Lösungen zu den Übungsaufgaben

Lösungsskizzen zu den Aufgaben aus dem Buch „**Grundkurs Software-Engineering mit UML**“ von Stephan Kleuker aus dem Verlag Stringer Vieweg (alle Rechte vorbehalten)

Die folgenden Lösungen stellen meist jeweils eine der möglichen Antworten auf die jeweilige Aufgabenstellung dar. Eine wichtige Erkenntnis des SoftwareEngineerings ist, dass, wenn zwei Leute das Gleiche sagen, sie nicht unbedingt das Gleiche meinen. Dies gilt insbesondere, wenn ein Auftragnehmer die Wünsche des Kunden verstehen will. Ähnliches gilt für Aufgabenstellungen und deren Lösungen. Da Aufgabenstellungen teilweise Interpretationen erlauben, erweitert sich die Anzahl sinnvoller Lösungen noch mehr.

Bilder und Programmcode der Lösungen sind von der Web-Seite des Buches herunterladbar.

Lösungen zu Kapitel 2

zu 1)

Anmerkungen: Man sieht am Produkt Verhandlungsbedingungen, dass Produkte auch optional für Aktionen sein können, also nicht vorliegen müssen.

Der Kunde hat in den Verhandlungen nach der folgenden Modellierung keine Ausstiegsmöglichkeit mehr.

Es gibt eine Optimierungsmöglichkeit, wenn Entscheider frühzeitig in Vertragsverhandlungen eingebunden werden.

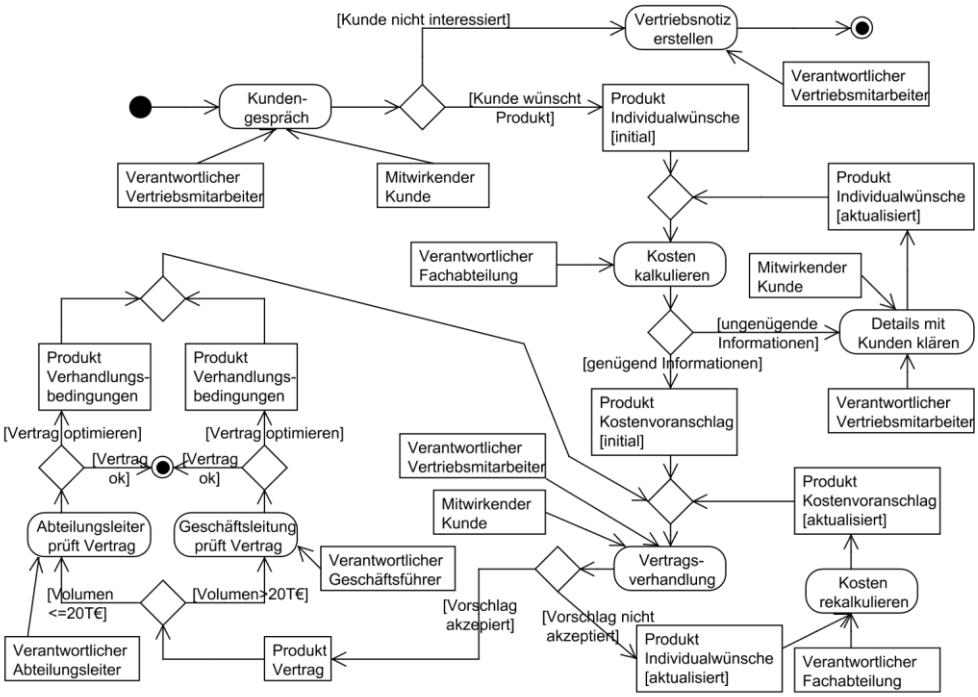


Abb. 1: Lösungsskizze zu Aufgabe 1

zu 2)

Der Fall, dass einige Aufgaben gemeinsam, andere getrennt bearbeitet werden, könnte noch ergänzt werden.

