UNIVERSITÄT ZU LÜBECK
INSTITUTE FOR SOFTWARE ENGINEERING
AND PROGRAMMING LANGUAGES

isp

Software Engineering Wintersemester 2018/19
Prof. Dr. Martin Leucker, Torben Scheffel 28.03.2019

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 jedem zusätzlichen Zettel, den Sie benutzen, wie vorher erläutert.

Vorname: _____

Nachname: _____

Geburtsdatum: _____

Matrikelnummer: _____

Studiengang: _____

Viel Erfolg!

Aufgabe 1: Phasen der Softwareentwicklung **7 Punkte (3/2/2)**

a) Beschreiben Sie kurz die Implementierungsphase und die Integrationsphase. Was passiert in welcher Phase? Nennen Sie auch Unterschiede der beiden Phasen.

b) Nennen Sie zwei Aktivitäten, die typischerweise in der Wartungsphase ausgeführt werden.

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
Die Entwurfsphase findet nach der Spezifikationsphase statt.	☐	☐
Ergebnisse der Spezifikationsphase werden im Anforderungsdokument festgehalten.	☐	☐
UML-Paketdiagramme werden typischerweise in der Integrationsphase erstellt.	☐	☐
UML-Zustandsdiagramme werden zum Festhalten der Anforderungen eingesetzt.	☐	☐

Aufgabe 2: Management **9 Punkte (3/4/2)**

a) Nennen Sie drei formale Qualifikationen, die in der Software- und Systementwicklung relevant sind.

b) Beschreiben Sie, wofür die vier Ohren im Vier-Ohren-Modell 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
LGPL-lizensierter Source-Code kann in proprietären Produkten verwendet werden.	☐	☐
Bei Inspektionen wird mit JUnit getestet.	☐	☐
Es ist immer besser, wenn die Wartung nicht der ursprüngliche Entwickler durchführt.	☐	☐
Review ist eine manuelle Prüfmethode.	☐	☐

Aufgabe 3: Vorgehensmodelle **8 Punkte (4/4)**

a) Beschreiben Sie, was nicht-inkrementelle Vorgehensmodelle ausmacht.

b) Beschreiben Sie, was prototyp-orientierte Vorgehensmodelle ausmacht. Erklären Sie auch, wofür der Prototyp genutzt wird.

Aufgabe 4: Planungsphase **7 Punkte (5/2)**

a) Nennen Sie zwei Methoden zur Aufwandsschätzung und beschreiben Sie jeweils kurz, wie diese funktionieren.

b) 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
Meilensteine können nicht in Netzplänen dargestellt werden.	☐	☐
In einem Projektstrukturplan wird der genaue zeitliche Ablauf eines Projektes dargestellt.	☐	☐
Eine Meilensteintrendanalyse stellt den Projektfortschritt dar.	☐	☐
Ein kritischer Pfad ist eine Folge von Vorgängen ohne Pufferzeit.	☐	☐

Aufgabe 5: Programmablaufplan **10 Punkte (2/8)**

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 und Struktogramme stellen die selben Zusammenhänge dar.	☐	☐
Programmablaufpläne stellen den Datenfluss dar.	☐	☐
UML-Zustandsdiagramme und Programmablaufpläne haben den selben Zweck.	☐	☐
In Struktogrammen können keine Schleifen dargestellt werden.	☐	☐

b) Zeichnen Sie einen Programmablaufplan für folgenden Codeausschnitt:

```
public int func(int x, int y) {
  int z = x + y;
  for(int i = 0; i < z; i++) {
    if (i > 7) {
      i = i * i;
    } else if (i < 4) {
      z = z - i;
      i = x - y;
    }
    if (z < 3) {
      z = func(i, z);
    }
  }
  z = z - 2;
  return z;
}
```

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

Aufgabe 6: Testen **10 Punkte (2/8)**

a) 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
Mit Testen kann man die komplette Abwesenheit von Fehlern zeigen.	☐	☐
Mit JUnit werden Akzeptanztests durchgeführt.	☐	☐
Wenn man Runtime Verification nutzt, ist Model Checking überflüssig.	☐	☐
Code-Coverage gibt an, wie viel Prozent der Testfälle fehlerfrei durchliefen.	☐	☐

