



Klausur Software Engineering

Aufgaben und Punkte

Die Bearbeitungszeit der Klausur umfasst 90 Minuten. Es gibt 10 Aufgaben mit insgesamt 100 erreichbaren Punkten. Die erreichbaren Punkte pro Aufgabe sind für jede Aufgabe oben rechts angegeben, ebenso die erreichbaren Punkte pro Teilaufgabe (in Klammern dahinter).

Notieren Sie Ihre Lösungen 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 dieser zusätzlichen Blätter.

Hilfsmittel

Es sind keine Hilfsmittel zugelassen mit Ausnahme eines eigenhändig (beidseitig) beschriebenen DIN A4-Zettels.

Persönliche Daten

Notieren Sie im Folgenden Ihre persönlichen Daten. Notieren Sie Ihren Namen und Ihre Matrikelnummer außerdem auf jedem zusätzlich gestellten Blatt.

Vorname: _____

Nachname: _____

Matrikelnummer: _____

Viel Erfolg!

Aufgabe 1: Phasen der Softwareerstellung

10 Punkte (4/6)

- a) Nennen Sie vier Phasen der Softwareerstellung, **außer** der Spezifikationsphase, und ordnen Sie sie in der Reihenfolge an, in der sie normalerweise im Entwicklungsprozess durchlaufen werden.

- b) Erläutern Sie, was während der Entwicklung einer Software fehlt, wenn die Spezifikationsphase ausgelassen wird (alle anderen Phasen jedoch durchlaufen werden). Erläutern sie auch, welche Probleme dadurch im weiteren Verlauf der Entwicklung der Software auftreten können.

Aufgabe 2: Vorgehensmodelle

9 Punkte (3/3/3)

- a) Benennen Sie zwei Unterschiede zwischen dem Lebenszyklusmodell und einem prototyporientierten Vorgehensmodell.

- b) In jedem Vorgehensmodell sind Tests vorgesehen, welche in zwei Kategorien „Verifikation“ und „Validierung“ unterschieden werden können. Was bedeuten diese beiden Kategorien jeweils?

- c) Beschreiben Sie zwei Elemente/Rollen, die spezifisch für das Scrum-Vorgehensmodell sind.

Aufgabe 3: Management

10 Punkte (3/3/4)

- a) Beschreiben Sie in einem Satz, was man unter einem Legacy-System versteht. Nennen Sie zudem zwei Eigenschaften, die ein Legacy-System typischerweise aufweist.

- b) Grenzen Sie die Begriffe Re-Engineering und Reverse Engineering voneinander ab, indem Sie beide Begriffe in je einem Satz beschreiben.

- c) Zur Qualitätssicherung von Software bieten sich sowohl i) manuelle als auch ii) (teil-)automatische Prüfmethode an. Geben Sie jeweils zu i) und ii) ein Beispiel an sowie eine Beschreibung in einem Satz, wie die Methode funktioniert.

Aufgabe 4: Anwendungsfalldiagramme

8 Punkte

Zur Verwaltung ihres Zugbestands möchte ein deutscher Bahnkonzern eine Monitor-App entwickeln lassen, mit der die ZugbegleiterInnen eine Übersicht über die aktuellen Positionen und Bewegungen der Zugflotte erhalten können. Es gibt zwei Systeme: Die mobile App und das Server-Backend.

Die App läuft auf den mobilen Endgeräten der ZugbegleiterInnen. Als ZugbegleiterIn kann man sich eine Übersichtskarte anzeigen lassen. Ebenso kann man sich mit einer Filterfunktion alle Züge gemäß eines Suchkriteriums anzeigen lassen. Sowohl von der Übersichtskarte als auch von der Filterergebnisanzeige aus kann man anschließend eine Detail-Ansicht für einen Zug anfordern. Zur internen Kommunikation gibt es noch eine integrierte Chat-Funktion, die man bei Problemen und Fragen verwenden kann.