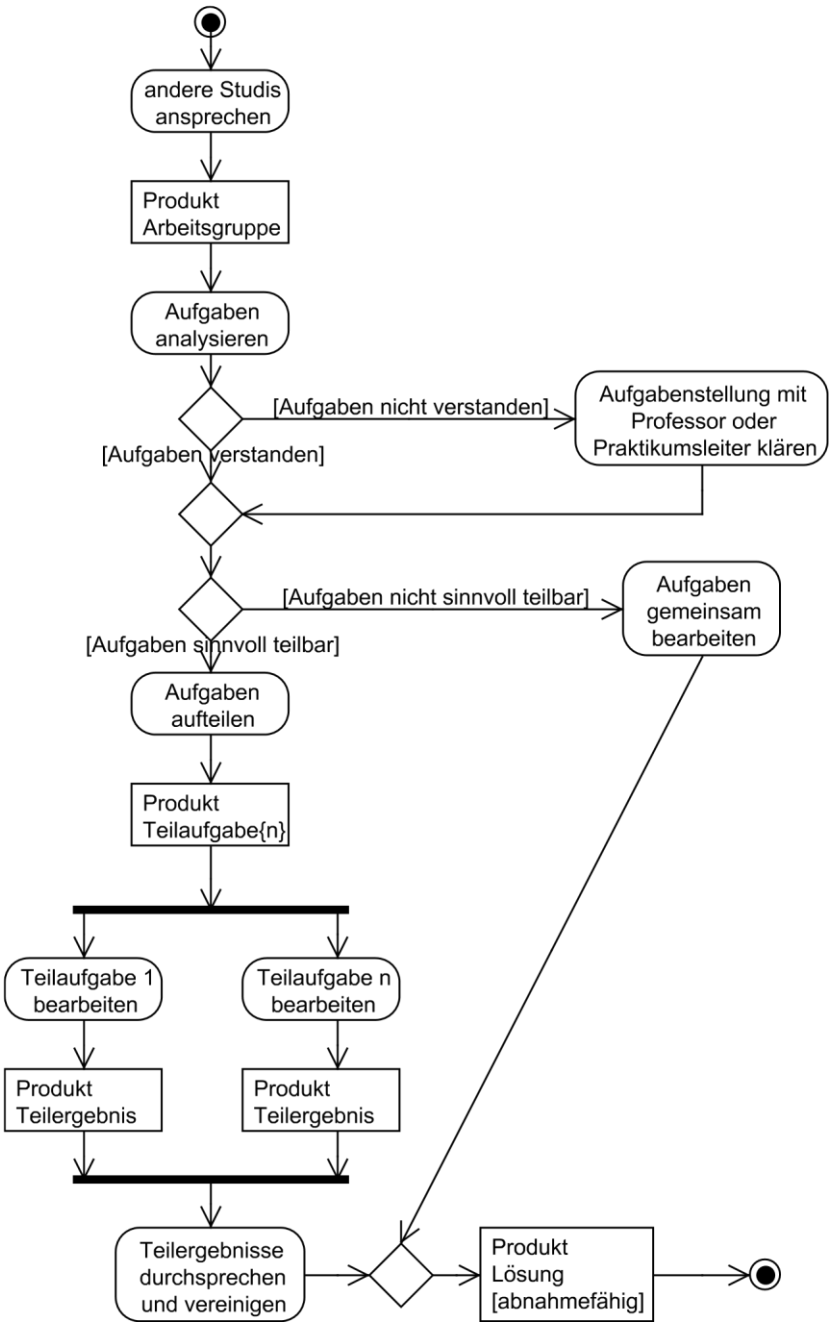


Abb. 2: Lösungsskizze zu Aufgabe 2

zu 3)

i) Werden Entscheidungen vom Kunden verlangt, ist unklar, wer auf Kundenseite entscheiden darf.
   Das Verhalten des Kunden bei späteren Änderungswünschen ist unklar, ein informeller Prozess für gütliche Einigungen ist noch nicht eingespielt.

ii) Mitarbeiter mit Kernkompetenzen stehen nicht zum gewünschten Zeitpunkt zur Verfügung.
   Mitarbeiter werden vor der Fertigstellung ihrer Arbeit aus dem Projekt abgezogen, da eine Fehlkalkulation vorlag. Da ein anderes Projekt höhere Priorität hat, kann die Arbeit erst zu einem späteren Zeitpunkt beendet werden.

iii) Man entschließt sich, dass Ihr Projekt nicht mehr zum Kerngeschäftsfeld gehört und deshalb nicht mehr die zugesagten Mitarbeiter oder Mittel bekommt.
   Die Geschäftsführung beschließt, für ein neues Tätigkeitsfeld eine große Anzahl neuer Mitarbeiter einzustellen und wünscht, dass Sie sich in Ihrem Projekt in das Unternehmen einarbeiten.

Lösungen zu Kapitel 3

zu 1)

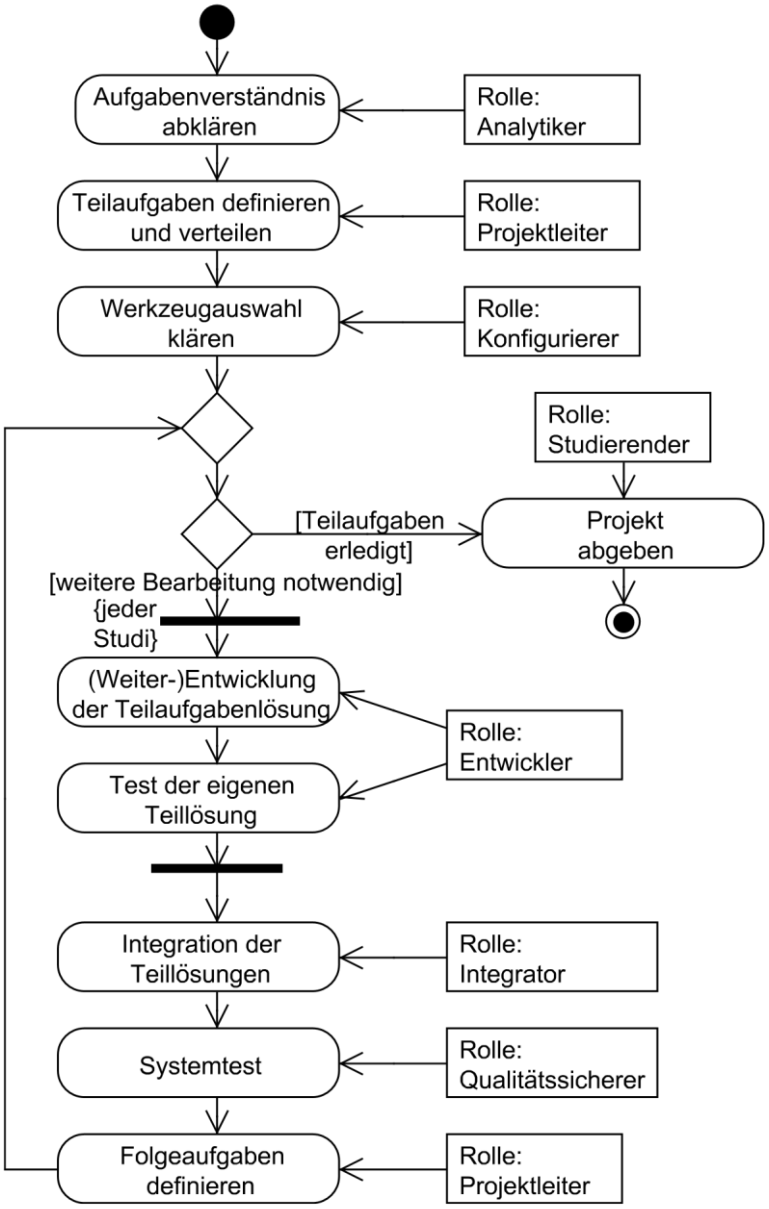


Abb. 3: Lösungsskizze zu Aufgabe 1

zu 2)

