

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?

Lösung:

- Stakeholder mit direktem Interesse (Interactor viewpoints)
 - Studierende: Möchte sich im System zu Prüfungen/Kursen an- und abmelden sowie Lernmaterialien herunterladen können.
 - DozentInnen: Möchten Lernmaterialien hochladen und Prüfungsergebnisse eintragen sowie Nachrichten versenden können.
 - MitarbeiterInnen der Prüfungsbüros: Möchten effizient Prüfungsergebnisse einsehen und veröffentlichen sowie Leistungsspiegel erstellen können.
 - Weitere nicht im Text beschriebene
 - * EDV/Hochschulrechenzentrum: Möchte ein leicht zu wartendes System.
 - * Universitätsmanagement: Möchte Statistiken über Studierendenanzahl, Durchfallquoten etc. erhalten können.
- Stakeholder mit indirektem Interesse (Indirect viewpoints)
 - Staat/Datenschützer: Geltende Gesetze, bspw. aus dem Bereich des Datenschutzes, müssen eingehalten werden.
 - Anbieter vorhandener Anwendungen: Kompatibilität muss gewährleistet werden (flexible Schnittstellen sollen verfügbar sein).

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

Lösung:

1. Die funktionale Anforderung ist geeignet, um eine fundierte Bewertung bzgl. der Erfüllung abzugeben. Es ist genau spezifiziert, welche Inhalte welcher Dokumententypen übernommen werden sollen. Die Erfüllung kann beispielsweise mit einem Worddokument (DOCX), welches die benannten Elemente enthält, überprüft werden.
2. Die nicht-funktionale Anforderung ist nicht geeignet, um eine fundierte Bewertung ob bzw. in wie weit die Anforderung erfüllt ist abzugeben, denn „wenig Speicher“ ist unklar formuliert (Wieviel ist das? Durchschnitt oder „Peak“-Verbrauch? Auf welchen Endgeräten? Usw.)

Neu formulierte nicht-funktionale Anforderung:

Die Anwendung darf zu keinem Zeitpunkt mehr als 100 MB Hauptspeicher auf Desktop-Geräten sowie 30 MB Hauptspeicher auf mobilen Endgeräten beanspruchen.

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.

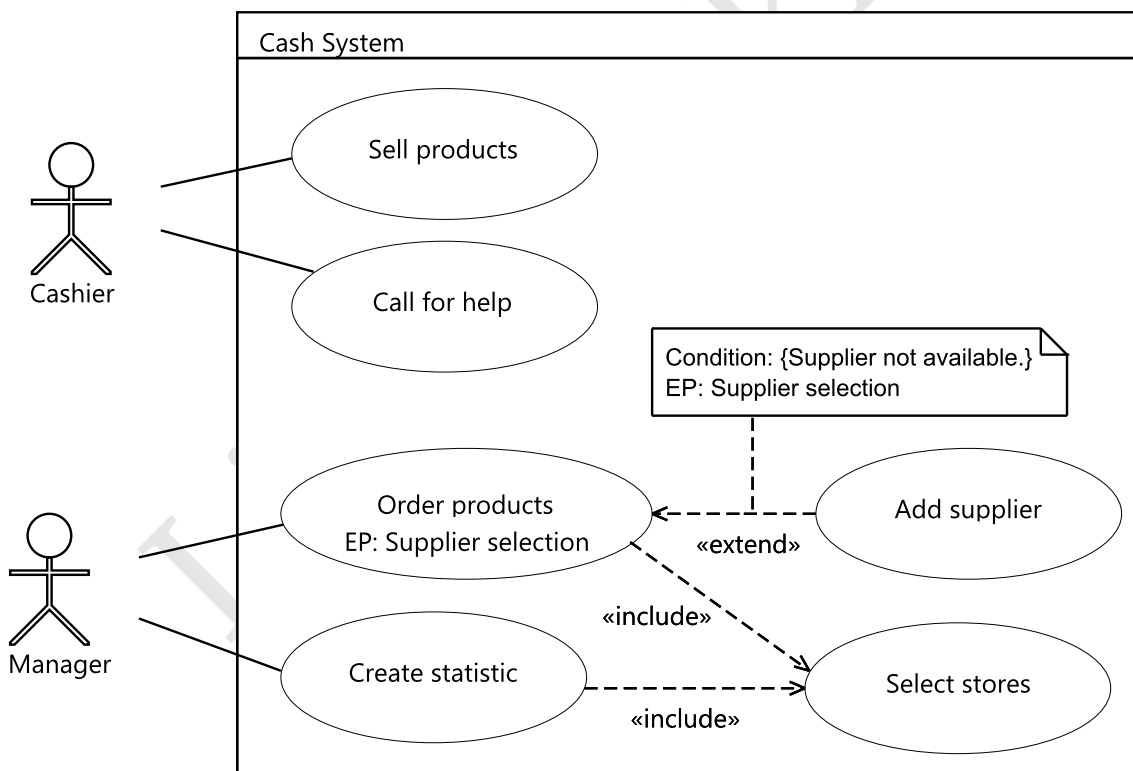
Hinweis: Denken Sie auch daran, sofern geboten, gemeinsame „Subfunction“-Use Cases herauszufaktorisieren.

