



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: Softwarequalität

7,5 Punkte (1/1/1,5/4)

- a) Nennen Sie zwei analytische Maßnahmen zur Qualitätssicherung.

- b) Nennen Sie zwei organisatorische Maßnahmen zur Qualitätssicherung.

- c) Beschreiben Sie kurz die drei Schritte bei der Qualitätssicherung durch formale Methoden.

- d) Erläutern Sie kurz die Art und Weise und in welchem Schritt Transitionssysteme und Temporallogiken bei der formalen Qualitätssicherung angewendet werden können und wie sie ineinander greifen.

Name:

Matrikelnummer:

Aufgabe 2: UML in der Softwareentwicklung

9,5 Punkte (6,5/3)

- a) Ordnen Sie die folgenden Diagramme in die richtige Sicht ein. Es gibt +0.5 Punkte für korrekte Kreuze und −0.5 Punkte für falsche.
- b) Nennen Sie einige Vor- und Nachteile von UML, insgesamt jedoch sechs.

Name:

Matrikelnummer:

Aufgabe 3: Sequenzdiagramme

11,5 Punkte

In dieser Aufgabe soll der Aufbau einer Kommunikation und die anschließende Interaktion zwischen zwei Agenten in einem dynamischen Multiagenten System modelliert werden.

Ein Agent besteht aus einem `HostingService` und potentiell mehreren `HostedServices`. Zudem gibt es `Clients`, die die Dienste nutzen.

Der Discovery-Vorgang kann auf zwei alternative Arten erfolgen: Entweder *explicit*, der `Client` sendet eine `Probe` Nachricht an den `HostingService`, die mit einer `ProbeMatches` Nachricht beantwortet wird. Daraufhin wird eine `Resolve` Nachricht an den `HostingService` gesendet, die mit `ResolveMatches` beantwortet wird.

Oder alternativ *implicit*, dabei sendet der `HostingService` eine `Hello` Nachricht an den `Client`.

Nachdem der Discovery-Vorgang abgeschlossen ist, ruft der `Client` mit einer `Get` Nachricht die Menge der `HostedServices` ab. Die Antwort darauf ist eine `GetResponse` Nachricht.

Als nächstes wird die Beschreibung des `HostedService` durch eine `GetMetaData` Nachricht abgerufen und mit `GetMetaDataResponse` beantwortet.

Erst jetzt kann die eigentliche Interaktion stattfinden. Der `Client` fragt den Wert einer Metrik durch `GetMetricValue` ab. Der `HostedService` berechnet intern den Wert der Metrik und beantwortet die Anfrage mit `GetMetricValueResponse`. Gehen Sie davon aus, dass es sich hierbei um eine numerische Metrik mit `Integer`-Wertebereich handelt.

Zeichnen Sie genau ein Sequenzdiagramm, das den gesamten beschriebenen Ablauf darstellt. Gehen Sie dabei von jeweils einem `Client`, `HostingService` und einem `HostedService` aus.

Benutzen Sie aus Platzgründen die nächste Seite für die Bearbeitung.

Name:

Matrikelnummer:

Name:

Matrikelnummer:

Aufgabe 4: Anforderungsfestlegung und Artefakte

6,5 Punkte (1/1/4,5)

- a) Nennen Sie zwei Teilgebiete der Anforderungsfestlegung.

- b) Nennen Sie zwei Funktionen des Anforderungsdokuments.

- c) Nennen Sie drei Untergruppen nicht-funktionaler Anforderungen und geben Sie jeweils zwei Beispiele.

Name:

Matrikelnummer:

Aufgabe 5: Algebraische Spezifikation

12 Punkte (12)

Im Folgenden soll ein abstrakter Datentyp `Vec` für zweidimensionale Vektoren über den natürlichen Zahlen angegeben werden. Vektoren sollen durch die Funktion `vector` erzeugt werden. Außerdem sollen drei weitere Operationen unterstützt werden,

- die Operation `addv` addiert zwei Vektoren elementweise,
- die Operation `dot` bestimmt das Skalarprodukt zweier Vektoren und
- die Operation `orth` bestimmt ob zwei Vektoren orthogonal zueinander stehen.