Risiko	**Maßnahme in 1)**
Die Aufgabenstellung wird unterschiedlich verstanden	Aktion zur Klärung des Aufgabenverständnisses
Studierende nutzen unterschiedliche Werkzeuge bzw. unterschiedliche Versionen, deren Ergebnisse nicht zusammenpassen	Aktion zur Klärung der Werkzeuglandschaft
Bei der Nutzung vom Code anderer Studierender wird auf veraltete Lösungen zugegriffen, wodurch spätere Teillösungen nicht integrierbar sind	teilweise durch iterative Entwicklung verhindert; sinnvoll wäre ein Werkzeug zur Verwaltung von Software-Ständen (zum Konfigurationsmanagement) mit gemeinsamem Repository
einer der Studierenden ist mit seiner Teilaufgabe überfordert bzw. unzuverlässig	iterative Entwicklung in kleinen Schritten mit der Aktion „Folgeaufgaben definieren“ zur möglichen Anpassung der Aufgaben
Gemeinsame Interpretation der Aufgabenstellung passt nicht zu den Vorstellungen des Aufgabenstellers	es fehlt eventuell eine Klärung der Aufgabenstellung nach der ersten Aktivität, auch wäre eine Durchsprache mit dem Aufgabensteller nach einer Integration möglich

Lösungen zu Kapitel 4

zu 1)

a)

Endanwender: Studierende, Mitarbeiter im Immatrikulations- und Prüfungsamt

Management des Auftraggebers: Geschäftsführung unserer IT-Firma

Käufer des Systems: Hochschulleitung

Prüfer: Verantwortliche der Hochschule zur Abnahme des Systems

Entwickler: unsere Entwickler, deren Know-how bei der Auswahl des Realisierungsansatzes berücksichtigt werden sollte

Wartungs- und Servicepersonal: SW-Installateur, z. B. Web-Master (evtl. auch als Anwender)

Produktbeseitiger: spielt bei reiner SW keine Rolle

Schulungs- und Trainingspersonal: Personen, die die Mitarbeiter schulen, evtl. nur technische Redakteure der Nutzungsbeschreibung

Marketing und Vertriebsabteilung unseres Unternehmens (z. B. Wunsch nach der Übertragbarkeit auf andere Hochschulen), man kann auch über das Marketing des Kunden nachdenken /z. B. der Wunsch nach einem einheitlichen fortschrittlichen Internet-Auftritt)

Systemschützer: Datenschutzbeauftragter, Verantwortlicher für InternetAnbindung (Security)

Standards und Gesetze: Qualitätsvorgaben unseres Unternehmens, Datenschutzgesetz

Projekt- und Produktgegner: muss konstruiert werden, z. B. Studis, die die Anonymisierung der Menschen und Dienstleistungen durch das Internet ablehnen

Kulturkreis: Studierende unterschiedlicher Herkunft und Muttersprache

Meinungsführer (und öffentliche Meinung): spielen hier kaum eine Rolle

b)

Ziel	**Studierende sollen sich schnell über Ihre Noten informieren können**
Stakeholder	Studierende
Auswirkungen auf Stakeholder	Studierende können sich schneller informieren, Sekretariate werden von Notennachfragen verschont
Randbedingungen	Internet-Seiten müssen funktionieren
Abhängigkeiten	-
Sonstiges	-

Ziel	**SW muss in die vorhandene Verwaltungssoftware eingegliedert sein**
Stakeholder	Mitarbeiter im Immatrikulations- und Prüfungsamt
Auswirkungen auf Stakeholder	Arbeitsprozess zur Verwaltung von Studierende und Noten werden verändert, gegebenenfalls auf andere Mitarbeiter verlagert oder gar überflüssig
Randbedingungen	Vorhandene SW muss offene dokumentierte Schnittstellen haben
Abhängigkeiten	-
Sonstiges	Entwicklungsplanung für die weitere SW-Infrastruktur der
	Hochschule beachten
Ziel	Internationalität der Hochschule dokumentieren
Stakeholder	(potenzielle ausländische) Studierende, Hochschulleitung
Auswirkungen auf Stakeholder	ausländische Studierende können sich ohne Zuhilfenahme Dritter über ihre Noten informieren, Hochschule dokumentiert durch die Mehrsprachigkeit ihres Internetauftritts ihre internationalen Ambitionen
Randbedingungen	Es müssen Übersetzer für die unterstützten Sprachen gefunden werden
Abhängigkeiten	-

Sonstiges	Layout-Vorgaben können sich in Sprachen unterscheiden (z. B. Arabisch von rechts nach links)

c)