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.

Lösung:



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)	

Lösung:

Use Case	Beschreibung
Name des Anwendungsfalls (Use Case Name)	Erstelle Frage
Kurzbeschreibung des Anwendungsfallziels (Goal in Context)	Der Spieldesigner fügt der Fragesammlung eine neue Frage hinzu.
Umfang (Scope)	Die Anwendung „The Quiz“
Zielart (Level)	Benutzerziel
Interessensgruppe und Interessen (Stakeholders and Interests)	<ul style="list-style-type: none">• Spieldesigner: Möchte Fragesammlung erweitern.• Spieler: Möchte neue Frage spielen.
Minimale Garantien (Minimal Guarantees)	Alle existierenden Fragen bleiben unverändert. Existierende Kategorien wurden erhalten.
Erfolgsgarantien (Success Guarantees)	Eine neue Frage des gewählten Typs mit definierten Werten wurde der Fragesammlung hinzugefügt. Neu erstellte Kategorien wurden hinzugefügt.
Hauptakteur (Primary Actor)	Spieldesigner
Vorbedingungen (Precondition)	Eine Fragesammlung ist geöffnet.
Haupterfolgsszenario (Main Success Scenario)	<ol style="list-style-type: none">1 System zeigt verfügbare Fragetypen an.2 Spieldesigner wählt einen Fragetyp aus.3 System erstellt neue Frage und zeigt sie zum Bearbeiten an.4 Spieldesigner legt Werte der Frage fest und bestätigt diese.
Erweiterungen (Extensions)	<ol style="list-style-type: none">2a Spieldesigner bricht Frage-Erstellung ab.<ol style="list-style-type: none">2a1 System zeigt Fragesammlung an.4a Benötigte Kategorie fehlt.<ol style="list-style-type: none">4a1 System ermöglicht Kategorie gemäß Anwendungsfall „Erstelle Kategorie“ der Fragesammlung hinzuzufügen.4b Spieldesigner bricht Frage-Erstellung ab.<ol style="list-style-type: none">4b1 System zeigt Fragesammlung an. Neue Kategorien werden erhalten.

