Name	Vorname	Matrikelnummer

Musterklausur mit Lösungshinweisen

Klausur Software Engineering
Bachelor

14. Februar 2006

Bitte freilassen!

Teil 1	Teil 2	Summe

Teil 1: Wissensfragen (insgesamt 38 Punkte, ca. 38 Minuten Bearbeitungszeit)
Kreuzen Sie Zutreffendes an. Zu jeder Frage können mehrere Antworten richtig sein.

Bitte beachten Sie, dass Ihnen für jedes falsch gesetzte Kreuz gleich viele Punkte abgezogen werden, wie Sie für eine korrekte Ankreuzung erhalten. Negative Punktzahlen ergeben null Punkte für die betreffende Frage.

Frage 1.1: Spezifikation von Anforderungen (4P)

- ☒ Der Aufwand für das Requirements Engineering soll umgekehrt proportional zum Risiko sein, das man bereit ist, einzugehen.
- ☐ Eine Spezifikation soll die Wünsche und Vorstellungen des Entwicklers adäquat, vollständig und widerspruchsfrei fassen.
- ☒ Der Spezifikationsprozess umfasst die Gewinnung, Darstellung und Prüfung von Anforderungen.
- ☒ Formale Spezifikationen sind auf Eindeutigkeit hin prüfbar.
- ☐ Informale Spezifikationen sind wertlos, da sie keine Prüfung der Vollständigkeit und Konsistenz erlauben.
- ☐ In der Anforderungsspezifikation werden ausschliesslich die funktionalen Anforderungen an ein System erfasst.
- ☐ In der konstruktiven Darstellung von Anforderungen wird das zu spezifizierende System als Black-Box betrachtet.
- ☒ Requirements Engineering ist am wirtschaftlichsten, wenn die Gesamtkosten, bestehend aus den Kosten für den Aufwand der Anforderungsspezifikation und den Fehlerkosten, minimal sind.

Frage 1.2: Prozessmodelle (4P)

- ☒ Das Wasserfall-Modell ist ein systematisches Prozessmodell für die Software-Entwicklung.
- ☐ Das Wasserfall-Modell ist ein typischer Vertreter der iterativen Software-Prozessmodelle
- ☐ Das Wasserfall-Modell ist ein ergebnisorientiertes Phasenmodell
- ☒ Wachstumsmodelle sind ein geeignetes Instrument zur Projektführung.
- ☒ Wachstumsmodelle bieten sich an, wenn ein Basisprodukt schnell verfügbar sein muss.
- ☐ In einem Wachstumsmodell wird die gewünschte Applikation aus Komponenten synthetisiert.
- ☐ Wachstumsmodelle bieten den Vorteil, dass die Planung der einzelnen Lieferungen und Phasen nur eine untergeordnete Rolle spielen.
- ☒ Phasenmodelle haben den Vorteil, dass sie dem natürlichen Software-Lebenslauf folgen.
- ☐ Ergebnisorientierte Phasenmodelle bringen den Nachteil mit sich, dass sie die Projektführung deutlich erschweren, da jede Phase explizit geplant werden muss.

Frage 1.3: Prototyping (3P)

- ☒ Prototypen sind ein zentrales Mittel zur frühzeitigen Erkennung und Lösung von Problemen in der Software-Entwicklung.
- ☐ Das Arbeiten mit Pilotsystemen ist eine Form des Phasenmodells.
- ☐ Exploratives Prototyping wird auch als Labormuster bezeichnet.
- ☐ Ein Prototyp im engeren Sinne wird parallel zur Modellierung des Anwendungsbereichs erstellt, lediglich um spezifische Aspekte der Benutzungsschnittstelle zu veranschaulichen.
- ☒ Es werden drei Arten von Prototyping unterschieden: exploratives, experimentelles und evolutionäres Prototyping
- ☒ Ein Pilotsystem ist ein Prototyp, der nicht zur experimentellen Erprobung oder zur Veranschaulichung dient, sondern selbst der Kern des Produkts ist.
- ☐ In der Regel werden alle drei Arten von Prototypen schnell aufgebaut, unter Vernachlässigung softwaretechnischer Standards.

Frage 1.4: Systematisches Entwerfen I (4 Punkte)

- ☒ Beim vertragsorientierten Entwurf ist die Nachbedingung bzw. die Invariante nicht in jedem Fall erfüllt.
- ☒ In einem guten Software-Design werden unsinnige Änderungen vorgenommen: (i) Modul A wird in die Module A1 und A2 zerlegt. (ii) Die Module B1 und B2 werden zu einem Modul B zusammengelegt. Im neuen Software-Design ist somit die Kohäsion kleiner und die Kopplung grösser.
- ☐ Sei S' eine Subschnittstelle von S. eine Schnittstellenvererbung nach dem Prinzip „Steinbruch" bedeutet, dass die von S' offerierten Leistungen ein Spezialfall der von S offerierten Leistungen sind.
- ☒ Nur echte Subschnittstellen garantieren die universelle Verwendbarkeit der Implementierung.
- ☐ Das Geheimnisprinzip besagt, dass die Entwurfspläne von zu entwickelnder Software nicht durch die Organisation, welche die Software entwickelt, offengelegt werden dürfen.
- ☐ Das EVA-Architekturmuster (Eingabe-Verarbeitung-Ausgabe) zählt zu den Strukturmustern.
- ☐ Die Abstrakte Fabrik ist ein Architekturmuster.
- ☒ Architekturstile definieren eine Familie von Software-Systemen bezüglich ihrer strukturellen Organisation.

Frage 1.5: Systematisches Programmieren (2 Punkte)

☐ GOTO-freie Programme sind gut strukturierte Programme.

☒ Bei Programmiersprachen mit geschlossenen Ablaufkonstrukten hat jedes Konstrukt genau einen Eintritts- und einen Austrittspunkt.

☐ Annehmende Schleifen sind sicherer als abweisende.

☒ Aus der Invariante einer Schleife S kann man die Initialisierung, die Abbruchbedingung und die Fortschaltung von S ableiten.

