UNIVERSITÄT ZU LÜBECK
INSTITUTE FOR SOFTWARE ENGINEERING
AND PROGRAMMING LANGUAGES

Software Engineering im Wintersemester 2021/2022
Prof. Dr. Martin Leucker, Malte Schmitz, Stefan Benox, Julian Schulz, Benedikt Stepanek, Friederike Weilbeer, Tom Wetterich

Übungszettel 6 (Lösungsvorschlag)

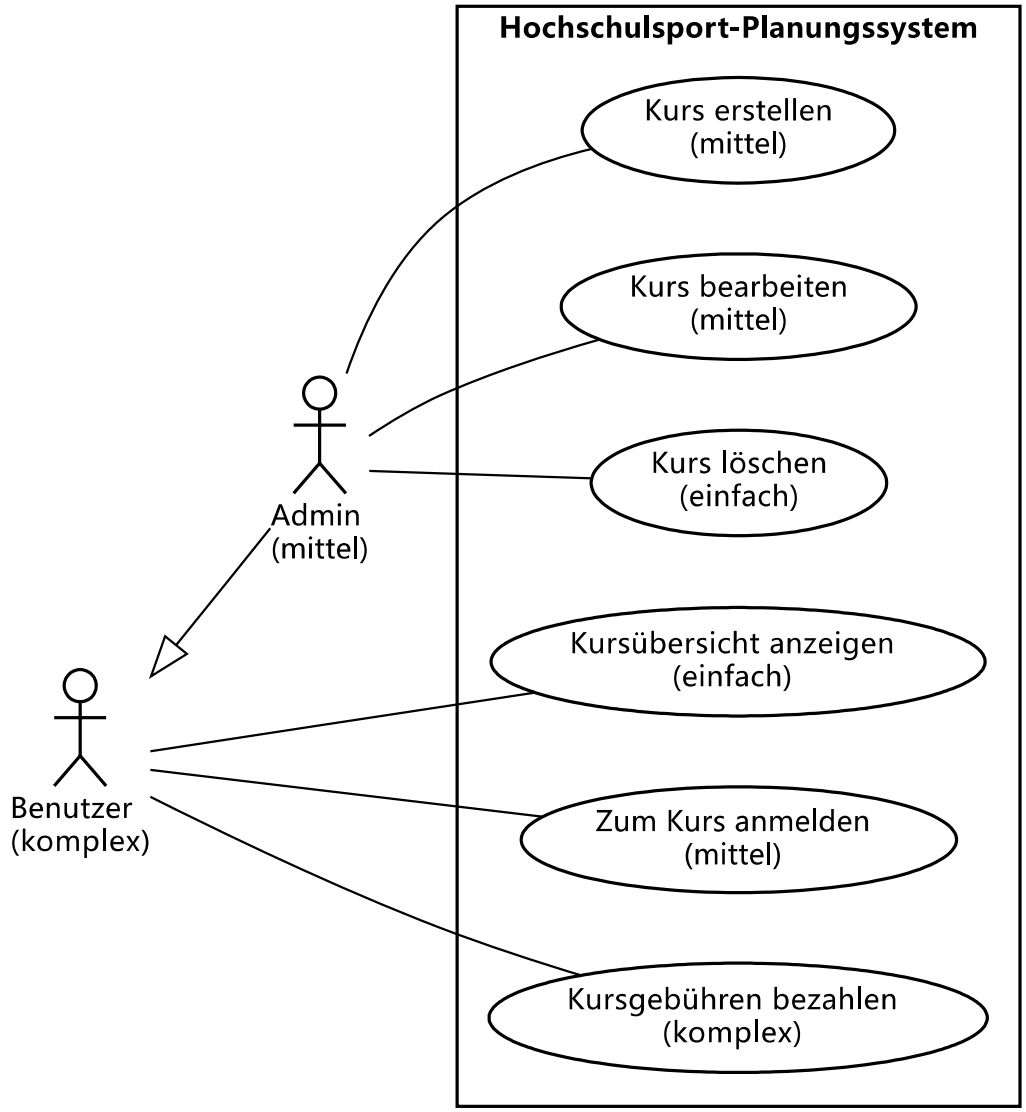
02.12.2021

Abgabe bis Donnerstag, 2. Dezember um 23:59 Uhr online im Moodle.