Geben Sie eine algebraische Spezifikation für `Vec` an. Dabei dürfen die aus der Vorlesung bekannten Spezifikationen für `Bool` und `Nat` benutzt werden, die über verschiedene Operationen verfügen. Zur Erinnerung: Zwei Vektoren stehen orthogonal zueinander, wenn ihr Skalarprodukt 0 ist und das Skalarprodukt berechnet sich wie folgt:

$$\begin{pmatrix} a_1 \\ a_2 \end{pmatrix} \bullet \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} = a_1 b_1 + a_2 b_2$$

Name:

Matrikelnummer:

Aufgabe 6: Planung der Use-Cases

13 Punkte (7/2/3/1)

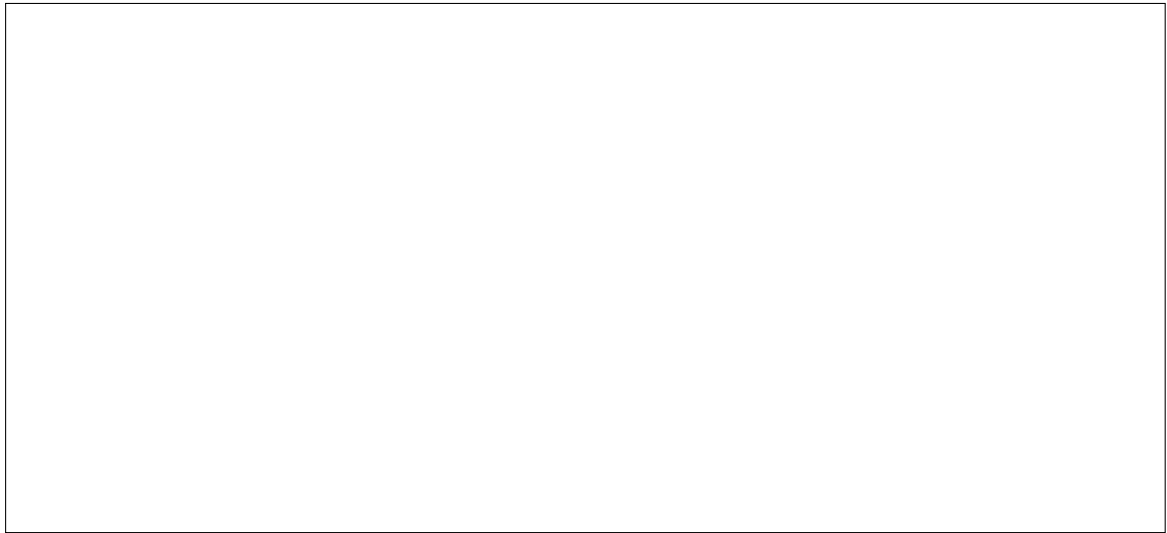
Im Folgenden wird ein System zur Organisation des Übungsbetriebs beschrieben. Mit dem System sollen Dozenten in die Lage versetzt werden einen Übungstermin festzulegen. Dafür müssen sie auch immer einen Raum buchen.

Dozenten und Tutoren sollen die Punkte einer Übung festlegen können.

Studenten sollen sich für die Übungstermine einschreiben können. Zudem müssen sie Übungen als besucht markieren können, dazu gehört auch das Einreichen einer Lösung, falls ihre Punkte noch unter dem Zielwert für die Zulassung liegen.

Aus Kostengründen werden ausschließlich Studenten als Tutoren eingesetzt.

- a) Erstellen Sie ein Use-Case-Diagramm (Anwendungsfalldiagramm) zu dem beschriebenen System.



