



## Klausur Software Engineering

### Aufgaben und Punkte

Die Bearbeitungszeit der Klausur umfasst 90 Minuten. Es gibt 8 Aufgaben mit insgesamt 80 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 neben einer Aufgabe stehen die Punkte für die 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).

### Persönliche Daten

Notieren Sie im Folgenden Ihre persönlichen Daten. Geben 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 an.

Vorname: \_\_\_\_\_

Nachname: \_\_\_\_\_

Geburtsdatum: \_\_\_\_\_

Matrikelnummer: \_\_\_\_\_

Studiengang: \_\_\_\_\_

Viel Erfolg!

**Aufgabe 1: Entwicklung von Softwaresystemen**

**8 Punkte (4/2/2)**

- a) Nennen Sie die acht Phasen der Softwareentwicklung und geben Sie kurz (in einem Satz) an, was in diesen Phasen passiert.

- b) Welche der Phasen der Softwareentwicklung ist üblicherweise die kostenintensivste. Welche Erkenntnis kann man daraus für den Softwareentwicklungsprozess ziehen?

*Matrikelnummer:*

- c) Beschreiben Sie das Vier-Ohren-Modell in eigenen Worten und erklären Sie warum die Kenntnis davon im Software-Entwicklungsprozess von Vorteil sein kann.

**Aufgabe 2: Zustandsbasierte Modellierung**

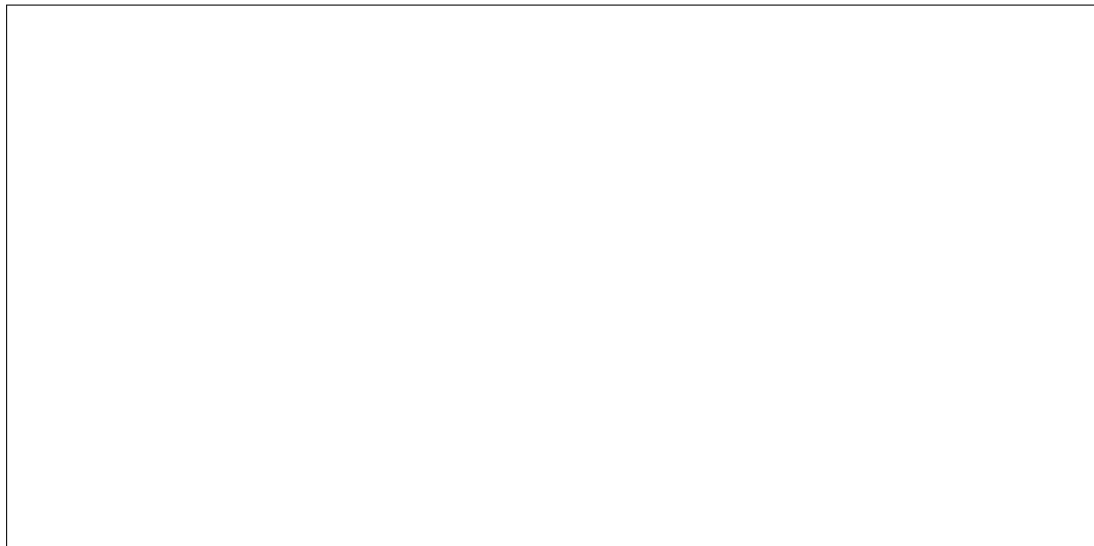
**9 Punkte (1/8)**

Ihre Firma soll einen Geldautomaten inklusive Software für ein großes Bankhaus entwickeln. Der Kunde beschreibt Ihnen die Funktionalität des Automaten wie folgt:

„Anfangs zeigt der Automat einen Wartebildschirm. Der Kunde kann sich dann einloggen, indem er seine Geldkarte in den Automaten schiebt. Außerdem muss er seinen PIN korrekt eingeben. Dann kommt ein Auswahlbildschirm, wo er verschiedene Sachen machen kann: Geld auszahlen lassen, Geld einzahlen, Kontostand anzeigen. Der Kunde kann eine dieser Aktionen wählen. Daraufhin wird die Aktion ausgeführt und der Kunde sieht erneut den Auswahlbildschirm. Die Option Auszahlen ist aber nur verfügbar, wenn der Kontostand des Kunden im Plus ist. Wird der Kontostand angezeigt kann der Kunde mit einem Klick auf OK bestätigen, dass er ihn gesehen hat, sodass der Automat zum Auswahlbildschirm zurückkehren kann. Beim Geld Auszahlen muss der Kunde auch noch den Betrag wählen. Er kann dort einen gängigen Betrag einfach auf dem Bildschirm anklicken. Beim Einzahlen kann der Kunde einfach immer wieder Geld durch den entsprechenden Schlitz einführen und wenn er fertig ist mit OK bestätigen. Sowohl beim Ein- als auch beim Auszahlen drückt der Automat außerdem eine Quittung nach Abschluss des Vorgangs. Vom Auswahlbildschirm aus kann der Kunde außerdem noch Beenden wählen. Dabei wird die Geldkarte wieder ausgegeben und der Automat kehrt zum Wartebildschirm zurück. Solange der Wartebildschirm aktiv ist, zeigt der Automat übrigens auch Werbung unserer Bank an.