Das zentrale Server-Backend wird von SystemadministratorInnen verwaltet. Als SystemadministratorIn kann man einen neuen Zug hinzufügen, Informationen zu einem Zug aktualisieren sowie Züge aus dem System löschen. Auch als SystemadministratorIn hat man über eine direkte Datenbankschnittstelle Zugriff auf die Chat-Funktion.

Beschreiben Sie die Anwendungsfälle der zwei Systeme durch UML-Anwendungsfalldiagramme.

Aufgabe 5: Sequenzdiagramme

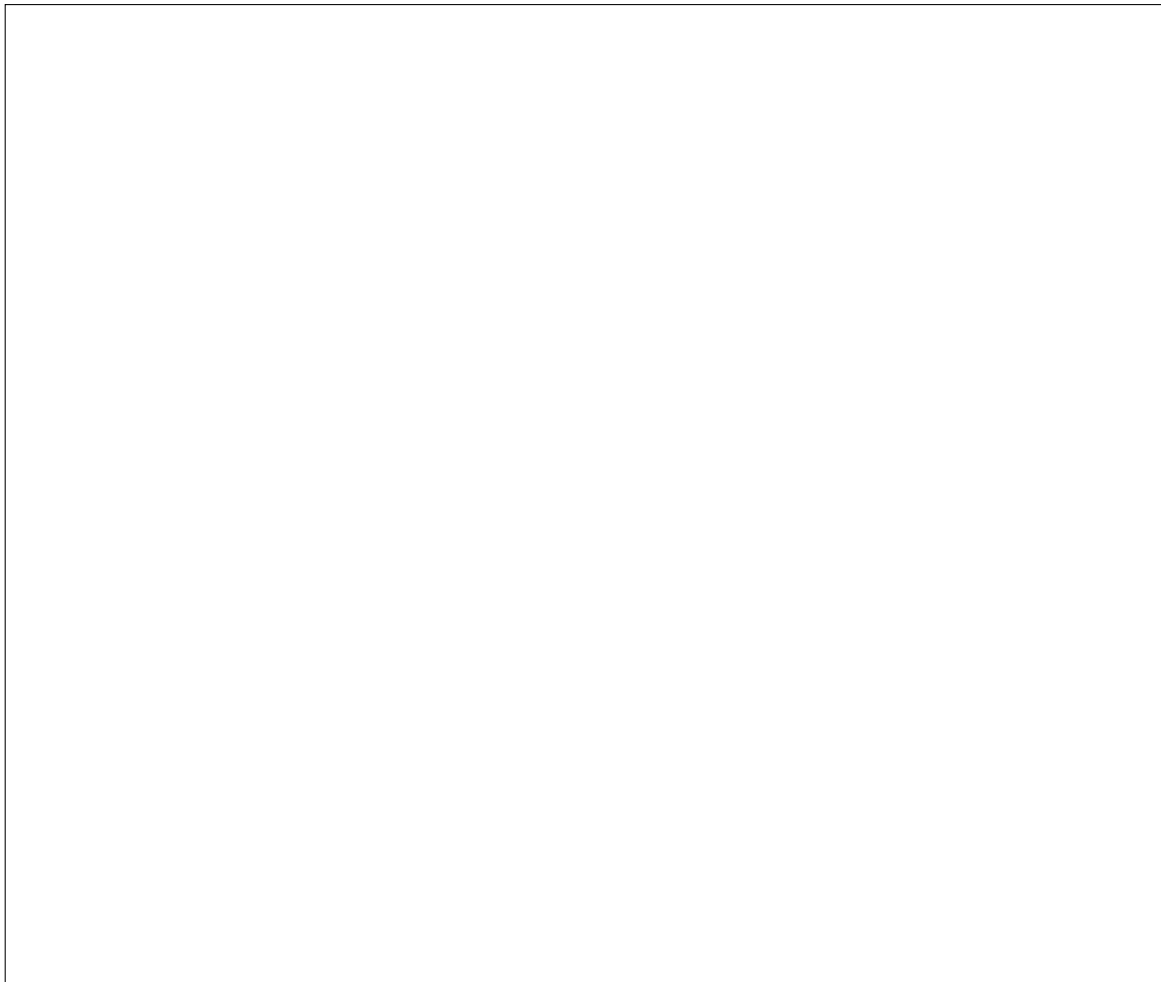
9 Punkte

Eine App für die MitarbeiterInnen eines deutschen Bahnkonzerns soll eine integrierte Chat-Funktion beinhalten. Die Chat-Funktion teilt sich in mehrere Chats (z.B. gibt es einen pro Zug, einen pro Bahnhof etc.). Sowohl Mitarbeiter-Accounts als auch Chats werden vom System als Objekte modelliert.

Durch einen Aufruf von außen (Mitarbeiter drückt auf *Login*) wird ein Registrierungsvorgang ausgelöst. Der zum Mitarbeiter gehörige *Account* kann sich bei einem *Chat* registrieren, wobei das *Account*-Objekt eine Referenz auf sich selbst übergibt. Die Registrierung ist abgeschlossen, nachdem der *Chat* eine Registrierungsbestätigung zurückgeschickt hat. Die Registrierung eines *Accounts* hat zur Folge, dass ihm bei einer neuen Chat-Nachricht diese übermittelt wird (auch wenn er selbst Sender dieser Nachricht war).

