sich überlegen, wie genau die gegebenen nicht-funktionalen Anforderungen sichergestellt werden können.

1. Die Importfunktion muss in der Lage sein, Überschriften, (unformatierten) Text und Bilder aus DOCX-Dokumenten in das aktuelle Dokument zu übernehmen.
2. Die Anwendung darf nur wenig Speicher (RAM) benötigen

Aufgabe 3: Anwendungsfälle erkennen und modellieren

Aufgabe: Identifizieren Sie die im folgenden beschriebenen Akteure und Anwendungsfälle sowie deren Beziehungen. Stellen Sie sie als UML-Anwendungsfalldiagramm dar.

Hinweis: Denken Sie auch daran, sofern geboten, gemeinsame „Subfunction“Use Cases herauszufaktorisieren.

Kassensystem einer Supermarktkette

Der zentrale Anwendungsfall des Kassensystems ist das Verkaufen von Produkten. Dabei erfasst der Kassierer zunächst die Produkte, die die/der Kund_in kaufen möchte. Anschließend wird der Zahlvorgang abgewickelt und ggf. ein Kassenbeleg für die/den Kund_in ausgedruckt. Benötigt der/die Kassierer_in Hilfe, zum Beispiel weil ihm ein Verkaufspreis unbekannt ist, kann er direkt über das System um Hilfe rufen. In diesem Fall wird umgehend ein(e) Kolleg_in informiert und zu ihr/ihm geschickt.

Manager_innen können über das System neue Produkte bei Lieferanten bestellen. Sollte ein Lieferant nicht im System vorhanden sein, kann ein neuer Lieferant im Bestellvorgang bei der Lieferanten-Auswahl dem System hinzugefügt werden. Manager nutzen das System ebenfalls um Statistiken zu erstellen. Sowohl beim Bestellen als auch beim Anfertigen von Statistiken müssen relevante Filialen definiert werden. Dies geschieht durch Eingabe des Landes und durch zusätzliche Filter wie z.B. dem Bundesland oder dem Ort.

Aufgabe 4: Detaillierte Anforderungsanalyse mittels „fully dressed“ Use Cases

Aufgabe: Im Rahmen der Anforderungsanalyse wurde folgende Beschreibung festgehalten. Erstellen Sie, wie in der Vorlesung beschrieben, eine „fully dressed“ Use Case Beschreibung zur Funktionalität des Anwendungsfalls „Erstelle Frage“ der Anwendung „The Quiz“.

Zusammenfassung des Interviews zum Anwendungsfall „Erstelle Frage“

Der/Die Spieldesigner_in (genannt SD) entschließt sich, eine geöffnete Fragesammlung um eine neue Frage zu erweitern.

Das System zeigt zunächst die verfügbaren Fragetypen an. Nachdem SD einen Fragetypen ausgewählt hat, erstellt das System eine neue Frage und zeigt diese zum Bearbeiten an. Bricht sie/er das Erstellen der Frage bei der Fragetyp-Auswahl ab, bleibt die Fragesammlung unverändert und das System zeigt die geöffnete Fragesammlung an.

Als nächstes legt SD nun alle Werte der neuen Frage fest. Stellt sie/er hierbei fest, dass eine benötigte Kategorie fehlt, ermöglicht das System diese gemäß Anwendungsfall „Erstelle Kategorie“ der Fragesammlung hinzuzufügen. Bestätigt SD die neue Frage, wird diese vom System der Fragesammlung hinzugefügt. Anschließend kann sie in Einzel- und Mehrbenutzer-Spielen von den Spielenden gespielt werden. Bricht SD jedoch das Erstellen der Frage ab, zeigt das System die geöffnete Fragesammlung an. In beiden Fällen bleiben vorhandene Fragen unverändert und angelegte Kategorien werden stets erhalten.