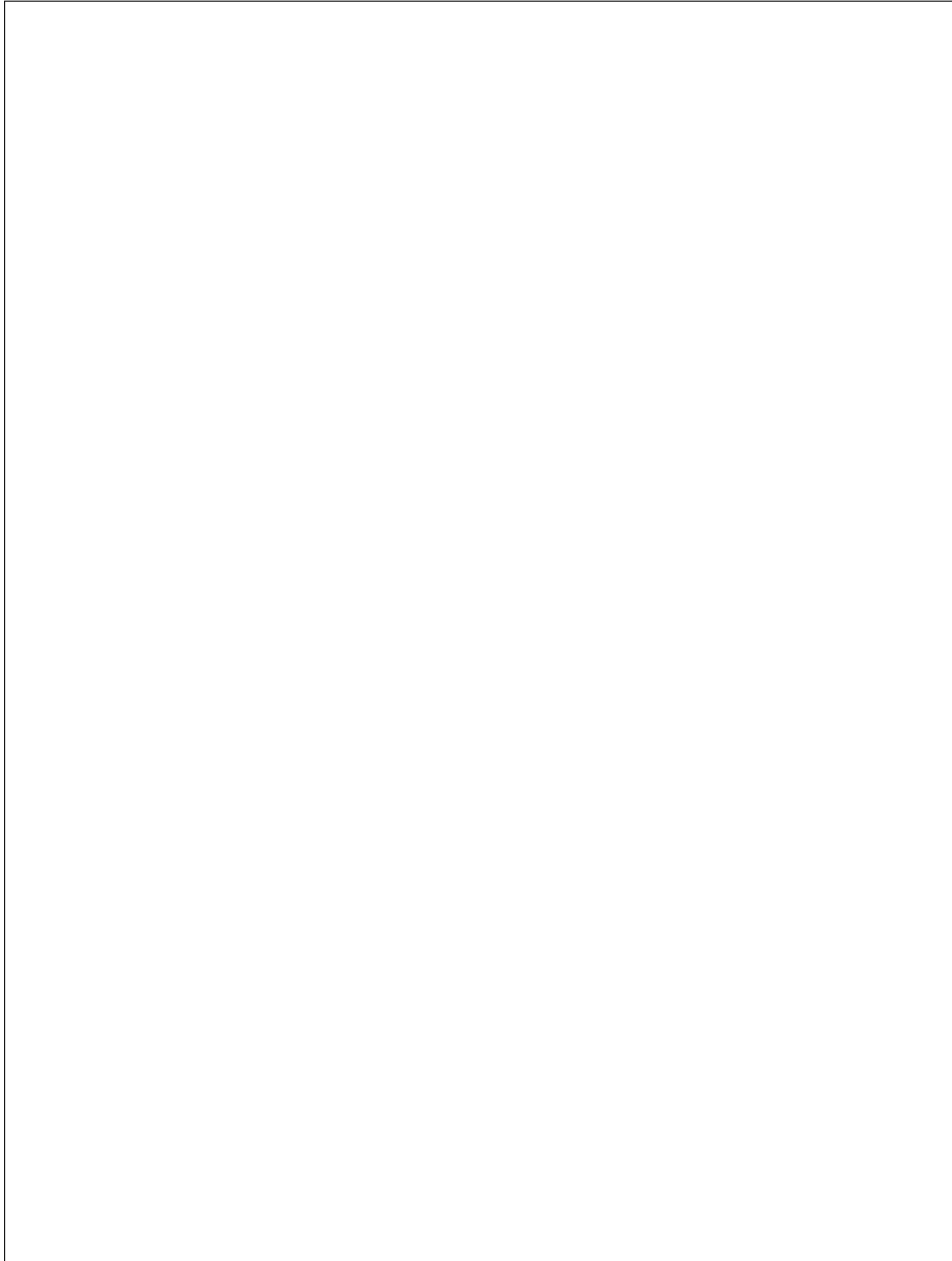
Wenn der Automat einmal gewartet werden muss, kann man außerdem von jeder Schaltfläche in einen Wartungsmodus kommen. Dazu muss man aber noch einen Geheimcode, bei uns ist das 0000, eingeben. Man hat dort die Optionen „Automat öffnen“ und „Ausschalten“. Verlässt man den Wartungsmodus ohne den Automaten auszuschalten gelangt man wieder genau zu der Schaltfläche, von der aus man den Wartungsmodus aufgerufen hat.“

- a) Beschreiben Sie den Unterschied zwischen Zuständen mit einer H-Markierung, Zuständen mit einer H\*-Markierung und Zuständen ohne eine solche Markierung in einem UML Zustandsdiagramm.



