UNIVERSITÄT ZU LÜBECK
INSTITUTE FOR SOFTWARE ENGINEERING
AND PROGRAMMING LANGUAGES

isp

Software Engineering Wintersemester 2015/16
Prof. Dr. Martin Leucker, Torben Scheffel 19.02.2016

Klausur Software Engineering

Aufgaben und Punkte

Die Bearbeitungszeit der Klausur umfasst 90 Minuten. Es gibt 10 Aufgaben mit insgesamt 100 zu erreichenden Punkten.
Notieren Sie Ihre Lösungen wenn möglich direkt auf dem Aufgabenblatt. Sollte der Platz nicht ausreichen, verwenden Sie zusätzliche Blätter, die Ihnen gestellt werden. Benutzen Sie jede Seite der zusätzlichen Blätter nur für genau eine Aufgabe und notieren Sie Ihren Namen und Ihre Matrikelnummer am oberen Rand des Blattes.
Rechts oben auf jeder Seite der Klausur stehen die Punkte für eine gesamte Aufgabe. Die in Klammern gesetzten Zahlen dahinter geben die Punkte für die einzelnen Teilaufgaben an, von links nach rechts, jeweils beginnend mit Teilaufgabe a).
Wenn Sie in einer Aufgabe Lösungen ankreuzen müssen, so erhalten Sie für jedes richtig gesetzte Kreuz Punkte und für jedes falsch gesetzte Kreuz werden Ihnen Punkte abgezogen. Wenn Sie kein Kreuz setzen bekommen Sie weder Punkte, noch werden Ihnen Punkte abgezogen. Die genauen Punktzahlen stehen dabei an jeder Aufgabe, bei der Sie etwas ankreuzen müssen, dabei. Sie können in jedem Aufgabenteil aber nicht weniger als 0 Punkte bekommen.

Persönliche Daten

Notieren Sie im Folgenden Ihre persönlichen Daten. Notieren Sie Ihren Namen und Ihre Matrikelnummer außerdem auf jedem weiteren Blatt der Klausur am oberen Rand sowie auf jeden zusätzlichen Zettel, den Sie benutzen, wie vorher erläutert.

Vorname: _____

Nachname: _____

Geburtsdatum: _____

Matrikelnummer: _____

Studiengang: _____

Viel Erfolg!

Name: *Matrikelnummer:*

Aufgabe 1: Softwareentwicklungsprozess **7 Punkte (1/2/4)**

a) Aus welchem Vorgehensmodell ist das V-Modell hervorgegangen?

b) Nennen Sie alle Testphasen des V-Modells, die der Validierung dienen.

c) Nennen Sie zwei spezielle Rollen, die für den Einsatz von Scrum vergeben werden und beschreiben Sie jeweils kurz deren Aufgaben.

Name: *Matrikelnummer:*

Aufgabe 2: Systemsichten und Diagrammarten **6 Punkte**

Wir betrachten folgende Sichten auf ein System:

- Funktionale Sicht
- Datenorientierte Sicht
- Algorithmische Sicht
- Regelbasierte Sicht
- Zustandsorientierte Sicht

Nennen Sie sechs verschiedene **nicht**-UML-Diagrammarten oder Modellierungsmethoden und ordnen Sie diese jeweils einer der oben genannten Sichten zu. Es können mehrere Diagramme und Modellierungsmethoden der selben Schicht zugeordnet werden.

Name: *Matrikelnummer:*

Aufgabe 3: Phasen der Softwareentwicklung und Diagramme **10 Punkte (3/3/4)**

a) Nennen Sie drei UML-Diagrammarten, die für die Modellierung in der Entwurfsphase benutzt werden können.

b) Nennen Sie drei Elemente, die auf Storycards stehen.

c) Entscheiden Sie, ob folgende Aussagen wahr oder falsch sind. Es gibt $+0.5$ Punkte für korrekte Kreuze und -0.5 Punkte für falsche.