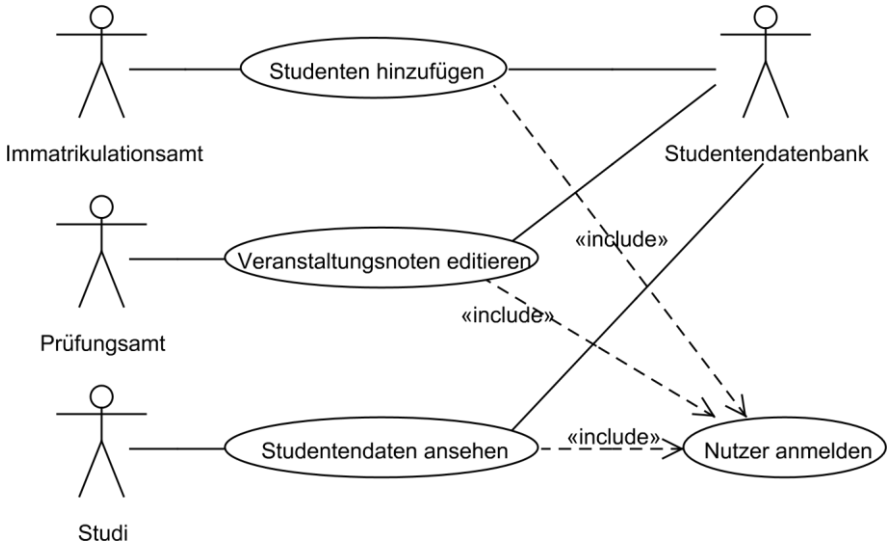


Abb. 4: Lösungsskizze zu Aufgabe 1c)

Hinweis: Den Aktor Studentendatenbank gibt es nur, wenn diese nicht als Teil des Projekts entwickelt wird.

Name des Use Case	Studenten hinzufügen
Nummer	U1
Paket	-
Autor	„ich“
Version	1.0
Kurzbeschreibung	Die Daten eines neuen Studierenden werden in das System eingetragen.
beteiligte Aktoren (Stakeholder)	Mitarbeiter im Immatrikulationsamt

Fachverantwortlicher	Leiter Immatrikulationsamt
Referenzen	Prozessbeschreibung des Immatrikulationsamtes
Vorbedingungen	System läuft
Nachbedingungen	neuer Studierender im System gesichert; für ihn können Noten eingetragen werden, System bereit neue Aufgabe zu erfüllen
typischer Ablauf	1. Mitarbeiter meldet sich in seiner Rolle am System an 2. Mitarbeiter wählt Möglichkeit zur Ergänzung eines Studierenden 3. Mitarbeiter trägt Daten ein 4. Mitarbeiter fordert System zur Datenübernahme auf 5. System bestätigt Datenübernahme
Kritikalität	unabdingbar, muss ausfallsicher laufen
Verknüpfungen	-

Mitarbeiter werden sich sicherlich nicht für jede Aktion beim System neu anmelden. Die Alternative, dass der Nutzer schon beim System angemeldet ist, würde in einem alternativen Ablauf berücksichtigt.

Name des Use Case	Veranstaltungsnoten editieren
Nummer	U2
Paket	-
Autor	„ich“
Version	1.0
Kurzbeschreibung	Die Note zu einer bestimmten Veranstaltung eines im System bekannten Studierenden wird neu eingetragen bzw. geändert.
beteiligte Aktoren (Stakeholder)	Mitarbeiter im Prüfungsamt

Fachverantwortlicher	Leiter Prüfungsamt
Referenzen	Prozessbeschreibung des Prüfungsamtes
Vorbedingungen	System läuft
Nachbedingungen	Neue bzw. aktualisierte Note eines Studierenden ist im System gesichert, System bereit neue Aufgabe zu erfüllen
typischer Ablauf	1. Mitarbeiter meldet sich in seiner Rolle am System an 2. Mitarbeiter wählt Möglichkeit zur Ergänzung bzw. Änderung von Veranstaltungsnoten 3. Mitarbeiter trägt Daten ein 4. Mitarbeiter fordert System zur Datenübernahme auf 5. System bestätigt Datenübernahme
Kritikalität	muss absolut zuverlässig laufen, eventuelle Funktionseinschränkung am Anfang akzeptabel
Verknüpfungen	-

Name des Use Case	Studentendaten ansehen
Nummer	U3
Paket	-
Autor	„ich“
Version	1.0
Kurzbeschreibung	Studierende können sich im System über ihre aktuellen Noten informieren
beteiligte Aktoren (Stakeholder)	Student
Fachverantwortlicher	(z. B. Fachbereichssekretariat)
Referenzen	-

Vorbedingungen	Web-Komponente läuft, nutzbare Internetverbindung
Nachbedingungen	Student konnte seine Noten einsehen, System bereit neue Aufgabe zu erfüllen
typischer Ablauf	1. Student meldet sich in seiner Rolle am System an 2. Student wählt die ihn interessierenden Informationen zur genauen Betrachtung aus 3. Student meldet sich vom System ab
Kritikalität	muss absolut zuverlässig laufen, eventuelle Funktionseinschränkung am Anfang akzeptabel
Verknüpfungen	-

d)

Risiko	**Es können nicht alle Veranstaltungsvarianten und Notenberechnungsarten im System verwaltet werden**
Gegenmaßnahme	Klärung mit den Prüfungsämtern und Prüfungsausschüssen, welche Veranstaltungsarten und Noten im System einsehbar sein sollen
Messung	Vorliegen einer schriftlichen Aufstellung möglicher Varianten zu jedem Studiengang

Risiko	**Web-Interface passt nicht zur einheitlichen Darstellung der Hochschule**
Gegenmaßnahme	Einbindung des Verantwortlichen der Hochschule zum verwendeten Corporate Design
Messung	Abstand der dokumentierten Kontakte zum Verantwortlichen

Risiko	**Integration in existierende Systeme funktioniert nicht**
Gegenmaßnahme	Analyse aller möglichen einzubindenden Systeme
Messung	Vorliegen einer Schnittstellendokumentation und eines Tests für jedes einzubindende System

e) Mit der folgenden Beschreibung wird festgehalten, dass erfolgreiche Anmel-dungen nicht protokolliert werden (da diese Information einmal im Diagramm erwähnt wird, muss es erlaubt sein, auf die anderen Fälle zu schließen), dies ist mit dem Kunden zu klären.

