Klausur Softwaretechnik

13.03.2009

Prof. Dr. Walter F. Tichy
Dipl.-Inform. T. Gelhausen
Dipl.-Inform. A. Höfer
Dipl.-Inform. D. Meder

Hier das Namensschild aufkleben.

Zur Klausur sind keine Hilfsmittel und kein eigenes Papier zugelassen.
Die Bearbeitungszeit beträgt 60 Minuten.
Die Klausur ist vollständig und geheftet abzugeben.
Mit Bleistift oder roter Farbe geschriebene Angaben werden nicht bewertet.

Aufgabe	**1**	**2**	**3**	**4**	**5**	**6**	∑
Maximal	16	7	6	8	10	13	60
K1							
K2							
K3							

Aufgabe 1: Aufwärmen (16P)

a) Kreuzen Sie an, ob die folgenden Aussagen wahr oder falsch sind. (5P)

Hinweis: Jedes korrekte Kreuz zählt 0,5 Punkte, jedes falsche Kreuz bewirkt 0,5 Punkte Abzug. Die Teilaufgabe wird mindestens mit 0 Punkten bewertet.

Aussage	**Wahr**	**Falsch**
Die Kosten für Wartung und Pflege von Software kann man mit maximal 1/3 der Gesamtkosten abschätzen.		
Die letzte Phase des klassischen Wasserfallmodells ist „Testen und Abnahme“.		
In der Planungsphase muss die Verfügbarkeit qualifizierter Fachkräfte für die Entwicklung überprüft werden.		
Funktionale Eigenschaften beschreiben auch, was das Produkt nicht tun sollte.		
Das Lastenheft ist die genaue Vorschrift für die Entwickler.		
Ein Vorteil der Ethnografie ist, dass sie detailreiche und leicht zu analysierende Aufzeichnungen liefert.		
Bei Microsofts „Synchronisiere und Stabilisiere“-Prozess gibt es in jedem Team in etwa gleich viele Tester wie Entwickler.		
Ein Modell ist die Abstraktion von existierenden oder imaginären Dingen, Personen, Abläufen und deren Beziehungen.		
Die dynamische Sicht auf das System zeigt, wie sich Elemente verhalten, sich gegenseitig beeinflussen (steuern) und/oder ihre Beziehungen zueinander ändern.		
Laut den Ergebnissen der Metastudie von Dybå et al. führt die Paarprogrammierung zu höherer Qualität als die Einzelprogrammierung bei ungefähr gleichem Aufwand.		

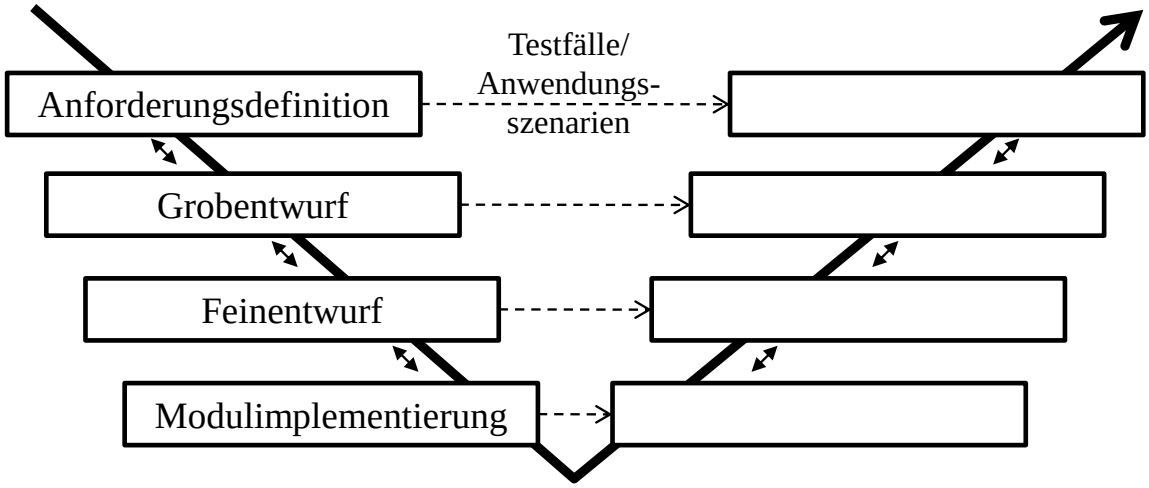
b) Erklären Sie den Unterschied zwischen den Begriffen „Attribut“ im Modell und „Instanzvariable“ im Code. (2P)