	Wahr	Falsch
Netzpläne können in der Planungsphase eingesetzt werden.	☐	☐
Zustände sind Teile von Anwendungsfalldiagrammen.	☐	☐
Aggregationen können in Klassendiagrammen vorkommen.	☐	☐
JUnit-Tests werden im Allgemeinen in der Implementierungsphase geschrieben.	☐	☐
Wird Runtime-Verification eingesetzt, ist Model Checking überflüssig.	☐	☐
In Objektdiagrammen gibt es Kompositionen.	☐	☐
Die Planungsphase wird vor der Entwurfsphase durchgeführt.	☐	☐
Die Function-Point-Methode nimmt Anwendungsfälle als direkte Grundlage.	☐	☐

Name: *Matrikelnummer:*

Aufgabe 4: Management **7 Punkte (3/2/2)**

a) Nennen Sie drei Unterschiede des Software-Reviews im Vergleich zur Software-Inspektion.

b) Nennen Sie zwei Nachteile, die auftreten können, wenn die ursprünglichen Entwickler auch die Tests schreiben.

c) Entscheiden Sie, ob folgende Aussagen wahr oder falsch sind. Es gibt $+0.5$ Punkte für korrekte Kreuze und -0.5 Punkte für falsche.

	Wahr	Falsch
Ein Ziel des Re-Engineerings kann es sein, die Dokumentation wieder herzustellen.	☐	☐
BSD ist eine eher restriktive Lizenz.	☐	☐
Ein Code-Walkthrough ist für kleine Entwicklerteams geeignet.	☐	☐
Der Risikofaktor ergibt sich aus der Schadenshöhe und der Eintrittswahrscheinlichkeit.	☐	☐

Name: *Matrikelnummer:*

Aufgabe 5: Struktogramm **9 Punkte (2/7)**

a) Entscheiden Sie, ob folgende Aussagen wahr oder falsch sind. Es gibt $+0.5$ Punkt für korrekte Kreuze und -0.5 Punkt für falsche.

	Wahr	Falsch
Programmablaufpläne können Datenstrukturen nicht differenziert darstellen.	☐	☐
Programmablaufpläne können Schleifen darstellen.	☐	☐
Struktogramme können lineare Kontrollstrukturen darstellen.	☐	☐
Programmablaufpläne können keine Auswahlen darstellen.	☐	☐

b) Zeichnen Sie ein Struktogramm für folgenden Codeausschnitt:

```
public String func(int x, int y) {
  switch(x) {
    case 1: y = y + 1;
    case 2: while(x > 1) {
              y = y * y;
              x = x - 1;
            }
    case 3: y = func(x - 1, y * x);
    default: y = 0;
  }
  if (y >= 25) {
    return y;
  } else {
    return x;
  }
}
```

Aus Platzgründen sollten Sie die nächste, komplett freie Seite für die Lösung benutzen!

Name: *Matrikelnummer:*

Aufgabe 6: Netzplan **14 Punkte (5/3/3/2/1)**

Gegeben ist folgendes Software-Projekt zur Entwicklung eines Kaffeeautomaten. Die Vorgangsdauer wird in Tagen angegeben und die Modellierung soll mit Tag 1 beginnen.

- Zum Projektstart wird zuerst eine gemeinsame **Schnittstelle** für die Herstellung des Kaffees, die Abrechnung und die Plugin-Schnittstelle festgelegt (Dauer: 1 Tag).
- Nachdem die Schnittstelle definiert wurde, können die Entwürfe beginnen. Der **Abrechnungs-Entwurf** braucht 4 Tage, der **Herstellungs-Entwurf** braucht 2 Tage und der **Plugin-Schnittstellenentwurf** benötigt auch 2 Tage. Alle drei Entwürfe sind unabhängig voneinander.
- Wenn alle Entwürfe abgeschlossen wurden, kann das Test-Team erste **Testfälle** erstellen (Dauer: 4 Tage).
- Nach den jeweiligen Entwürfen können parallel zu den Testfällen die Module implementiert werden. Die **Plugin-Schnittstelle** benötigt 7 Tage, das **Herstellungsmodul** braucht 5 Tage, der erste Teil der Implementierung des **Abrechnungsmodul**s 7 Tage, der zweite Teil benötigt 4 Tage. Die beiden Teile des Abrechnungsmoduls müssen nacheinander implementiert werden.
- Wenn die Implementierung des Herstellungsmoduls, die Implementierung der Plugin-Schnittstelle und der erste Teil der Implementierung des Abrechnungsmoduls abgeschlossen sind, kann bereits mit der **Integration** begonnen werden (3 Tage).
- Wenn die Implementierung des zweiten Teils des Abrechnungsmoduls und die Integration abgeschlossen sind, gibt es noch einen 4-tägigen **Integrationstest**.
- Zum **Abschluss** des Projekts werden die Ergebnisse aufbereitet und dokumentiert und dem Kunden vorgestellt (4 Tage).

