Hochschule
Bonn-Rhein-Sieg
University of Applied Sciences

Prof. Dr. Sascha Alda
Moritz Balg und Dr. Cornely

4. Dezember 2024

Übungen zur Vorlesung Software Engineering I
WS 2024 / 2025

Übungsblatt Nr. 7
(Abgabe 7-1 bis: Mittwoch, den 11. Dezember 2024, **10:00 Uhr**)
(Abgabe 7-2 bis: Mittwoch, den 11. Dezember 2024, **21:00 Uhr**)

Aufgabe 1 (Entwicklung UML-Modelle, 15 Punkte):

Die Entwicklung eines UML Use Case Modells sowie eines UML-Klassendiagramms wird auf jeden Fall ein zentraler Bestandteil der Software Engineering I Klausur sein! Daher werden sie in diesem Aufgabenblatt nochmals auf Basis einer textuellen Use Case Beschreibung sowie User Stories UML-Modelle ableiten. Diese Anforderungen sind im Dokument „Fallstudie Coll@HBRS, v1.3“ beschrieben (siehe LEA). Ihre Aufgaben:

a.)
Entwickeln sie aus dem gegebenen textuellen Use Case sowie aus den User Stories (S. 2, bitte beachten!) *ein* UML Use Case Modell. Zeichnen sie auch die Systemgrenze ein. Achtung: das Use Case Modell ist „nicht besonders groß“ ;-) Die Use Case aus der Vorbedingung des textuellen Use Cases brauchen sie nicht zu berücksichtigen. Beeinflussen die User Stories das Use Case Modell? Antwort bitte kurz im Modell andeuten z.B. über einen Kommentar-Block.

b.)
Wenden sie auf den textuellen Use Cases die Abbott-Methode an, und entwickeln sie ein Klassendiagramm zur Darstellung der relevanten Analyse-Objekte. Jede Klasse sollte mit einem Objekttyp annotiert werden. Modellieren sie auch die wichtigsten Beziehungen zwischen den Klassen. Modellieren sie ihre Beziehungen ausgehend von der zentralen Control-Klasse „Jobangebot suchen“. Berücksichtigen sie nur die Datentransferobjekte, die sich explizit aus der Spezifikation des textuellen Use Case ergeben. Zwischen den Entities bitte auch die Kardinalitäten berücksichtigen (bitte auch hier insbesondere die Stories beachten).

Aufgabe 2 (Pattern zur flexiblen Weitergabe von Reports, UML; 15 Punkte)

Es soll ein Anwendungsfall „Sende Reports“ weiter entworfen werfen. Konkret sollen in Zukunft zwei Formen von Report-Objekten verschickt werden können: `SkillReport`-Objekte und `CompanyReport`-Objekte. Für jeden `Report`-Typ soll es jeweils eine eigene konkrete `Publisher`-Klasse (z.B. `SkillReportPublisher`) geben, welche *aktuell registrierte* `Consumer`-Objekte nach der Erzeugung eines `Report`-Objekts benachrichtigt. Die Benachrichtigung soll über eine Methode `update` erfolgen, welche von den `Consumer`-Objekten implementiert werden muss.

Die `Report`-Objekte sollen in den Publisher-Objekten abgespeichert werden. Die Erzeugung der `Report`-Objekte erfolgt intern in den `Publisher`-Objekten durch eine von dem `Publisher` angebotene Methode `produce`.

`Consumer`-Objekte (z.B. `SkillConsumer`) können die `Report`-Objekte *bei Bedarf* beziehen, falls das zugehörige Topic („Thema“) des `Report`-Objekts passend ist. Dazu kann ein `Consumer`-Objekt eine beliebige Anzahl von relevanten Topics *intern* definieren (z.B. in Form einer internen Liste). Das Topic wird durch ein String-Objekt repräsentiert. Bei `SkillReport`-Objekten entspricht das Topic dem Attribut `skill` (z.B. Java, C++). Ein `SkillReport`-Objekt hat zudem das Attribut `studentID` sowie eine `reportID`.

`Consumer`-Objekte können durch eine externe `Main`-Klasse bei den `Publisher`-Objekten *flexibel* registriert oder aber auch de-registriert werden. Die Speicherung der Consumer-Objekte sollte in den Publishern durch eine Datenstruktur realisiert werden.

Ihre Aufgaben:
a.) (Bearbeitung auch in der Übung am 11.12.2024)
Modellieren sie das aus der obigen Beschreibung resultierende UML-Klassen-Diagramm. Berücksichtigen sie auch Methoden und eventuelle Attribute der Klassen; modellieren sie die Signaturen der Methoden exakt. Objekt-Typen brauchen sie keine zu annotieren. Die Klasse `CompanyReport` brauchen sie nicht weiter zu spezifizieren, sollte aber als Klasse in das Diagramm aufgenommen werden. Denken Sie auch über Generalisierungen nach wie z.B. die Ableitung von abstrakten Klassen oder Interfaces.

b.)
Modellieren sie die Interaktionen der oben beschriebenen Objekte als *konkretes* Szenario in Form *eines* UML-Sequenz-Diagramms. Verwenden sie hierzu die *synchrone* Interaktion. Akteure brauchen sie keine zu modellieren. In diesem Sequenz-Diagramm sollen drei verschiedene Interaktionen hintereinander (sequentiell) verdeutlicht werden:

- Die Erzeugung und die Registrierung eines Consumer-Objekts in einen Publisher durch eine Main-Klasse. Sie können davon ausgehen, dass der Publisher schon existiert und nicht neu erzeugt werden muss.

- Das Erzeugen *eines* SkillReport-Objekts nebst den anschließenden Interaktionen zur Benachrichtigung und zum Beziehen des SkillReport-Objekts durch das Consumer-Objekt. Bei der Erzeugung eines SkillReport-Objekts sollten die Werte zu den Attributen beispielhaft in dem Konstruktor-Aufruf (`new`(...)) übergeben werden. Das interne Abspeichern von Objekten brauchen sie nicht modellieren.

- Die De-Registrierung des erzeugten Consumer-Objekts durch eine Main-Klasse

c.)
Modellieren sie ein UML-Package Diagramm. Gehen sie davon aus, dass das `Consumer`-Interface nebst Klasse, welche das Interface implementiert, in einem separaten „UI“-Package liegt. Alle anderen Klassen liegen in einem „AL-Package“. Die Packages sollten die Klasse bzw. Interfaces explizit aufführen. Berücksichtigen sie auch elementare <<import>>-Beziehungen *zwischen* den Packages.

Hinweis: Das hier verwendete Muster zur Benachrichtigung von Objekten über eine Zustandsänderung wird als Observer Pattern bezeichnet ([Gamma, 1995], vgl. Kapitel 6, SE-1). Dies ist ein grundlegendes Muster zur flexiblen Interaktion von Objekten sowie von Subsystemen, das in vielen Bereichen verwendet wird! Sie sollten sich, durch eine erste Recherche, mit den grundlegenden Aspekten des Musters vertraut machen! Als geeignete Quelle verweise ich auf (Brügge und Dutoit, 2013), Anhang A.7.