Name:		Vorname:		Matr.Nr.:	

Technische Universität München
Fakultät für Informatik
Prof. Dr. Dr. h.c. M. Broy

Wintersemester 2006/2007
26. Februar 2007

Klausur zu Softwaretechnik

Aufgabe 1 – Multiple Choice (8 Punkte)

Kreuzen Sie für die folgenden Multiple-Choice-Fragen genau die richtigen Antworten deutlich an. Es kann mehr als eine Antwort richtig sein. Jede Frage wird mit einem Punkt gewertet, der nur gegeben wird, wenn genau die richtigen Antworten angekreuzt wurden. Falls zu viele oder zu wenige Antworten genannt werden, wird die Frage mit 0 Punkten bewertet.

Welche Prozessmodelle sind für Projekte mit häufigen Änderungen geeignet?
- ☒ **Extreme Programming (XP)**
- ☐ V-Modell 97
- ☒ **V-Modell XT**

Welche der folgenden sind gewünschte Eigenschaften von Anforderungen?
- ☒ **Konsistent**
- ☒ **Überprüfbar**
- ☐ Implementierungsnah

Die Architektur eines Softwaresystems sollte Rücksicht nehmen auf
- ☒ **die Arbeitsteilung im Entwicklungsunternehmen**.
- ☒ **die Änderungshäufigkeit einzelner Teile**.
- ☒ **Performanzanforderungen**.

Wofür verwenden wir Abstraktionen im Software Engineering?
- ☒ **Um komplexe Zusammenhänge erfassbar zu machen**.
- ☐ Zur Verbesserung der Zuverlässigkeit.
- ☒ **Um automatische, modellbasierte Analysen zu ermöglichen**.

Das Prinzip des „Information Hiding“ bedeutet im Software Engineering,
- ☐ dass Informationen sicher vor dem Zugriff Unbefugter sind.
- ☒ **dass gewisse Entwurfsentscheidungen hinter einer Schnittstelle gekapselt werden**.
- ☐ dass in Spezifikationen keine Implementierungsdetails enthalten sind.

Was sind die Hauptfaktoren, die die Definition von sinnvollen Software-Metriken (z.B. für die Größe einer Software) erschweren?
- ☐ Software ist häufigen Änderungen unterworfen.
- ☐ Softwaresysteme sind häufig sehr umfangreich.
- ☒ **Software ist immateriell.**

Was sind Nachteile des Wasserfallmodells?
- ☐ Keine Verfolgbarkeit von Anforderungen über die Phasen hinweg.
- ☒ **Unflexibel gegenüber Änderungen von Projektbedingungen.**
- ☐ Keine Werkzeugunterstützung.

In welcher Phase lohnt es sich in Bezug auf Kosten am meisten, Fehler zu vermeiden und zu entfernen?

☒ **Anforderungsanalyse**

☐ Implementierung

☐ Integration

Aufgabe 2 – Nutzungsfälle und Szenarien (5 Punkte)

Sie bekommen die Aufgabe, die Anforderungen für ein neu zu entwickelndes System zu erfassen und zu analysieren. Das System ist ein webbasiertes Publikationssystem für Zeitschriften, d.h. Artikel und Werbeanzeigen werden online von den Autoren eingepflegt. Der Redakteur generiert und prüft dann zentral die jeweilige Zeitschrift.