c) Beschreiben Sie, wie Funktionalität gemäß der testgetriebenen Entwicklung implementiert wird. (4 × 0,5P = 2P)

d) Nennen Sie die Vor- und Nachteile der Erhebungstechnik „Anwendungsfälle“. (2P)

e) Geben Sie die optimierte Form des „ijk“-Algorithmus zur Multiplikation zweier n × n-Matrizen a und b an. (1P)

f) Ergänzen Sie das folgende V-Modell um die korrespondierenden Softwaretests. (2P)



g) Beschreiben Sie in je einem Satz die Aufgabe der Softwaretests des V-Modells (vgl. Aufgabenteil f)). (2P)

Aufgabe 2: Optimierungstechniken (7P)

Gegeben sei die folgende spärlich besetzte Matrix:

Zeile/Spalte	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	0	0	4	0
2	0	0	0	0	0	0
3	0	0	1	0	5	0
4	0	0	0	0	2	0
5	0	0	0	0	0	3

a) In der Vorlesung wurde ein Verfahren vorgestellt, um spärlich besetzte Felder zu komprimieren. Wie müssen Sie die folgenden drei Java-Felder initialisieren, damit Sie den Inhalt des oben gezeigten spärlich besetzten Feldes in komprimierter Form wiedergeben? (3P)

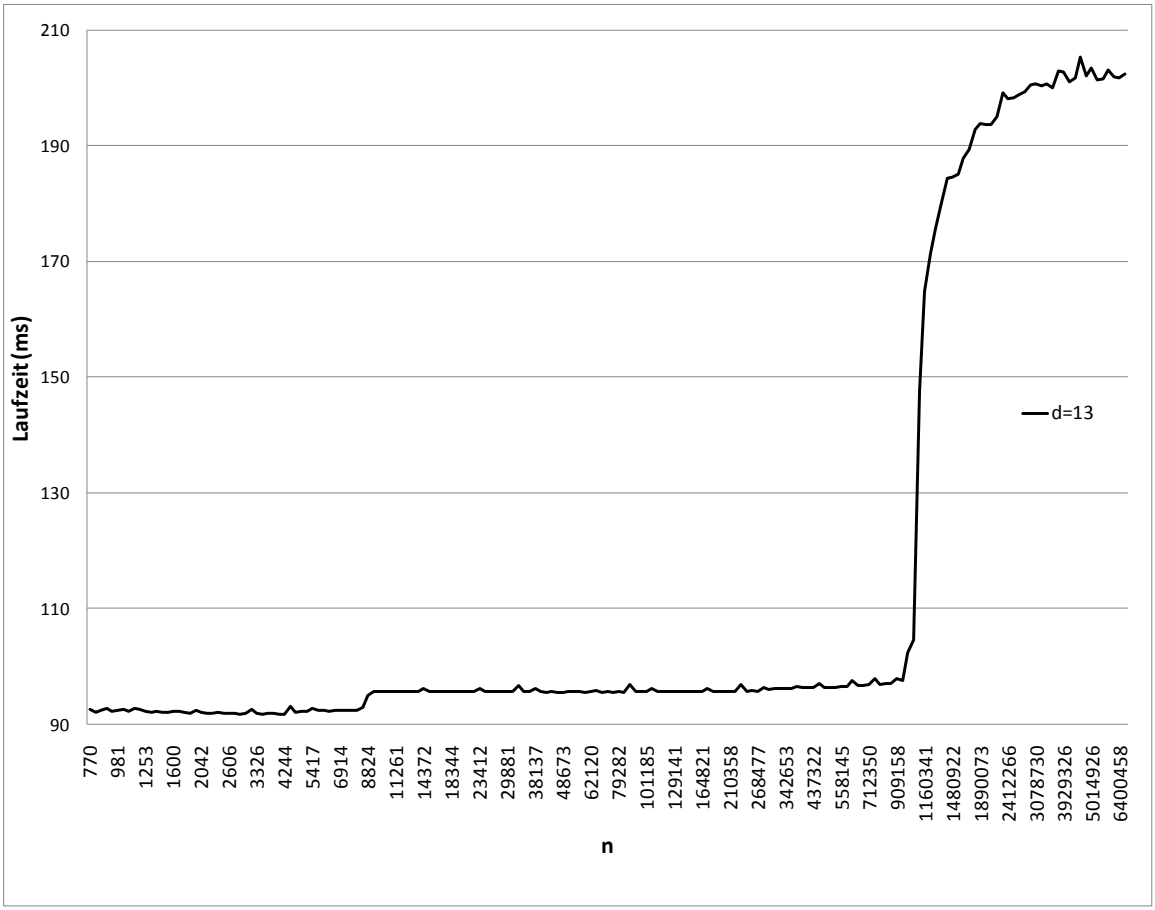
```
          int[] wert = {                                          };

         int[] zeile = {                                          };

 int[] spaltenErster = {                                          };
```

