Hochschule
Bonn-Rhein-Sieg
University of Applied Sciences

Prof. Dr. Sascha Alda
Moritz Balg, Dr. Rita Cornely

8. Oktober 2024

Übungen zur Vorlesung Software Engineering I
WS 2024 / 2025

Übungsblatt Nr. 2
(Abgabe bis: Mittwoch, den 16. Oktober 2024, **09:00 Uhr**)

Aufgabe 1 (Modellierung eines Prozesses zur Abnahme, 15 Punkte):

Die Abnahme (engl.: *sign-off*) von Dokumenten ist insbesondere bei großen Software-Projekten, die nach den Vorgaben eines Wasserfall-Modells (vgl. Kapitel 2, ggf. vorarbeiten) entwickelt werden, ein oftmals unterschätzt komplexer Prozess. Einer offiziellen Abnahme geht in der Regel ein Kontrollprozess (engl.: *review*) voraus, in dem die *beteiligten* Stakeholder die Dokumente Korrektur lesen und ihre Kritiken äußern können.

Die Firma HollaDieWaldfee Software GmbH wünscht sich von ihnen eine ausführliche Modellierung eines Prozesses eines Fachfeinkonzepts (Technisches Design), das üblicherweise als Ergebnis der Phase „System Entwurf“ in der Firma erstellt wird. Nehmen sie für Ihr Prozessmodell folgende Anforderungen und Überlegungen mit in Betracht, die von der Firma gestellt wurden:

- Das Fachfeinkonzept wird von einem *internen* Software-Architekten erstellt. Von diesem Stakeholder sollte dann der weitere Prozess zur Abnahme durchgeführt werden.

- Das Project Management Office (PMO) ist verantwortlich, nach Erhalt von den zu kontrollierenden Dokumenten durch den Software-Architekten, diese zunächst aufzuarbeiten (z.B. Dokumente in ein einheitliches Format bringen) und in einem *weiteren* Schritt diese auf einem internen Projekt-Sharepoint abzulegen. Anschließend werden die betreffenden Stakeholder benachrichtigt, die an einem Review beteiligt sind.

- An einem Review seien folgende Stakeholder beteiligt, welche *parallel* ein Review durchführen: der Kunde (interner Projektleiter), ein externer Gutachter sowie der IT-Leiter (intern). Jeder Stakeholder übergibt nach seinem Review seine Anmerkungen an das PMO. Erst wenn *alle* Bewertungen eingetroffen sind, bereitet das PMO, innerhalb *einer* Aktivität, die Reviews auf und führt eine Gesamtbewertung aus.

- Was passiert, wenn in dem Fachfeinkonzept Fehler entdeckt werden? Beachten sie dabei die Fallunterscheidung bzgl. kritischer und wenig kritischer Fehler. Bei dem Vorhandensein von *kritischen* Fehlern soll, nach einer entsprechenden Überarbeitung des Fachfeinkonzepts durch den Software-Architekten, der Kontrollprozess erneut durchgeführt werden. Bei *wenig kritischen* Fehlern erfolgt ebenfalls eine Überarbeitung, aber ohne erneute Kontrolle. Anschließend kann eine Abnahme *ohne* Vorbehalt

offener Punkte durch das PMO vorbereitet werden. Letzteres wird natürlich auch vorbereitet, wenn keine Fehler vorliegen.

- Was passiert, wenn aus zeitlichen Gründen gemäß Projektplan die Abnahme des Dokuments dringend erfolgen muss, es jedoch immer noch kritische Fehler in der Spezifikation gibt? Welche Art Abnahme muss man in dem Kontext spezifizieren? Hier ist sich die Firma nicht einig und fordert einen Vorschlag ihrerseits.

- Wer ist für die finale Abnahme zuständig und wie wird diese eingeholt? Alle relevanten *internen* Stakeholder müssen dabei einen zuvor durch das PMO ausgegebenen „Letter of Acceptance“ (inklusive Abnahmeprotokoll) *parallel* unterschreiben. Erst wenn *alle* Unterschriften eingeholt sind, kann das PMO die nächste Phase („Implementierung“) einleiten.

Skizzieren Sie ihr Prozessmodell zur Abnahme eines Fachfeinkonzepts mit Hilfe eines Aktivitätsdiagramms von der UML. Dokumente und Swimlanes brauchen sie in ihrem Modell keine zu berücksichtigen. Die in dieser Aufgabe formulierten Fragen dienen als Hilfestellung für die Erstellung des Modells und müssen nicht explizit beantwortet werden. Informationen zu diesem Diagrammtyp finden sie in dem Buch:

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

Dieses Buch wird in dieser Vorlesung *ausschließlich* als Referenz verwendet! Bitte verwenden sie keine „Online“-Quellen (Wiki etc.)! Eine erste Einführung dazu findet in Woche KW 41-2024 (9.10.2024) in den Übungen statt.

Aufgabe 2 (Intensivierung Java und Objektorientierung, 15 Punkte)

Es soll ihre Aufgabe sein, eine Klasse `Container` zu implementieren, die zur Laufzeit verschiedene (mehrere) „Member-Objekte“ aufnehmen und *intern innerhalb* des `Container`-Objekts abspeichern kann. Diese „Member-Objekte“ implementieren das Interface `Member`.