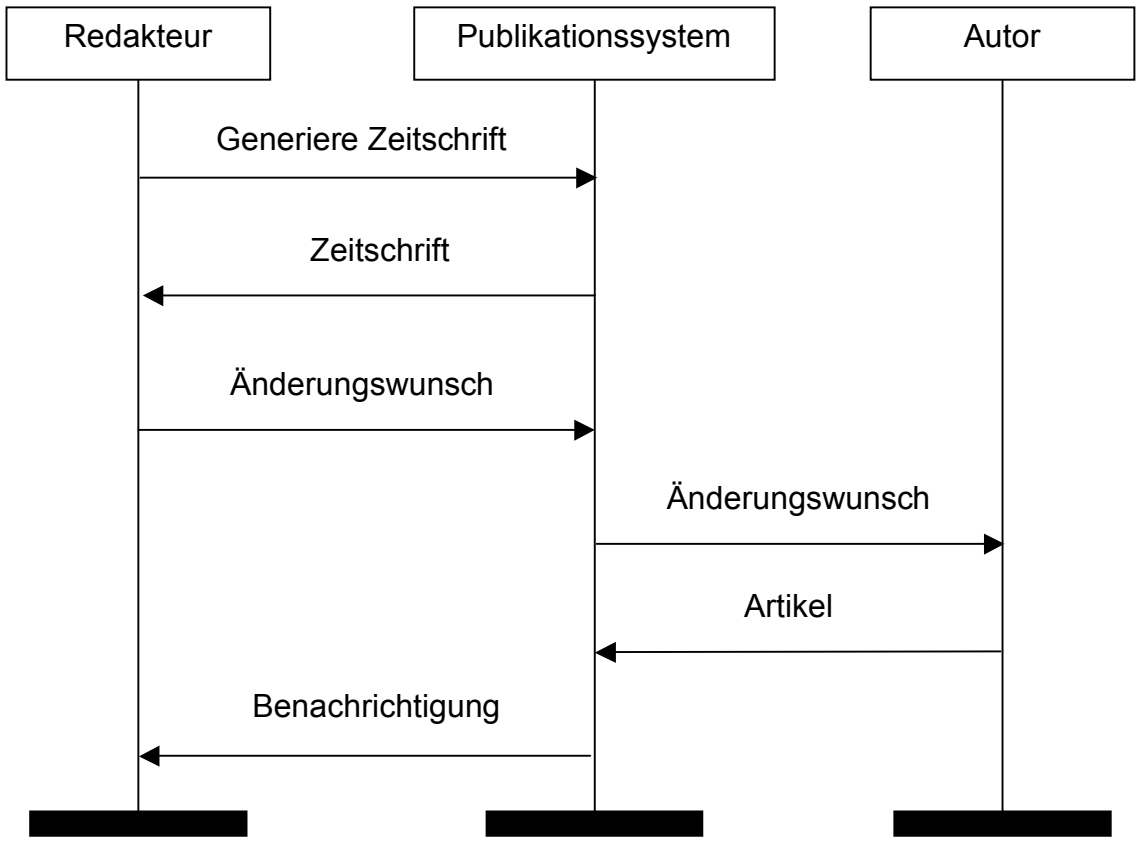
a) Ein Nutzungsfall stellt das Erstellen eines Artikels für eine Zeitschrift dar. Erstellen Sie eine tabellarische Beschreibung dieses Nutzungsfalls.

1 Punkt für sinnvollen Ereignisfluss, 1 Punkt für sinnvolle Anfangs- und Abschlussbedingungen, 1 Punkt für sinnvolle Qualitätsanforderungen.

Anwendungsfallname	ErstelleArtikel
Akteure	Initiiert von Autor Kommuniziert mit Redakteur
Ereignisfluss	1. Der Autor öffnet die Oberfläche des Systems 2. Er meldet sich am System an 3. Er betätigt den Link „Artikel erstellen“ 4. Er trägt Titel und Text ein 5. Er lädt möglicherweise Bilder hoch 6. Er betätigt den Link „Im System speichern“
Anfangsbedingungen	- Der Autor hat eine Kennung am System.
Abschlussbedingungen	- Der Artikel ist im System verfügbar - Der Autor hat eine Erklärung dafür erhalten, warum die Transaktion nicht bearbeitet werden kann.
Qualitätsanforderungen	- Der Artikel darf nicht von Unbefugten gelesen werden können.

b) Beschreiben Sie das Szenario „Zeitschrift generieren mit Änderungswunsch“ als Message Sequence Chart bzw. UML Interaktionsdiagramm. Der Redakteur generiert die Zeitschrift und prüft das angezeigte Resultat. Er ist mit dem Ergebnis unzufrieden und legt einen Änderungswunsch im System an. Dieser wird an den zuständigen Autor weitergeleitet. Nach erfolgter Änderung durch den Autor wird der Redakteur wiederum benachrichtigt.

1 Punkt für die korrekten Lebenslinien, 1 Punkt für die korrekten Nachrichten



Aufgabe 3 – Software-Test (8 Punkte)

Zwei wichtige Arten von Verfahren zur Bestimmung von Testfällen sind strukturelle (Glass-Box) und funktionale (Black-Box) Tests.

a) Im Folgenden ist die Prozedur „prod_incoming“ aus einem betrieblichen Informationssystem spezifiziert. Sie beschreibt das Eintreffen bestellter Ware. Dabei gibt „ordered“ die bestellte Anzahl, „onstock“ die gelagerte Anzahl und „in“ die gelieferte Anzahl eines Produktes an.

```
proc prod_incoming(var Product prod, Int in)
pre  ordered(prod) ≥ in
post ordered(prod') = ordered(prod) − in ∧
     onstock(prod') = onstock(prod) + in
```

Bilden Sie mindestens 6 Äquivalenzklassen für „in“, „ordered” und „onstock” und geben Sie jeweils einen funktionalen Testfall mit erwarteter Ausgabe an.