☐ Bei Unterprogrammen erfolgt die Parameterübergabe typischerweise mit „call by value“.

Frage 1.6: Software-Metriken und Skalen I (4 Punkte)

☒ Das Mass nach McCabe kann aus Programmflussgraphen berechnet werden.

☐ Bei der zyklomatischen Komplexität handelt es sich um ein additives Mass.

☒ Eine Intervallskala kann mit der Transformation $f(x) = ax + b$ in eine andere Intervallskala transformiert werden, wobei a und b konstante Grössen sind und x der zu transformierende Wert.

☐ Das Software-Science-Mass nach Halstead ist kein Komplexitätsmass.

☒ Die Anzahl aufgetretene Fehler (errors) bei der Verwendung eines Softwaremoduls liegen auf einer Absolutskala.

☐ Die Repräsentationsbedingung (auch Homomorphiebedingung genannt) sagt aus, dass zwei Skalen durch die Multiplikation mit einem konstanten Faktor ineinander überführt werden können.

☒ Auf der Absolutskala sind keine Transformationen möglich.

☐ Intervallskalen haben einen absoluten Nullpunkt.

Frage 1.7: Software-Metriken und Skalen II (2 Punkte)

☒ Zählt man die Anzahl Kinder einer Familie, so handelt es sich um eine Absolutskala

☐ Die Namen von Familienmitglieder werden durch eine Ordinalskala ausgedrückt

☐ Die Farben der Judogürtel sind eine Nominalskala

☒ Die Zeiten, die bei 100m-Sprints gemessen werden, sind Werte einer Verhältnisskala.

☐ Die Anzahl gemessene Fehler in einem Programm sind Werte auf einer Intervallskala.

Frage 1.8: Qualitätsmanagement I (2P)

- ☒ Totales Qualitätsmanagement ist eine Führungsmethode, die Kundenzufriedenheit als oberstes Unternehmensziel postuliert.
- ☒ Für die Qualitätslenkung können Audits eingesetzt werden.
- ☐ Für Software ist ein Qualitätsmanagement auch ohne Spezifikation der Anforderungen möglich.
- ☐ Qualität ist ein absolutes Mass für die Güte einer Einheit.
- ☐ Es gibt systematische, ingenieurmässige Vorgehensweisen, welche die Erreichung gegebener Qualitätsanforderungen garantieren.
- ☐ Die Qualitätspolitik ist geheim und darf deshalb nur der Geschäftsleitung bekannt sein.

Frage 1.9: Software-Lebenslauf und Evolution I (2P)

- ☐ Nur S-Typ Software nach der Lehman'schen Klassifikation weist einen vollständigen Software-Lebenslauf auf.
- ☐ Typischerweise umfasst der Lebenslauf einer Software-Komponente die zwei Stadien Initiierung und Entwicklung.
- ☒ In der Entwicklung von Software laufen für die Erstellung einer Komponente die folgenden Tätigkeiten ab: Analysieren und Spezifizieren, Architekturentwurf, Detailentwurf, Codierung und (Einzel-)Test, Integration & Test und Installation & Systemtest plus Dokumentieren als übergreifende Aktivität.
- ☐ Jede Art von Software ist instabil und entwickelt sich evolutionär über die Zeit.
- ☒ Das Gesetz über die fortwährende Veränderung (Continuing Change) eines Software-Systems (nach Lehman) sagt aus, dass ein System, welches im Betrieb ist, laufend angepasst werden muss. Wird dies nicht gemacht, so verliert das System seinen Nutzen.

Frage 1.10: Software-Lebenslauf und Evolution II (2P)

- ☐ Reverse-Engineering bedeutet, die beim Modultest gefundenen Fehler rückgängig zu machen.
- ☒ Forward Engineering bezeichnet den traditionellen Prozess, der von abstrakten, maschinenunabhängigen Architekturen und Entwürfen zu einer physischen Systemimplementierung führt.
- ☐ Reengineering bedeutet Analyse des Codes eines bestehenden Systems mit dem Ziel, Komponenten und Beziehungen zu identifizieren und eine abstrakte Repräsentation des Systems zu gewinnen.
- ☒ Die Resultate eines Reverse-Engineering Prozesses sind aus modelltheoretischer Sicht eine deskriptive Form der Modellierung.

Frage 1.11: Statische Analyse (2 Punkte)

- ☒ Ziele der statischen Analyse sind u.a. die Bewertung von Qualitätsmerkmalen und die Erkennung von Fehlern.
- ☐ Statische Programmanalysen sind nicht automatisierbar und ähneln daher stark dem Review.
- ☐ Rekursive Algorithmen können nicht statisch analysiert werden.
- ☒ Die Untersuchung der Verschachtelungstiefe ist Teil der statischen Analyse von Programmen.

Frage 1.12: Qualitätsmanagement II (3P)

- ☒ Qualitätsfachleute benötigen einen unabhängigen Berichtspfad für Qualitätsbelange bis hinauf in die Geschäftsleitung.
- ☐ Modernes Qualitätsmanagement bedeutet, dass alle Qualitätsbelange an speziell ausgebildete Qualitätsfachleute delegiert werden.
- ☒ Testprotokolle und Reviewberichte sind Qualitätsaufzeichnungen und müssen aufbewahrt werden.
- ☐ Anforderungsspezifikationen stellen keine qualitätsbezogene Dokumentation dar.
- ☐ Der Band 2 des Qualitätshandbuches einer Unternehmung wird üblicherweise an Kunden abgegeben, da er u.a. einen Überblick über die getroffenen Qualitätsmassnahmen gibt.
- ☒ In der Qualitätslenkung werden konstruktive Massnahmen so weit als möglich eingesetzt, um das generelle Qualitätsniveau zu heben.