Aufgabe 6.1: Use-Case-Methode

4 Punkte, leicht

Gegeben sei folgendes Use-Case Diagramm:



▶ Quelltext des Diagramms

Weiter wurden folgende Einflussfaktoren geschätzt:

Technische Einflüsse	**Schätzung**
Verteiltes System	1
Performanz	1
End-User Effizienz	0
Complex processing	0
Wiederverw. Code	1
Leicht installierbar	2
Leicht zu benutzen	2
Portierbar	0
Leicht veränderbar	3
Nebenläufig	1
Security Features	2

Technische Einflüsse	Schätzung
Zugriff von Dritten	2
Spezielle Schulung	1

Umgebungsbezogene Einflüsse	Schätzung
RUP ist bekannt	1
Anwendungserfahrung	3
OO-Erfahrung	2
Chef Analyst verfügbar	0
Motivation	4
Stabile Anforderungen	1
Teilzeit-Kräfte	0
Schwierige Programmiersprache	1

Führen Sie eine Aufwandsabschätzung mit der Use-Case-Methode durch. Berechnen Sie dazu UAW, UUCW, UUCP, TCF, ECF und UCP.

▼ Lösungsvorschlag

Wir betrachten im ersten Schritt die Akteure des Systems und bestimmen das Unadjusted-Actor-Weight (UAW):

$$\mathbf{UAW} = 2 + 3 = 5$$

Nun bestimmen wir aus den Use-Cases des Systems das Unadjusted Use-Case-Weight (UUCW):

$$\mathbf{UUCW} = 5 \cdot 2 + 10 \cdot 3 + 15 = 10 + 30 + 15 = 55$$

Die Unadjusted Use-Case-Points (UUCP) ergeben sich jetzt aus der Summe aller Schnittstellen-Punkte und aller Use-Case-Points:

$$\mathbf{UUCP} = \mathbf{UAW} + \mathbf{UUCW} = 5 + 55 = 60$$

Der Technische Faktor (TCF) ergibt sich als gewichtete Summe aller technischer Einflussfaktoren:

$$\mathbf{TCF} = 0.6 + 0.01 \cdot \sum_{i=1}^{13} t_i \cdot T_i = 0.6 + 0.01 \cdot 15 = 0.6 + 0.15 = 0.75$$

Der Environmental Factor (ECF) ergibt sich als gewichtete Summe aller umgebungsbezogener Einflussfaktoren:

$$\mathbf{ECF} = 1.4 - 0.03 \cdot \sum_{i=1}^{8} e_i \cdot E_i = 1.4 - 0.03 \cdot 10 = 1.4 - 0.3 = 1.1$$

Für die finale Berechnung der Use-Case-Points (UCP) gilt

$$\mathbf{UCP} = \mathbf{UUCP} \cdot \mathbf{TCF} \cdot \mathbf{ECF} = 60 \cdot 0.75 \cdot 1.1 = 45 \cdot 1.1 = 49.5$$

Beachten Sie, dass es nicht möglich ist, den zeitlichen Aufwand direkt aus den berechneten UCP abzuleiten. Die Idee ist, dass sich über verschiedene Projekte hinweg eine stabile Relation zwischen berechneten UCP und Arbeitsaufwand für das konkrete Team ergibt. Ohne das Team näher zu kennen, lässt sich keine konkrete Zeit benennen.

Die Use-Case-Methode dient insbesondere dazu, die Schätzungen von Arbeitsaufwand strukturiert durchzuführen. Die genauen Faktoren sind dabei eher willkürlich und nicht der Kern der Methode. Es geht vielmehr darum, bei jeder Schätzung alle und insbesondere immer die gleichen Aspekte zu bewerten, sodass sich über mehrere Schätzungen hinweg vergleichbare Ergebnisse einstellen.