*Matrikelnummer:*

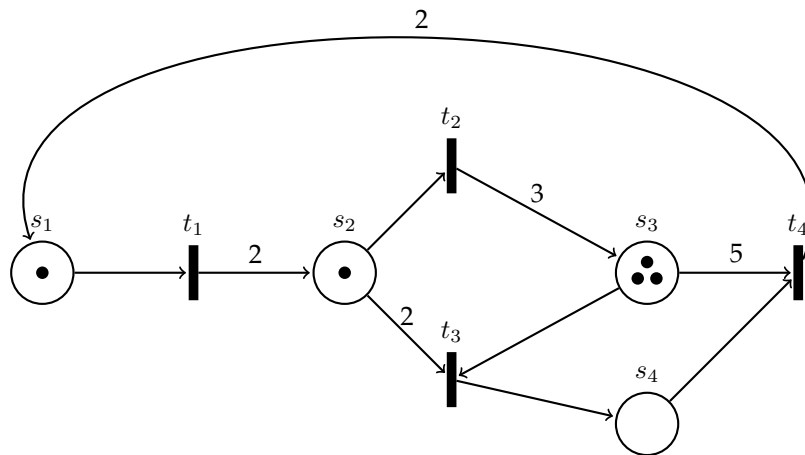
- b) Modellieren Sie den beschriebenen Geldautomaten in einem UML Zustandsdiagramm.



**Aufgabe 3: Petri-Netze**

**13 Punkte (1/7/1/1/2/1)**

Betrachten Sie folgendes Petri-Netz:



- a) Nennen Sie alle Transitionen im Petri-Netz, die gerade aktiviert sind.

*Matrikelnummer:*

- b) Zeichnen Sie den Erreichbarkeitsgraphen des Petri-Netzes. Stellen Sie die Markierungen dabei als Vektor dar (für die obige Markierung also z.B.  $(1,1,3,0)$ )

*Matrikelnummer:*

c) Ist das Petri-Netz lebendig? Begründen Sie Ihre Antwort.

d) Kann das Petri-Netz in eine Verklemmung geschaltet werden? Begründen Sie Ihre Antwort.

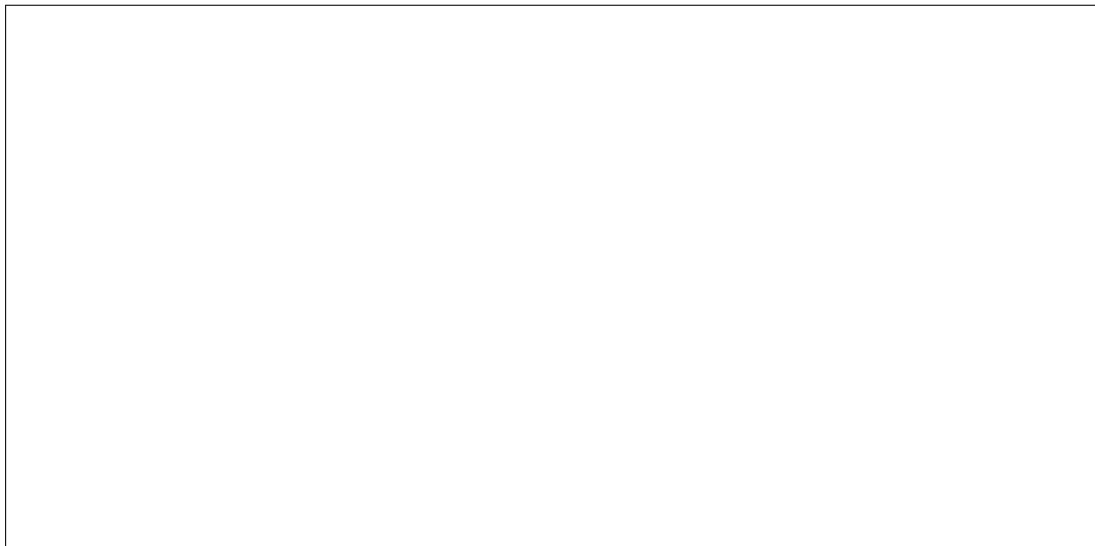


e) Welche der folgenden Aussagen sind richtig? Begründen Sie Ihre Antwort.

1. Wenn ein Petri-Netz nicht lebendig ist, kann es immer auch in eine Verklemmung geschaltet werden.
2. Ein lebendiges Petri-Netz kann nicht in eine Verklemmung geschaltet werden.
3. Ein Petri-Netz, welches nicht verklemmen kann, ist lebendig.
4. Keine der obigen Aussagen ist korrekt.