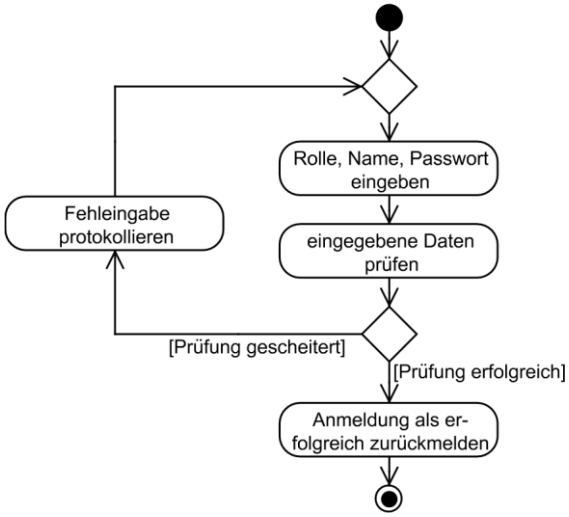


Abb. 5: Nutzer anmelden

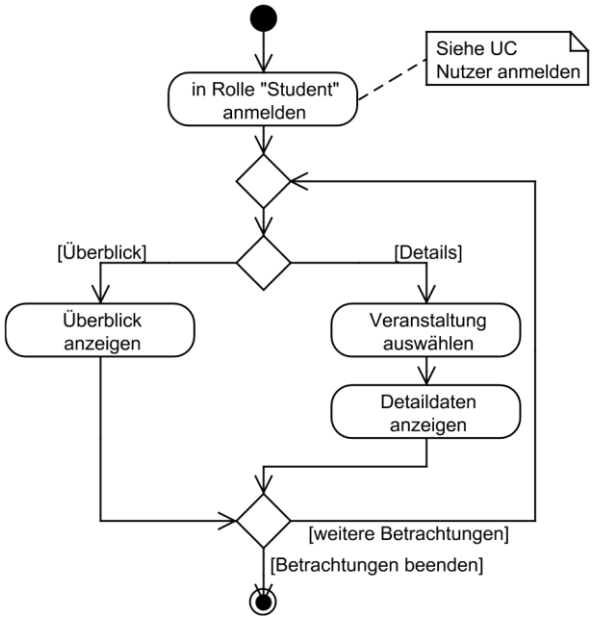


Abb. 6: Studentendaten ansehen

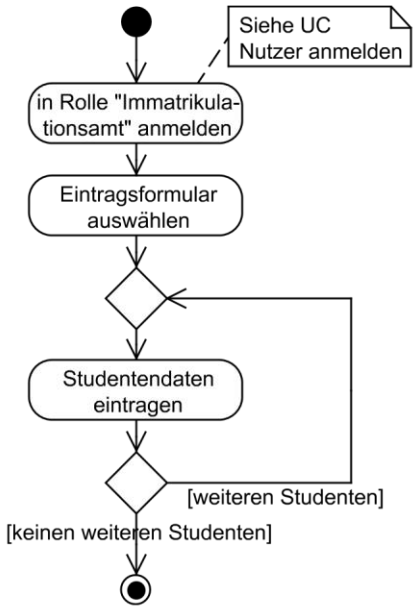


Abb. 7: Studenten hinzufügen

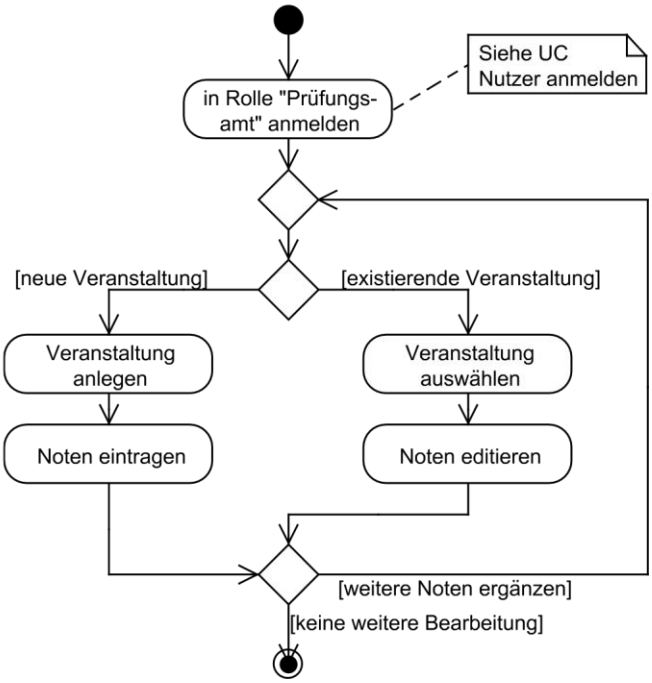


Abb. 8: Veranstaltungsnoten editieren

f) zu „Nutzer anmelden“

Aus der Aktion „Name, Passwort und Rolle eingeben“ kann abgeleitet werden:

ANF 1: Anmeldemöglichkeit:

Im Ausgangszustand muss das System einem potenziellen Nutzer die Möglichkeit bieten, einen Namen, eine Rolle und ein Passwort einzugeben.