Frage 1.13: Reviews (4P)

- ☐ Das Walkthrough ist die effektivste Art eines Reviews.
- ☐ Nur bei der Teaminspektion ist es nötig, weitere Unterlagen zum Prüfling, wie z.B. Prüflisten, Referenzunterlagen, etc. an die einzelnen Teilnehmer des Reviews abzugeben.
- ☒ Beim Walkthrough besteht die Gefahr, dass der Autor durch geschicktes Vortragen das Resultat des Reviews stark beeinflussen kann.
- ☒ Das Ziel eines Reviews ist es, Stärken und Schwächen und daher die Qualität des Prüflings aufzuzeigen.
- ☒ Checklisten können beim Finden von Befunden eine wertvolle Hilfestellung für die Gutachter sein. Es muss jedoch darauf geachtet werden, dass die Checklisten auf den Prüfling abgestimmt sind.
- ☐ Bei der Team-Inspektion, dem Walkthrough und der N-Individuen-Inspektion findet eine Sitzung statt, in welcher alle Befunde durch die Gutachter vorgetragen werden.
- ☒ Für das Durchführen einer effektiven Team-Inspektion müssen mehr als ein Gutachter in der Sitzung anwesend und vorbereitet sein.
- ☐ Reviews gehören zu den dynamischen Prüfverfahren und sind deshalb geeignet jede Art von Dokumenten zu prüfen.

Teil 2: Anwendungsaufgaben (insgesamt 92 Punkte, ca. 92 Minuten Bearbeitungszeit)

A. Anforderungstechnik, Planung

Aufgabe 2.1: Anforderungsanalyse (10 P)

Bei der Analyse, die am Anfang einer Systementwicklung für eine kleine, aber stetig wachsende Fastfoodkette durchgeführt wird, ist ein Interview aufgezeichnet worden, aus dem der folgende Ausschnitt stammt:

Zurzeit verwalten alle Filialen unserer Kette ihr Lager selber. Jedes Mal wenn der Vorrat einer Zutat kurz vor dem Ausgehen ist, bestellen die zuständigen Mitarbeiter der Filiale per Telefon beim zentralen Lager nach. Wenn das Lager zu wenig kontrolliert wird, erleben wir häufig eine Überraschung, weil z.B. plötzlich die Hamburger oder die Brötchen ausgegangen sind.
In Zukunft wollen wir ein automatisches Lagerverwaltungssystem, welches automatisch beim Unterschreiten eines Mindestbestandes eine Bestellung auslöst.
Pro Woche passiert es zurzeit höchstens etwa drei Mal, dass die Filialen beim zentralen Lager nachbestellen müssen. Unser CEO schätzt, dass sich binnen Jahresfrist auf Grund unseres Wachstums der Lagerumschlag verdreifachen wird.

a. Extrahieren Sie stichwortartig den Ist-Zustand anhand des oben aufgeführten Interview-Ausschnitts. (**1,5P**)

b. Analysieren Sie den Text und extrahieren Sie die Anforderungen an das zu schaffende System aus dem oben stehenden Text. Gliedern Sie in funktionale und nicht-funktionale Anforderungen. (**4P**)

c. Legen Sie ein Glossar zu den relevanten Begriffen im oben stehenden Text an. Beschreiben Sie diese Begriffe stichwortartig. (**3P**)

d. Welche weiteren Techniken (neben dem Interview) würden Sie im genannten Fall einsetzen, um die Anforderungen detaillierter zu machen (sprich zu erheben)? Nennen Sie mindestens 3 Techniken. (**1,5 P**)

Lösung:

a.

Ist-Zustand:

- Filialen verwalten Lager selber.
- Manuelle Erfassung von Lagerbestand.
- Manuelle Nachbestellung von fehlenden Zutaten.

b.

Anforderungen:

Funktional	Nicht Funktional
• Automatische Lagerverwaltung • Automatische Nachbestellung bei Unterschreitung Mindestbestand	• Leistungsanforderung: 3 Nachbestellungen für das Lager pro Woche → in einem Jahr werden es 9 Nachbestellungen pro Woche sein. • Skalierbarkeit → Kann auf Grund des Wachstums abgeleitet werden.

c.

Begriff	**Beschreibung**
Bestelllung	Siehe Nachbestellung
Filiale	Ladenlokal der Fastfoodkette, welches Kunden mit Produkten (Fastfood) versorgt.
Filiallager	Das lokale Lager einer Filiale. Es wird durch das zentrale Lager der Kette beliefert.
Kette	Die Fastfoodunternehmung als Ganzes, welche Dienstleistungen für alle Filialen bereitstellt, z.B. das zentrale Lager.
Lager	Siehe Filiallager
Lagerverwaltungssystem	Das Lagerverwaltungssystem der Filialen. Es führt den Bestand des Lagers und löst automatisiert Nachbestellungen aus.
Mindestbestand	Ein Schwellwert für das Lager, der vom Lagerverwaltungssystem periodisch kontrolliert wird. Bei Unterschreitung des Schwellwertes wird durch das Lagerverwaltungssystem automatisch eine Nachbestellung an das zentrale Lager geschickt.
Nachbestellung	Bei Unterschreitung des Mindestbestandes wird eine Bestllung an das zentrale Lager der Kette ausgelöst, welche das Lager wieder auf einen bestimmten Bestand füllt.
Zutat	Zutaten werden gebraucht, um Produkte in den Filialen herzustellen. Zutaten werden im Filiallager gelagert und vom zentralen Lager geliefert.
Zentrales Lager	Grosslager, welches von den Zulieferern verschiedene Zutaten zwischenlagert. Es werden Bestellungen von den Filialen an das zentrale Lager geschickt. Welche dann die entsprechenden Zutaten anliefert.

Hinweis: Die fett geschriebenen Begriffe im Glossar sind ein Muss für die korrekte Lösung.

d.

- Gemeinsame Arbeitstagungen
- Beobachtungen
- Befragungen
- ....

Aufgabe 2.2: Make or Buy (2 P)
Wenn eine Firma Software braucht, steht sie oft vor der Alternative, selbst zu entwickeln oder einzukaufen. Geben Sie zwei (mögliche) Vorteile des einen und des anderen Verfahrens an!