3 Punkte für richtige Äquivalenzklassen. 0,5 Punkte pro richtigem Testfall.

Klassenbedingung	***in***	***ordered***	***onstock***	***Erwartet: ordered,onstock***
in < ordered	17	23	10	6, 27
in = 0	0	23	10	23, 10
in < 0	-5	23	10	Exception
in > ordered	17	10	10	Exception
ordered = 0	17	0	10	Exception
ordered < 0	17	-10	10	Exception

b) Die Prozedur wurde in Java als Methode der Klasse „Product“ implementiert.

```
public void prodIncoming(int in) throws TooMuchIncomingException
{
if (state.getOrdered() < in) {
throw new TooMuchIncomingException();
}
state.setOrdered(state.getOrdered() – in);
state.setOnstock(state.getOnstock() + in);
return;
}
```

Schreiben Sie entsprechende strukturelle Testfälle, so dass Pfadüberdeckung erreicht wird.

1 Punkt für Pfadüberdeckung, 0,5 pro richtigem Testfall.

in	***ordered***	***onstock***	***Erwartet: ordered,onstock***
17	23	10	6, 27
25	23	10	TooMuchIncomingException

Aufgabe 4 – Architekturbeschreibung (11 Punkte)

Erstellen Sie eine Architekturbeschreibung für ein einfaches Studentenverwaltungssystem, das die Prüfungsbeteiligung (mit Status und Note) von Studenten (mit Name, Vorname, Matrikelnummer) an angebotenen Prüfungen (mit Bezeichnung, Ort, Zeit) verwalten soll. Eine Komponente „Prüfungsbeteiligung“ soll folgende typische Aufgaben unterstützen:

(I) Anmeldung und Abmeldung eines Studenten zu einer bestimmten Prüfung,

(II) Abfrage des Status einer Prüfungsbeteiligung (angemeldet, abgemeldet, teilgenommen, bestanden, durchgefallen) zu einer bestimmten Prüfung durch einen Studenten,

(III) Abfrage der Note eines Studenten zu einer Prüfung,

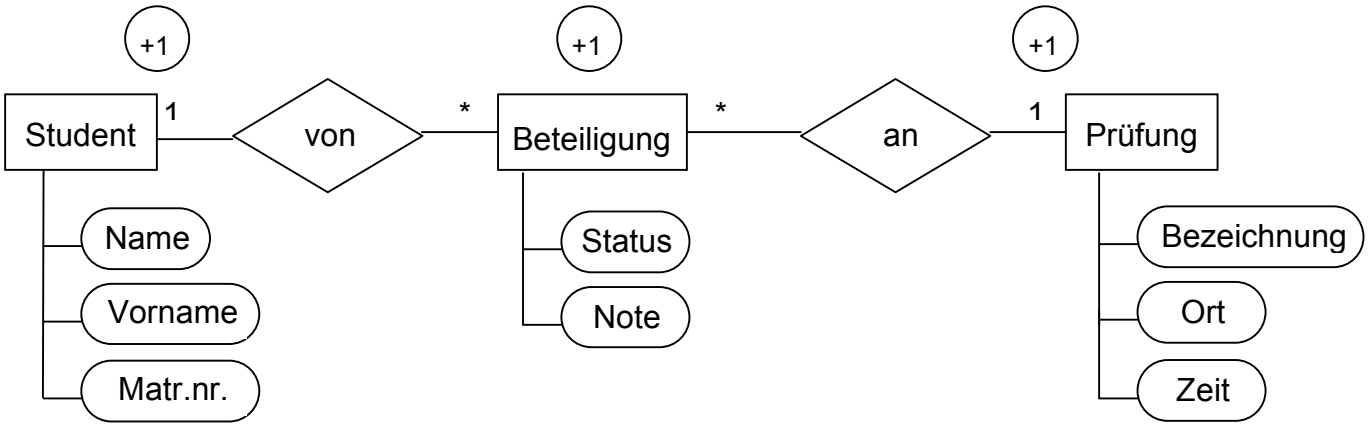
(IV) Eintragung der Teilnahme eines Studenten an einer Prüfung durch die Übungsleitung,

(V) Eintragung der Note eines Studenten an einer Prüfung durch die Übungsleitung.