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!**

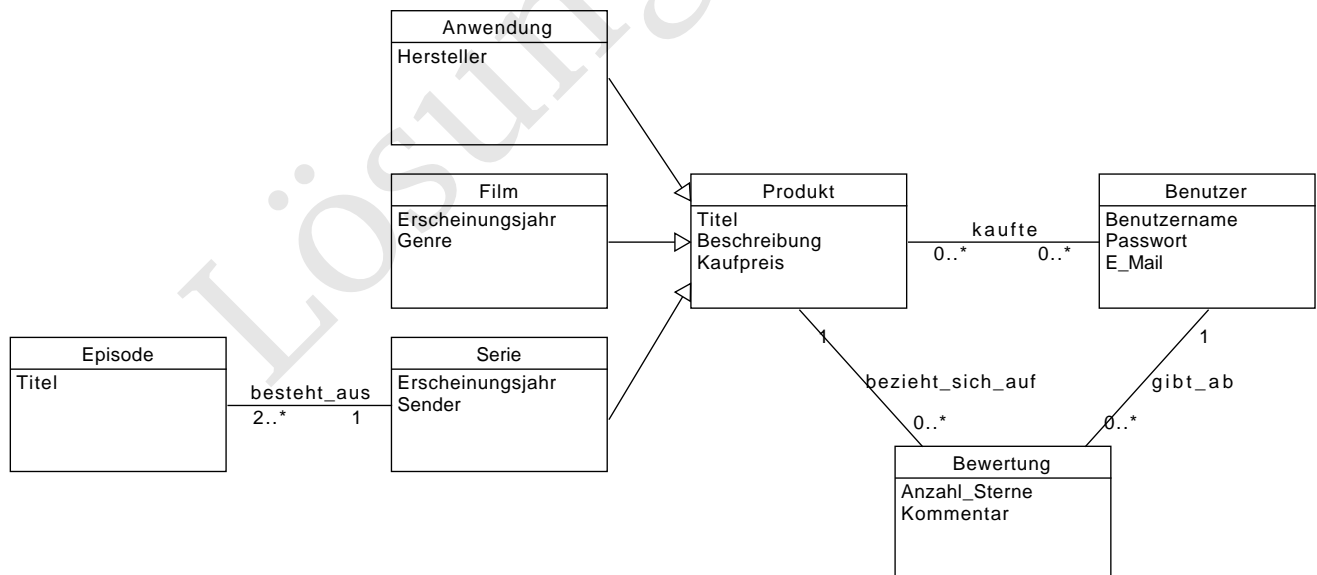
Hinweis: Identifizieren Sie zunächst die relevanten Konzepte und modellieren Sie diese anschließend als Klassen.

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.

Lösung:



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

Lösungsskizze

Lösung:

1. Software inspections/peer reviews

- **Ziel:** Komplexe Sachverhalten bewerten und semantische Fehler finden.
- **Funktionsweise:** Ein Review ist eine menschliche Prüfung von Dokumenten. Sie kann auf alle Arten von Dokumenten angewendet werden und findet im Allgemeinen eher Fehlerzustände als Wirkungen. Worauf bei einem Review geachtet wird und welche Rollen dafür nötig sind, muss im Vorfeld festgelegt werden. Die Inspektion ist ein spezielles Review mit dem Ziel Fehler zu finden, bei dem das Dokument meist anhand einer Checkliste bewertet wird. Ein weiterer positiver Effekt von Reviews ist, dass mehrere Leute die Qualität des Dokuments bewerten und so das Vertrauen darin erhöht wird.
- **Einsatz:** Im gesamten Entwicklungsprozess möglichst auf alle Dokumente, denn es findet eher Fehlerzustände während z.B. Testen eher Fehlerwirkungen aufdeckt.

2. Automated static analyses

- **Ziel:** Automatisches Aufdecken von Fehlerzuständen oder fehlerträchtigen Stellen in formalen Dokumenten.
- **Funktionsweise:** Ein Werkzeug überprüft automatisch formale Dokumente auf die Verletzung von definierten Eigenschaften. Bei Quellcode kann zum Beispiel die Verletzung der Syntax, Fehler oder Warnungen, nicht erreichbarer Code sowie die Abweichung von Konventionen erkannt werden.
- **Einsatz:** Im Vorfeld von Reviews, da sie automatisch ausführbar sind. Wurden Konventionen festgelegt, muss deren Einhaltung unbedingt automatisch durchgesetzt werden, denn ansonsten werden sie früher oder später missachtet und ignoriert. Unterstützende Werkzeuge die Fehlerstellen aufzeigen stets einsetzen und gefundene Warnungen beheben sofern sie berechtigt sind.