Entwickeln Sie einen Netzplan wie folgt:

a) Stellen Sie die gegebenen Aufgaben in einem Netzplan dar. Die (gerichteten) Kanten entsprechenden dabei den Abhängigkeiten.

b) Rechnen Sie vorwärts: Berechnen Sie, wann die Vorgänge frühestens begonnen werden können und annotieren Sie diese Werte im Netzplan.

c) Rechnen Sie rückwärts: Berechnen Sie ausgehend vom Projektende, wann die Vorgänge spätestens begonnen werden können (ohne dass sich das Projektende verändert) und annotieren Sie diese Werte im Netzplan.

d) Bestimmen Sie sämtliche Pufferzeiten.

e) Geben Sie einen kritischen Pfad an.

Die Vorgangsknoten sollen die Aufteilung des linken Knoten haben. Beginnen Sie mit dem rechten Knoten.

Name des Vorgangs	
Min. Start	Max. Start
Dauer	Puffer

Schnittstelle	
1	

Aufgabe 7: Algebraische Spezifikation **12 Punkte (12)**

Im Folgenden soll ein abstrakter Datentyp `ArithMap` für eine Map mit Datenwerten aus der Sorte `Data` als Schlüssel und natürlichen Zahlen als Werte angegeben werden, für die es noch zusätzlich arithmetische Operationen gibt. Dabei können arithmetische Maps aus der leeren, arithmetischen Map `empty`, bei der für alle Datenwerte initial *zero* aus der Sorte `Nat` zurückgegeben wird, durch das Anwenden der Operation `put` erzeugt werden. `put` fügt dabei für einen Datenwert eine natürliche Zahl ein. Außerdem gibt es noch zwei weitere Operationen:

- die Operation `get`, die für einen Datenwert die darunter gespeicherte Zahl zurückgibt und
- die Operation `addm`, die zwei arithmetische Maps und zwei Datenwerte nimmt und die Summe aus der Zahl, die in der ersten Map unter dem ersten Datenwert gespeichert ist und der Zahl, die in der zweiten Map unter dem zweiten Datenwert gespeichert ist, zurückgibt.

Geben Sie eine algebraische Spezifikation für `ArithMap` an. Dabei darf die aus der Vorlesung bekannte Spezifikation für `Nat` benutzt werden. Außerdem kann `Data` als gegeben angesehen und genutzt werden.

Name: *Matrikelnummer:*

Aufgabe 8: Lineare Temporallogik **13 Punkte (2/6/5)**

Sei im Folgenden $\text{AP} = \{a, b, c\}$ und $\Sigma = 2^{\text{AP}}$.

a) Sei $(\mathcal{G}\, a)\,\mathcal{U}(b \wedge \mathcal{X}\,\mathcal{X}\,\mathcal{F}\, c)$ eine LTL-Formel.

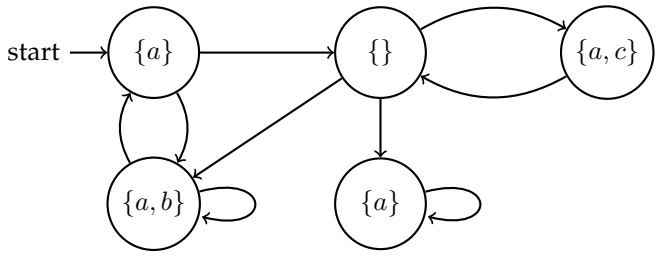
Geben Sie einen Lauf über Σ an, der die Formel erfüllt.