Nicht zu berücksichtigen sind dabei Authentifizierung und Autorisierung.

Beschreiben Sie folgende Bestandteile einer Architektur des Systems:

1. Datenmodell (als E/R- oder Klassendiagramm), *3 Punkte*



2. Spezifikation der Methoden der Komponente „Prüfungsbeteiligung“ nach „design by contract“, *6 Punkte*

Prüfungsbeteiligung

```
sort Beteiligung = (status: Status, note: Note);
sort Püfungsbeteiligung = (Student × Prüfung → Beteiligung);

map: Püfungsbeteiligung;

method anmelden (r s: Student; r p: Prüfung);
   pre:  s, p existieren;                                              +1
   post: map(s,p).status = angemeldet;

method abmelden (r s: Student; r p: Prüfung);
   pre:  (s, p existieren) ∧ map(s,p).status = angemeldet);            +1
   post: map(s,p).status = abgemeldet;

method meinStatus (r s: Student; r p: Prüfung): Status;
   pre:  s, p existieren;                                              +1
   post: meinStatus(s,p) = map(s,p).status;

method meineNote (r s: Student; r p: Prüfung): Note;
   pre:  (s, p existieren) ∧                                           +1
         map(s,p).status ∈ {bestanden, durchgefallen};
   post: meineNote(s,p) = map(s,p).note;

method teilnahme (r s: Student; r p: Prüfung);
   pre:  (s, p existieren) ∧ map(s,p).status = angemeldet);            +1
   post: map(s,p).status = teilgenommen;

method benote (r s: Student; r p: Prüfung; r n: Note);
   pre:  (s, p, n existieren) ∧ map(s,p).status = teilgenommen);
   post: [(1,0 ≤ n ≤ 4,0) ⇒ map(s,p).status = bestanden] ∧             +1
         [(4,0 < n ≤ 5,0) ⇒ map(s,p).status = durchgefallen] ∧
         map(s,p).note = n;
```

3. Integritätsbedingung zwischen Status und Note von Prüfungsbeteiligungen.
   2 Punkte

```
∀ (s: Student; p: Prüfung):
   (map(s,p).status = bestanden)      ⇔ (1,0 ≤ map(s,p).note ≤ 4,0) ∧
   (map(s,p).status = durchgefallen)  ⇔ (4,0 < map(s,p).note ≤ 5,0)
```

Aufgabe 5 Zustandmodellierung (8 Punkte)

Das Zustandsverhalten einer Prüfungsbeteiligung von Student Max an der Prüfung SWE gemäß Aufgabe 4 soll nun genauer modelliert werden durch ein formales Zustandsübergangsdiagramm mit Ein- und Ausgabe sowie Vor- und Nachbedingungen. Zu modellieren sind dabei:

- die An- und Abmeldung mit Bestätigung,
- Eingabe der Teilnahme mit Bestätigung,
- Eingabe einer Note nach Teilnahme mit Bestätigung,
- Notenabfrage nach Noteneingabe,
- Statusabfrage in jedem beliebigen Zustand.

Nicht zu berücksichtigen sind dabei Authentifizierung und Autorisierung.

Die Parameter (Student, Prüfung) sind in der Aufgabenstellung fixiert zu (Max, SWE) und werden daher weggelassen.

a) Geben Sie die möglichen Ein- und Ausgaben an.

Eingaben: anmelden, abmelden, meinStatus, meineNote, teilnahme, benote(note) +1

Ausgaben: ok, status, note +1

wobei status ∈ Status, note ∈ Note

b) Geben Sie alle Zustandsattribute an, die für die Modellierung notwendig sind und beschreiben Sie deren Verwendungszweck.

note: Note speichert die Note der Beteiligung von Max an der Prüfung SWE +1

c) Geben Sie die Zustände der Prüfungsbeteiligung an.

Status = {angemeldet, abgemeldet, teilgenommen, bestanden, durchgefallen} +1

d) Zeichnen Sie das Zustandsübergangsdiagramm. Verwenden Sie hierzu die Syntax mit Ein- und Ausgabe, Vor- und Nachbedingungen.