Lösung:

Vorteile Software kaufen:

- Konzentration auf Kernkompetenzen.
- Für Massensoftware gilt: Kostengünstiger als selber entwickeln.
- ...

Vorteile Software selber bauen:

- Kritische Elemente (geschäftsrelevante Informationen) bleiben in der Firma.
- U.u. flexibler bei der Anpassung der Informationssysteme an neue Geschäftsgegebenheiten.
- Keine Abhängigkeit von Dritten.

B. Software-Architektur

Aufgabe 2.3: Architekturelle Stile (6 P)

Geben sie zu den unten beschriebenen Szenarien jeweils an:
- Welchen Architektur-Stil Sie verwenden würden
- Erklärungen zu einer möglichen Implementierung
- Eine kurze Begründung zur Wahl des Stils

a. (3P) Aus Dokumenten in einer Datenbank sollen für verschiedene Klienten Ausgaben in unterschiedlichen Formaten generiert werden. Die Dokumente können in unterschiedlichen Formaten gespeichert sein. Die Anfragen bestehen aus einfachen get(document_id, format) Nachrichten. Es soll leicht möglich sein, neue Formate für andere Arten von Clients hinzuzufügen. Clients können Endbenutzer-Applikationen als auch Programme sein, welche die Daten weiterverarbeiten.

b. (3P) Die Bundespolizei möchte ein neues System zur besseren Verbrechensbekämpfung einführen. Es soll den Datenaustausch zwischen verschiedensten Abteilungen in allen Kantonen ermöglichen. Der Datenaustausch darf dabei nur fallbezogen stattfinden, dass heisst, es werden keine fixen Verbindungen zwischen den Systemen etabliert. Es soll möglich sein, dass während des Verfahrens neue Abteilungen hinzustossen können. Abteilungen können z.B. sein: Labore die Analysen zur Verfügung stellen aber auch der Dorfpolizist, der feststellen will, ob ein gefundenes Auto im Zusammenhang mit dem untersuchten Fall steht.

Lösung (teilweise ohne die geforderten Erklärungen und Begründungen)

- **a.** Pipe and Filter oder MVC
- **b.** Blackboard: Unabhängigkeit von Produzenten und Konsumenten

---

C. Entwurf von Software und Systematisches Programmieren

2.4 Systematisches Programmieren und Vertragsorientierter Entwurf (15 P)

Gegeben ist folgender korrekt lauffähiger Java Code für das binäre Suchen über ein Array von verschiedenen Elementen:

```
0   package search.binary;
1
2   /**
3    * Klasse für die binaere Suche.
4    */
5   public abstract class BinSuchen {
6
7       /**
8        * Suche ueber Array v mit zu Element element.
9        */
10      public Ele srch(Ele[] v, Ele theElementWhichWeAreSearchingFor, int c)
11      {
12          v = permute(v);
13          int l, zR; l = c; // setze linken index für start der suche
14          zR = v.length - 1;
15
16          do {
17              c = (l + zR) / 2 ; // reuse c
18              if (v[c].comp2(theElementWhichWeAreSearchingFor) == 4) {
19                  zR = c - 1;
20              } else {
21                  l = c + 1;
22              }
23          } while ( v[c].comp2(theElementWhichWeAreSearchingFor) != 2 &&
24                    l <= zR );
25
26          Ele e = null; if (v[c].comp2(
27              theElementWhichWeAreSearchingFor) == 2) {
28          e = v[c];
29          } return e;
30      }
31
32      // Liste aufsteigend sortieren: Muss in einer Sub-Klasse
33      // implementiert werden
34      protected abstract Ele[] permute(Ele[] vx);
35  }
```

```
0   package search.binary;
1
2   /**
3    * Muss von Elementen, nach denen gesucht werden können soll,
4    * implementiert werden.
5    * Suchelemente müssen ebenfalls diese Schnittstelle implentieren..
6    *
7    */
8   public interface Ele {
9
10      /**
11       * Methode gibt 1 zurueck fuer groesser,
12       * 2 fuer gleich
13       * 4 fuer kleiner.
14       */
15      int comp2(Ele e);
16  }
```

Diese kleine Software-Bibliothek wurde gemäss folgender Spezifikation erstellt:

i. Eine Schnittstelle soll für das Suchen von Elementen eine Methodenschnittstelle zur Verfügung stellen. Die Methode nimmt als Argument ein anderes Element entgegen und liefert zurück, ob das übergebene Element kleiner, gleich oder grösser ist, als dasjenige Element, dessen Vergleichsmethode aufgerufen wurde. Jedes Objekt, nach dem gesucht werden kann, muss diese Schnittstelle implementieren.

ii. Eine Klasse soll eine Methode für die Suche innerhalb eines Feldes (Array) von Elementen zur Verfügung stellen. Neben dem zu durchsuchenden Array, soll ein einzelnes Element übergeben werden können, welches das zu suchende Element beschreibt. Die Suche soll binär erfolgen. Der linke Startindex für die binäre Suche soll mitübergeben werden.

iii. Bei der Suche darf das Eingabearray nicht verändert werden. Als Rückgabewert wird das gefundene Element aus dem Array zurückgegeben oder ein *null* Wert, sofern kein Element gefunden wurde.

iv. Da der grösste Aufwand bei der (für binäres Suchen notwendigen) Sortierung der Liste entsteht und die ganze Bibliothek flexibel sein soll, soll der Sortieralgorithmus in einer separaten Methode in einer Unterklasse bereitgestellt werden.

v. Unter keinen Umständen soll durch die Klasse bzw. eine der Methoden eine Ausnahme (Exception) geworfen werden.