Geben Sie einen Lauf über Σ an, der die Formel nicht erfüllt.

b) Entscheiden Sie, ob folgende Aussagen wahr oder falsch sind. Es gibt $+1$ Punkt für korrekte Kreuze und -1 Punkt für falsche.

	Wahr	Falsch
$\mathcal{G}(a \rightarrow \neg a)$ ist semantisch äquivalent zu *false*.	☐	☐
Jeder Lauf, der $\mathcal{F}\, b$ nicht erfüllt, erfüllt auch $a\,\mathcal{U}\, b$ nicht.	☐	☐
$(\mathcal{X}\,\mathcal{X}\, a)\,\mathcal{U}\,\mathcal{F}\,\mathcal{F}(b\,\mathcal{U}\, b)$ ist eine syntaktisch korrekte LTL-Formel.	☐	☐
Der Lauf $\{b\}(\{a\})^\omega$ erfüllt $(b \rightarrow \neg a) \wedge \mathcal{F}\,\mathcal{G}\, a$.	☐	☐
Der Lauf $\{a\}\{a,b\}\{a\}\{b\}\{c\}(\{\}\{a\})^\omega$ erfüllt $(a\,\mathcal{U}\, b)\,\mathcal{U}\, c$.	☐	☐
Jeder Lauf, der $\mathcal{G}((\mathcal{G}\, a) \wedge \mathcal{F}\, b)$ erfüllt, erfüllt auch $\mathcal{G}(a\,\mathcal{U}\, b)$.	☐	☐

c) Entscheiden Sie, welche der unten angegebenen LTL-Formeln auf **allen** unendlichen Läufen des Transitionssystems erfüllt sind und welche nicht. Es gibt $+1$ Punkt für korrekte Kreuze und -1 Punkt für falsche.



	Auf allen Läufen erfüllt	Nicht auf allen Läufen erfüllt
$\mathcal{F}(b \vee c)$	☐	☐
$\mathcal{G}(a \vee (\neg a \wedge \neg b \wedge \neg c))$	☐	☐
$\mathcal{G}((c \rightarrow \neg b) \wedge (\mathcal{F}\, b \rightarrow \mathcal{X}\, a)$	☐	☐
$\mathcal{F}\,\mathcal{G}\, a$	☐	☐
$a \rightarrow (b \vee \mathcal{X}\,\mathcal{X}\,\mathcal{X}\,\mathcal{X}\, b)$	☐	☐

Name: *Matrikelnummer:*

Aufgabe 9: UML-Objektdiagramm **8 Punkte (8)**

Stellen Sie die Objekte, deren Attribute und deren Beziehungen, die bei der Markierung `!Objektdiagramm bis hier!` bei Ausführung der `main`-Methode des Codes im Anhang existieren, in einem UML-Objektdiagramm dar.

Name: *Matrikelnummer:*

Aufgabe 10: Design-Pattern **14 Punkte (6/4/4)**

a) Welches Design-Pattern wurde bei dem Code im Anhang angewendet? Erläutern Sie, woran Sie dies festmachen. Wenn Sie nicht wissen, welches Design-Pattern genutzt wurde, nennen Sie Design-Pattern, die nicht genutzt wurden und erläutern Sie, woran Sie dies festmachen (Dies ermöglicht nicht das Erreichen der vollen Punktzahl).

b) Leider wurde während des Einladens des Mülls auch radioaktives `Uranium` eingeladen. Jeder Abfall kann über die Methode `isHazardousWaste` angeben, ob er gefährlich ist. Welches Pattern könnte nun in dem Code verwendet werden um vor dem Ausladen (`dispose`) des `SanitationTruck`s zu prüfen, ob kein gefährlicher Müll in dem `SanitationTruck` ist? Wie würde das dann funktionieren?

c) Nennen Sie zwei Anti-Pattern und erläutern Sie diese kurz.