Im konkreten Fall dieser Übungsaufgabe ist das Use-Case-Diagramm sehr grob. Der Verdacht liegt nahe, dass sich hinter den einzelnen Use-Cases noch sehr viele Detailaufgaben verbergen, die bislang nicht erfasst wurden. Wenn allerdings Use-Cases immer auf diesem abstrakten Niveau erfasst werden, kann sich trotzdem eine stabile Relation zwischen berechneten UCP und Arbeitsaufwand ergeben. Wenn allerdings beim nächsten Projekt alle Aufgaben detailliert im Use-Case-Diagram modelliert werden, so würden sich deutlich höhere und entsprechend unvergleichbare UCP ergeben.

Aufgabe 6.2: Function-Point-Methode

4 Punkte, leicht

Ein System zum Einschreiben in den Übungsbetrieb soll neu entwickelt werden. Folgende Anforderungen wurden vom Kunden genannt:

- Das System soll folgende Daten erfassen: Dozierende (einfach), deren Vorlesungen (einfach) und zugehörige Übungsgruppen (mittel) und welche Studierende sich in die Übungsgruppen eingeschrieben haben (mittel).
- Das System soll über eine Eingabemaske für Dozierende zum Einrichten der Übungsgruppen verfügen (einfach).
- Dozierende sollen die eingegebenen Daten auch wieder abrufen (einfach) und ändern (einfach) können.
- Studierende können sich in Übungsgruppen einschreiben (mittel).
- Dozierende können einsehen, welche Studierende in Ihre Gruppen eingeschrieben sind (mittel).
- Dozierende können die Liste der Studierenden in Ihren Übungsgruppen editieren und zum Beispiel Studierende in andere Übungsgruppen verschieben (komplex).
- Studierende und Dozierende werden über jede Änderung per E-Mail benachrichtigt (mittel).
- Für die Abrechnung im Rahmen der Lehrverpflichtungsverordnung (LVVO) kann das System ausgeben, welcher Dozierende für welche Vorlesung wieviele Übungsgruppen angeboten hat und wie diese ausgelastet waren (mittel).
- Für die Planung des kommenden Semesters speichert das System die Auslastung in vergangenen Semestern als Referenz (mittel) und kann nach Eingabe der Studierendenzahlen (mittel) automatisch berechnen, wieviele Übungsgruppen im kommenden Semester angeboten werden müssen (komplex).

Folgende Einflussfaktoren wurden geschätzt:

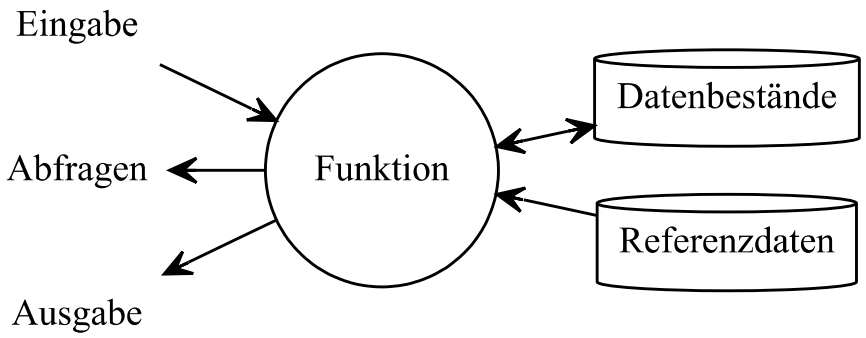
Faktor	Wert
Datenkommunikation	3
Verteilte Funktionen	3
Performanz Anforderungen	1
Hardware Konfiguration	0
Hohe Rate an Transaktionen	2
Dateneingang über Onlinefunktionen	3
End-user Effizienz	3

Faktor	Wert
Online updates	0
Komplexe Berechnungen	1
Wiederverwendbarkeit	3
Einfachheit der Installation	2
Einfachheit des Betriebs des Systems	2
Verschiedene technische Plattformen	0
Änderbarkeit des Codes	5

Schätzen Sie mit der Function-Point-Methode den Aufwand für dieses Projekt.