Hinweise: Wenn „einzugeben“ nicht weiter spezifiziert wird, ist die Form des Nutzungsdialogs offen. Dieser kann hier explizit erklärt oder gegebenenfalls aus einem für das Projekt allgemeingültigen Styleguide abgeleitet werden.

„Ausgangszustand“ wird im Glossar als Zustand nach dem Starten des Systems und dem Aufbau der Oberfläche definiert.

Anmerkung: Das sich der Zugang der Studierenden vom Zugang der Hochschulmitarbeiter unterscheiden könnte/sollte, kann hier kritisch hinterfragt werden, ob wirklich alle anderen Use Cases den Use Case „Nutzer anmelden“ nutzen. Dieser Gedanke, der z. B. genau zu diesem Entwicklungszeitpunkt auftreten kann, stößt eventuell eine neue Iteration in der Use Case-Modellierung an.

Aus der Aktion „Eingegebene Daten prüfen“ kann abgeleitet werden:

ANF 2: Überprüfung der Anmeldung:

Nach der Eingabe der Daten Name, Rolle und Passwort muss das System in der Nutzerverwaltung überprüfen können, ob eine Anmeldung erlaubt ist.

„Nutzerverwaltung“ wird mit der Art der Prüfungsmöglichkeit in das Glossar aufgenommen.

Aus der Aktion „Fehleingabe protokollieren“ kann abgeleitet werden:

ANF 3: Protokollierung fehlerhafter Anmeldungen

Nach einer gescheiterten Prüfung in der Nutzerverwaltung muss das System die gescheiterte Anmeldung in der Form (Name, Rolle, Passwort, Datum) auf dem Server XY in einer Datei protokollieren.

ANF 4: Herstellung des Ausgangszustands

Nachdem der Nutzer das System verlassen hat, nach einem Start des Systems und nach einer gescheiterten Anmeldung, muss das System zur Anmeldungsmöglichkeit im Ausgangszustand zurückkehren.

Man beachte, dass aus ANF 3 die technische Anforderung kommt, dass die Systeme grundsätzlich eine Verbindung zum Server XY aufnehmen können. Was passiert, wenn diese Verbindung nicht existiert, ist nicht spezifiziert. (Läuft die Anwendung nur, wenn auch der Server XY läuft, handelt es sich nicht um ein Spezifikationsloch.)

ANF 4 ist so formuliert, dass sie aus mehreren Aktionen abgeleitet werden kann, es ist u. a. eine Aufgabe der Strukturierung der Anforderungen (z.B. der Zuordnungen von Attributen, wie „Systemterminierung“) sicher zu stellen, dass eine solche Anforderung nur einmal im System steht.

Aus der Aktion „Anmeldung als erfolgreich zurückmelden“ kann abgeleitet werden:

ANF 5: Erfolgreiche Systemanmeldung verwalten

Nach einer erfolgreichen Anmeldung muss das System die Information über den angemeldeten Nutzer mit seiner Rolle intern speichern und die rollenspezifische Funktionalität starten.

Die formulierte Anforderung konkretisiert den Titel der Aktion und fordert, dass die erfolgte Anmeldung zwischenzeitlich gespeichert wird. An dieser Stelle wurde „implizites IT-Know-how“ angewandt, da diese Information höchstwahrscheinlich bei Synchronisationen auf den Datenbeständen hilfreich ist.

Der Begriff „rollenspezifische Funktionalität“ ist ein Platzhalter, der als Formulierung in einer Anforderung als sehr kritisch eingestuft werden muss. Der Begriff muss entweder im Glossar oder in einer weiteren Iteration über den Anforderungen konkretisiert werden. Ebenfalls ist „intern speichern“ z. B. im Glossar wie folgt zu konkretisieren: „Intern gespeicherte Daten stehen der Applikationsausführung für einen Nutzer in einer bestimmten Rolle bis zu dem Zeitpunkt schreibend und lesend zur Verfügung, bis sich der Nutzer abmeldet.“.

Zum Aktivitätsdiagramm stellt sich die Frage, ob „Fehleingabe protokollieren“ nicht auch zu einem Endzustand führen kann/soll/muss. Diese Frage kommt insbesondere durch die ANF 4 auf, da diese einen allgemeinen „Endzustand“ beschreibt. Da sich inhaltlich kein Unterschied ergibt, weil die Anmeldung in diesem System immer am Anfang passiert, ist die Modellierung hier egal. Eventuell ist die gewählte Modellierungsart einfacher zu dokumentieren, da die Nachbedingung des UC „Nutzer mit seiner Rolle angemeldet“ kompakt formuliert werden kann.

2)

a) Grundsätzlich ist die Aufgabe, einen Text, der von einer Person geschrieben wurde, die Anforderungen nicht systematisch erstellt, in qualitativ hochwertige Anforderungen zu verwandeln, nicht trivial. Das folgende Beispiel zeigt, dass häufig der Ansatz, Sätze eins-zu-eins zu transformieren, nicht klappt.

Das System soll zur Verwaltung der komplexen Projektstrukturen in unserer Firma dienen, dabei ist es interessant, wer in welchem Projekt arbeitet, aber auch zu welchem Bereich ein Projekt gehört. Bereiche und Projekte werden von wichtigen Mitarbeitern geleitet, wobei es sich als nicht wünschenswert herausgestellt hat, dass Bereichsleiter auch Projektleiter sind.

Die ersten Sätze sind einleitend und haben erklärenden Charakter und sind hilfreich für das Glossar. Diese Informationen können nur schwer in Anforderungen umgesetzt werden. Es ist sinnvoller, diese Informationen in die klarer ersichtlichen funktionalen Anforderungen einfließen zu lassen.

