SE Besprechung

Übung 5 – Verträge, Aufwand- und Risikoschätzung

SE, 01.12.09

Dustin Wüest

Studierende nach PPO 2001

- Bitte so bald wie möglich bei mir **melden**
- Bei geringer Anzahl an Studierenden nach PPO 2001 findet für diese eine normale Schlussklausur und eine **zusätzliche mündliche Prüfung** statt
- Bei grösserer Anzahl an Studierenden nach PPO 2001 findet für diese eine **erweiterte Schlussklausur** statt

Aufgabe 5.1 – Verträge

- Schnittstellendefinition mit Verträgen
  - *@post result = ...* wenn die Methode einen Rückgabewert hat
- Automatische Überprüfung (mit Hilfe des Interfaces, bzw. der Methoden)
  - *tree = tree@pre* kann mit dem Interface nicht geprüft werden

Aufgabe 5.1 – Verträge

- Wenn eine Methode eine Zustandsvariable nicht verändert, so muss dies explizit zugesichert werden (allgemein Gültiges kann in Klasseninvariante verschoben werden)
  - children = children@pre (?)
- Gefahr implementierungsabhängiger Spezifikationen → nur solche Zustandsvariablen verwenden, welche eine Entsprechung im Problembereich / in der Anwendungsdomäne haben
  - ~~children = children@pre~~

Aufgabe 5.1 – Verträge

- Preconditions (Voraussetzungen)
- Postconditions (Ergebniszusicherungen)
- Invariants (Invarianten)
- Obligations (Verpflichtungen)

„Wenn Du **das** tust (@pre), dann garantiere ich Dir **diese** Ergebnisse (@post)“.

Etwas gilt über die ganze Zeit → Klasseninvariante (@inv).

„Wenn Du diese Methode aufrufst, musst Du auch **folgendes** tun (@obligation)“.

Aufgabe 5.1 – Verträge

- Voraussetzen (@pre), wenn es dem Aufrufer zugemutet werden kann
- Prüfen, wenn mit Falscheingaben gerechnet werden muss, (und nur wenn eine sinnvolle Behandlung von Fehlern möglich ist; sonst Exception werfen)

Aufgabe 5.1 – Verträge

- Tree ?

→ Üblich: z.B. Klasse *Node* mit *children* Attribut

→ In dieser Übung: eine Klasse *Tree* und eine Klasse *Node*

```
private Node root;                    //Wurzel
private Vector<Node> children;        //Speichert alle Knoten des Baumes
```

isEmpty() == true <=> children.size() == 0

getSize() = children.size()

exists(Node node) == true <=> node ε children

Link zwischen *child node* und *parent node*?

Ganz ohne Annahmen geht es nicht… (z.B. Regeln bei *attach()*)

Aufgabe 5.1 – Verträge

```
* …
* A newly created tree consists of a root node. This node can
* not be deleted.
*
* @inv exists(root) && getSize() >= 0
*        with a = sum of added nodes with attach() and
*             r = sum of removed nodes with remove()
*        holds getSize() = a – r
*/
public class Tree {
...
```

Garantiert, dass die Grösse des Baumes nur durch attach() und remove() verändert wird.

Aufgabe 5.1 – Verträge

```
* …
* @pre    –
*
* @post  result = (children.size() == 0)
*           && children = children@pre
*
* @return true if the tree is empty
*/
public boolean isEmpty() {...}
```


Aufgabe 5.1 – Verträge

```
* …
* @pre    –
*
* @post  result = (children.size() == 0)
*           && children = children@pre
*
* @return true if the tree is empty
*/
public boolean isEmpty() {...}
```

Implementations-abhängige Variable.
Besser wäre z.B.:
result = (getSize() == 0)
&& elements of the tree haven't changed

Aufgabe 5.1 – Verträge

```
* …
* @pre   –
*
* @post  result = # of nodes in tree
*           && elements of the tree haven't changed
*
* @return the number of nodes in the tree
*/
public int getSize() {...}
```

- Aufpassen vor Endlos-Rekursion! *result = getSize()* geht hier z.B. nicht. *getSize() = getSize()@pre* geht ebenfalls nicht.