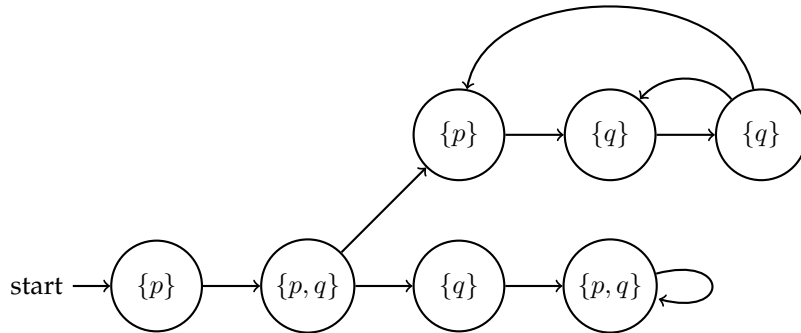


f) Geben Sie ein beliebiges Petri-Netz an, welches nicht in eine Verklemmung geschaltet werden kann und zusätzlich lebendig ist.



**Aufgabe 4: Lineare Temporallogik****10 Punkte (je 2)**

Betrachten Sie folgendes Transitionssystem:



Geben Sie für jede der folgenden LTL-Formeln an, ob Sie auf allen Läufen des obigen Transitionssystems erfüllt ist und begründen Sie Ihre Antwort kurz:

a)  $(p \wedge \mathcal{X} q) \wedge true$

b)  $\mathcal{GF} p$

*Matrikelnummer:*

c)  $((\mathcal{F} q) \mathcal{R}(\mathcal{F} p)) \wedge \neg(\mathcal{G} r)$

d)  $\neg(\neg(\mathcal{F} r) \wedge \neg(\mathcal{G} p))$

e)  $\mathcal{G}(\neg p) \vee ((\mathcal{G} r) \mathcal{U}(q \rightarrow \mathcal{F} p))$

**Aufgabe 5: Algebraische Spezifikation****8 Punkte (1/3/1/2/1)**

Gegeben sind folgende Algebraische Spezifikationen:

```

1  spec Bool =
2  sort
3      bool = true | false
4  ops
5      not: (bool) bool
6  vars
7      x: bool
8  axioms
9      false  $\neq$  true,
10     Hier sollen die Axiome aus Aufgabe a) ergänzt werden
11 end
12
13 spec MNat = Bool then
14 sort
15     mnat = zero | succ(mnat)
16 ops
17     add: (mnat, mnat) mnat
18     mult: (mnat, mnat) mnat
19     iszero: (mnat) bool
20 vars
21     m, n: mnat
22 axioms
23     succ(zero)  $\neq$  zero
24     succ(succ(zero))  $\neq$  zero
25     succ(succ(succ(zero))) = zero
26
27     add(zero, n) = n
28     add(succ(m), n) = succ(add(m, n))
29
30     mult(zero, n) = zero
31     mult(succ(m), n) = add(mult(m, n), n)
32
33     m = zero  $\rightarrow$  iszero(m) = true
34     m  $\neq$  zero  $\rightarrow$  iszero(m) = false
35 end

```

*Matrikelnummer:*

- a) In der Algebraischen Spezifikation für `Bool` fehlen die Axiome für die Funktion `not`. `not` entspricht der logischen Negation. Stellen Sie Axiome für `not` auf:

- b) Geben Sie für folgende Aussagen an, ob sie aus den Axiomen folgen oder nicht und begründen Sie Ihre Antwort.

- `succ(succ(succ(succ(zero)))) = succ(zero)`

- `mult(succ(succ(zero)), zero) = zero`

*Matrikelnummer:*

- `iszero(add(succ(succ(zero)), succ(zero)))  $\neq$  true`

c) Erklären Sie in eigenen Worten, was die Axiome in den Zeilen 23-25 bewirken.