b) Ergänzen Sie folgende Funktion an den durch Strichen gekennzeichneten Stellen, so dass sie die Einträge aus dem komprimierten spärlich besetzten Feld zurückliefert. (2P)

```
// z = Zeile, s = Spalte
int holeEintrag(int z, int s) {
   for (                                                          ) {

      if (                                                        ) {

         return                                                   ;

      }
   }

   return                                                         ;
}
```

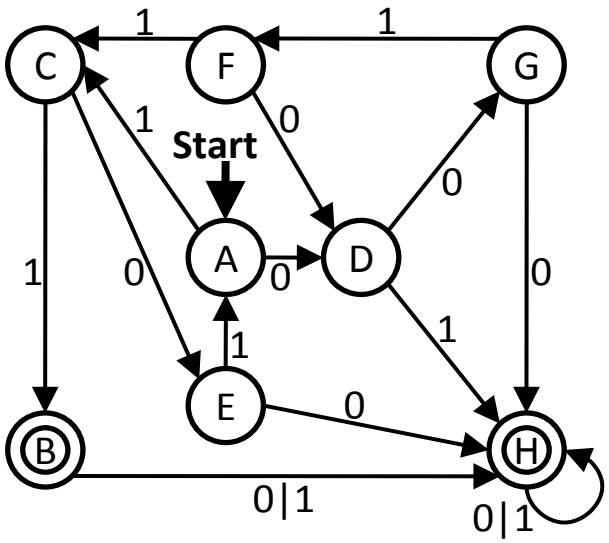
c) In der Vorlesung wurde eine Methode besprochen, mit der man Cache-Effekte sichtbar machen kann. Der Code wurde mehrfach auf einem aktuellen Rechner ausgeführt. Die erhaltenen Zugriffszeiten sind in nachfolgendem Diagramm dargestellt. Wie viele Caches besitzt der Rechner und wie groß sind diese? Begründen Sie Ihre Antwort. (2P)

Hinweis: Machen Sie geeignete Annahmen und dokumentieren Sie diese.



Aufgabe 3: Implementierung von Zustandsautomaten (6P)

Gegeben sei der folgende deterministische, endliche Automat (DEA):



A								
B	-							
C		-						
D		-						
E		-						
F		-						
G		-						
H	-		-	-	-	-	-	
	A	**B**	**C**	**D**	**E**	**F**	**G**	**H**

a) Minimieren Sie den gegebenen DEA. Verwenden Sie dazu den in der Vorlesung vorgestellten Algorithmus zur Ermittlung des Äquivalenzklassenautomaten. Geben Sie in Ihrer Lösung die vollständige Tabelle der äquivalenten Zustände an. (4P)

Markieren Sie das jeweilige Feld in der Tabelle mit einem „+“ (Plus), wenn zwei Zustände äquivalent sind, wenn nicht, dann mit einem „-“ (Minus).