Ich könnte mir vorstellen, dass bei der Eingabe eines neuen Projektes, bei dem der Projektleiter ein Bereichsleiter ist, ein trauriger Smiley erscheint und ein Text aufpoppt „das wollen wir doch nicht mehr".

Anf. 1: Im Ausgangszustand muss das System dem Nutzer die Möglichkeit bieten, einen Menü-Punkt zur Eingabe neuer Projekte auszuwählen.

Anf. 2: Nachdem der Nutzer sich für die Eingabe eines neuen Projekts entschieden hat, muss das System dem Nutzer die Möglichkeit bieten, Informationen über ein neues Projekt (<Nutzer nach konkreten Daten fragen, einige im Text erwähnt>) in einer Eingabemaske einzugeben.

Anf. 3: Nachdem die Eingabemaske für neue Projektinformationen eingeblendet wurde, muss das System dem Nutzer die Möglichkeit bieten, einen Knopf zur Aufforderung zur Übernahme der Daten zu drücken.

Anf. 4: Nachdem der Nutzer das System zur Übernahme neuer Projektinformationen aufgefordert hat, muss das System den Sachverhalt, ob der angegebene Projektleiter bereits Bereichsleiter ist, prüfen.

Anf. 5: Ist ein für ein neues Projekt angegebener Projektleiter bereits Bereichsleiter, muss das System eine Fehlermeldung „Bereichsleiter darf nicht Projektleiter sein" ausgeben.

Anmerkung: In Anf. 5 könnte auch auf den Smiley eingegangen werden. Es wird hier von einer Menü-Steuerung ausgegangen, was formal schon eine DesignEntscheidung ist. Alternativ könnte man über die „Auswahlmöglichkeit von Funktionalitäten“ schreiben.

Anf. 6: Ist ein für ein neues Projekt angegebener Projektleiter nicht bereits Bereichsleiter, muss das System die eingegebenen Daten im System speichern.

Neben dem Anlegen der Informationen muss man natürlich schauen können, wie viele Projekte in welcher Abteilung sind und wo wie viele Mitarbeiter in Projekten sind. Dabei ist natürlich zusätzlich zu beachten, dass Mitarbeiter nur anteilig in Projekten arbeiten.

Anf. 7: Im Ausgangszustand muss das System dem Nutzer die Möglichkeit bieten, einen Menü-Punkt zur Übersicht über die Anzahl der Projekte pro Abteilung auszuwählen.

Anf. 8: Nachdem der Nutzer die Übersicht über die Anzahl der Projekte pro Abteilung ausgewählt hat, muss das System eine Tabelle aller Abteilungen mit der Anzahl der zugeordneten Projekte darstellen. <Nutzer fragen, ob wirklich nur diese Informationen gewünscht sind.>

Anf. 9: Im Ausgangszustand muss das System dem Nutzer die Möglichkeit bieten, einen Menü-Punkt zur Übersicht über die Anzahl der Mitarbeiter pro Projekt auszuwählen.

Anf. 10: Nachdem der Nutzer die Übersicht über die Anzahl der Mitarbeiter pro Projekt ausgewählt hat, muss das System eine Tabelle aller Projekte mit der Anzahl der zugeordneten Mitarbeiter und der Mitarbeiteranzahl unter Berücksichtigung ihrer anteiligen Arbeit in Projekten darstellen.

Insgesamt sollte man alle wichtigen Informationen über Projekte leicht abfragen können.

Anmerkung: Eine solche Zeile deutet auf die Angst des Erstellers hin, Anforderungen zu vergessen. Hier hilft nur eine Nachfrage beim Ersteller.

Natürlich gehört auch der finanzielle Bearbeitungsanteil zum System, da jeder Bereichsleiter nur ein bestimmtes Budget hat, das durch neue Projekte oder aktualisierte Budgets von alten Projekten nicht überschritten werden darf. Aber diese Anpassungsmöglichkeiten sind ja für eine solche Verwaltungssoftware selbstverständlich.

Anf. 11: Nachdem der Nutzer das System zur Übernahme neuer Projektinformationen aufgefordert hat, muss das System den Sachverhalt, ob das Budget des Bereichsleiters durch das neue Projekt nicht überschritten wird, prüfen.

Anf. 12: Wird durch ein neu eingegebenes Projekt das Budget eines Bereichsleiters überschritten, muss das System eine Fehlermeldung „Budget des Bereichsleiters überschritten“ ausgeben.

Die Anforderung 6 muss ergänzt werden (weiterhin ist es sinnvoll, die Anforderungen nach der Erstellung neu zu sortieren) :

Anf. 6: Ist ein für ein neues Projekt angegebener Projektleiter nicht bereits Bereichsleiter *und wird das Budget des Bereichsleiters nicht überschritten*, muss das System die eingegebenen Daten im System speichern.

Der letzte Satz des Textes ist weitere Verzweiflung des Erstellers, die zu hinterfragen ist.

b)

Technische Anforderung: Das System muss auf dem vorhandenen Unternehmensserver laufen.

Anforderungen an die Benutzungsschnittstelle: Die Oberfläche berücksichtigt die im Dokument „Einheitliches Erscheinungbild unserer Unternehmenssoftware“ beschriebenen Design-Regeln.

Anforderungen an die Dienstqualität: Anfragen an das System dauern für den zum Test vom Kunden zur Verfügung gestellten Datensatz maximal drei Sekunden bis zur vollständigen Ergebnisanzeige.