Um eine neue Nachricht in den Chat zu schreiben, muss zunächst ein Aufruf von außen kommen (Mitarbeiter drückt auf *Senden*). Dieser Aufruf hat zur Folge, dass das *Account*-Objekt die neue Nachricht an das *Chat*-Objekt sendet. Nun wird die Nachricht allen Objekten zugestellt, die sich im Chat zuvor registriert haben.

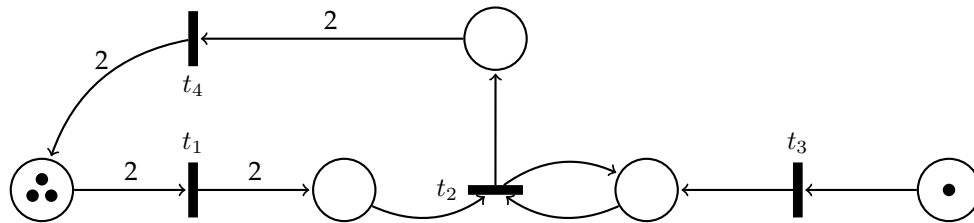
Geben Sie ein Sequenzdiagramm an für einen Chat und die Accounts der Mitarbeiter Peter und Maria: Initial führen sowohl Peter als auch Maria einen Registrierungsvorgang beim Chat durch (beide drücken auf *Login*). Anschließend drückt Peter nach Eingabe seiner Nachricht „Hallo von Peter!“ auf *Senden*. Der Chat soll sich dabei so verhalten, wie oben beschrieben.



Aufgabe 6: Petri-Netze

8 Punkte (5 / 1,5 / 1,5)

Es sei folgendes Petri-Netz gegeben:



- a) Zeichnen Sie den Erreichbarkeitsgraphen des oben gezeigten Petri-Netzes. Verwenden Sie zur Beschriftung der Knoten eine Vektordarstellung der Markierungen (von links nach rechts im gegebenen Netz, z.B. $\langle 3, 0, 0, 0, 1 \rangle$ für die initiale Markierung).

- b) Besitzt das Petri-Netz eine Verklemmung? Erläutern Sie ihre Antwort kurz.

- c) Ist das Petri-Netz lebendig? Erläutern Sie ihre Antwort kurz.