- b) Sie erhalten von den Nutzern einige Rückmeldungen: So schätzen die Dozenten ihre Oberfläche als recht komplex ein, da sie den Aufwand für Festlegung eines Termins für durchschnittlich halten und die Raumbuchung sogar für sehr komplex. Ebenso beschweren sich die Studierenden über eine komplexe Schnittstelle. Sie empfinden die Einschreibung und Markieren einer Übung zwar als einfach durchführbar, allerdings empfinden sie das Einreichen einer Lösung als sehr komplex. Nur die Tutoren schätzen ihren Zugang als einfach ein, auch wenn sie die Punktefestlegung bereits im mittleren Komplexitätsbereich sehen.

Bestimmen Sie die Unadjusted Use-Case-Points(UUCP).

Machen Sie ihren Rechenweg kenntlich



Name:

Matrikelnummer:

- c) Eine erfahrenere Kollegin hat entschieden, dass alle technischen Einflussfaktoren mit mittlerer Bedeutung von 4 bewertet werden und alle umgebungsbezogenen Einflussfaktoren mit einer Bedeutung von 2.

Bestimmen Sie den TCF- und den ECF-Wert.

Die einzelnen Gewichtungsfaktoren finden Sie im Anhang.

Machen Sie ihren Rechenweg kenntlich

- d) Setzen Sie die Werte in die Formel zur Berechnung des UCP-Werts ein. Das Ergebnis muss nicht berechnet werden.

Machen Sie ihren Rechenweg kenntlich

Name:

Matrikelnummer:

Aufgabe 7: Programmablaufplan

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.

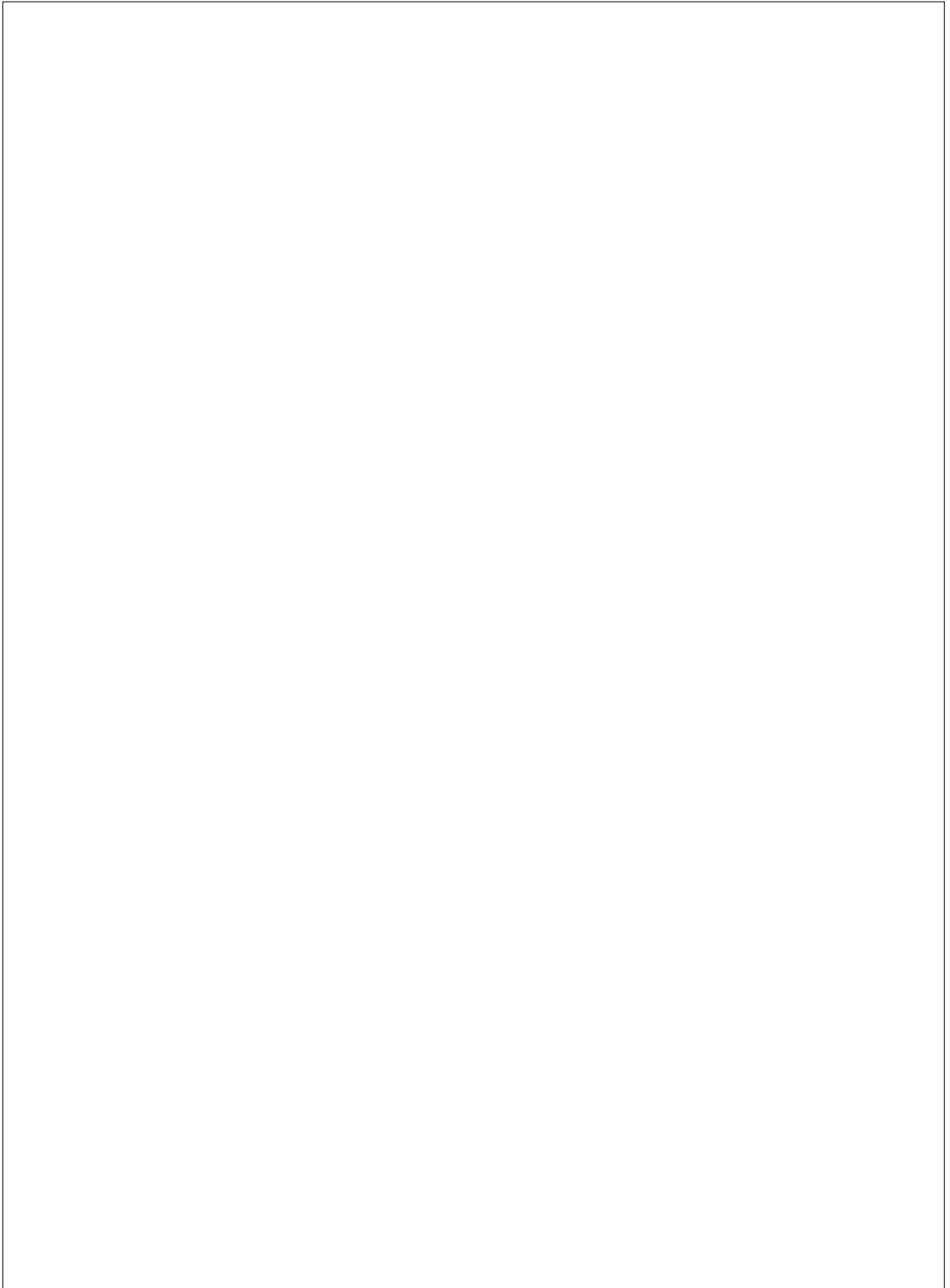
b) Zeichnen Sie einen Programmablaufplan für folgenden Pseudocode: Bei `initialize` und `distance_update` handelt es sich um Teilalgorithmen, die nicht im Detail betrachtet werden.

```
Dijkstra(graph, start) {  
    initialize(graph, start, distance[], pre[], q)  
3    while (q is not empty) {  
        u := node in q with smallest value in distance  
        remove u from q  
6        for (every neighbour v of u) {  
            if (v is in q) {  
                distance_update(u, v, distance, pre)  
9            }  
        }  
12    return pre  
}
```