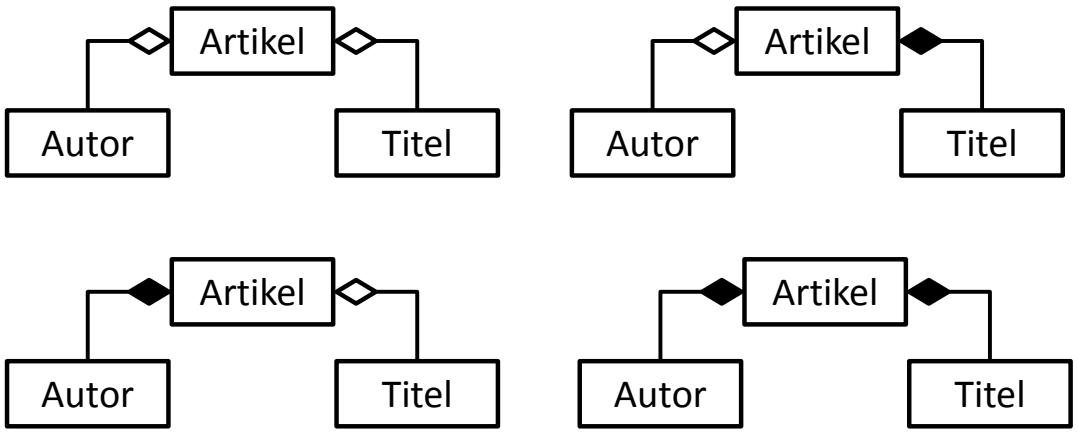
Hinweis: Für jedes korrekt gesetzte Feld gibt es 0,25 Punkte. Für jedes falsch gesetzte Feld werden 0,25 Punkte abgezogen. Die Teilaufgabe wird mit mindestens 0 Punkten bewertet.

b) Zeichnen Sie den in Teilaufgabe a) minimierten DEA. (2P)

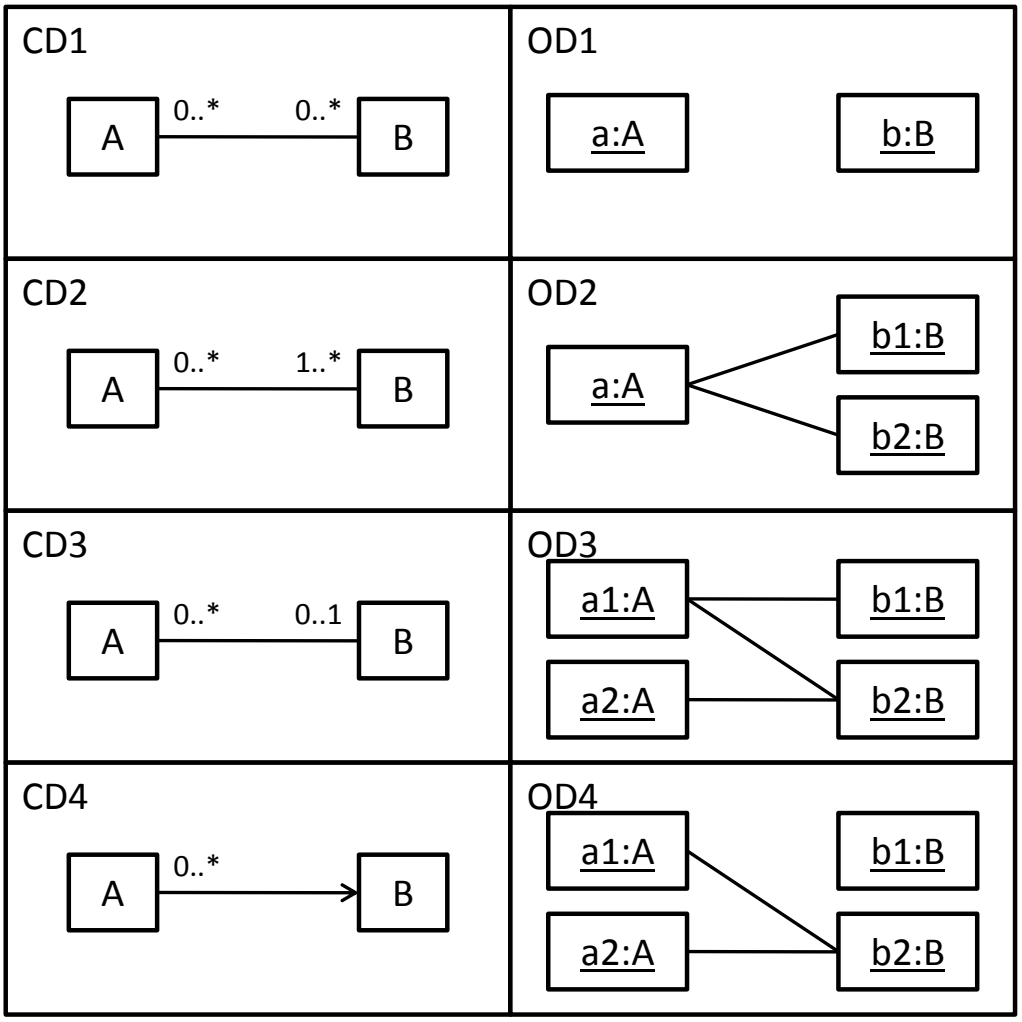
Aufgabe 4: Semantik von UML (8P)