*Matrikelnummer:*

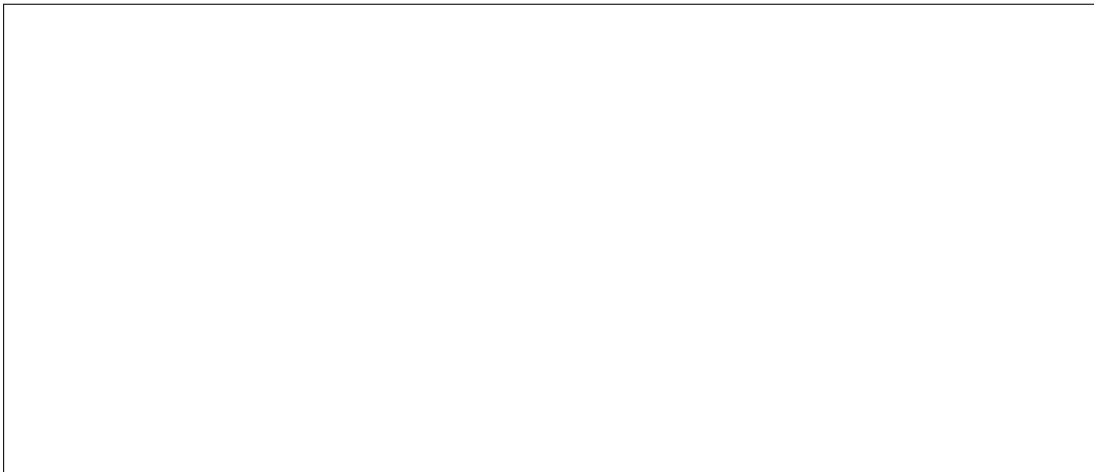
Sie möchten nun ein Modell  $\mathcal{M}$  für die Algebraische Spezifikation `MNat` finden. Dabei sei  $\mathcal{B}$  ein Modell für `Bool`

- Als Trägermenge für `mnat` wählen Sie:  $\text{mnat}^{\mathcal{M}} = \mathbb{N}_0$

d) Vervollständigen Sie das begonnene Modell, indem Sie  $\text{zero}^{\mathcal{M}}$ ,  $\text{succ}^{\mathcal{M}}$ ,  $\text{add}^{\mathcal{M}}$ ,  $\text{mult}^{\mathcal{M}}$  und  $\text{iszero}^{\mathcal{M}}$  definieren.



e) Gilt für Ihr Modell das Erzeugungsprinzip? Begründen Sie.



### Aufgabe 6: Klassen- und Objektdiagramm

14 Punkte (8/6)

Sie möchten eine Datenstruktur für sortierte Binärbäume, basierend auf dem Entwurfsmuster Kompositum entwickeln.

Dort möchten Sie Strings abspeichern. Allerdings möchten Sie die Möglichkeit haben, die Strings im Baum nach deren Länge, als auch alphabetisch zu sortieren.

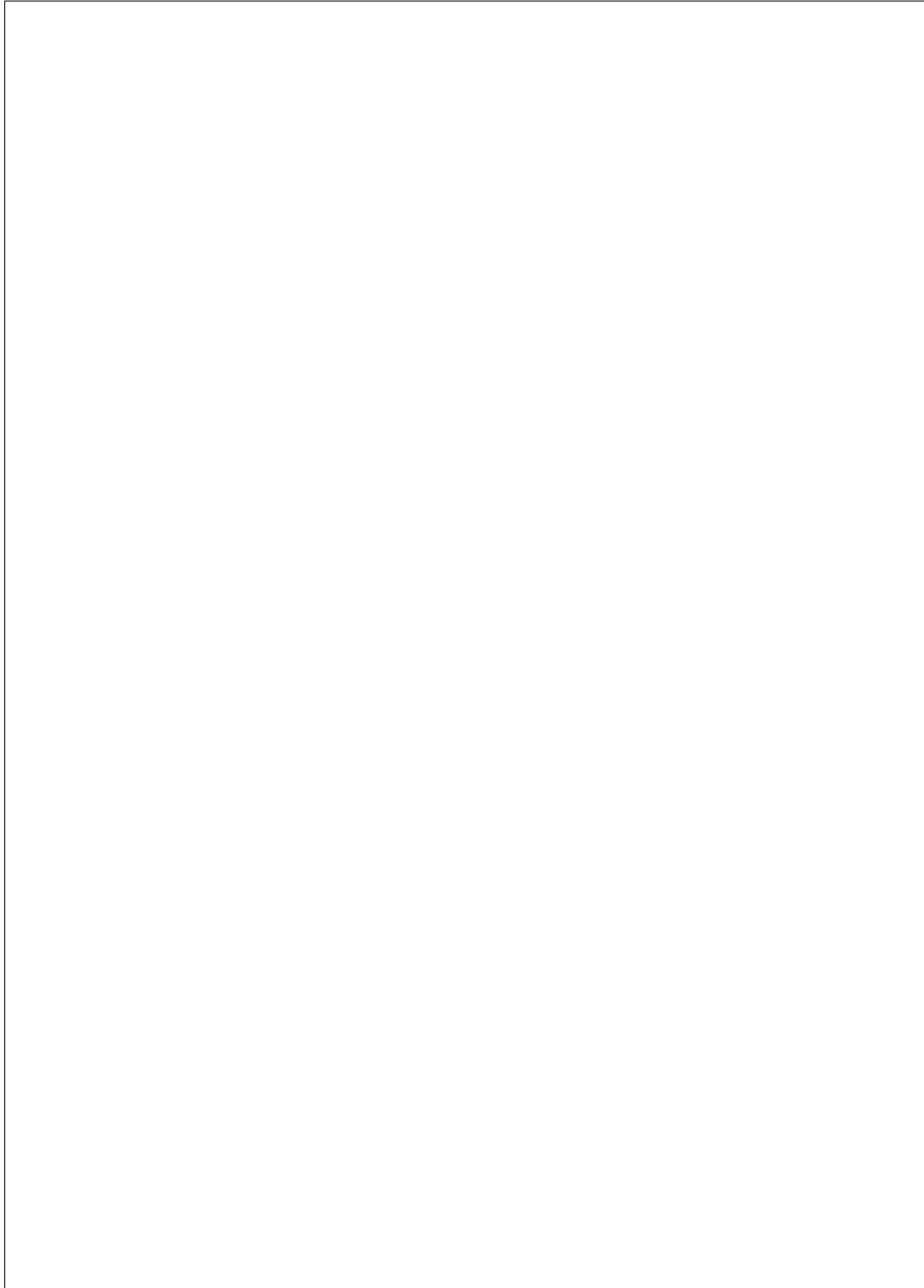
Sie haben sich für die Realisierung folgendes grobes Konzept überlegt:

- Die Knoten im Baum sollen mittels einer Klasse `TreeNode` realisiert werden. Ein `TreeNode` besitzt jeweils bis zu zwei Kinder.
  - Ein solcher Knoten soll über die Methoden `contains`, `insert` und `height` verfügen. `contains` stellt dabei fest, ob der Teilbaum unterhalb des Knotens ein bestimmtes Element enthält. `insert` fügt ein neues Element in den Teilbaum ein und `height` errechnet die Höhe des Teilbaums.
  - An Stellen, wo ein `TreeNode` kein Kind besitzt, soll er stattdessen auf ein Objekt der Klasse `Fin` referenzieren, dieses Objekt soll die gleichen Methoden wie `TreeNode` anbieten, damit in der Klasse `TreeNode` nicht unterschieden werden muss, je nachdem wie viele Kinder ein `TreeNode` besitzt.
  - Für die Sortierung des Baums wird es nötig sein beim Einfügen zwei String-Instanzen vergleichen zu können. Sie möchten das alphabetische Vergleichen in eine Klasse `AlphaComp` und das Vergleichen bezüglich Länge in die Klasse `LengthComp` kapseln. Allerdings ist Ihnen auch wichtig zukünftig noch weitere Vergleichs-Arten hinzufügen zu können ohne viel an Ihrem Programm verändern zu müssen.
  - Schließlich möchten Sie noch eine Hauptklasse `Tree`, über die Sie auf ihren Baum zugreifen können. Beim Erstellen eines solchen `Tree`-Objekts sollte man gezwungen sein sich auf die Sortiermethode festzulegen. Ein nachträglicher Wechsel soll nicht möglich sein. Die Klasse `Tree` soll außerdem über Methoden verfügen um Elemente zum Baum hinzuzufügen, die Höhe des Baums auszugeben und zu prüfen, ob ein bestimmter String bereits im Baum enthalten ist.
- a) Zeichnen Sie das oben ausgeführte Konzept als UML Klassendiagramm. Stellen Sie darin jede Klasse und jedes Interface inklusive aller Methoden und Attribute dar, welche für eine Realisierung des Konzeptes benötigt werden, außer die Klasse `String`. Die oben explizit erwähnten Klassen/Interfaces und Methoden stellen nur eine Teilmenge aller sinnvollerweise benötigten dar, ergänzen Sie weitere Klassen/Interfaces, Attribute und Methoden wo für eine Implementierung nötig.  
Geben Sie Beziehungen zwischen Klassen mithilfe von Assoziationen inklusive Multiplizitäten an. Verwenden Sie nach Möglichkeit Aggregationen und Kompositionen. Geben Sie bei den Methoden die Datentypen aller Attribute sowie den Rückgabedatentyp an.
- b) Sie möchten nun das Zusammenspiel der Klassen in einem Objektdiagramm darstellen. Erstellen Sie ein vollständiges UML Objektdiagramm für folgendes Szenario: In den Baum werden nacheinander die Strings "ab", "a", "abcd", "abc" in der angegebenen Reihenfolge eingefügt. Als Sortiervariante wurde „nach Länge“ gewählt. Auch hier sollen alle Objekte außer die vom Typ `String` dargestellt werden.