Benutzen Sie aus Platzgründen die nächste Seite für die Bearbeitung.

Name:

Matrikelnummer:

A large, empty rectangular box with a thin black border, occupying the central portion of the page. It is intended for a drawing or a detailed answer.

Name:

Matrikelnummer:

Aufgabe 8: Lineare Temporallogik

13 Punkte (2/6/5)

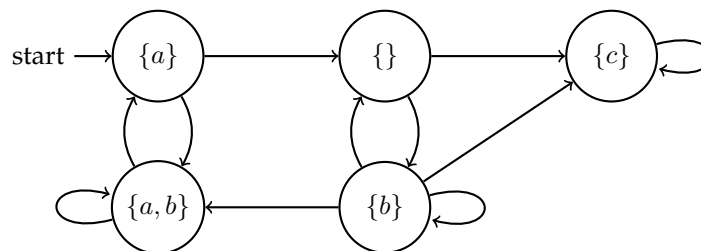
Sei im Folgenden $AP = \{a, b, c\}$ und $\Sigma = 2^{AP}$.

- a) Sei $(\mathcal{F}c) \rightarrow (\neg b\mathcal{U}(a \vee c))$ 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.
- 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.



Name:

Matrikelnummer:

Aufgabe 9: Management

6 Punkte (2/2/1/1)

- a) Entscheiden Sie, ob folgende Aussagen wahr oder falsch sind. Es gibt +0.5 Punkt für korrekte Kreuze und −0.5 Punkt für falsche.
- b) Nennen Sie jeweils zwei Vor- und Nachteile, die auftreten können, wenn die ursprünglichen Entwickler auch die Wartung der Software übernehmen.

- c) Nennen Sie zwei Vorteile des Reverse Engineering

- d) Nennen Sie zwei Rollen, die beim Release Management beteiligt sind.

Name:

Matrikelnummer:

Aufgabe 10: Design-Pattern

12 Punkte (6/2/4)

Das Programm im Anhang dient der Verwaltung von Projekten mit ihren Aufgaben und Teilergebnissen.