a) In den Vorlesungsunterlagen wurde die Restriktion **{unique}** nur für zweistellige Assoziationen definiert: „Sei $\rho \subseteq K_1 \times K_2$ die einer Assoziation zugrunde liegende Relation…“. Geben Sie analog eine Definition für n-stellige Relationen an, in denen **{unique}** an genau einem Ende angegeben ist. (5P)

b) Welche der folgenden Modellierungen ist richtig, wenn man geeignete Multiplizitäten ergänzt? Markieren Sie die richtige Modellierung und ergänzen Sie sinnvolle Multiplizitäten. (1P)



c) Welche Objektdiagramme passen zu welchen Klassendiagrammen? (2P)



Markieren Sie das jeweilige Feld in der Tabelle mit einem „+“ (Plus), wenn das Objektdiagramm zum Klassendiagramm passt, wenn nicht, dann mit einem „-“ (Minus).

Hinweis: Nur vollständig ausgefüllte Zeilen ergeben Punkte!

	OD1	**OD2**	**OD3**	**OD4**
CD1				
CD2				
CD3				
CD4				

Aufgabe 5: Entwurfsmuster (10P)

Zeigen Sie, dass Bilder in Applets einen schönen Anwendungsfall für das Entwurfsmuster „Fliegengewicht“ abgeben. Die wichtigsten Indikatoren aus der Schnittstellendokumentation sind:

java.applet

Class Applet

java.lang.Object
  └ […]
    └ **java.applet.Applet**

All Implemented Interfaces: ImageObserver, […]

public class **Applet** extends Panel

An applet is a small program that is intended not to be run on its own, but rather to be embedded inside another application.

The Applet class must be the superclass of any applet that is to be embedded in a Web page or viewed by the Java Applet Viewer. The Applet class provides a standard interface between applets and their environment.

Since: JDK1.0

Method Summary	
[…]	[…]
Image	**getImage**(URL url) – Returns an Image object that can then be painted on the screen.
[…]	[…]

[…]

getImage

public Image **getImage**(URL url)

Returns an Image object that can then be painted on the screen. The url that is passed as an argument must specify an absolute URL.

This method always returns immediately, whether or not the image exists. When this applet attempts to draw the image on the screen, the data will be loaded. The graphics primitives that draw the image will incrementally paint on the screen.

Parameters:
url - an absolute URL giving the location of the image.
Returns:
the image at the specified URL.
See Also:
Image

java.awt

Class Image

java.lang.Object
  └ **java.awt.Image**

public abstract class **Image** extends Object

The abstract class Image is the superclass of all classes that represent graphical images. The image must be obtained in a platform-specific manner.