Use Case	Beschreibung
Name des Anwendungsfalls (Use Case Name)	
Kurzbeschreibung des Anwendungsfallziels (Goal in Context)	
Umfang (Scope)	
Zielart (Level)	
Interessengruppe und Interessen (Stakeholders and Interests)	
Minimale Garantien (Minimal Guarantees)	
Erfolgsgarantien (Success Guarantees)	
Hauptakteur (Primary Actor)	
Vorbedingungen (Precondition)	
Haupterfolgsszenario (Main Success Scenario)	
Erweiterungen (Extensions)	

Aufgabe 5: Identifikation und Modellierung der Konzepte einer Domäne

Aufgabe: Erstellen Sie gemäß der folgenden Beschreibung ein sinnvolles Domänenmodell (domain model) in Form eines UML-Klassen-Diagramms. Neben den konzeptionellen Klassen sind die Beziehungen der Klassen untereinander in Form von Vererbungsbeziehungen und ungerichteten Assoziationen (inkl. Multiplizitäten auf beiden Seiten und einem aussagekräftigen Assoziationsnamen), sowie die wesentlichen Eigenschaften (Attribute) der Klassen zu verdeutlichen. **Weitere Elemente, wie Operationen, spezialisierte Assoziationen (Aggregation und Komposition), gerichtete Assoziationen sowie Datentypen und Sichtbarkeiten von Attributen sind NICHT zu verwenden!**

Hinweis: Identifizieren Sie zunächst die relevanten Konzepte und modellieren Sie diese anschließend als Klassen.

App-Store

Im App-Store werden unterschiedliche digitale Produkte, wie Anwendungen (Apps), Filme oder Serien zum Kauf angeboten. Alle digitalen Produkte besitzen einen Titel, eine aussagekräftige Beschreibung und einen Kaufpreis. Zusätzlich besitzt jede Produktart weitere Detailinformationen. Bei Anwendungen ist dies die Herstellerfirma, bei Filmen das Erscheinungsjahr und das Genre sowie bei Serien das Erscheinungsjahr und der Sender, für den sie produziert wurde. Eine Serie besteht zusätzlich aus mehreren Episoden, von denen sich jede durch einen eigenen Titel auszeichnet.

Bevor ein_e Benutzer_in digitale Produkte erwerben kann, muss sie/er sich einmalig im System mit einem frei wählbaren Benutzernamen und Passwort sowie einer gültigen E-Mail-Adresse registrieren. Anschließend kann sie/er unter dem angelegten Benutzerprofil digitale Produkte erwerben. Das System merkt sich für jede Benutzer_in die gekauften Produkte. Unabhängig davon ob ein Produkt gekauft wurde, kann der/die Benutzer_in alle Produkte durch die Vergabe von einem bis hin zu fünf Sternen bewerten. Je mehr Sterne vergeben werden, desto besser gefällt das Produkt. Ergänzende Informationen zur Bewertung können in einem optionalen Kommentar publiziert werden.

Aufgabe 6: Maßnahmen zur Verifikation und Validierung verstehen und einschätzen können

Aufgabe: Die folgenden Techniken können zur Verifikation und Validierung von Software eingesetzt werden. Beschreiben Sie je dessen Ziel und Funktionsweise. Überlegen Sie sich zusätzlich, wann Sie eine Technik einsetzen würden.

1. Software inspections/peer reviews
2. Automated static analyses
3. Formal verification
4. Testing
5. Runtime assertion checking

Aufgabe 7: Architekturstile

Ziel: Einordnung von Architekturstilen bzgl. Architekturmerkmalen

Beschreibung:

Beurteilen Sie die Architekturstile *Layered*, *Model-View Controller* und *Service Based* bzgl. der unten aufgezählten Eigenschaften unter Verwendung der Skala gut, mittel und schlecht.

Begründen Sie Ihre Beurteilung. Ihre Beurteilung muss sich auf die Architekturebene beziehen, d.h. wie gut werden die Eigenschaften *aus der Architektur heraus* unterstützt. (Mit entsprechend viel Aufwand und hoher zusätzlicher Komplexität lässt sich fast alles mit allem realisieren.)