Aufgabe 7: Algebraische Spezifikation

13 Punkte (2/3/5/3)

Zur Verwaltung ihres Zugbestands möchte ein deutscher Bahnkonzern einen abstrakten Datentyp für Züge entwerfen lassen, um diesen anschließend als konkretes Modell zu implementieren. Eine Zuglokomotive (`lok`) ist von einer bestimmten Zuggattung (`gattung`), entweder ICE oder RE. Ein Wagon wird an einen Zug angehängt (`mitwagon`) und ist von einer bestimmten Wagenklasse (`klasse`), entweder erste Klasse oder zweite Klasse.

```
1 spec Zuege =  
2   sorts  
3     gattung = ice | re  
4     klasse = erste | zweite  
5     zug = lok (gattung) | mitwagon (klasse, zug)  
6   vars  
7     g: gattung  
8     k: klasse  
9     z: zug  
10  axioms  
11    ice  $\neq$  re  
12    erste  $\neq$  zweite  
13    lok(g)  $\neq$  mitwagon(k, z)  
14 end
```

- a) Geben Sie einen Ausdruck an, der eine ICE-Lok mit zwei Wagons beschreibt, beide zweite Klasse.

- b) Die Zuggattung der Lokomotive bestimmt die Zuggattung des gesamten Zuges. Die zusätzliche Operation `getGattung: (zug) gattung` soll die Zuggattung eines Zuges bestimmen. Geben Sie Axiome an, um diese zusätzliche Operation korrekt zu spezifizieren.

- c) Geben Sie für die Spezifikation `zuege` ein Modell \mathcal{A} an, welches auf Bits und Bitfolgen (Listen/Vektoren von Bits) basiert. Geben Sie dabei auch eine Interpretation für `getGattung` an (siehe *Aufg.teil b*). Die Trägermengen sind vorgegeben.

$$gattung^{\mathcal{A}} := \{0, 1\}$$

$$klasse^{\mathcal{A}} := \{0, 1\}$$

$$zug^{\mathcal{A}} := \{0, 1\}^* \times \{0, 1\}$$

Hinweis: Bitfolgen können als Vektoren notiert werden, z.B. $\langle \rangle \in \{0, 1\}^$ und $\langle 1, 0, 1 \rangle \in \{0, 1\}^*$.*

- d) Es soll eine neue Operation `nurErste: (zug) bool` geben, welche überprüft, ob alle Wagons eines Zuges der ersten Klasse angehören. Geben Sie Axiome an, um diese zusätzliche Operation korrekt zu spezifizieren.

Sie dürfen dabei annehmen, dass die Spezifikation

```
1 spec Bool =  
2   sorts bool = false | true  
3 end
```

bereits vordefiniert wurde und zur Verfügung steht.

Aufgabe 8: Lineare Temporallogik

13 Punkte (1/1/6/5)

Es seien $AP = \{a, b, c\}$ und $\Sigma = 2^{AP}$.

- a) Wie viele unendliche Läufe über Σ erfüllen die LTL-Formel $\mathcal{G}(a \wedge b \wedge c)$?

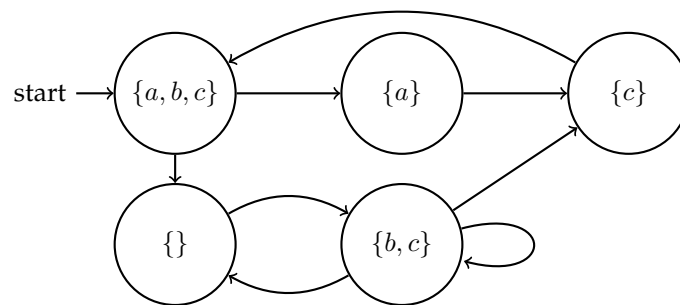
- b) Wie viele unendliche Läufe über Σ erfüllen die LTL-Formel $a \mathcal{U} b$?

- c) Entscheiden Sie, ob folgende Aussagen wahr oder falsch sind. Es gibt +1 Punkt für korrekte Kreuze und -1 Punkt für falsche (jedoch insgesamt mind. 0 Punkte).