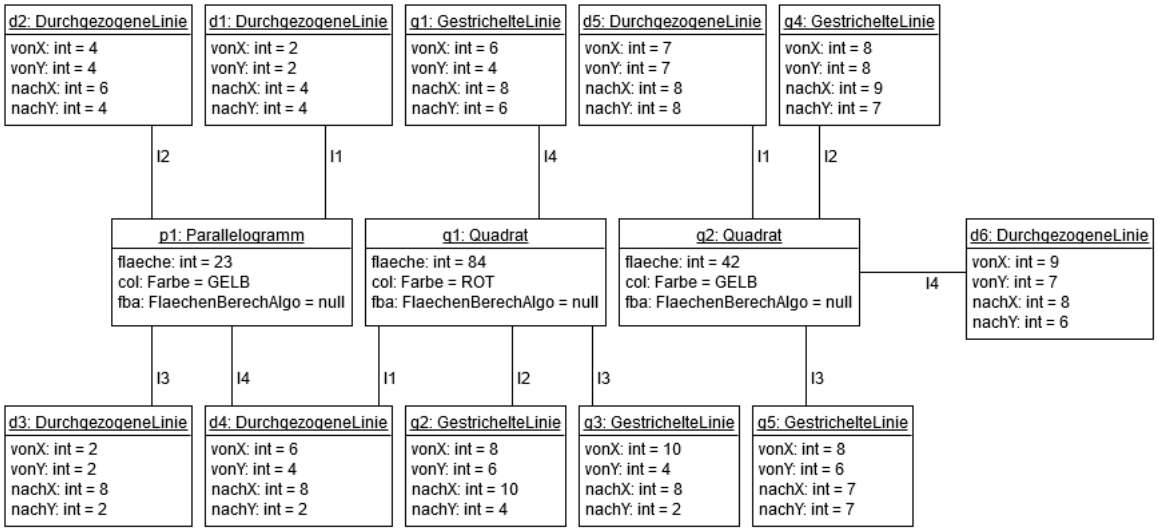
b) Vervollständigen Sie den folgenden JUnit-Testfall an den gekennzeichneten Stellen geeignet. Sie dürfen sowohl JUnit 4 als auch JUnit 5 Syntax nutzen. Gehen Sie davon aus, dass alle benötigten Imports vorhanden sind.

Der Testfall testet zwei Methoden einer Klasse `IntSet` mit dem Konstruktor `public IntSet()`. Die zu testenden Methoden sind `size()` und `contains(int i)`. Erstere soll die Größe der Menge (als `int`) zurückgeben, `contains(i)` soll `true` zurückgeben, falls der Wert `i` in der Menge vorkommt und sonst `false`. Der Aufruf `remove(i)` entfernt das Element `i` aus der Menge, wenn es enthalten ist.

```
public class IntSetTest {
  Set set;

  @Before
  public void setUp() {

    ________________________
  }

  ________________________
  public void testSize() {
    set.add(3);
    set.add(7);

    ________________________(set.size(), 2);
    set.remove(5);
    assertTrue(set.size() == ______);

    ________________________

    ________________________
    assertTrue(set.size() == 0);
    set.add(6);
  }

  ________________________
  public void testContains() {
    set.add(2);
    set.add(8);

    ________________________(set.contains(2));

    ________________________(set.contains(6));
  }
}
```

Aufgabe 7: UML-Objekt- und Klassendiagramm **10 Punkte**

Betrachten Sie folgendes UML-Objektdiagramm:



Erstellen Sie ein UML-Klassendiagramm, aus dem dieses Objektdiagramm hervorgegangen sein könnte. Hierbei soll jede Klasse dargestellt werden, die nicht zur Java-Standard-API gehört. Entwerfen Sie dabei sinnvolle Oberklassen und Vererbungshierachien, wenn angebracht. Benutzen Sie Assoziationen wo immer möglich und vermeiden Sie Redundanzen.

Aufgabe 8: Algebraische Spezifikation **14 Punkte (2/7/2/3)**

Betrachten Sie folgende algebraische Spezifikation. Dabei sei `Nat` wie aus der Vorlesung bekannt.

```
1  spec Poly = Nat then
2  sorts
3  poly = empty | make(poly,nat)
4  ops
5  terms: (poly) nat,
6  addPoly: (poly,poly) poly,
7  evaluate: (poly,nat) nat
8  vars
9  p,r: poly,
10 a,b,n: nat
11 axioms
12 terms(empty) = zero
13 terms(make(p,zero)) = terms(p)
14 terms(make(p,succ(a))) = succ(terms(p))
15 addPoly(empty,p) = addPoly(p,empty) = p
16 addPoly(make(p,a),make(r,b)) = make(addPoly(p,r),add(a,b))
17 evaluate(empty,n) = zero
18 evaluate(make(p,a),n) = add(mult(evaluate(p,n),n),a)
19 end
```

a) Beschreiben Sie, was Zeile 18 informell bedeutet.