1. Fehlertoleranz (Fault-Tolerance): Stabilität des Systems bei auftretenden Fehlern wie z.B. Abstürzen einzelner Module/Komponenten
2. Parallelisierbarkeit

Aufgabe 8: Softwaremetriken anwenden und Anomalien erkennen

Aufgabe:

a) Vervollständigen Sie den Kontrollflussgraphen (CFG) aus Abbildung 1 für die Methode „magic()“ gemäß Listing 1. Vergeben Sie für jeden Knoten ein eindeutiges Label, d.h. eine fortlaufende Zahl wie in Abbildung 1.

b) Geben Sie alle „linear unabhängigen Pfade“ in Ihrem Kontrollflussgraphen als Menge von Listen der Knoten-Labels im Graphen an, also bspw. $\{(1, 2, 4, 5), \dots\}$, wobei $(1, 2, 4, 5)$ einen linear unabhängigen Pfad in Abbildung 1 darstellt.

c) Berechnen Sie die zyklomatische Komplexität nach McCabe's ursprünglicher Formel $C := E - N + 2P$ mit N = Anzahl Knoten, E = Anzahl Kanten und P = Anzahl der Zusammenhangskomponenten (engl. „Connected Components“). Ergänzen Sie dazu ggf. den CFG aus Aufgabenteil a) um weitere benötigte Elemente.

d) Eine zyklomatische Komplexität größer als 10 deutet auf zu komplexe Strukturen hin. Begründen Sie kurz, ob die berechnete Komplexität in Aufgabenteil c) Ihrem Empfinden der Komplexität der Methode entspricht.

e) Listen Sie unnötige und niemals ausgeführte Anweisungen in Listing 1 durch Angabe der Zeile auf. Begründen Sie ebenfalls, warum eine Anweisung überflüssig ist bzw. nicht ausgeführt werden kann.
Hinweis: Beachten Sie das der Konstruktor den möglichen Wertebereich der Felder `a` und `b` einschränkt.

Listing 1: Magic

```
1  public class Magic {
2    private int a, b;
3
4    public Magic(int a, int b) {
5      if (a < 0 || b < 0) {
6        throw new IllegalArgumentException("Negative values are not allowed");
7      }
8      this.a = a; this.b = b;
9    }
10
11   public void magic() {
12     int x = a;
13     int y = 0, z = b;
14     int tmp = 0;
15     if (x == z) {
16       return;
17     } else {
18       if ((x & tmp) <= y) {
19         tmp = x;
20         x = x^z;
21       } else {
22         throw new RuntimeException();
23       }
24
25       z = x^z;
26       y = tmp;
27       x = x^z;
28
29       if (y <= a) {
30         tmp = 0;
31         a = x;
32         while (y > 0) {
33           y--;
34           tmp++;
35         }
36       }
37     }
38
39     b = 2 * z - tmp;
40     return;
41   }
42 }
```

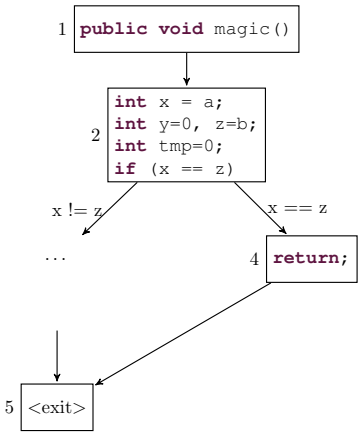

Abbildung 1: Anfang des Kontrollflussgraphen für Listing **??**

Aufgabe 9: Bewerten eines Software-Designs durch Betrachtung der Kopplung (Abhängigkeiten) und Kohäsion

In einem in Java implementierten Kassensystem finden Sie die in Listing 3 abgebildete Klasse `ReceiptPrinter`. Ihre primäre Aufgabe ist es, in der Methode `print(PrintStream)` einen Kassenbeleg auf einem Ausgabemedium, wie z.B. der Konsole, auszugeben. Der auszugebende Kassenbeleg wird dazu aus einer Datei geladen. Ein Kassenbeleg selbst ist eine Instanz der Klasse `Receipt` (siehe Listing 4), welcher mehrere Positionen, Instanzen von `ReceiptPosition` (siehe Listing 5), enthält.

a) **Ziel:** Kopplung einer Klasse erkennen.

   Aufgabe: Listen Sie alle Typen (Klassen, Interfaces, Enumerationen und Annotationen) auf, von denen `ReceiptPrinter` gemäß des Quellcodes direkt abhängig ist.

b) **Ziel:** Kopplung zwischen den Klassen der Anwendung bewerten.