3. Formal verification

- **Ziel:** Abwesenheit von Fehlern beweisen.
- **Funktionsweise:** Es wird ein mathematischer Beweis geführt, indem die Konformität des Quellcodes zu definierten Eigenschaften bewiesen wird. Es setzt voraus, dass der Code, dessen Spezifikationen und die Konformität formal definiert sind.
- **Einsatz:** Ergänzend zum Testen, um den letzten Fehler zu finden. Von großer Bedeutung für Anwendungen mit hohen Safety- und Security-Anforderungen.

4. Testing

- **Ziel:** Anwesenheit von Fehlern aufzeigen.
- **Funktionsweise:** Die Anwendung wird dynamisch ausgeführt und die Eingaben gemäß eines Testskripts vorgenommen. Im Testorakel vorgegebene Sollwerte werden mit den aktuellen Werten verglichen, um fehlerhaftes Verhalten aufzudecken.
- **Einsatz:** So früh wie möglich und kontinuierlich, um den ersten Fehler zu finden. Möglichst automatisierte Tests implementieren damit diese regelmäßig (z.B. täglich) den aktuellen Stand testen.

5. Runtime assertion checking

- **Ziel:** Einhaltung von Rahmenbedingungen zur Laufzeit sicherstellen.
- **Funktionsweise:** Assertions sind spezielle Anweisungen, die anhand einer logischen Aussage sicherstellen, dass die Anwendung im erwarteten Zustand ist. Wird die Aussage zu falsch ausgewertet, wird zum Beispiel durch das Werfen einer Exception die Weiterausführung verhindert. Assertions sind zum Teil fester Bestandteil von Programmiersprachen oder werden mit Hilfe von Präprozessoren realisiert. Zum Teil ist es möglich die Überprüfung von Assertions vor Programmausführung zu aktivieren bzw. deaktivieren. Bei Java Anwendungen muss die Überprüfung von „assert“-Anweisungen mit Hilfe des JVM-Parameters „-ea“ aktiviert werden.
- **Einsatz:** Hilfreich während der Entwicklung und unterstützend beim Debugging.

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

Lösungsskizze

Lösung:

Fehlertoleranz:

- Layered: Schlecht. Bei Ausfall einer Schicht sind zumindest auch die oberen Schichten i.d.R. direkt betroffen, da diese zur Erledigung ihrer Aufgaben die unteren Schichten benötigen. Schichtenarchitekturen zeichnen sich auch häufig durch längere Zeiten beim Neustarten nach einem Ausfall aus, unter anderem ist das System erst Einsatzbereit, wenn die am langsamsten startende Schicht bereit ist.
- MVC: Mittelmässig. Vom Absturz einer View (Sicht) oder eines Controllers sind die anderen Views und Controller nicht betroffen. Beim Ausfall des Modells fällt jedoch normalerweise das ganze System aus.

Die Fehlertoleranz des Modells hängt allerdings auch von dessen Architektur ab.

- Service-Based: Gut. Der Ausfall eines einzelnen Dienstes (Services) schränkt die Funktionalität zwar ein, die anderen Dienste sind jedoch nicht (direkt) betroffen.

Die Auswirkungen hängen insbesondere auch von der Granularität der Aufteilung in Dienste ab und inwieweit Nutzerinteraktionen nicht immer alle Dienste benötigen. Bei der Möglichkeit mehrere Instanzen gleicher Dienste laufen zu lassen ist die Fehlertoleranz sogar sehr gut.

Parallelisierbarkeit:

- Layered: von der Architektur nicht unterstützt, da strikter sequentieller Durchlauf (Überspringen einzelner sog. offener Schichten möglich) der Schichten

Mit Aufwand lässt sich auch innerhalb der Schichten parallelisieren, aber dies wird nicht aus der Architektur heraus unterstützt.

- MVC: (mittel) Views, Controller und Modell können einzelnen Prozessen leicht zugeordnet werden; die Architektur unterstützt aber von sich aus z.B. nicht den sicheren geteilten Zugriff auf das Modell.
- Service-Based: Als verteilter (distributive) Stil sind service-basierte Architekturen schon von Natur aus auf Nebenläufigkeit ausgelegt. Die Dienste (Services) können als eigene Prozesse auf dem gleichen oder unterschiedlichen Rechnern laufen.