Field Summary	
protected float	**accelerationPriority** – Priority for accelerating this image.
static […]	[…] (nur noch Klassenvariablen)

Method Summary	
abstract void	**flush**() – Flushes all resources being used by this Image object.
float	**getAccelerationPriority**() – Returns the current value of the acceleration priority hint.
ImageCapabilities	**getCapabilities**(GraphicsConfiguration gc) – Returns an ImageCapabilities object which can be inquired as to the capabilities of this Image on the specified GraphicsConfiguration.
abstract Graphics	**getGraphics**() – Creates a graphics context for drawing to an off-screen image.
abstract int	**getHeight**(ImageObserver observer) – Determines the height of the image.
Image	**getScaledInstance**(int width, int height, int hints) – Creates a scaled version of this image.
abstract ImageProducer	**getSource**() – Gets the object that produces the pixels for the image.
abstract int	**getWidth**(ImageObserver observer) – Determines the width of the image.
void	**setAccelerationPriority**(float priority) – Sets a hint for this image about how important acceleration is.

a) Geben Sie ein Einsatzbeispiel an, bei dem sich die Implementierung der **Bilder** als Fliegengewicht auszahlen würde. (1P)

b) Welche der existierenden Klassen, Methoden und Parameter nehmen welche Rollen des Musters ein? (3P)

Rolle	**Existierende Klasse/Methode/Parameter**
Fliegengewicht	
FliegengewichtFabrik	
gibFliegengewicht()	
Schlüssel	
Operation	
Extrinsischer Zustand	

c) Welches weitere Entwurfsmuster kann man in der Methode `getImage` erkennen? (1P)

d) Was verstehen wir unter „extrinsischer Zustand“ und „intrinsischer Zustand“? Geben Sie jeweils die Definition und ein Beispiel bezogen auf das Fliegengewicht **dieser Aufgabe** an! (4P)

e) Welches Optimierungsprinzip läge der Implementierung der Bilderverwaltung in Applets zugrunde, wenn hier das Entwurfsmuster Fliegengewicht eingesetzt wird? (Wir erwarten eine Antwort der Form: „Prinzip X99: Nimm Java und spar dir Speicherlecks!“ aus dem Kapitel über Optimierung) (1P)

Aufgabe 6: Kontrollflussorientierte Testverfahren (13P)

Gegeben sei die folgende Java-Funktion:

```
01 public static boolean istPrim(int z) {
02    boolean result = true;
03    if (z != 2) {
04       if ((z < 2) || ((z % 2) == 0)) {
05          result = false;
06       } else {
07          for (int i = 3; i * i <= z; i += 2) {
08             if (z % i == 0) {
09                result = false;
10                break;
11             }
12          }
13       }
14    }
15    return result;
16 }
```

a) Erstellen Sie den Kontrollflussgraphen der Funktion istPrim(...). Bitte schreiben Sie den Quelltext in die Kästchen, Verweise auf Zeilennummern der Funktion sind nicht ausreichend. (5P)

b) Geben Sie eine minimale Testfallmenge an, mit der Sie die Anweisungsüberdeckung erfüllen. Geben Sie die durchlaufenen Pfade an. (3P)

c) Erfüllt die Testfallmenge auch die Zweigüberdeckung? Begründen Sie Ihre Antwort. (1P)

d) Nennen Sie die Teilpfade aller Grenztests (Boundary-Interior-Test). (1P)

e) Betrachten Sie die folgende Bedingung aus der **Java-Funktion `istPrim(…)`**. Geben Sie eine minimale Testfallmenge an, die die minimale-mehrfache Bedingungsüberdeckung erfüllt. Geben Sie für jeden Testfall der Menge an, welchen Zustand die atomaren Bedingungen und die zusammengesetzte Bedingung einnehmen. (1P)

```
if ((z < 2) || ((z % 2) == 0)) …
```

f) Gibt es eine kleinere minimale Testfallmenge, die die minimale-mehrfache Bedingungsüberdeckung erfüllt, wenn die Bedingung wie folgt modifiziert wird? Begründen Sie Ihre Antwort. (2P)

```
if ((z < 2) | ((z % 2) == 0)) …
```