Aufgabe 5.1 – Verträge

```
/*
 * @post  result = ( getSize() == 0 )
 */
public boolean isEmpty() {...}

/*
 * @post  if isEmpty() == true then result = 0 else result > 0
 */
public int getSize() {...}
```

zyklische Abhängigkeit
→Endlosschleife

Hier praktisch nur mit teilformaler Sprache möglich

Aufgabe 5.1 – Verträge

```
* …
* @pre   node != null && parent != null
*          && !exists(node)
*
* @post  getSize() = (getSize()@pre + 1)
*          && !isEmpty()
*          && exists(node)
* …
*/
public void attach( Node node, Node parent ) {...}
```


Aufgabe 5.1 – Verträge

```
* …
* @pre   node != null
*
* @post  !exists(node)
* …
*/
public void remove( Node node ) {...}
```

Aufgabe 5.1 – Verträge

```
* …
* @pre   node != null
*
* @post  if there exists k: k >= 0 and k < children.size() and
*            children.get(k) == node then result = true
*                                    else result = false endif
* …
*/
public boolean exists( Node node ) {...}
```


Aufgabe 5.1 – Verträge

```
* …
* @pre   node != null
*
* @post  if there exists k: k >= 0 and k < children.size() and
*            children.get(k) == node then result = true
*                                    else result = false endif
* …
*/
public boolean exists( Node node ) {...}
```

Implementations-abhängige Variable.
Besser wäre z.B.:
if *node ε tree* ***then*** *result == true*
else *result == false* ***endif***

Aufgabe 5.2 – Function Points

- FP Resultate

  Min: 9.81

  Max: 63.22

Aufgabe 5.2 – Function Points

- Dateneingaben
- Datenausgaben = 0
- Anfragen
- Externe Schnittstellen
- Interne Datenbestände

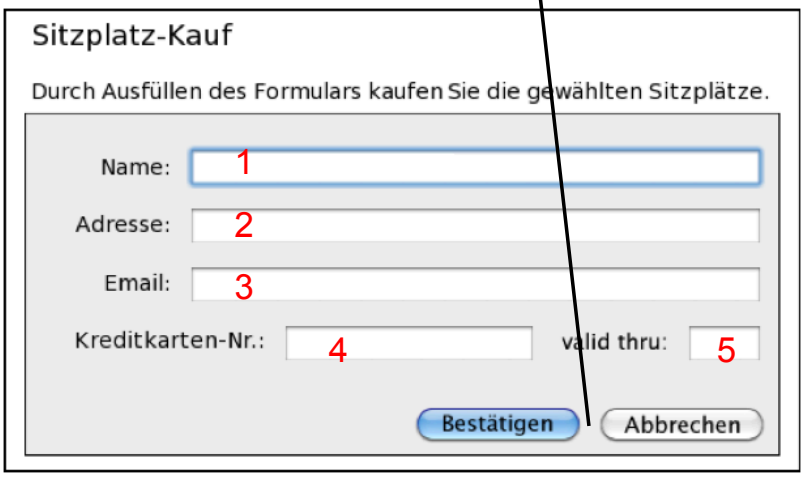
} (falls vorhanden → Annahme: *einfach*)

Element	Schwierigkeitsgrad einfach	mittel	komplex	Summe
Dateneingaben	_____ x 3 = _____	_____ x 4 = _____	_____ x 6 = _____	_____
Datenausgaben	_____ x 4 = _____	_____ x 5 = _____	_____ x 7 = _____	_____
Anfragen	_____ x 3 = _____	_____ x 4 = _____	_____ x 6 = _____	_____
Ext. Schnittstellen	_____ x 5 = _____	_____ x 7 = _____	_____ x10 = _____	_____
Int. Datenbestände	_____ x 7 = _____	_____ x10 = _____	_____ x15 = _____	_____
			Function Point Rohwert (UFP)	