Beantworten Sie folgende Fragen:

a. Untersuchen Sie den Code hinsichtlich der systematischen Programmierung. Achten Sie besonders auf die Kommentierung, Namengebung, Verträge der Schnittstellen, Formatierung und die Verwendung von geschlossenen Konstrukten, etc. Gehen Sie systematisch durch den Code und geben Sie für jedes gefundene Problem die Zeilennummer und eine Beschreibung an. Finden Sie bessere Namen wo dies für das Verständnis des Codes nötig ist. (8P)

Lösung:
Klasse BinSuchen

Zeile	Problem
2-3	Klassenkommentar ist mehr als dürftig. Zu einem Klassenkommentar gehört der Zweck der Klasse, die Verantwortlichkeiten, sowie die Elemente des Vertrages der Klasse (z.B. im Klassenkommentar die Invariante). Weiter sollten noch Projektinformationen, Autor etc. untergebracht werden.
5	Name *BinSuchen* kann besser gewählt werden. Den Verwender interessiert nicht das wie (binär) sondern nur das was (Suchen). Besser Namen nicht abgekürzt schreiben.
7-9	Methodenkommentar schlecht. In einen Methodenkommentar gehört, was die Methode macht. Eine genaue Beschreibung der Parameter und der Rückgabewerte hilft die Funktionsweise besser zu verstehen. Vertragselemente (Zusicherungen), wie Pre- und Postcondition sowie Obligations fehlen.
10	Namen *Ele*, *v*, *c* entweder zu kurz oder zu schlecht gewählt. Name *theElementWhichWeAreSearchingFor* ist nicht passend und zu lang.
11	Permute ist falscher Methodenname (siehe auch Zeile 15 von Ele)
13	*l*, *zR* sind eher zu kurze und schlecht gewählte Namen. Aussagekräftigere Namen, wie z.B. *leftBoundary*, *rightBoundary* wären besser. Zudem sollte die Zeile auf mehrere Zeilen aufgeteilt werden.
1-24	Annehmende Schleifen sind u.U. problematisch. Hier könnte durchaus eine abweisende Schleife verwendet werden. Falls *v != null && v.length == 0* dann gibt es eine Ausnahme (Exception), weil auf das 0 Element zugegriffen wird. Entweder ist

	eine Abfrage auf *v.length == 0* nötig oder eine abweisende Schleife, welche nicht eintritt, falls *v.length == 0*. Alternativ kann das Problem auch über eine Formulierung im Vertrag behoben werden.
17	Die Variable *c* wird in einem anderen Kontext wieder verwendet. Dies ist schlecht und kontraintuitiv.
18	*Comp2* ist ein schlechter Name. Es sollten symbolische Konstanten verwendet werden. (siehe auch Zeile 15 von Interface *Ele*).
23	Symbolische Konstanten verwenden.
26-29	Schlecht formatiert und schlecht eingerückt!
27	Symbolische Konstanten
32-33	Besserer Methodenkommentar (siehe Zeile 7-9), einheitlich kommentieren.
34	Falscher Name führt den Leser des Codes in die Irre. Schlecht! Parametername *vx* ist schlecht.

Interface Ele

Zeile	**Problem**
2-7	Schlechter Interface-Kommentar (siehe Zeile 2-3 von *BinSuchen*)
8	Schlechter Name für Interface (Abkürzung schlecht).
8, 10-14	Symbolische Konstanten sollten definiert werden.
15	*Comp2* und *e* sind schlechte Namen.
10-14	Besserer Methodenkommentar nötig (siehe oben)
	Generell: zu wenig Kommentar im Code selber.
	Generell: Verträge fehlen.

---

b. Schreiben Sie einen Klassenkommentar für die Klasse *BinSuchen*, sowie einen Kommentar für die beiden dazugehörenden Methoden in einer Art und Weise, dass diese von Software-Entwicklern ohne Einblick in den Code und die Spezifikation der Bibliothek verwendet werden können. Beschreiben Sie dazu auch die Elemente des Vertrags der Klasse *BinSuchen* (Vorbedingungen, Nachbedingungen, Verpflichtungen und Invariante), indem Sie diese in einer formalen oder informalen Weise im Kommentar unterbringen. (7 P)

Lösung:

```
/**
 * Stellt die Möglichkeit einer binären Suche zur Verfügung. Zu diesem
 * Zweck wird ein Feld von gegebenen Elementen sortiert.
 * Um den Suchmechanismus zu verwenden, ist eine Instanziierung dieser
 * Klasse nötig.
 * @Author      Silvio Meier
 * @Invariante  true
 */
public abstract class Search {

    /**
```

```
     * Sucht über ein gegebenes Array von Objekten (elementVector),
     * welche die Schnittstelle Element implementieren und
     * das zu suchende Element beschreibt. leftBondaryIndex
     * gibt den linken Start-Index für die binäre Suche an.
     * @pre  elementVector != null and elementVector.length > 0
     * @pre  elementToSearchFor != null
     * @pre  leftBoundaryIndex >= 0 &&
     *       leftBoundaryIndex <= elementVector.length
     * @post elementVector@pre == elementVector &&
     *       elementVector.length@pre == elementVector.length
     */
    public Element search(Element[] elementVector,
                          Element elementToSearchFor,
                          int leftBoundaryIndex)    {
    }

    /**
     * Diese Methode muss durch eine Unterklasse dieser Klasse
     * implementiert werden. Es muss ein Sortieralgorithmus
     * implementiert werden, der aufsteigend sortiert.
     * Diese Methode gibt eine sortierte Kopie des Eingabearrays
     * von Objekten zurück, die die Schnittstelle Element implementieren.
     * Das originale Array wird nicht verändert.
     * @pre   elementVector != null
     * @post  elementVector.length > 0 implies
     *        for all i : int = 0..elementVector.length-1 in result |
     *          result[i] < result[i + 1]
     * @post  elementVector@PRE == elementVector
    protected abstract Element[] sort(Element[] elementVector);
}
```