Anhang

```
public class Main {

  public static void main(String[] args) {

    // create trashcans
    Trashcan plot1Bio = new Trashcan("Bio");
    Trashcan plot1Plastic = new Trashcan("Plastic");

    Trashcan plot2Bio = new Trashcan("Bio");

    Trashcan plot3Bio = new Trashcan("Bio");

    // fill trashcans and create other trash
    plot1Bio.throwIn(new BioWaste("Apple"));
    plot1Bio.throwIn(new BioWaste("Banana"));
    plot1Plastic.throwIn(new PlasticWaste("Foil"));

    plot2Bio.throwIn(new BioWaste("Pear"));
    BulkyWaste plot2BulkyStuff = new BulkyWaste("Broken Table");

    HazardousWaste plot3Uranium = new HazardousWaste("Uranium")
    plot3Bio.throwIn(plot3Uranium);
    plot3Uranium.parent = plot3Bio;

    // load everything in a sanitation truck
    SanitationTruck truck = new SanitationTruck("Fancy Truck");

    truck.loadIn(plot1Bio);
    truck.loadIn(plot1Plastic);
    truck.loadIn(plot2Bio);
    truck.loadIn(plot2BulkyStuff);
    truck.loadIn(plot3Bio);

    // !Objektdiagramm bis hier!

    truck.dispose();
  }
}

public abstract class Trash {

  protected String name;

  protected Trash(String name) {
    this.name = name;
  }

  public void dispose() {
    System.out.println(name + " disposed.");
  }
}
```

```java
public class Trashcan extends Trash {

  private Set<Trash> content = new HashSet<>();

  public Trashcan(String name) {
    super(name);
  }

  public void throwIn(Trash t) {
    if (t instanceof Trashcan || t instanceof SanitationTruck || t instanceof BulkyWaste) {
      throw new IllegalArgumentException("Only small trash allowed!");
    }
    content.add(t);
  }

  @Override
  public void dispose() {
    for (Trash trash : content) {
      trash.dispose();
    }
  }
}

public class SanitationTruck extends Trash {

  private Set<Trash> content = new HashSet<>();
  private int size = 27;

  public SanitationTruck(String name) {
    super(name);
  }

  public void loadIn(Trash t) {
    if (t instanceof SanitationTruck) {
      throw new IllegalArgumentException("Can not load in other trucks!");
    }
    content.add(t);
  }

  @Override
  public void dispose() {
    for (Trash trash : content) {
      trash.dispose();
    }
  }
}
```

```
public class BioWaste extends Trash {

  private boolean hazardous = false;

  public BioWaste(String name) {
    super(name);
  }

  public boolean isHazardousWaste() {
    return hazardous;
  }
}

public class PlasticWaste extends Trash {

  private boolean hazardous = false;

  public PlasticWaste(String name) {
    super(name);
  }

  public boolean isHazardousWaste() {
    return hazardous;
  }
}

public class BulkyWaste extends Trash {

  private boolean hazardous = false;
  private int volume = 25;

  public BulkyWaste(String name) {
    super(name);
  }

  public boolean isHazardousWaste() {
    return hazardous;
  }
}

public class HazardousWaste extends Trash {

  private boolean hazardous = true;
  public Trash parent;

  public HazardousWaste(String name) {
    super(name);
  }

  public boolean isHazardousWaste() {
    return hazardous;
  }
}
```

Korrektur

Diese Seite wird von den Korrektoren ausgefüllt.

Aufgabe	Erreichte Punkte	Mögliche Punkte
1 Softwareentwicklungsprozess		7
2 Systemsichten und Diagrammarten		6
3 Phasen der Softwareentwicklung und Diagramme		10
4 Management		7
5 Struktogramm		9
6 Netzplan		14
7 Algebraische Spezifikation		12
8 Lineare Temporallogik		13
9 UML-Objektdiagramm		8
10 Design-Pattern		14
Gesamtpunktzahl		100

Note: _____