Wahr Falsch

$(a \mathcal{U} b) \wedge \neg(a \mathcal{U} b)$ ist semantisch äquivalent zu <i>false</i> .	<input type="checkbox"/>	<input type="checkbox"/>
Jeder Lauf, der $\mathcal{G} b$ nicht erfüllt, erfüllt auch b nicht.	<input type="checkbox"/>	<input type="checkbox"/>
$a \mathcal{U} \mathcal{F} b \mathcal{F}$ ist eine syntaktisch korrekte LTL-Formel.	<input type="checkbox"/>	<input type="checkbox"/>
Der Lauf $\{\}\{\}\{a\}\{b\}\{a\}\{a, b\}\{c\}^\omega$ erfüllt $\mathcal{G}(b \rightarrow \mathcal{X} \mathcal{F} a)$.	<input type="checkbox"/>	<input type="checkbox"/>
Der Lauf $\{a, c\}\{a\}\{\}\{b\}(\{a, c\}\{b\}\{a\})^\omega$ erfüllt $\mathcal{F}(a \mathcal{U}(b \wedge c))$.	<input type="checkbox"/>	<input type="checkbox"/>
Jeder Lauf, der $\neg(a \vee b \vee c) \wedge \mathcal{F} b$ erfüllt, erfüllt auch $\mathcal{X}((a \vee \neg b \vee c) \mathcal{U} b)$.	<input type="checkbox"/>	<input type="checkbox"/>

- d) 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 (jedoch insgesamt mind. 0 Punkte).



Auf allen Läufen erfüllt Nicht auf allen Läufen erfüllt

$\mathcal{X} \mathcal{F}(b \vee c)$	<input type="checkbox"/>	<input type="checkbox"/>
$(\mathcal{G} \mathcal{F} a) \vee (\mathcal{G} \mathcal{F} b)$	<input type="checkbox"/>	<input type="checkbox"/>
$\mathcal{X} \mathcal{X} c$	<input type="checkbox"/>	<input type="checkbox"/>
$\mathcal{G}(\mathcal{X} c \rightarrow \neg c)$	<input type="checkbox"/>	<input type="checkbox"/>
$\mathcal{X}(c \mathcal{R} \neg a)$	<input type="checkbox"/>	<input type="checkbox"/>

Aufgabe 9: Implementierungsphase

8 Punkte (2/2/2/2)

- a) In einer Commit-Historie werden die Commits als gerichteter Graph dargestellt, wobei die Knoten die Commits sind und die Kanten anzeigen, welche Commits aufeinander basieren. Hat dieser Graph immer die Form eines Baumes? Begründen Sie kurz.

- b) Benennen Sie zwei Unterschiede zwischen einer Versionsverwaltung mit zentralem Repository und einer Versionsverwaltung mit verteilten Repositories.

- c) Benennen Sie zwei verschiedene Abläufe/Situationen, die man mit einem Build-Werkzeug automatisieren kann.

- d) Welche Aussagen zum Testen als Verifikationstechnik sind wahr? Es gibt +0,5 Punkte für korrekte Kreuze und -0,5 Punkte für falsche (jedoch insgesamt mind. 0 Punkte).