Aufgabe 8: Softwremetriken anwenden und Anomalien erkennen

Aufgabe:

- 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.
- 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.
- 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.
- 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.
- 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 }
```

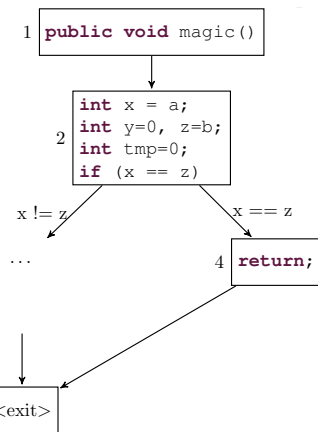


Abbildung 1: Anfang des Kontrollflussgraphen für Listing ??

Lösung a):

Siehe Abbildung ??.

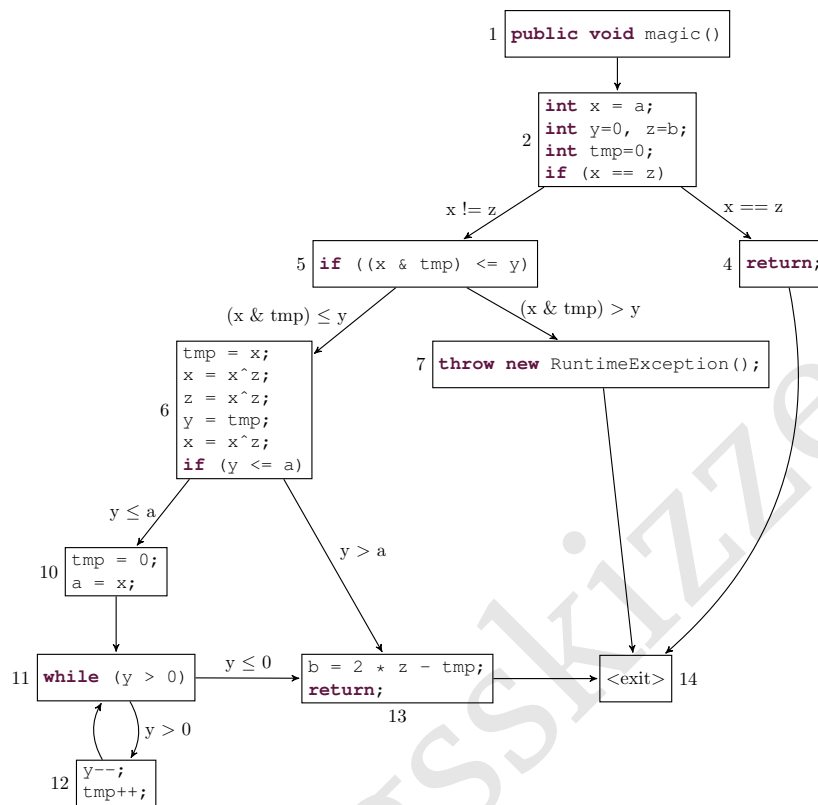


Abbildung 2: Vollständiger Kontrollflussgraph

Lösung b):

Die Menge der linear unabhängigen Pfade ist $\{(1, 2, 4, 14), (1, 2, 5, 7, 14), (1, 2, 5, 6, 13, 14), (1, 2, 5, 6, 10, 11, 13, 14), (1, 2, 5, 6, 10, 11, 12, 11, 13, 14)\}$.

Lösung c):

$$E = 14, N = 11, P = 1$$

$$C = E - N + 2P = 14 - 11 + 2 \cdot 1 = 5$$

Lösung d):

Der Kontrollflussgraph ist noch einigermaßen überschaubar; dennoch ist die Funktionalität des Tauschs („swap“) der Werte zweier Instanzvariablen unnötig kompliziert realisiert und sollte vereinfacht werden. Daher erscheint die berechnete Komplexität in diesem Beispiel angemessen zu sein.

Lösung e):

Listing ?? enthält die Klasse Magic mit der kürzesten Variante der Methode magic(). Prinzipiell sind alle anderen Anweisungen unnötig, inklusive der Benutzung der temporären Variablen x und z. Im Detail:

- Zeile 16: Dies ist eine Abkürzung, die allerdings keinen funktionalen Unterschied macht. Auf das If-Statement in Zeile 15 kann also verzichtet werden (das Statement durch den Else-Block ersetzt werden).
- Zeile 22: Toter Code, da die Bedingung des If-Statements in Zeile 18 immer wahr ist. Dieses If-Statement kann also ebenfalls entfernt werden (If-Statement durch Then-Block ersetzen).

- Zeilen 19, 26, 30, 32–35: Hier wird der ursprüngliche Wert von x (dieser sei x_0) in tmp gespeichert, dann mit y getauscht und in der Schleife wieder zurück gesetzt, sodass nach der Schleife y den Wert 0 und tmp den ursprünglichen Wert von x hat. Der Ausdruck in Zeile 39 vereinfacht sich, da z an dieser Stelle auch den ursprünglichen Wert von x (x_0) innehat, zu $2 \cdot x_0 - x_0 = x_0$. Vereinfacht man diesen Ausdruck somit zu „b = z;“, kann man all die Statements mit tmp incl. der Schleife entfernen.
- Zeile 29: Da an dieser Stelle y den ursprünglichen Wert von x, und damit a angenommen hat, evaluiert die If-Bedingung immer zu true. Das If-Statement kann also durch den entsprechenden Then-Block ersetzt werden.
- Schlussendlich kann man an Stelle der temporären Variablen x und z gleich a und b verwenden; außerdem ist die Variable y unnötig, da sie nur in Zusammenhang mit der Variablen tmp verwendet wird.

Listing 2: Magic vereinfacht