Aufgabe 5.2 – Function Points

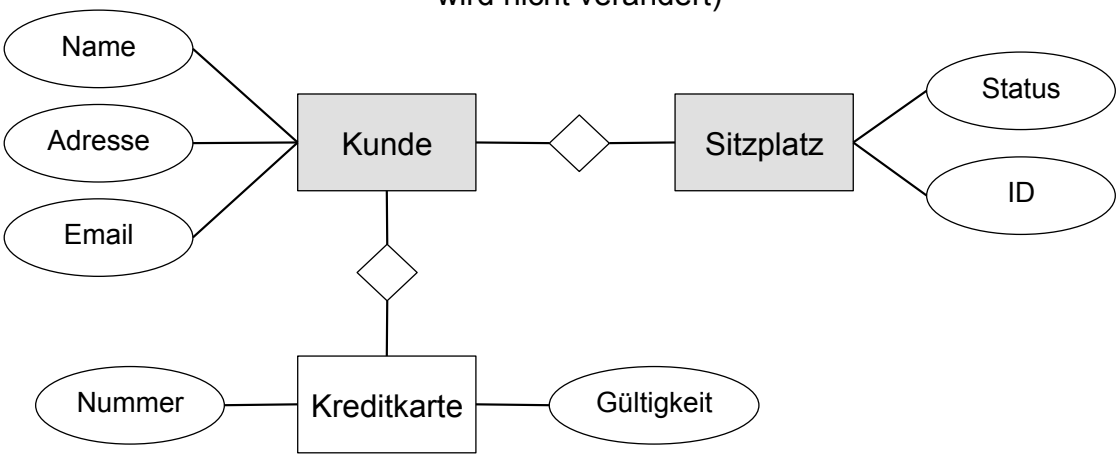
- Dateneingaben
  - 5 Datenelemente

- Bei Klick auf *Abbrechen*: Eingaben werden nicht verwertet
- Klick auf *Bestätigen*: keine zusätzliche Eingabe von Daten



Aufgabe 5.2 – Function Points

- Dateneingaben
  - 2 Datenbestände werden verarbeitet (Kreditkarten-Datensatz wird nicht verändert)



Aufgabe 5.2 – Function Points

Anzahl bearbeiteter Datenbestände	Anzahl unterscheidbarer Datenelemente in der Eingabe		
	1–4	5–15	>15
0–1	einfach	einfach	mittel
2	einfach	mittel	komplex
>2	mittel	komplex	komplex

Element	Schwierigkeitsgrad			
	einfach	mittel	komplex	Summe
Dateneingaben	_____ x 3 = _____	1 x 4 = _____	_____ x 6 = _____	4
Datenausgaben	_____ x 4 = _____	_____ x 5 = _____	_____ x 7 = _____	_____
Anfragen	_____ x 3 = _____	_____ x 4 = _____	_____ x 6 = _____	_____
Ext. Schnittstellen	_____ x 5 = _____	_____ x 7 = _____	_____ x10 = _____	_____
Int. Datenbestände	_____ x 7 = _____	_____ x10 = _____	_____ x15 = _____	_____
			Function Point Rohwert (UFP)	

Aufgabe 5.2 – Function Points

- Dateneingaben
- Datenausgaben = 0
- Anfragen → Kreditkartendaten müssen geprüft werden
- Externe Schnittstellen
- Interne Datenbestände

Element	Schwierigkeitsgrad			
	einfach	mittel	komplex	Summe
Dateneingaben	_____ x 3 = _____	1 x 4 = _____	_____ x 6 = _____	4
Datenausgaben	_____ x 4 = _____	_____ x 5 = _____	_____ x 7 = _____	_____
Anfragen	1 x 3 = _____	_____ x 4 = _____	_____ x 6 = _____	3
Ext. Schnittstellen	_____ x 5 = _____	_____ x 7 = _____	_____ x10 = _____	_____
Int. Datenbestände	_____ x 7 = _____	_____ x10 = _____	_____ x15 = _____	_____
			Function Point Rohwert (UFP)	

Aufgabe 5.2 – Function Points

- Dateneingaben
- Datenausgaben = 0
- Anfragen
- Externe Schnittstellen → Kreditkartendaten, Sitzplatzdaten (altes System)
- Interne Datenbestände

Element	Schwierigkeitsgrad			
	einfach	mittel	komplex	Summe
Dateneingaben	____ x 3 = ____	1 x 4 = ____	____ x 6 = ____	4
Datenausgaben	____ x 4 = ____	____ x 5 = ____	____ x 7 = ____	
Anfragen	1 x 3 = ____	____ x 4 = ____	____ x 6 = ____	3
Ext. Schnittstellen	2 x 5 = ____	____ x 7 = ____	____ x10 = ____	10
Int. Datenbestände	____ x 7 = ____	____ x10 = ____	____ x15 = ____	
			Function Point Rohwert (UFP)	

Aufgabe 5.2 – Function Points

- Dateneingaben
- Datenausgaben = 0
- Anfragen
- Externe Schnittstellen
- Interne Datenbestände → Kundendaten

Element	Schwierigkeitsgrad			
	einfach	mittel	komplex	Summe
Dateneingaben	____ x 3 = ____	1 x 4 = ____	____ x 6 = ____	4
Datenausgaben	____ x 4 = ____	____ x 5 = ____	____ x 7 = ____	
Anfragen	1 x 3 = ____	____ x 4 = ____	____ x 6 = ____	3
Ext. Schnittstellen	2 x 5 = ____	____ x 7 = ____	____ x10 = ____	10
Int. Datenbestände	1 x 7 = ____	____ x10 = ____	____ x15 = ____	7
			Function Point Rohwert (UFP)	24