▼ **Lösungsvorschlag**

Das Diagramm auf Folie Plan-27 zeigt die verschiedenen Kategorien:



▶ Quelltext des Diagramms

- **Datenbestände** sind Daten, die von der Funktion gelesen und geschrieben werden. (Der Pfeil im Diagramm geht in beide Richtungen.)
- **Referenzdaten** sind Daten, die von der Funktion nur gelesen, aber nicht aktualisiert werden. (Der Pfeil im Diagramm geht nur in eine Richtung.)
- Bei einer **Eingabe** werden Daten von der Funktion eingelesen.
- Bei einer **Abfrage** werden Daten von der Funktion ausgegeben.
- Bei einer **Ausgabe** werden Daten von der Funktion ausgelesen und über eine bloße Abfrage hinaus analysiert und aufbereitet.

Die Unterscheidung zwischen Abfragen und Ausgaben ist nicht immer ganz klar. Wikipedia erläutert:

> Eine Ausgabe oder Abfrage haben den Hauptzweck der Präsentation von Informationen an der Anwendungsgrenze. Für eine Ausgabe ist zusätzlich gefordert, dass ihre Verarbeitungslogik mathematische Berechnungen oder Formeln, die Bildung abgeleiteter Daten, die Pflege eines internen Datenbestands oder eine Veränderung des Systemverhaltens beinhaltet.

Eine Abfrage ist also eine reine Abfrage von Daten aus dem Datenbestand ohne weitergehende Aufbereitung oder Analyse der Daten. Eine Ausgabe hingegen beinhaltet zusätzlich zum Auslesen der Daten aus dem Datenbestand auch noch eine Verarbeitung und Aufbereitung der Daten. Entsprechend wird eine Ausgabe in der Tabelle auf Folie Plan-29 auch jeweils mit einem Punkt mehr gewichtet als die einfachere Abfrage:

Kategorie	niedrig	mittel	hoch
Datenbestände	7	10	15
Referenzdaten	5	7	10
Eingabe	3	4	6
Abfrage	3	4	6
Ausgabe	4	5	7

Für die Bestimmung der Function-Points ergibt sich mit dieser Tabelle:

Kategorie	Anforderung	Anzahl	Grad	Gewicht	Summe
Datenbestände	Dozierende, Vorlesungen	2	einfach	7	14
Datenbestände	Übungsgruppen, Studierende	2	mittel	10	20
Datenbestände		0	komplex	15	0
Referenzdaten		0	einfach	5	0
Referenzdaten	Vergangene Auslastungen	1	mittel	7	7
Referenzdaten		0	komplex	10	0
Eingabe	Übungsgruppen anlegen, ändern	2	einfach	3	6
Eingabe	Einschreiben, Studierendenzahlen	2	mittel	4	8
Eingabe	Studierende verschieben	1	komplex	6	6

Kategorie	Anforderung	Anzahl	Grad	Gewicht	Summe
Abfrage	Übungsgruppen anzeigen	1	einfach	3	3
Abfrage	Studierende anzeigen, E-Mails	2	mittel	4	8
Abfrage		0	komplex	6	0
Ausgabe		0	einfach	4	0
Ausgabe	LVVO-Ausgabe	1	mittel	5	5
Ausgabe	Prognose	1	komplex	7	7
Summe					84

Wir erhalten also $\mathbf{FP_1} = 84$.

Für die Einflussfaktoren ergibt sich

$$\mathbf{FP_2} = 0.65 + 0.01 \cdot \sum_{i=1}^{14} g_i = 0.65 + 0.01 \cdot 28 = 0.93$$

Insgesamt erhalten wir damit

$$\mathbf{FP} = \mathbf{FP_1} \cdot \mathbf{FP_2} = 84 \cdot 0.93 \approx 78$$

Genau wie bei der Use-Case-Methode ist es auch hier weder möglich, noch sinnvoll, den zeitlichen Aufwand direkt aus den berechneten Function-Points abzuleiten. Auch hier ist die Idee, dass sich im Laufe der Zeit bei kontinuierlicher Anwendung der Methode ein firmenspezifischer Umrechnungsfaktor ergibt.