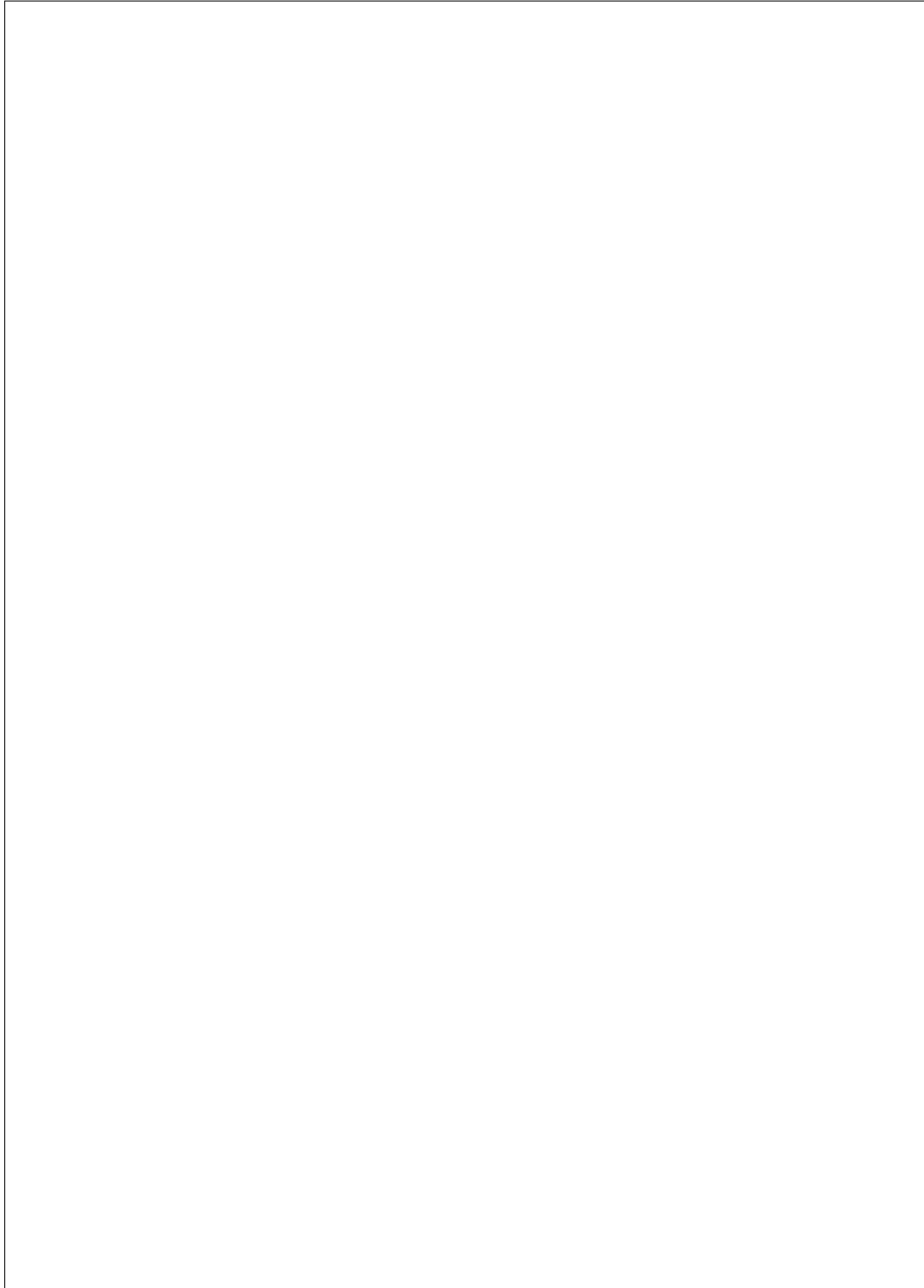
*Matrikelnummer:*

Lösung für Teilaufgabe a)

A large, empty rectangular box with a thin black border, intended for the student to write their solution for part a) of the task.

*Matrikelnummer:*

Lösung für Teilaufgabe b)

A large, empty rectangular box with a thin black border, intended for the student to write their solution for part b of the task.

*Matrikelnummer:*

Eine weitere fast leere Seite für Aufgabe 6)

**Aufgabe 7: Netzplan****11 Punkte (6/2/2/1)**

Im Folgenden soll die Entwicklung eines Online-Rollenspiels mithilfe eines Netzplans geplant werden. Für die Planung wird angenommen, dass Team und Ressourcen ausreichend groß bemessen sind.

Die Vorgangsdauer wird in Tagen angegeben und die Modellierung soll mit Tag 1 beginnen. In einer ersten Analyse wurde folgendes Vorgehen als Basis für die Entwicklung gewählt:

- Zum Projektstart (an Tag 1, *Hinweis: Eigener Knoten im Netzplan*) wird die Datenmodellierung für Charaktere und die Welt (4 Tage) vorgenommen, Ideen für das Kampfsystem gesammelt (3 Tage), sowie Entwürfe für Server (4 Tage) und Client (2 Tage) erstellt.
- Sobald die Datenmodellierung abgeschlossen ist und die Ideen für das Kampfsystem bereit sind, werden Grafiken für Charaktere und die Welt erstellt (8 Tage). Ist zusätzlich zur Datenmodellierung und dem Kampfsystem auch der Entwurf für den Server abgeschlossen, wird mit dessen Implementierung begonnen (5 Tage).
- Wurden die Grafiken erstellt und der Entwurf des Clients abgeschlossen, wird mit dessen Implementierung begonnen (3 Tage).
- Ist eine Implementierung abgeschlossen, wird für diese noch eine Testphase durchgeführt (jeweils 4 Tage).