Aufgabe 5.2 – Function Points

- TDI: 4 Faktoren sehr hoch, 2 nicht vorhanden, Rest durchschnittlich

Nr.	Faktor	Wert	Einzusetzen sind Werte zwischen 0 und 5
1	Datenkommunikation		
2	Verteilte Funktionen		0 nicht vorhanden, kein Einfluss
3	Leistungsanforderungen		1 unbedeutender Einfluss
4	Belastung der Hardware		2 mäßiger Einfluss
5	Verlangte Transaktionsrate		3 durchschnittlicher Einfluss
6	Online-Dateneingabe		4 erheblicher Einfluss
7	Effiziente Benutzerschnittstelle		5 starker Einfluss
8	Online-Datenänderungen		
9	Komplexe Verarbeitungen		
10	Wiederverwendbarkeit		
11	Einfache Installation		
12	Einfache Benutzbarkeit		
13	Installation an mehreren Orten		
14	Änder- und Erweiterbarkeit		
	Summe der Faktoren (TDI)		

$4 * 5 + 8 * 3 = 44$ (TDI)

$0.65 + 0.01 * 44 = 1.09$ (VAF)

UFP * VAF = FP

$24 * 1.09 =$ **<u>26.16</u>**

Aufgabe 5.2 – Function Points

- Was braucht es noch?
  - Mittlerer Aufwand pro Function Point muss bekannt sein
  - Faktoren müssen unternehmensspezifisch kalibriert und projektspezifisch angepasst werden

Aufgabe 5.3 – COCOMO2, a)

- Wahl Projektleiter

Skalierungsfaktoren	IT4U	OnlineSystemsIT
Präzedenz	tiefer	höher
Zusammenarbeit	höher	tiefer

Kostenfaktoren	IT4U	OnlineSystemsIT
Personnel continuity	höher	tiefer
Application Experience	tiefer	höher

Aufgabe 5.3 – COCOMO2, b)

- Outsourcing?

Skalierungsfaktoren	IT4U	ITIndia
Zusammenarbeit	höher	tiefer

Kostenfaktoren	IT4U	ITIndia
Platform Experience	tiefer	höher
Language and Tool Exp.	tiefer	höher
Team Co-location and communications support	höher	tiefer

Aufgabe 5.3 – COCOMO2, c)

Eigenschaften, welche den Faktor *Flexibilität* beeinflussen

- Schon vorhandene, wiederzuverwendende Schnittstellen / Systemteile (→ weniger flexibel)
- Termindruck / Zeitvorgaben durch den Kunden (→ weniger flexibel)
- Einflussnahme des Kunden (je mehr der Kunde mitentscheidet, desto weniger flexibel sind wir. Auf der anderen Seite müssen wir flexibel sein, um auf laufende Kundenwünsche einzugehen.)
- Grösse des Teams (je grösser, desto weniger flexibel)

Aufgabe 5.3 – COCOMO2, d)

<u>KSLOC Berechnung</u>

- Reuse required: von **1.00** auf **1.29**
  (oder von 1.14 auf 1.49 (Fehler in Folie))
- Formeln auf Folie 17 (Kapitel 16)

$$\text{Aufwand} = 2.45 * \text{KSLOC}^{B} * \prod_{i=1}^{17} EM_i$$

$$B = 1.01 + 0.01 \sum_{i=1}^{5} Sf_i$$

$$= 1.153$$

$$(\text{KSLOC}_O) \quad \prod_{i=1}^{17} EM_i = 17 * 1.00 = 1.00$$

$$(\text{KSLOC}_N) \quad \prod_{i=1}^{17} EM_i = 16 * 1.00 * 1.29 = 1.29$$

Aufgabe 5.3 – COCOMO2, d)

KSLOC Berechnung

- Reuse required: von **1.00** auf **1.29**
  (oder von 1.14 auf 1.49 (Fehler in Folie))
- Formeln auf Folie 17 (Kapitel 16)

$$\text{Aufwand} = 2.45 * \text{KSLOC}^{B} * \prod_{i=1}^{17} EM_i$$

$$B = 1.01 + 0.01 \sum_{i=1}^{5} Sf_i$$

$$= 1.153$$

Faktor	Sehr gering	Gering	Nominal	Hoch	Sehr hoch	Extra hoch
Präzedenz	4,05	3,24	2,43	1,62	0,81	0
Flexibilität	6,07	4,86	3,64	2,43	1,21	0
Risiko-Umgang	4,22	3,38	2,53	1,69	0,84	0
Zusammenarbeit	4,94	3,95	2,97	1,98	0,99	0
Prozessreife	4,54	3,64	2,73	1,82	0,91	0