Genau wie die Use-Case-Methode ist auch die Function-Point-Methode eine Möglichkeit, die Schätzungen von Arbeitsaufwand strukturiert durchzuführen. Die genauen Faktoren sind dabei eher willkürlich und nicht der Kern der Methode. Es geht vielmehr darum, bei jeder Schätzung alle und insbesondere immer die gleichen Aspekte zu bewerten, sodass sich über mehrere Schätzungen hinweg vergleichbare Ergebnisse einstellen.

Im Vergleich zur Use-Case-Methode ist die Funtion-Point-Methode etwas älter und legt daher den Schwerpunkt mehr auf die Eingabe, Verarbeitung und Ausgabe der Daten als auf die Akteure, die mit dem System interagieren. Da die konkrete Ausgestaltung der Methoden und dabei insbesondere die Gewichtung und Auswahl der Faktoren eh für den Einsatz in einem Unternehmen an die tatsächlichen Gegebenheiten angepasst werden müssen, kann dies auf der

Grundlage beider Methoden erfolgen. Wenn bereits systematisch Use-Case-Diagramme erfasst werden, können diese als Basis zwar bei der Use-Case-Methode besser genutzt werden, aber bei näherer Betrachtung werden auch bei der Function-Point-Methode Funktionen des Systems als Basis der Bewertung herangezogen, sodass diese Fokussierung auf Use-Cases nicht als gewichtiger Unterschied der Methoden bezeichnet werden kann.

Aufgabe 6.3: LTL über linearen Läufen

4 Punkte, mittel

Geben Sie für die folgenden Eigenschaften LTL-Formeln an, sodass geanu die Läufe, die die jeweilige Eigenschaft erfüllen auch Modell der zugehörigen Formel sind. Hierbei gilt für die Menge der Propositionen $\mathbf{AP} = \{a, b, c\}$ und für das Alphabet $\Sigma = 2^{\mathbf{AP}}$.

1. Das Wort w erfüllt die Eigenschaft $\exists i, j \in \mathbb{N} : i \neq j \wedge w_i = w_j = \{a, b, c\}$.

   ▼ **Lösungsvorschlag**

   Es muss zwei beliebige, aber unterschiedliche Positionen i und j im Wort geben, an denen eine Eigenschaft φ erfüllt ist. Das kann mit der Eigenschaft $\mathcal{F}(\varphi \wedge \mathcal{X}\,\mathcal{F}\,\varphi)$ erfüllt werden, die nach einer Position sucht, an der φ und irgendwann später nochmal φ gilt. An den Positionen soll nun $w_i = w_j = \{a, b, c\}$ gelten, d.h. die Propositionen a, b und c müssen gelten. Mit $\varphi = a \wedge b \wedge c$ erhalten wir:

   $$\mathcal{F}(a \wedge b \wedge c \wedge \mathcal{X}\,\mathcal{F}(a \wedge b \wedge c))$$

2. Die Eigenschaft a *gilt solange bis* b *gilt*, muss solange gelten bis c gilt.

   ▼ **Lösungsvorschlag**

   Die Eigenschaft a *gilt solange bis* b *gilt* lässt sich durch $\varphi = a\,\mathcal{U}\,b$ ausdrücken. Nun soll φ gelten bis c gilt, also erhalten wir $\varphi\,\mathcal{U}\,c$ und damit insgesamt:

$$(a \mathcal{U} b) \mathcal{U} c$$

3. Immer wenn a gilt muss im vorherigen Schritt b gelten.

▼ **Lösungsvorschlag**

Wir können mit den aus der Vorlesung bekannten LTL-Operatoren nicht auf die vorherige Position im Wort zugreifen, deswegen müssen wir die Eigenschaft umgedreht formulieren: Wenn b nicht gilt, dann darf im nächsten Schritt nicht a gelten. Außerdem darf in der ersten Position nicht a gelten, denn es gab keine vorherige Position:

$$\neg a \wedge \mathcal{G}(\neg b \rightarrow \neg \mathcal{X} a)$$

Ergänzende Erläuterungen

Es gilt $\neg\varphi \rightarrow \neg\psi \equiv \psi \rightarrow \varphi$. (Dieser Zusammenhang ist als Kontraposition bekannt und leicht herzuleiten über $\psi \rightarrow \varphi \equiv \neg\psi \vee \varphi$.)