```
1 public class Magic {
2     int a, b;
3
4     public Magic(int a, int b) {
5         this.a = a;
6         this.b = b;
7     }
8
9     public void magic() {
10         a = a^b;
11         b = a^b;
12         a = a^b;
13     }
14 }
```

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

In einem in Java implementierten Kassensystem finden Sie die in Listing 2 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 3), welcher mehrere Positionen, Instanzen von `ReceiptPosition` (siehe Listing 4), 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 3: ReceiptPrinter

```
1 import java.io.File;
2 import java.io.FileInputStream;
3 import java.io.ObjectInputStream;
4 import java.math.BigDecimal;
5 import java.util.Date;
6
7 import model.Receipt;
8
9 public class ReceiptPrinter {
10     private Receipt receipt;
11
12     private Date loadDate;
13
14     public ReceiptPrinter(File file) throws Exception {
15         initLoadDate();
16         this.receipt = read(file);
17     }
18
19     private void initLoadDate() {
20         loadDate = new Date();
21     }
22
23     public void print(java.io.PrintStream ps) {
24         ps.print(toString());
25     }
26
27     @Override
28     public String toString() {
29         StringBuilder sb = new StringBuilder();
30         for (model.ReceiptPosition rp : receipt.getPositions()) {
31             sb.append(rp.getProduct())
32               .append(": ")
33               .append(rp.getPrice()).append("\n");
34         }
35         sb.append("=====\n")
36           .append(computePrice())
37           .append("\n\n").append(loadDate).append(" ");
38         return sb.toString();
39     }
40
41     private BigDecimal computePrice() {
42         BigDecimal sum = BigDecimal.ZERO;
43         for (model.ReceiptPosition rp : receipt.getPositions()) {
44             sum = sum.add(rp.getPrice());
45         }
46         return sum;
47     }
48
49     private Receipt read(File file) throws Exception {
50         ObjectInputStream in = new ObjectInputStream(new FileInputStream(file));
51         try {
52             Object read = in.readObject();
53             if (read instanceof Receipt) {
54                 return (Receipt)read;
55             }
56             else {
57                 throw new Exception("No Receipt found.");
58             }
59         }
60         finally {
61             in.close();
62         }
63     }
64 }
```

Listing 4: Receipt