   Aufgabe: Bewerten Sie kurz unter Berücksichtigung der gefundenen Abhängigkeiten, wie gut sich die Klasse `ReceiptPrinter` in anderen Projekten wiederverwenden lässt.

c) **Ziel:** Lack of Cohesion of Methods (LCOM) berechnen.

   Aufgabe: Berechnen Sie für die Klasse `ReceiptPrinter` den LCOM-Wert, mit Hilfe der in der Vorlesung vorgestellten verbesserten Version von Li und Henry. Erstellen Sie dazu zunächst den Graphen und nutzen Sie diesen anschließend um den LCOM-Wert zu bestimmen.

d) **Ziel:** Kohäsion einer Klasse mittels LCOM bewerten.

   Aufgabe: Beschreiben Sie kurz, wie Sie den errechneten LCOM-Wert aus Aufgabenteil c) bezüglich der Kohäsion der Klasse `ReceiptPrinter` interpretieren. Entspricht das Ergebnis Ihrer Auffassung bezüglich der Kohäsion der Klasse?

e) **Ziel:** Kohäsion einer Klasse bewerten.

   Aufgabe: Die Methoden `print(PrintStream)` und `toString()` greifen auf keine gemeinsame Instanzvariable zu. Ist es ratsam diese auf zwei Klassen zu verteilen, um die errechnete Kohäsion mittels LCOM zu verbessern?

Listing 2: ReceiptPrinter

```
import java.io.File;
import java.io.FileInputStream;
import java.io.ObjectInputStream;
import java.math.BigDecimal;
import java.util.Date;

import model.Receipt;

public class ReceiptPrinter {
   private Receipt receipt;

   private Date loadDate;

   public ReceiptPrinter(File file) throws Exception {
      initLoadDate();
      this.receipt = read(file);
   }

   private void initLoadDate() {
      loadDate = new Date();
   }

   public void print(java.io.PrintStream ps) {
      ps.print(toString());
   }

   @Override
   public String toString() {
      StringBuilder sb = new StringBuilder();
      for (model.ReceiptPosition rp : receipt.getPositions()) {
         sb.append(rp.getProduct())
           .append(": ")
           .append(rp.getPrice()).append("\n");
      }
      sb.append("===========\n")
        .append(computePrice())
        .append("\n\n(").append(loadDate).append(")");
      return sb.toString();
   }

   private BigDecimal computePrice() {
      BigDecimal sum = BigDecimal.ZERO;
      for (model.ReceiptPosition rp : receipt.getPositions()) {
         sum = sum.add(rp.getPrice());
      }
      return sum;
   }

   private Receipt read(File file) throws Exception {
      ObjectInputStream in = new ObjectInputStream(new FileInputStream(file));
      try {
         Object read = in.readObject();
         if (read instanceof Receipt) {
            return (Receipt)read;
         }
         else {
            throw new Exception("No Receipt found.");
         }
      }
      finally {
         in.close();
      }
   }
}
```

Listing 3: Receipt

```
package model;

import java.io.Serializable;
import java.util.LinkedList;
import java.util.List;

public class Receipt implements Serializable {
   private static final long serialVersionUID = -8760949217949784771L;

   private final List<ReceiptPosition> positions = new LinkedList<ReceiptPosition>();

   public void addPosition(ReceiptPosition rp) {
      if (rp != null) {
         positions.add(rp);
      }
   }

   public ReceiptPosition[] getPositions() {
      return positions.toArray(new ReceiptPosition[positions.size()]);
   }
}
```

Listing 4: ReceiptPosition

```
package model;

import java.io.Serializable;
import java.math.BigDecimal;

public class ReceiptPosition implements Serializable {
   private static final long serialVersionUID = 5269887755509983958L;

   private final String product;

   private final BigDecimal price;

   public ReceiptPosition(String product, BigDecimal price) {
      this.product = product;
      this.price = price;
   }

   public String getProduct() {
      return product;
   }

   public BigDecimal getPrice() {
      return price;
   }
}
```