2.5 Rekursion vs. Iteration (10 P)
Folgende iterative Version des Euklischen GgT-Algorithmus ist gegeben.

```
0    // Der Algorithmus basiert auf der Eigenschaft
1    // ggt(u, v) == ggt(u-v,v), wenn u >= v
1    public int ggt(int u, int v) {
2      int t = 0; // Hilfsvariable für den Tausch von u und v
3
4      while (u > 0) {
5       if (u < v) {
6          t = u;
7          u = v;
8          v = t; //u und v getauscht
9       }
10      u = u - v;
11     }
12     return v;
```

a. Die Iteration kann durch eine Rekursion ersetzt werden. Schreiben Sie eine möglichst kurze rekursive Methode für den oben stehenden Algorithmus in korrektem Java. **(6 P)**

b. Welche Zeile Ihrer rekursiven Funktion enthält die Verankerung? Begründen Sie Ihre Antwort. **(1,5 P)**

c. Welche Zeile ihres rekursiven Algorithmus ist für die Reduktion zuständig? Begründen Sie Ihre Antwort. **(1,5 P)**

d. Ist die rekursive oder die iterative Version in diesem Fall vorzuziehen? Begründen Sie Ihre Antwort. **(1 P)**

Lösung:

a. Rekursive Lösung des Problems:

```
public static int ggt(int u, int v) {
      int t = 0;
      if (u > 0) {
            if (u < v) {
                  t = u;
                  u = v;
                  v = t;
            }
      } else return v;
      return ggt(u - v, v);
}
```

b.
Die Verankerung ist durch die Zeile if (u > 0) bzw. else return v; gegeben.

c. Zeile *return ggt(u – v, v)* ist für die Reduktion zuständig.

d. Obwohl die rekursive Version mathematisch gesehen klarer ist, dürfte die iterative Version für die meisten Menschen besser verständlich sein. Daher ist die iterative Version der rekursiven vorzuziehen, zudem entstehen keine Speicherprobleme durch einen wachsenden Kellerspeicher (Stack).

D. Prüfverfahren
Aufgabe 2.6: Testen (19P)

Gegeben sei die folgende Spezifikation einer Funktion zur Berechnung des Grössten Gemeinsamen Teilers:

Die Funktion nimmt zwei 8-Bit Ganzahlwerte *u* und *v* und berechnet daraus die grösstmögliche Zahl als Rückgabewert, welche u und v ganzzahlig (ohne verbleibenden Rest) teilt.

```
/**
 * Berechnet den groessten gemeinsamen Teiler von a und b.
 *
 * @pre 0 < u <= 255
 * @pre 0 < v <= 255
 * @post Rueckgabewert entspricht dem groessten gemeinsamen Teiler.
 */
 int ggT(int u, int v);
```

a) Beschreiben Sie kurz die Kriterien, nach denen Sie die Testfälle entwerfen. Entwerfen Sie dann konkrete Testfälle für einen Black-Box-Test für die beschriebene Funktion und stellen Sie die von Ihnen ausgewählten Testfälle tabellarisch dar. Geben Sie an, nach welchen Kriterien Sie Ihre Tests ausgewählt haben. Achten Sie auf eine ökonomische Auswahl Ihrer Testfälle. (10P)

b) Welche Elemente des Flussgraphen eines zu testenden Programmes müssen beim White-Box Test jeweils für die Anweisungs-, Zweig- und Pfadüberdeckung im Graphen wie oft besucht werden? Nennen Sie die Elemente und die Anzahl der nötigen Besuche. (3P)

c) Folgende Implementierung der oben beschriebenen Funktion ist gegeben:

```
1    public int ggT(int u, int v) {
2      int t = 0; // Hilfsvariable für den Tausch von u und v
3
4      while (u > 0) {
5       if (u < v) {
6          t = u;
7          u = v;
8          v = t; //u und v getauscht
9       }
10      u = u - v;
11     }
12     return v;
13    }
```

Erstellen Sie einen Flussgraphen für diese Implementierung. (4P)

d) Für einen Black-Box Test wird die Implementierung nicht benötigt. Kann man umgekehrt sagen, ein White-Box Test benötigt die Spezifikation nicht? Begründen Sie! (2P)

Lösung:

a. Äquivalenzklassen, Unterteilung der Äquivalenzklassen nach z.B. folgenden Kriterien:

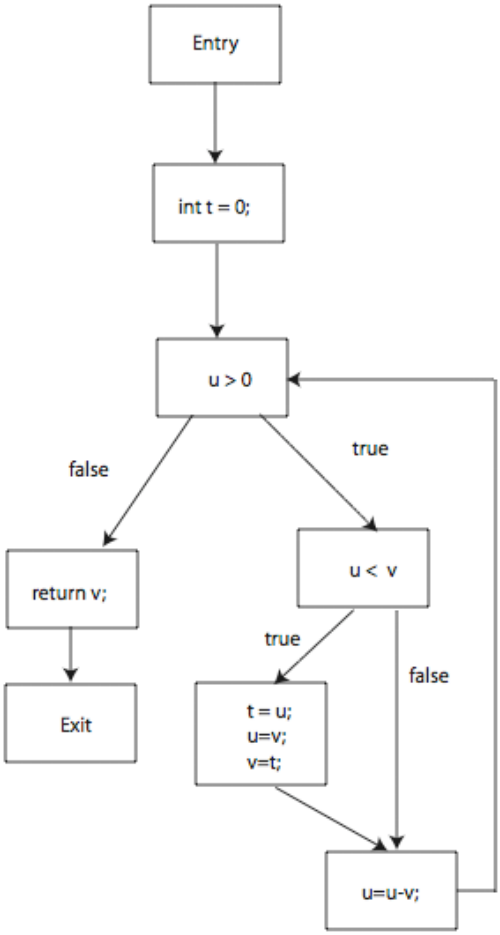
I. Äquivalenzklasse für Eingabewerte u und v, bei denen eine Primzahl resultiert
II. Äquivalenzklasse für Eingabewerte u und v, bei denen eine nicht-Primzahl resultiert
III. Äquivalenzklasse für Eingabewerte u und v, wo u das grössere Element ist und v das kleinere
IV. Äquivalenzklasse für Eingabewerte u und v, wo u das kleinere Element ist und v das grössere
V. Grenzwerte: Verwendung der unteren Schranke des Wertebereichs für beide Argumente u=1, v=1
VI. Grenzwerte: Verwendung der oberen Schranke des Wertebereichs für beide Argumente u=255, v=255
VII. Grenzwerte: Verwendung einer unteren und einer oberen Schranke des Wertebereichs u=1, v=255
VIII. Grenzwerte/Äquivalenzklasse: Verwendung der oberen Schranke des Wertebereichs und eines Wertes der nicht Schranke ist, u=255, v=45
IX. Grenzwerte/Äquivalenzklasse: Verwendung der unteren Schranke des Wertebereichs und eines Wertes, der nicht eine Schranke ist, u=66, v=1

Das Testen von Werten ausserhalb des Wertebereichs macht keinen Sinn, da für diese Werte kein Resultat zugesichert wird.