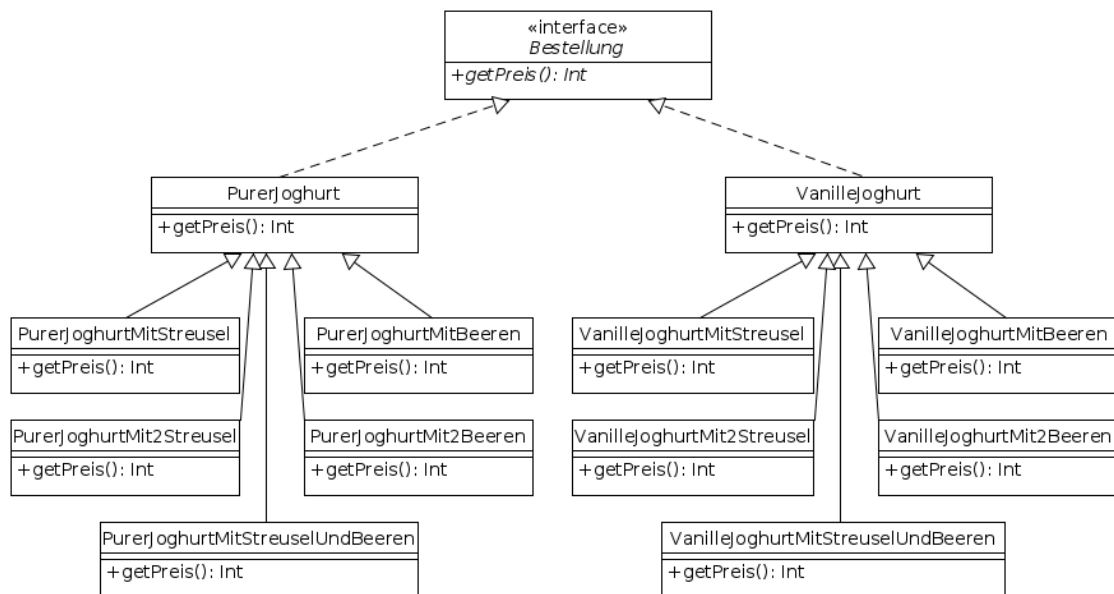
	Wahr	Falsch
Das Ziel des Testens ist die Abdeckung der <i>erwarteten</i> Eingaben.	<input type="checkbox"/>	<input type="checkbox"/>
Ein ausführlich getestetes Programm kann als fehlerfrei bezeichnet werden.	<input type="checkbox"/>	<input type="checkbox"/>
Es kann passieren, dass ein mit JUnit geschriebener Testfall bei Ausführung nicht terminiert.	<input type="checkbox"/>	<input type="checkbox"/>
In JUnit wird ein Testfall durch eine statische Klassenmethode repräsentiert.	<input type="checkbox"/>	<input type="checkbox"/>

Aufgabe 10: Design Patterns

12 Punkte (4/2/6)

Ein Hersteller von gefrorenem Joghurt möchte sein Abrechnungssystem von Ihnen verbessern lassen. Essentiell ist im Entwurf das Interface `Bestellung` mit der Methode `getPreis()`, da das System über diese Schnittstelle den Preis für den Kunden anzeigt und auch intern erfasst. Die verschiedenen Joghurtsorten *pur* und *Vanille* sind ebenso vorhanden wie die beiden Zusätze *Streusel* und *Beeren*. Zusätze können kombiniert und auch mehrfach bestellt werden (z.B. „Purer Joghurt mit zweifach Streusel“) und es ist absehbar, dass in Zukunft neue Joghurtsorten und neue Zusätze hinzukommen werden. Jede Joghurtsorte und jeder Zusatz hat einen Grundpreis, der dann bei einer Bestellung entsprechend aufaddiert wird.

Ein erster Entwurf sieht wie folgt aus.



Bei einer Bestellung „Purer Joghurt mit zweifach Streusel“ soll dabei folgender Java-Code ausgeführt werden.

```
Bestellung bestellung = new PurerJoghurtMit2Streusel();
```

a) Nennen Sie zwei Schwachpunkte des Entwurfs hinsichtlich der gegebenen Anforderungen.

- b) Zur Verbesserung des Entwurfs eignet sich das Decorator Pattern. Wie könnte nach Anwendung des Decorator Patterns der Java-Code zur Erzeugung der Bestellung „Purer Joghurt mit zweifach Streusel“ aussehen?

Bestellung bestellung =

- c) Ändern Sie das UML-Klassendiagramm, indem Sie das Decorator Pattern anwenden. Zeichnen Sie hierfür ein vollständiges UML-Klassendiagramm.

Korrektur

Diese Seite wird von den Korrektoren ausgefüllt.

Aufgabe	Erreichte Punkte	Mögliche Punkte
1 Phasen der Softwareerstellung		10
2 Vorgehensmodelle		9
3 Management		10
4 Anforderungsfestlegung		8
5 Sequenzdiagramme		9
6 Petri-Netze		8
7 Algebraische Spezifikation		13
8 Lineare Temporallogik		13
9 Implementierungsphase		8
10 Design Patterns		12
Gesamtpunktzahl		100

Note: _____