Aufgabe 5.3 – COCOMO2, d)

KSLOC Berechnung

- Reuse required: von **1.00** auf **1.29**
  (oder von 1.14 auf 1.49 (Fehler in Folie))

Factor	Very low	Low	Nominal	High	Very High	Extra high
Reliability required	0.75	0.88	1.00	1.15	1.39	
Database size		0.93	1.00	1.09	1.19	
Product complexity	0.75	0.88	1.00	1.15	1.30	1.66
Reuse required		0.91	1.00	1.14	1.29	1.49
Documentation required	0.89	0.95	1.00	1.06	1.13	
Execution time constraint			1.00	1.11	1.31	1.67
Storage constraint			1.00	1.06	1.21	1.57
Platform volatility		0.87	1.00	1.15	1.30	
Analyst capability	1.50	1.22	1.00	0.83	0.67	
Programmer capability	1.37	1.16	1.00	0.87	0.74	
Personnel continuity (turnover)	1.24	1.10	1.00	0.92	0.84	
Application experience	1.22	1.10	1.00	0.89	0.81	
Platform experience	1.25	1.12	1.00	0.88	0.81	
Language and tool experience	1.22	1.10	1.00	0.91	0.84	
Use of software tools	1.24	1.12	1.00	0.86	0.72	
Team co-location and communications support	1.25	1.10	1.00	0.92	0.84	0.78
Required development schedule	1.29	1.10	1.00	1.00	1.00	

$$\prod_{i=1}^{17} EM_i = 17 * 1.00 = 1.00$$

$$\prod_{i=1}^{16} EM_i = 16 * 1.00 * 1.29 = 1.29$$

Aufgabe 5.3 – COCOMO2, d)

$$\text{Aufwand} = \prod_{i=1}^{17} EM_i * 2.45 * KSLOC^B$$

$$\text{Aufwand}_{Old} >= \text{Aufwand}_{Neu}$$

$$1.00 * 2.45 * KSLOC_O^B = 1.29 * 2.45 * KSLOC_N^B \quad | /2.45$$

$$KSLOC_O^B = 1.29 * KSLOC_N^B \quad | KSLOC_N^B = (x * KSLOC_O)^B$$

$$KSLOC_O^B = 1.29 * (x * KSLOC_O)^B$$

$$KSLOC_O^B = 1.29 * x^B * KSLOC_O^B \quad | /KSLOC_O^B$$

$$1 = 1.29 * x^B \quad | /1.29$$

$$1 / 1.29 = x^{1.153} \quad | \sqrt[1.153]{}$$

$$(1 / 1.29)^{1 / 1.153} = x$$

$$0.802 = x$$ ➔ KSLOC muss um mind. **20%** reduziert werden

Aufgabe 5.3 – COCOMO2, e)

Voraussetzung, damit COCOMO2 zuverlässige Werte liefert?

- Faktoren müssen unternehmensspezifisch kalibriert werden
  - Erfahrungen aus vergangenen Projekten (vom eigenen Unternehmen oder evtl. von anderen mit ähnlichen Projekten)
  - Berechnungen auf konkrete Bedingungen anpassen

(stabile Umgebung & einfache Anwendungssoftware gilt nur für COCOMO (1))

Aufgabe 5.4 – Risikoschätzung

- Risikoabschätzung für das konkrete Projekt
  - Möglicher Verlust (Zeit, Geld, Kontrolle, Qualität)
- **Vorbeugung** gegen Bedrohungen
  - Risiko vermeiden, mindern, auf Andere abwälzen, Planung für Worst Case
- Risiko = Schadenshöhe * Eintrittswahrscheinlichkeit
- Keine Risiken aus dem Skript (z.B. falsche Funktionalität)

Aufgabe 5.4 – Risikoschätzung

- Unmotivierte Entwickler
  - E = 3, S = 4, Risiko = 12
  - Arbeitsbedingungen verbessern, Firmenfeier, Anreize schaffen
- Feuer / Virus → Datenverlust
  - E = 2, S = 9, Risiko = 18
  - Regelmässige Backups (auswärtiger Server), Rauchverbot, Firewalls
- Zahlungsunfähigkeit der Theaterleitung
- Übersetzungsprobleme (mehrsprachiges GUI)
- Falscher Technologieeinsatz
- Schweinegrippe