Hochschule
Bonn-Rhein-Sieg
University of Applied Sciences

Prof. Dr. Andreas Hackelöer

18. April 2024

Übungen zur Vorlesung Software Engineering I
Sommersemester 2024

Übungsblatt Nr. 2

(Abgabe in Teams von max. 3 Personen bis: Mittwoch, den 24. April 2024, **9:00 Uhr**)

Wichtige Anmerkungen:

- Bitte reichen Sie sämtliche Abgaben nur noch über LEA ein, auch den Quellcode! Sie können dafür aus Ihrem GitLab- oder GitHub-Repository, sofern Sie eines verwenden, die Funktion „Download ZIP“ verwenden und das ZIP anschließend bei LEA einstellen.

- Bitte als Kompressionsformat ausschließlich ZIP verwenden, nicht RAR oder tar.gz!

- Abbildungen / UML-Diagramme idealerweise als PDF oder wenigstens als PNG oder JPG einreichen, bitte jedoch nicht in einem (XML-)Rohformat eines UML-Editors! Falls das Werkzeug Ihrer Wahl keine Bild-Exportfunktion bietet, können Sie dafür auch einfach die Taste PrintScrn/Drucken verwenden und den Screenshot anschließend in einem beliebigen Grafikprogramm bearbeiten.

Aufgabe 1 (Modellierung eines Abnahmeprozesses, 15 Punkte):

Insbesondere bei großen Software-Projekten, die nach dem Wasserfallmodell (vgl. Kapitel 2) entwickelt werden, kann die Abnahme (engl. *sign-off*) von Dokumenten eine hohe Komplexität besitzen. Üblicherweise ist der Abnahme eine Kontrolle (engl. *review*) vorgeschaltet, bei der alle beteiligten Stakeholder die Dokumente erhalten und Gelegenheit bekommen, diese im Dialog mit den Erzeugern der Dokumente zu überarbeiten.

Die Firma NullPointer Software GmbH bittet Sie um die Modellierung eines Prozesses für die Erstellung einer Anforderungsspezifikation (Fachkonzept), die als Ergebnis der Phase "Anforderungsentwicklung“ erstellt wird. Diese Spezifikation wird dann in der daran anschließenden Phase „Systementwurf“ einem Software-Architekten übergeben.

Beachten Sie für Ihr Prozessmodell die folgenden Vorgaben der Firma:

- Das Fachkonzept wird von einem *internen* Anforderungsingenieur (Requirements Engineer, RE) erstellt. Dieser Stakeholder soll dann den weiteren Abnahmeprozess durchführen.

- Es gibt eine Abteilung im Unternehmen namens „Project Management Office“ (PMO), das vom RE eingereichte Dokumente zunächst im Hinblick auf formale Unternehmensvorgaben überarbeitet (z.B. Layout). Im Anschluss werden die am Review beteiligten Stakeholder benachrichtigt.

- An einem Review sind die folgenden Stakeholder beteiligt, die parallel ein Review durchführen: Der Kunde (*interner* Projektleiter), ein externer Gutachter sowie die IT-Leiterin (intern). Jeder dieser Stakeholder übergibt nach dem Review seine Anmerkungen an das PMO. Sobald alle Bewertungen vorliegen, führt das PMO *innerhalb einer Aktivität* die Sichtung der Reviews durch und generiert daraus eine Gesamtbewertung.

- Wenn dabei in dem Fachkonzept Fehler entdeckt werden, wird unterschieden nach kritischen und weniger kritischen Fehlern. Bei kritischen Fehlern soll nach Überarbeitung des Fachkonzepts durch den RE der gesamte Kontrollprozess erneut durchlaufen werden. Bei weniger kritischen Fehlern überarbeitet der RE das Fachkonzept ebenfalls, es gibt aber keine erneute Kontrolle durch die Stakeholder (und das PMO). In beiden Fällen gilt: Daran anschließend kann eine Abnahme *ohne Vorbehalt offener Punkte* durch das PMO vorbereitet werden. Letzteres gilt natürlich auch, wenn in dem Fachkonzept gar keine Fehler entdeckt werden.

- Es kann passieren, dass aus zeitlichen Gründen die Abnahme des Dokuments sofort erfolgen muss, obwohl noch kritische Fehler offen sind. Welche Art Abnahme kann das PMO in diesem Fall vornehmen? Berücksichtigen Sie diesen Fall in Ihrem Prozessmodell.

- Wenn das PMO das Fachkonzept freigeben will, muss es zunächst allen *internen* Stakeholdern einen „Letter of Acceptance“ vorlegen, die *alle* diese Stakeholder *parallel* unterschreiben müssen. Erst wenn *alle* Unterschriften vorliegen, kann das PMO das Fachkonzept an den Software-Architekten übergeben, der dann die nächste Phase (Systementwurf) einleiten kann.

Modellieren Sie Ihr Prozessmodell zur Abnahme eines Fachkonzepts bei der Null Pointer Software GmbH mittels eines *UML-Aktivitätsdiagramms*. Dokumente und Swimlanes müssen dabei nicht berücksichtigt werden. Die hier formulierten Fragen müssen nicht explizit beantwortet werden und dienen nur als Hilfestellung für die Erstellung des Modells. Informationen zum genannten Diagrammtyp finden Sie in diesem Buch ab Seite 263:

Rupp, Chris et al.: *UML 2 Glasklar – Praxiswissen für die UML-Modellierung*. Hanser Verlag, 2012 (5. Auflage).