- a) Welches Design-Pattern wurde zur Implementierung des Kostenberechnungsalgorithmus angewendet? Erläutern Sie, woran Sie dies festmachen. Benennen Sie gegebenenfalls einzelne Akteure. 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) Das Programm soll nun neben den Gesamtkosten auch die insgesamt benötigte Zeit für das Projekt berechnen können. Beschreiben Sie kurz das prinzipielle Vorgehen.

- c) Das Programm soll nun neben Aufgaben und Teilergebnissen auch Projektressourcen verwalten, die ebenfalls mit Kosten verbunden sind. Vergleichen Sie diese Änderung mit der vorhergehenden in Bezug auf Umfang und Komplexität. Gehen Sie kurz auf mögliche Gründe ein.

Korrektur

Diese Seite wird von den Korrektoren ausgefüllt.

Aufgabe	Erreichte Punkte	Mögliche Punkte
1 Softwarequalität		7,5
2 UML in der Softwareentwicklung		9,5
3 Sequenzdiagramme		11,5
4 Anforderungsfestlegung und Artefakte		6,5
5 Algebraische Spezifikation		12
6 Planung der Use-Cases		13
7 Programmablaufplan		9
8 Lineare Temporallogik		13
9 Management		6
10 Design-Pattern		12
Gesamtpunktzahl		100

Note: _____

Diese Seite enthält keinen Inhalt.

Anhang

T	Beschreibung	Gewicht	T	Beschreibung	Gewicht
T1	Verteiltes System	2	T8	Portierbar	2
T2	Performanz	1	T9	Leicht veränderbar	1
T3	End-User Effizienz	1	T10	Nebenläufig	1
T4	Complex processing	1	T11	Security features	1
T5	Wiederverw. Code	1	T12	Zugriff von Dritten	1
T6	Leicht installierbar	0,5	T13	Spezielle Schulung	1
T7	Leicht zu benutzen	0,5			

T	Beschreibung	Gewicht	T	Beschreibung	Gewicht
E ₁	RUP	1,5	E ₅	Motivation	1
E ₂	Anwendungserfahrung	0,5	E ₆	Stabile Anforderungen	2
E ₃	OO-Erfahrung	1	E ₇	Teilzeit Kräfte	-1
E ₄	Chef Analyst verfügbar	0,5	E ₈	Schwierige Programmiersprache	-1

```

2 public interface ProjectItem {
    void accept(Foo foo);
}

```

java/src/ProjectItem.java

```

import java.util.ArrayList;
import java.util.List;
3
public class Project {
    String name;
6    List<ProjectItem> projectItems;

    public Project(String name) {
9        this.name = name;
        this.projectItems = new ArrayList<>();
    }
12
    public List<ProjectItem> getProjectItems() {
        return projectItems;
15    }
}

```

java/src/Project.java

```

public class Task implements ProjectItem {
2    String name;
    double timeRequired;

5    public Task(String name, double time) {
        this.name = name;
        this.timeRequired = time;
8    }

    @Override
11    public void accept(Foo foo) {
        foo.barTheTask(this);
    }
}

```

14 }

java/src/Task.java

```
1 public class Deliverable implements ProjectItem {
    String name;
    double cost;
4
    public Deliverable(String name, double cost) {
        this.name = name;
7        this.cost = cost;
    }
10
    @Override
    public void accept(Foo foo) {
        foo.barTheDeliverable(this);
13    }
}
```

java/src/Deliverable.java

```
1 public interface Foo {
    void barTheDeliverable(Deliverable deliverable);
    void barTheTask(Task task);
4 }
```

java/src/Foo.java

```
// Berechnet die Kosten aller Projektbestandteile
2 public class CostFoo implements Foo {
    double totalCost = 0;
5
    @Override
    public void barTheDeliverable(Deliverable deliverable) {
        totalCost += deliverable.cost;
8    }

    @Override
11    public void barTheTask(Task task) {
    }
}
```

java/src/CostFoo.java