Klausur zur Veranstaltung Softwareentwicklung 1

Sommersemester 2004
Hessische VWA
Bernd Ulmann

12. Mai 2004

Hinweise:

- Die Klausur besteht aus 24 Teilaufgaben
- Insgesamt sind 120 Punkte erreichbar, jede Teilaufgabe entspricht 5 Punkten.
- Als Hilfsmittel sind alle „unbelebten Hilfsmittel“, d.h. insbesondere Ihre Vorlesungsmitschriften, zugelassen.
- Bitte schreiben Sie leserlich! (Dankeschön! :-))
- Verwenden Sie bitte keinen (!) Bleistift!

Viel Erfolg!

Aufgabe 1

1. Nennen Sie zwei Nachteile, die das Wasserfallmodell gegenüber dem V-Modell aufweist.

2. Für die Nachfolgesendung zu „WiSo“, „Warum“ ist ein neues Einkommensteuerprogramm zu entwickeln. Da hierbei in der momentanen Situation durch den Gesetzgeber monatlich Änderungen vorzunehmen waren, hat das zuvor mit der Entwicklung der Software beauftragte Unternehmen entnervt aber verständlicherweise das Handtuch geworfen.

   Die kreativen Köpfe hinter WiSo sind verständlicherweise enttäuscht und verunsichert und möchten von Ihnen so schnell wie möglich einen Eindruck des geplanten Produktes erhalten.

   Welches Vorgehensmodell würden Sie (und warum) für die Entwicklung des Programmes „Warum“ wählen?

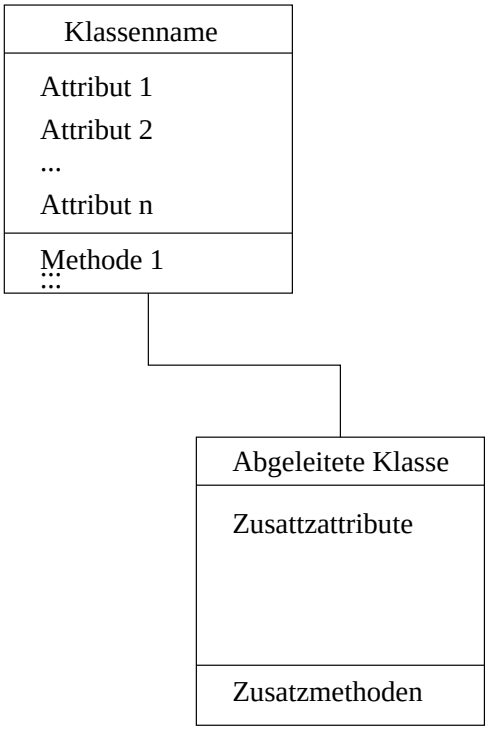
3. Zeichnen Sie das V-Modell. :-)

4. Beschreiben Sie den Unterschied zwischen Validierung und Verifikation.

Aufgabe 2

Das folgende Diagramm veranschaulicht eine Technik (in OO-freundlichen Kreisen auch als UML bekannt :-)), mit deren Hilfe die Beziehungen von Klassen und abgeleiteten Klassen untereinander dargestellt werden können.

In der Abbildung sind eine Oberklasse sowie eine hiervon abgeleitete Klasse dargestellt. Die Oberklasse vererbt alle ihre Attribute und Methoden, d.h. insbesondere auch Methoden zum Zugriff auf Werte von Attributen an die von ihr abgeleitete Unterklasse. Attribute und Methoden der Unterklasse, die über jene der Oberklasse hinausgehen beziehungsweise diese ergänzen, sind in der Darstellung der jeweiligen Unterklasse aufzuführen.



1. Erläutern Sie den Begriff der „Kapselung“[1] in der Objektorientierung.

2. Im Rahmen der Entwicklung eines Programmpaketes für Immobilienmakler sind zunächst eine Reihe von Klassen zu modellieren. Der Einfachheit halber beschränken wir uns auf Einfamilienhäuser sowie Gewerbeimmobilien, die jeweils einen Kaufpreis und eine Grundstücksfläche, die auch 0 sein darf, besitzen. Darüberhinaus können Einfamilienhäuser über ein Schwimmbad verfügen und benötigen einen Wert für die Wohnfläche. Im Gegensatz hierzu verfügen Gewerbeimmobilien über eine Gewerbenutzfläche sowie über die Angabe der Entfernung zum nächstgelegenen Flughafen.

   Zeichnen Sie nun für die Klasse „Einfamilienhaus“ ein Klassendiagramm, wie es in der vorangegangenen Abbildung dargestellt war.

[1] Auch als „Geheimnisprinzip“ bekannt.

3. Zeichnen Sie nun ein entsprechendes Klassendiagramm für die Klasse „Gewerbeimmobilie“.

4. Zeichnen Sie nun ein Klassendiagramm mit einer Oberklasse „Immobilie“ sowie zwei hiervon abgeleiteten Unterklassen „Einfamilienhaus“ beziehungsweise „Gewerbeimmobilie“.

5. Geben Sie nun für alle drei Klassen Ihres Diagramms die jeweils notwendigen Methoden für den Zugriff auf die jeweiligen Klassenattribute an.