Aufgabe 2 (Objektorientierter Entwurf in Java, 15 Punkte)

Eine Klasse `CardBox` soll implementiert werden, welche zur Laufzeit verschiedene Karteikarten von Personen (`PersonCard`-Objekte) innerhalb des `CardBox`-Objekts abspeichern kann. Diese `PersonCard`-Objekte implementieren das Interface `PersonCard`.

Das Interface können Sie von dem GitLab-Repository der Vorlesung herunterladen:

https://gitlab.com/hackeloeer/codesSS2024

Hierzu können Sie am einfachsten einen Pull von Git anweisen (git pull), um die Aktualisierungen herunterzuladen.

Es gibt zwei Arten von Personen-Karteikarten: Einmal `EnduserCard` und einmal `DeveloperCard`. Beide enthalten Nach- und Vorname (String), aber `DeveloperCard`s enthalten zusätzlich noch die Information, ob der Entwickler oder die Entwicklerin genug Kaffee hat (`hasEnoughCoffee`). Enduser benötigen keinen Kaffee, aber sie können hungrig sein (`isHungry`) oder auch nicht.

Die Klasse `CardBox` soll die folgenden funktionalen Anforderungen (FA) erfüllen:

FA1: Objekte vom Typ `PersonCard` können in einem Objekt der Klasse `CardBox` zur Laufzeit abgespeichert werden. Dafür sollten Sie konkrete Klassen `EnduserCard` und `DeveloperCard` bereitstellen, welche das Interface `PersonCard` implementieren. Bitte beachten Sie bei der Implementierung den Kommentar in dem Interface.

Eine Kontrolle, ob ein übergebenes `PersonCard`-Objekt mit einer ID bereits in dem `CardBox`-Objekt vorhanden ist, soll in der `CardBox`-Klasse implementiert werden. Falls eine bereits vorhandene ID hinzugefügt werden soll, soll eine *geprüfte* Exception vom Typ `CardBoxException` geworfen werden. Die Klasse `CardBoxException` soll selbst implementiert werden. Die Exception-Message soll diesen Text ausgeben:

„`Das CardBox-Objekt mit der ID [ID des Objekts] ist bereits vorhanden`“.

Die Spezifikation der Methode nach UML ist:

```
+ addPersonCard( personCard : PersonCard ) : void { throws CardboxException }
// Anmerkung: UML definiert keine explizite Semantik für Exceptions
```

FA2: Es gibt weiterhin eine Methode `deletePersonCard`, mit der Sie Objekte vom Typ `PersonCard` zur Laufzeit aus einem instanziierten Objekt der Klasse `CardBox` entfernen können. Diese Methode erhält dazu die ID des zu entfernenden Objekts.

Falls es zu der übergebenen ID kein vorhandenes Objekt in der `CardBox` gibt, soll über einen von Ihnen frei wählbaren Rückgabewert eine Fehlermeldung ausgegeben werden. Welche Nachteile ergeben sich aus Ihrer Sicht bei einer solchen Fehlerbehandlung gegenüber Exceptions? (Kurze Erläuterung)

Spezifikation der Methode nach UML:

```
+ deletePersonCard( id : int ) : String
```

FA3: Es soll mit der Methode `showContent` möglich sein, den gesamten Inhalt eines `CardBox`-Objekts auf der Konsole auszugeben. Dabei soll zu allen `PersonCard`-Objekten ID, Vor- und Nachname ausgegeben werden, zu Entwicklern aber auch der Kaffeestatus und zu Endusern der Hungrigkeitsstatus. Implementieren Sie diese Methode so, dass Sie dort keine Fallunterscheidung nach Typ des gerade auszugebenden `PersonCard`-Objekts treffen müssen.
Beispielausgabe für ein `DeveloperCard`-Objekt:
`„ID = [ID], Vorname = [Vorname], Nachname = [Nachname], hasEnoughCoffee = [Kaffeestatus]“`

Spezifikation der Methode nach UML:

```
+ showContent( ) : void
```

FA4: Es gibt eine Methode `size`, welche die Anzahl aller Objekte in einer `CardBox`-Instanz zurückliefert.

Spezifikation der Methode nach UML:

```
+ size( ) : int
```

Weiteres:
1.
Zur Umsetzung dieser Anforderungen müssen Sie in der `CardBox`-Klasse eine Listen-Datenstruktur aus der Java-Klassenbibliothek verwenden, um Objekte intern abzuspeichern. Suchen Sie dazu in dem Package java.util.* nach geeigneten Strukturen. Bitte keine eigene Listen-Implementierung umsetzen! Die Datenstruktur `HashMap` darf *nicht* verwendet werden.
2.
Testen Sie die Implementierung der Klasse `CardBox` mittels einer externen Testklasse (die Nutzung von JUnit 5 ist von Vorteil). Überlegen Sie, welche Testfälle für eine ausreichende Testabdeckung vonnöten sind und setzen Sie diese um.