Testfall Nr.	Eingabewerte	Erwartete Resultate	Kriterium
1	u=1, v=1	1	Grenzwert, V
2	u=255, v=255	255	Grenzwert,VI
3	u=14, v=49	7	Äquivalenzklasse, I + IV
4	u=242, v=11	11	Äquivalenzklasse, I + III
5	u=128, v=36	4	Äquivalenzklasse II
6	u=1, v=255	1	Grenzwerte VII
7	u=255, v=45	5	Grenzwert VIII
8	U=66, v=1	1	Grenzwert IX

b.
Für die Anweisungsüberdeckung muss mindestens jeder Knoten einmal besucht werden. Für die Zweigüberdeckung muss jede Kante mindestens einmal traversiert werden. Für die Pfadüberdeckung muss mindestens jeder mögliche Pfad durch den Graphen mindestens einmal durchlaufen werden.

c.
Der Flussgraph sieht wie folgt aus:



d. Die Struktur alleine reicht nicht. Um die Sollergebnisse zu ermitteln, wird auch beim White-Box Test die Spezifikation benötigt.

Aufgabe 2.7: Reviews (12 P)
Ein Review ist eine Massnahme zur Prüfung von Dokumenten und Programmen in der Software-Entwicklung. Folgendes Fallbeispiel beschreibt die Durchführung eines Reviews, bei dem verschiedene Review-Regeln befolgt und gegen andere verstossen wurde.

1	Silvio, Patrick, Harald, Martin und Nancy treffen zum vereinbarten Termin im Konferenzraum ein.
2	Martin nimmt sich einige Filzstifte und schreibt die Agenda an die Tafel: "Code-Review: Was ist
3	in Silvios Java-Klassen alles falsch". Nach kurzer Durchsicht der Vorbereitungsnotizen ermahnt
4	Martin Harald, sich zukünftig besser auf die Reviews vorzubereiten. Zu Beginn beanstandet
5	Patrick die mangelhafte Programmiererfahrung von Silvio und dessen mangelhafte
6	Arbeitsauffassung.
7	Aufgrund von Silvios Anregung werden die Prüfunterlagen klassenweise von vorne nach hinten
8	durchgearbeitet. Beim Durchgehen der ersten beiden Seiten des Prüflings ergeben sich gehäuft
9	Diskussionen über Stilfragen. Zudem wurden zwei kritische Befunde in den Referenzunterlagen
10	gefunden und von Nancy in einer separaten Befundliste aufgenommen. Auf den folgenden Seiten
11	bemängelt Harald bei diversen Klassen die gewählte Lösungsvariante und schlägt eine
12	elegantere Implementierung vor.
13	Nach zwei Stunden ist knapp die Hälfte der Java-Klassen geprüft. Die Review-Teilnehmer
14	entschliessen sich einhellig, den restlichen Code zügig zu Ende zu prüfen und hierfür die Zeit zu
15	verwenden, die für eine informelle Nachsitzung geplant war. Nach circa 3 Stunden wird die
16	Befundliste abgeschlossen. Patrick, Harald, Silvio und Nancy verabschieden sich zum
17	wohlverdienten Mittagessen. Martin entschliesst sich, das Prüfdokument als akzeptabel mit
18	kleinen Änderungen einzustufen und füllt entsprechend den Review-Bericht aus. Anschliessend
19	schliesst er sich den anderen beim Mittagessen an, um mit ihnen Haralds
20	Verbesserungsvorschläge weiter zu diskutieren.

a. Identifizieren Sie welche Person welche Rolle im Review einnimmt. (2P).

Lösung:
Autor: Silvio
Gutachter: Harald
Gutachter: Patrick
Schreiber: Nancy
Moderator: Martin

---

b. Im Fallbeispiel wurden einige Review-Regeln verletzt, andere wurden eingehalten. Geben Sie die entsprechenden Zeilen an, was gemacht wurde und ob es den Review-Regeln entspricht oder nicht. (10P)

Hinweis: Falsche Antworten geben Punktabzüge.

Lösung:

Zeile	Was?	Regelkoform / Verstoss gegen Regeln
1	Vorausgeplanter Termin	Regelkonform
2–3	Es sollten positive, wie auch negative Aspekte im Prüfling aufgezeigt werden.	Verstoss
3–4	Alle Review-Teilnehmer kommen vorbereitet. Harald kommt schlecht vorbereitet	Ermahnung ist regelkonform. Schlecht Vorbereitet ist ein Verstoss gegen die Regeln.
4–5	Es werden die zu Prüfenden Dokumente bewertet, nicht die Autoren.	Verstoss
7–8	Das Review sollte systematisch durch den Code durchgehen.	Verstoss (Schlecht geplant), bzw. Regelkonform (dann schlieslich doch noch strukturiert vorgegangen)
8–9	Keine Stilfragen diskutieren während der Sitzung.	Verstoss
9–10	Befunde durch Moderator in Befundliste eintragen	Regelkonform
11-12	Keine Diskussion von Lösungen	Verstoss
13-16	Zu viele Elemente für die Review-Prüfung	Verstoss
17-18	Alle Review-Teilnehmer haben den Reviewbericht abzusegnen	Verstoss
19-20	Diskussion von Lösungsvorschlägen ausserhalb des Reviews	Regelkonform