Aufgabe 3

Die folgenden Aufgaben widmen sich dem Werkzeug der Use-Cases, wie sie Ihnen sicherlich noch mit Schrecken aus der Vorlesung bekannt sind. :-) Um es für mich erträglicher zu gestalten, werden Sie mit einem mir sehr nahestehenden Geschäftsprozeß bekannt gemacht.

Es ist der Vorgang eines Coca-Cola-Verkaufs an einem von einer Getränkefirma aufgestellten Getränkeautomaten in Form eines Black-Box-Use-Case zu modellieren.

1. Identifizieren Sie alle an dem genannten Geschäftsprozeß beteiligten Stakeholder.

2. Bestimmen Sie mindestens vier Preconditions für den Use-Case.

3. Beschreiben Sie das Main-Success-Szenario.

4. Beschreiben Sie drei mögliche (und auch wahrscheinliche :-)) Fehlerszenarien.

5. Bestimmen Sie die Kategorie des Use-Cases.

6. Warum handelt es sich eigentlich um einen Black-Box-Use-Case?

7. Nachdem Sie nun fast alle relevanten Brösel zusammengetragen haben, schreiben Sie den kompletten Black-Box-Use-Case auf.

Aufgabe 4

1. Gegeben sei die folgende, einfache kontextfreie Grammatik[2]:

```
ausdruck ::= bezeichner |
             -ausdruck  |
             (ausdruck) |
             ausdruck operator ausdruck
operator ::= + |
             - |
             * |
             /
```

   Zeichnen Sie den Parsetree für den folgenden Ausdruck:

$$(a + b)/(c - d)$$

[2] Nicht vergessen: Kontextfreie Grammatiken bieten die Möglichkeit, reguläre Ausdrücke, mit deren Hilfe sich die Struktur syntaktisch korrekter Ausdrücke beschreiben läßt, rekursiv anzuwenden. Im übrigen stellt | den Auswahloperator dar, mit dessen Hilfe zwischen verschiedenen Varianten eines Ausdruckes gewählt werden kann.

2. Ein kleines Rätsel rund um den Sichtbarkeitsbereich von Variablen:

```
#include <stdio.h>

int i;

int summe (int wert)
{
  int summe;

  summe = 0;
  for (i = 1; i <= wert; i++)
    summe = summe + i;

  return summe;
}

int main ()
{
  int j;

  i = 10;
  j = summe (i) / i;
  printf ("j = %d\n", j);

  return 0;
}
```

Das obige C-Programm berechnet zunächst die Summe aller natürlichen Zahlen von 1 bis zu einem vorgegebenen Wert und dividiert diese dann durch den Inhalt der Variablen `i`. Welchen Wert hat `j` zum Zeitpunkt seiner Ausgabe durch das `printf`?

Aufgabe 5

Eine bekannte Zahlenfolge ist die sogenannte Fibonacci-Folge, benannt nach Leonardo von Pisa, genannt *Fibonacci*, die sich anhand des folgenden Bildungsgesetzes errechnen läßt:

f_i bezeichne die i-te Fibonacci-Zahl. Dann gilt zunächst (quasi als Initialwerte):

$$f_0 = f_1 = 1. \tag{1}$$

Alle folgenden Fibonacci-Zahlen errechnen sich zu

$$f_i = f_{i-1} + f_{i-2} \quad \text{für } i \geq 2. \tag{2}$$

1. Wie lauten die Werte für f_2 bis f_7?

2. Zeichnen Sie einen Baum der Funktionsaufrufe, die benötigt werden, um mit der folgenden Routine den Wert des Aufrufes (dieser initiale Funktionsaufruf zählt mit!) `fib_rek (5)` zu berechnen.

```
unsigned int fib_rek (unsigned int i)
{
  if (i < 2)
    return 1;

  return fib_ref (i - 1) + fib_rek (i - 2);
}
```

3. Geben Sie nun – analog zur vorangegangenen Aufgabe an, wieviele Funktionsaufrufe für die Berechnung von `fib_rek (6)` mit obiger rekursiver Funktion notwendig sind (nicht den gesamten Aufrufbaum!).

4. Wie verhält sich dieser Algorithmus bei steigendem Wert für den Übergabeparameter an die Funktion `fib_rek ()`? Linear? Quadratisch? Kubisch? Etwas anderes (wenn ja, was?)?

5. Die folgende Routine berechnet auch Zahlen der Fibonacci-Folge, im Gegensatz zu `fib_rek ()` jedoch nicht rekursiv, sondern stattdessen iterativ:

```
unsigned int fib_iter (unsigned int i)
{
  int j, f_1, f_2, f_i;

  if (i < 2)
    return 1;

  for (j = f_1 = f_2 = 1; j < i; j++)
  {
    f_i = f_1 + f_2;
    f_2 = f_1;
    f_1 = f_i;
  }

  return f_i;
}
```

   Wieviele Schleifendurchläufe benötigt die Berechnung von `fib_iter (5)` sowie die Berechnung von `fib_iter (6)`?

6. Unter der (etwas vereinfachenden) Voraussetzung, das ein Durchlauf der zentralen `for`-Schleife ähnlich aufwendig ist wie ein einzelner Aufruf der Routine `fib_rek ()` aus dem vorangegangenen Beispiel, stellt sich die Frage, welcher Routine Sie im Hinblick auf eine möglichst große Effizienz des Programmes den Vorzug bei der Berechnung geben würden – der *rekursiven* oder eher der *iterativen* Variante?