```
1 package model;
2
3 import java.io.Serializable;
4 import java.util.LinkedList;
5 import java.util.List;
6
7 public class Receipt implements Serializable {
8     private static final long serialVersionUID = -8760949217949784771L;
9
10    private final List<ReceiptPosition> positions = new LinkedList<ReceiptPosition>();
11
12    public void addPosition(ReceiptPosition rp) {
13        if (rp != null) {
14            positions.add(rp);
15        }
16    }
17
18    public ReceiptPosition[] getPositions() {
19        return positions.toArray(new ReceiptPosition[positions.size()]);
20    }
21 }
```

Listing 5: ReceiptPosition

```
1 package model;
2
3 import java.io.Serializable;
4 import java.math.BigDecimal;
5
6 public class ReceiptPosition implements Serializable {
7     private static final long serialVersionUID = 5269887755509983958L;
8
9     private final String product;
10
11    private final BigDecimal price;
12
13    public ReceiptPosition(String product, BigDecimal price) {
14        this.product = product;
15        this.price = price;
16    }
17
18    public String getProduct() {
19        return product;
20    }
21
22    public BigDecimal getPrice() {
23        return price;
24    }
25 }
```

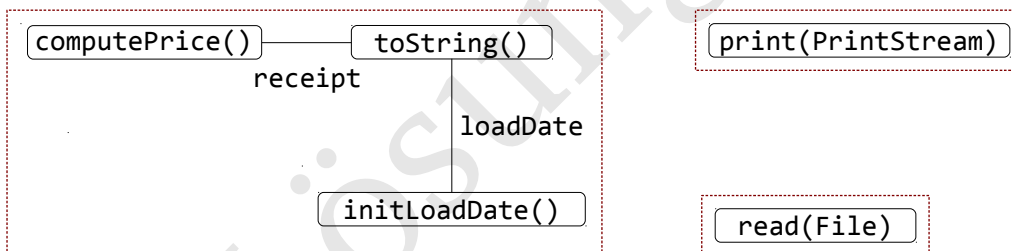
Lösung zu a):

- Java
 - `java.io.File`
 - `java.io.FileInputStream`
 - `java.io.ObjectInputStream`
 - `java.math.BigDecimal`
 - `java.util.Date`
 - `java.lang.Object`
 - `java.io.PrintStream`
 - `java.lang.Override`
 - `java.lang.String`
 - `java.lang.StringBuilder`
 - `java.lang.Exception`
- Anwendung
 - `model.Receipt`
 - `model.ReceiptPosition`

Lösung zu b):

Die Klasse `ReceiptPrinter` benötigt zur Erfüllung ihrer Funktionalität die Klassen `Receipt` und `ReceiptPosition`. Eine Wiederverwendung vom `ReceiptPrinter` ohne `Receipt` und `ReceiptPosition` ist somit nicht sinnvoll. Da alle drei Klassen selbst lediglich Abhängigkeiten zur Java-API haben, können sie leicht gemeinsam in anderen Projekten wiederverwendet werden.

Lösung zu c):



LCOM = 3 von 5. **Hinweis:** Der Konstruktor der Klasse, `ReceiptPrinter(File)`, wurde absichtlich nicht im Diagramm übernommen (siehe Bemerkung in der VL). Konstruktoren können dazu führen, dass eine Klasse sehr leicht einen (unberechtigten) LCOM-Wert von 1 erreicht.

Lösung zu d):

Der LCOM-Wert 3 von 5 deutet auf eine geringe Kohäsion hin. Dies ist zutreffend, da bspw. die Funktionalität zum Laden eines Kassenbelegs in der Methode `read(File)` völlig losgelöst von der eigentlichen Aufgabe der Klasse, der Ausgabe eines Kassenbelegs auf einem Ausgabemedium, ist.

Lösung zu e):

Nein, die Methoden sollten nicht über zwei Klassen verteilt werden, denn beide Methoden sind für die Erfüllung der Funktionalität erforderlich.