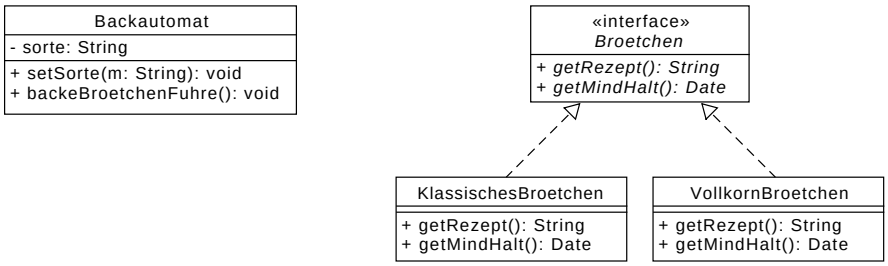
b) Entwickeln Sie ein Modell $\mathcal{A}$, welches das Erzeugungsprinzip erfüllt. Geben Sie auch die Trägermenge an. Nehmen Sie dabei an, dass $\mathcal{B}$ ein geeignetes Modell für Nat ist.

c) Weisen Sie für Ihr Modell das Erzeugungsprinzip nach. Nehmen Sie dabei an, dass $\mathcal{B}$ ein geeignetes Modell für Nat ist.

d) Zeigen Sie, dass das Axiom in Zeile 16 von Ihrem Modell erfüllt wird. Nehmen Sie dabei an, dass $\mathcal{B}$ ein geeignetes Modell für Nat ist.

Aufgabe 9: Design-Pattern **13 Punkte (3/4/6)**

Eine Bäckerei hat ihre Brötchen-Produktion automatisiert. Jeder Backautomat wird durch ein `Backautomat`-Objekt in Java repräsentiert, welches die Teigzusammenstellung sowie die Ofenansteuerung kontrolliert. Ein Aufruf der `backeBroetchenFuhre`-Methode führt zur Produktion von 50 Brötchen. Brötchen gibt es in zwei Sorten, in *klassisch* und in *Vollkorn*; die gewünschte Sorte kann mit `setSorte` vor dem Backvorgang eingestellt werden. Jedes konkrete Brötchen, das erst zubereitet und dann gebacken werden soll, wird durch genau ein `Broetchen`-Objekt repräsentiert. Dieses gibt Aufschluss z.B. über das Mindesthaltbarkeitsdatum sowie das Rezept (abrufbar über `getMindHalt` bzw. `getRezept`). Für die Zukunft sind weitere Sorten geplant, z.B. *Lübecker Spezial*.



Im Folgenden ist ein Auszug aus der `backeBroetchenFuhre`-Methode gegeben.

```
public void backeBroetchenFuhre() {
  List<Broetchen> inhalt = new LinkedList<>();

  // 1) Broetchen-Objekte erstellen
  for (int i = 0; i < 50; i++) {
    if (sorte.equals("klassisch")) {
      inhalt.add(new KlassischesBroetchen());
    } else if (sorte.equals("Vollkorn")) {
      inhalt.add(new VollkornBroetchen());
    }}
  // 2) Teig unter Beruecksichtigung der Elemente aus `inhalt` zubereiten
  // 3) Ofen ansteuern
}
```

a) Nennen Sie zwei Anti-Pattern und erläutern Sie, in welcher Form diese in dem Kontext der Aufgabenstellung auch hätten auftreten können.

b) Auch die Brotproduktion soll nun automatisiert werden, weshalb es eine neue Methode `backeBrotFuhre` geben soll, die das Backen von 20 Broten veranlasst. Brote soll es ebenfalls in allen Sorten geben, wobei das genaue Rezept bzw. die Berechnung des Mindesthaltbarkeitsdatums für jede Brot- und Brötchensorte einzeln festgelegt wird. Passen Sie das UML-Diagramm sinnvoll an (ohne gezielt ein Design-Pattern zu verwenden) und zeichnen Sie es vollständig neu.

c) Wenden Sie ein Design-Pattern(z.B. das Abstract Factory Pattern) an, um die Klasse `Backautomat` von den konkreten Sorten zu entkoppeln. Wenn die Bäckerei eine neue Sorte aufnimmt, soll die Klasse `Backautomat` nicht mehr verändert werden müssen. Passen Sie das **ursprüngliche** UML-Diagramm (ohne Brot) an und zeichnen Sie es als vollständiges UML-Diagramm neu. Notieren Sie, welches Pattern Sie genutzt haben.