Da nach einem inkrementellen Vorgehensmodell gearbeitet werden soll, wurde eine erste Demo des Spiels für eine große Spielemesse vereinbart. Dafür ist ein Meilenstein *Demo* nach Abschluss aller Tests geplant.

- Stellen Sie die gegebenen Aufgaben in einem Netzplan dar. Die (gerichteten) Kanten entsprechenden dabei den Abhängigkeiten.
- Rechnen Sie vorwärts: Berechnen Sie, wann die Vorgänge frühestens begonnen werden können und annotieren Sie diese Werte im Netzplan.
- Rechnen Sie rückwärts: Berechnen Sie ausgehend vom Meilenstein *Demo*, wann die Vorgänge spätestens begonnen werden können (ohne dass sich das Projektende verändert) und annotieren Sie diese Werte im Netzplan.
- Bestimmen Sie sämtliche Pufferzeiten und geben Sie einen kritischen Pfad an.

Verwenden Sie folgende Knotendarstellung:

Vorgang	
frühester Start	spätester Start
Dauer	Puffer

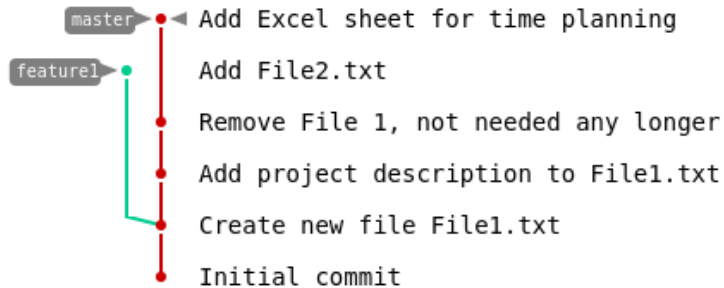
Benutzen Sie aus Platzgründen die nächste Seite im Querformat für die Bearbeitung.

*Matrikelnummer:*



**Aufgabe 8: Versionsverwaltung****7 Punkte (4/3)**

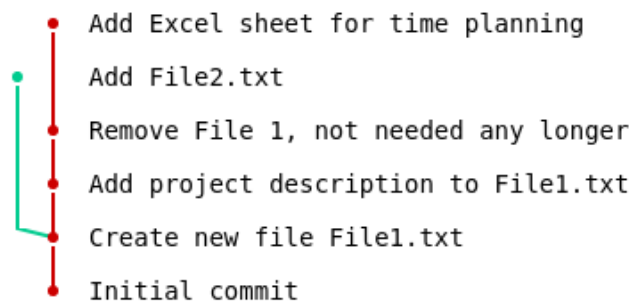
- a) Betrachten Sie für diese Aufgabe das unten abgebildete git-Repository. Aktuell ist dort der Branch `master` ausgecheckt.



Sie geben nun die unten aufgeführten Befehle ein. Sie können dabei davon ausgehen, dass die Ausführung aller Befehle erfolgreich verläuft. Zeichnen Sie den Graphen des Repositorys nach Ausführung dieser Befehle. Zeichnen Sie auch ein, auf welche Commits die Branches zeigen. Ergänzen Sie dafür den bereits gezeichneten Teil des Graphen in der Antwortbox:

```

git checkout feature1
git add newFile
git commit -m "Added new file to repo"
git checkout master
git merge feature1 -m "A special commit"
git add newFile2
git commit -m "Another new File"
  
```



*Matrikelnummer:*

- b) Erläutern Sie die Funktionsweise von Versionsverwaltungssystemen mit verteiltem und zentralem Repository. Nennen Sie jeweils ein Beispiel für ein solches Versionsverwaltungssystem.

*Matrikelnummer:*



**Korrektur**

Diese Seite wird von den Korrektoren ausgefüllt.

Aufgabe	Erreichte Punkte	Mögliche Punkte
1. Entwicklung von Softwaresystemen		8
2. Zustandsbasierte Modellierung		9
3. Petri-Netze		13
4. Lineare Temporallogik		10
5. Algebraische Spezifikation		8
6. Klassen- und Objektdiagramm		14
7. Netzplan		11
8. Versionsverwaltung		7
Gesamtpunktzahl		80

Note: \_\_\_\_\_