F. Statische Programmanalyse

2.8 Sprachen und Parsing (10P)

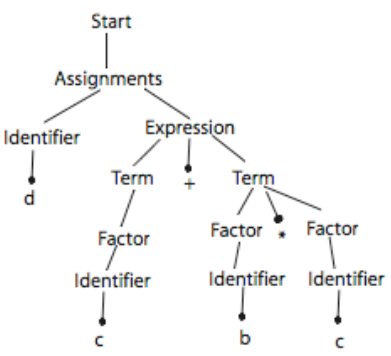
Folgende Grammatik einer kleinen „Programmiersprache“ *M* für das Formulieren von einfachen mathematischen Ausdrücken ist gegeben:

Start	::=	**'begin'** (Assignments)* **'end'**
Expression	::=	Term ((**'+'** Term) | (**'-'** Term))*
Term	::=	Factor ((**'*'** Factor) | (**'/'** Factor))*
Factor	::=	Identifier | (<DECIMAL_LITERAL>) | **'('** Expression **')'**
Assignments	::=	(Identifier **':='** Expression **';'**)
Identifier	::=	**'a'** | **'b'** | **'c'** | ..... | **'x'** | **'y'**

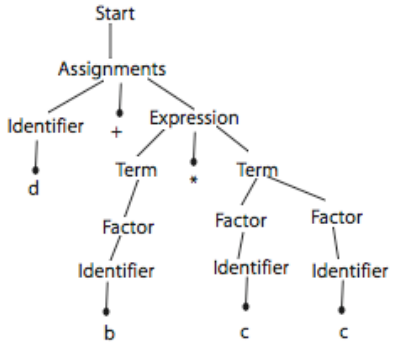
Legende:
( )* Element in der Klammer ist 0 bis n-Mal beliebig oft wiederholt
( )? Element in der Klammer ist optional
A | B Entweder A oder B wird gewählt
( ) Elemente in der Klammer werden vor den Elementen ausserhalb der Klammer ausgewertet.
Ein Terminalsymbol steht in Hochkommas (Bsp. 'terminal') und der Name einer Produktion beginnt mit einem Grossbuchstaben.
<DECIMAL_LITERAL> bezeichnet alle möglichen positiven und negativen ganzzahlige Werte.

Folgender Ausdruck in der obigen Sprache ist gegeben: *begin d:=c+b*c; end*

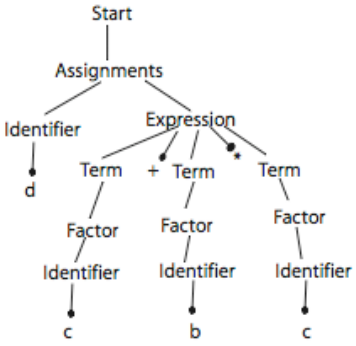
Folgende drei Abstrakte Syntaxbäume sind gegeben:



Baum A



Baum B



Baum C

a. Welche(r) der oben gegebenen abstrakten Syntaxbäume A, B, C sind korrekte abstrakte Syntaxbäume der Sprache *M*? Begründen Sie warum die anderen Bäume falsch sind (6 P).

b. Welche der folgenden Aussagen sind zu den oben gegebenen Sprache richtig? Begründen Sie, falls die Aussage nicht richtig ist: (4 P)

b.1 Die lexikalische Analyse von Spracheingaben der Sprache *M* befasst sich mit der Umwandlung von Quellcodes in einen Strom von Tokens.

b.2 Die Token der Grammatik der Sprache *M* sind die folgenden Symbole 0|1|2|3|4|5|6|7|8|9 und <DECIMAL_LITERAL>.

b.3 Syntaxbäume, welche semantische Informationen annotiert haben, sind die Grundlage für die Codeanalyse durch Analysatoren respektive die Grundlage für die Zwischencodegenerierung bei Compilern.

Lösung:
a.

Baum A ist richtig. 1P

Baum B ist falsch, weil die Operatoren an den falschen Knoten hängen und die Reihenfolge im Baum nicht stimmt. 2,5P

Baum C ist falsch, weil der Operator * am falschen Knoten hängt und der letzte Teilast nicht an den Expression Knoten angehängt werden darf. Zudem wird die Operatorbindungsstärke nicht mehr richtig ausgedrückt (Punkt-Vor-Strich). 2,5P

b.1 Aussage ist richtig.

b.2 Die Aussage ist falsch: I. Ein Token(strom) resultiert von der lexikalischen Analyse und nicht von einer Grammatik II. Es gibt noch weitere Elemente, die als Token bei der lexikalischen Analyse resultieren: +, -, *, etc.

b.3 Aussage ist richtig.

Teil F: Programmanalyse, Reengineering und Reverse Engineering und Software Evolution (8P)

Aufgabe 2.9: Reengineering und Reverse Engineering (4P)

Erklären Sie die folgenden Begriffe aus dem Reverse Engineering anhand der Tabelle durch Ankreuzen (falsch angekreuzte Elemente ergeben Abzug):

	Source Code analysieren	Neue Requirements integrieren	Domänen-Modell erstellen	Source Code Struktur (nicht Funktionalität) verändern	Source Code und Funktionalität verändern
Reverse Engineering	☒	☐	☒	☐	☐
Reengineering	☒	☒	☒	☐	☒
Restructuring	☒	☐	☐	☒	☐
Refactoring	☒	☐	☐	☒	☐

Lösung: in Aufgabe eingetragen

Aufgabe 2.10: Software Change (4 P)

a. Nennen Sie vier Gründe, warum eine Software im Laufe der Zeit verändert bzw. angepasst werden muss.

Lösung:

- Neue Anforderungen aus dem Gebrauch
- Das Geschäftsumfeld ändert sich
- Fehler beheben
- Neue Hardware
- Leistung, Verlässlichkeit muss verbessert werden
- Regulatorische Änderungen
- Etc.