Entsprechend können wir die Eigenschaft auch durch folgende äquivalente Formel ausdrücken:

$$\neg a \wedge \mathcal{G}((\mathcal{X} a) \rightarrow b)$$

Intuitiv entspricht auch diese Formel der beschriebenen Eigenschaft: Wenn im nächsten Zustand a gilt, dann muss im aktuellen Zustand b gelten. Auch in dieser Variante ist die Forderung $\neg a$ für den ersten Zustand notwendig, denn die Implikation verlangt nur etwas, wenn im *nächsten* Zustand a gilt.

4. Es darf so lange nicht a gelten, bis von einem Zeitpunkt an solange b gegolten hat bis c gilt.

▼ **Lösungsvorschlag**

Die Eigenschaft b *gilt solange bis* c *gilt* lässt sich ausdrücken als $\varphi = b \mathcal{U} c$. Nun darf solange a nicht gelten, bis φ erfüllt ist, also:

$$(\neg a) \mathcal{U} (b \mathcal{U} c)$$

5. Entweder a gilt solange nicht, bis b nie mehr gilt oder c darf niemals gelten.

▼ **Lösungsvorschlag**

Die Eigenschaft b *gilt nie* kann als $\mathcal{G}\,\neg b$ ausgedrückt werden. Entsprechend erhalten wir $\neg a\,\mathcal{U}(\mathcal{G}\,\neg b)$ für den vorderen Teil der Eigenschaft. Man beachte dabei, dass die Negation in $\neg a$ sich nur auf die Proposition a bezieht. Zusammen mit *oder* c *darf niemals gelten* ergibt sich:

$$(\neg a\,\mathcal{U}(\mathcal{G}\,\neg b)) \vee \mathcal{G}\,\neg c$$

6. Das Wort w erfüllt die Eigenschaft $w_2 = \{a, b\} \Leftrightarrow w_4 = \{a, b\}$.

▼ **Lösungsvorschlag**

Es soll genau dann an der zweiten Position des Wortes das Zeichen $\{a, b\}$ stehen, wenn es an der vierten Position des Wortes auch steht. Das Zeichen lässt sich als $a \wedge b \wedge \neg c$ beschreiben. Auf die zweite und vierte Position kann durch entsprechend viele Next-Operatoren zugegriffen werden und die Equivalenz wird direkt als aussagenlogischer Operator in LTL unterstützt:

$$\mathcal{X}\,\mathcal{X}((a \wedge b \wedge \neg c) \leftrightarrow \mathcal{X}\,\mathcal{X}(a \wedge b \wedge \neg c))$$

7. Es gilt unendlich oft a.

▼ **Lösungsvorschlag**

Wenn eine Eigenschaft im unendlich langen Wort an unendlich vielen Stellen gelten soll, dann muss an jeder Position gelten ($\mathcal{G}$), dass diese Eigenschaft in Zukunft wieder gelten wird ($\mathcal{F}$). In diesem Fall soll a unendlich oft erfüllt sein, sodass wir erhalten:

$$\mathcal{G}\,\mathcal{F}\,a$$

8. Es gibt einen Zeitpunkt, an dem a gilt und nach dem a nie mehr gilt, aber ab dem für immer b gilt, inklusive dem Zeitpunkt selber.

▼ **Lösungsvorschlag**

Ein solcher Übergang von einer Eigenschaft in die nächste lässt sich in LTL durch entsprechende Vierschachtelung von Operatoren erreichen.

Irgendwann ($\mathcal{F}$) muss a gelten und ab der nächsten Position ($\mathcal{X}$) darf a nie wieder gelten ($\mathcal{G}\,\neg a$). Außerdem muss ab dieser Position immer b gelten ($\mathcal{G}\,b$):

$$\mathcal{F}(a \wedge ((\mathcal{X}\,\mathcal{G}\,\neg a) \wedge \mathcal{G}\,b))$$