Das Interface können sie über das GitHub-Repository der Vorlesung beziehen (bitte auch den Kommentar über das Interface beachten!):

https://github.com/aldaGit/codesSEws24

Hierzu die neuen Codes über einen Pull beziehen. Eine kleine Anleitung dazu finden sie im vierten Teil des Tutoriums zur Integration von GitHub in IntelliJ:

https://www.youtube.com/watch?v=I4L0k33TNQ4

Die Klasse `Container` soll folgende (funktionale) Anforderungen (*FA*) erfüllen:

FA1: Es soll möglich sein, Objekte vom Typ `Member` in einem instanziierten Objekt der Klasse `Container` zur Laufzeit abzuspeichern. Um zur Laufzeit Objekte vom Typ `Member` zu erzeugen, sollten sie eine entsprechende Klasse (z.B. `ConcreteMember`) bereitstellen, welches das Interface `Member` implementiert. Bitte bei der Implementierung auch den Kommentar aus dem Interface beachten.

Eine Kontrolle, ob ein übergebenes `Member`-Objekt mit einer ID bereits in dem `Container`-Objekt enthalten ist, sollte ebenfalls in der `Container` -Klasse implementiert werden. Falls ein `Member`-Objekt mit einer bereits vorhandenen ID übergeben wird, so soll eine *geprüfte* Exception vom Typ `ContainerException` ausgeworfen werden. Die Klasse `ContainerException` bitte selbst implementieren! Die zugehörige Message dieser Exception sollte folgendes *exakt* ausgeben:

„Das Member-Objekt mit der ID [hier die ID des Objekts] ist bereits vorhanden!“

Spezifikation der Methode nach der UML:

```
+ addMember( member : Member ) : void {throws ContainerException}
// Bemerkung: streng genommen kennt die UML keine Exceptions
```

FA2: Es soll mit der Methode `deleteMember` möglich sein, Objekte vom Typ `Member` in einem instanziierten Objekt der Klasse `Container` zur Laufzeit zu löschen. Dazu muss die eindeutige ID des Objekts vom Typ `Member` dem `Container`-Objekt übergeben werden. Falls zu der übergebenen ID kein `Member`-Objekt gespeichert ist, sollte über einen von *ihnen freiwählbaren* Rückgabewert eine entsprechende Fehlermeldung ausgegeben werden. Welche Nachteile ergeben sich aus ihrer Sicht für ein solchen Fehlerhandling gegenüber einer Lösung mit Exceptions? Kurzes Statement!

Spezifikation der Methode nach der UML:

```
+ deleteMember( id : Integer ) : String
```

FA3: Es soll mit einer Methode `dump` möglich sein, die IDs der aktuell abgespeicherten Objekte vom Typ `Member` auszugeben (hier: direkt auf der Console.). Für die Ausgabe der ID eines einzelnen `Member`-Objekts soll innerhalb der Methode `dump` die Methode `toString()` des jeweiligen `Member`-Objekts verwendet werden, die dazu aus der Klasse `java.lang.Object` überschrieben werden muss. Die Ausgabe, die bei `toString()` erfolgt, sollte wie folgt für ein `Member`-Objekt aussehen:

„Member (ID = [hier die ID des Members])“

Spezifikation der Methode nach UML:

```
+ dump( ) : void
```

FA4: Es soll möglich sein, die Anzahl der aktuell abgespeicherten Objekte vom Typ `Member` in einem `Container`-Objekt mittels der Methode `size` zu ermitteln und an ein aufrufendes Objekt als Rückgabeparameter zu übergeben.

Spezifikation der Methode nach UML:

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

1.)
Zur Umsetzung dieser Anforderungen *müssen* sie in der Klasse `Container` eine Listen-Datenstruktur wiederverwenden, um Objekte intern abzuspeichern. Recherchieren sie dazu in dem Package `java.util.*` nach brauchbaren Strukturen. Bitte *keine* eigene Listen-Implementierung realisieren ;-) Diese List-Datenstruktur muss intern als Objektvariable bereitgestellt werden. Die Datenstruktur `HashMap` darf *nicht* verwendet werden.

2.)
Testen sie die Implementierung der Klasse `Container` mit Hilfe einer externen Testklasse (JUnit5 kann verwendet werden, falls bekannt). Erzeugen sie dabei zwei verschiedene `Member` -Objekte und testen sie ihre Implementierung *hinreichend* mit mindestens 10 aufeinanderfolgenden Testfällen (bei JUnit: 10 Assertions).

Hinweis dazu: formulieren sie ihre Testfälle auf Basis der Anzahl von intern gespeicherten `Member` –Objekten (= den Zustand des Container-Objekts), die beim Hinzufügen / Löschen von Objekten variieren kann (weitere Hinweise dazu in der Übung). Dokumentieren sie ihre Testfälle zusätzlich mit dem Excel-Template (LEA → Kapitel 1 → Material, Tab „Simple Test Suite“). Bitte auch Äquivalenzklassen definieren.

3.)
Alle Klassen und Interfaces und mögliche Unterpackages ihrer Lösung sollen in einem *neuen* Package `org.hbrs.se1.ws24.uebung2` abgelegt werden.