Aufgabe 10: Lineare Temporallogik **13 Punkte (2/6/5)**

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

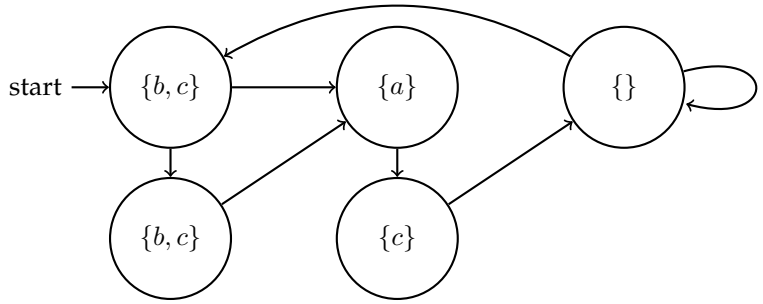
a) Geben Sie eine LTL-Formel an, die von dem Lauf $\{a, b\}\{\}(\{c\}\{a, b, c\})^\omega$ erfüllt wird und nicht semantisch äquivalent zu $true$ ist.

Geben Sie eine LTL-Formel an, die von dem Lauf $\{a, b, c\}^\omega$ nicht erfüllt wird und nicht semantisch äquivalent zu $false$ ist.

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

	Wahr	Falsch
Es gibt genau zwei unendliche Läufe, die die Formel $(\mathcal{G}\, a) \vee \neg\, \mathcal{G}\, a$ erfüllen.	☐	☐
$\mathcal{G}((a \vee \neg a)\, \mathcal{U}\, \neg a)$ ist semantisch äquivalent zu *true*.	☐	☐
Jeder Lauf, der $a\, \mathcal{U}(b \rightarrow c)$ erfüllt, erfüllt auch $(\mathcal{F}\, b) \wedge (\mathcal{F}\, c)$.	☐	☐
$c\, \mathcal{R}(a \rightarrow \mathcal{F}\, b)$ und $\mathcal{G}(a \rightarrow \mathcal{F}\, b) \vee ((a \rightarrow \mathcal{F}\, b)\, \mathcal{U}(c \wedge (a \rightarrow \mathcal{F}\, b)))$ sind semantisch äquivalent.	☐	☐
Der Lauf $\{\}^\omega$ erfüllt $\mathcal{G}(c \rightarrow \mathcal{X}\, b) \wedge \mathcal{G}(\neg a \rightarrow \mathcal{X}\, \mathcal{F}\, \neg c)$.	☐	☐
Der Lauf $(\{a\}\{b, c\}\{a\}\{a, c\}\{b\})^\omega$ erfüllt $\mathcal{G}((a\, \mathcal{U}\, b)\, \mathcal{U}\, c)$.	☐	☐

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
$\mathcal{G}(a \rightarrow \mathcal{X}\, \mathcal{X}\, \mathcal{X}\, \mathcal{X}\, a)$	☐	☐
$\mathcal{G}\, \mathcal{F}(\neg(a \vee b \vee c))$	☐	☐
$c \rightarrow b$	☐	☐
$\mathcal{X}(a\, \mathcal{U}\, c) \vee \mathcal{F}(a \wedge c)$	☐	☐
$(\mathcal{G}(a \vee c))\, \mathcal{U}(\neg a \wedge \neg b \wedge \neg c)$	☐	☐

Korrektur

Diese Seite wird von den Korrektoren ausgefüllt.

Aufgabe	Erreichte Punkte	Mögliche Punkte
1 Phasen der Softwareentwicklung		7
2 Management		9
3 Vorgehensmodelle		8
4 Planungsphase		6
5 Programmablaufplan		10
6 Testen		10
7 UML-Objekt- und Klassendiagramm		10
8 Algebraische Spezifikation		14
9 Design-Pattern		13
10 Lineare Temporallogik		13
Gesamtpunktzahl		100

Note: _____