Anforderungen an sonstige Lieferbestandteile: Das System wird mit einem Installations- und Benutzungshandbuch ausgeliefert.

Anforderungen an die Durchführung der Entwicklung und Einführung: Die Entwicklung muss nach dem V-Modell XT erfolgen, die Projektdurchführung ist für den Kunden sichtbar zu dokumentieren.

Rechtlich-vertragliche Anforderungen: Die Zahlung der restlichen 60% des Kaufpreises erfolgt nach der Abnahme durch den Kunden.

zu 3) a)

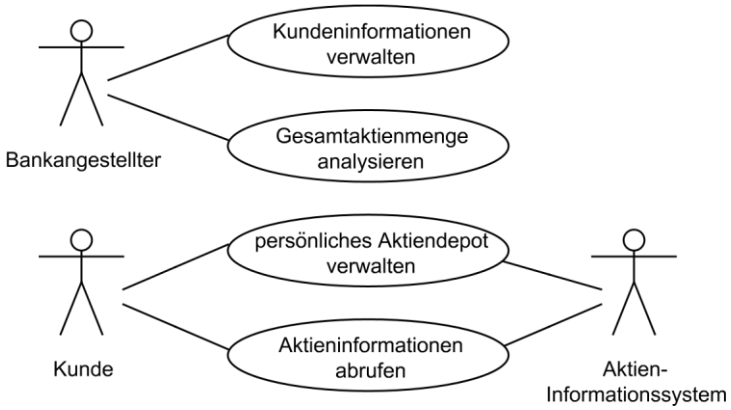


Abb. 9: Use Case Diagramm zu Aufgabe 3a)

Da es einige Übersichten für Bankangestellte und einige andere Übersichten für Kunden gibt, können diese jeweils zu einem getrennten Use Case zusammengefasst werden. Dies kann u. a. damit begründet werden, dass viele dieser Analysen später kaum getrennt genutzt (auch implementiert) werden.

zu b)

Der letzte Use Case soll nur andeuten, dass die Systempflege ein eigener Use Case sein kann. Im konkreten Fall müsste die Software warten, bis sie von der Hardware die Information bekommt, dass diese funktionsfähig ist.

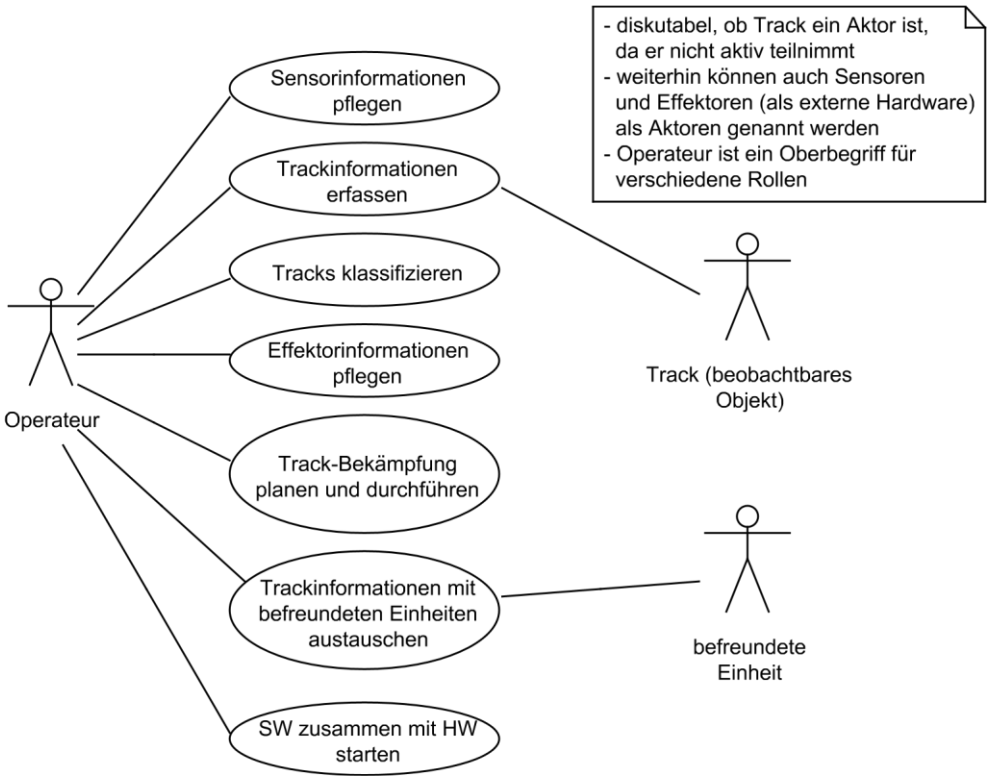


Abb. 10: Use Case Diagramm zu Aufgabe 3b)

Anmerkung zur Aufgabe 3: Sinn der Aufgabe ist es, ein gewisses Gefühl für Use Cases zu entwickeln. Ein Unbehagen, ob die Use Cases zu grob oder zu detailliert sind, ist berechtigt. Bei der Überarbeitung ist hauptsächlich zu beachten, dass die resultierenden Use Cases möglichst unabhängig voneinander sind. Ein „Verwaltungs-Use-Case“ darf z. B. auch in die Use Cases „erstellen“, „bearbeiten“ und ggfls. „löschen“ aufgeteilt werden.

Lösungen zu Kapitel 5

zu 1)

Hinweis: es sind stark abweichende Lösungen denkbar, da Mitarbeiterverwaltung entweder alles steuert und Mitarbeiter eher passiv ist oder Mitarbeiterverwaltung alle Informationen an Mitarbeiter durchschleust.

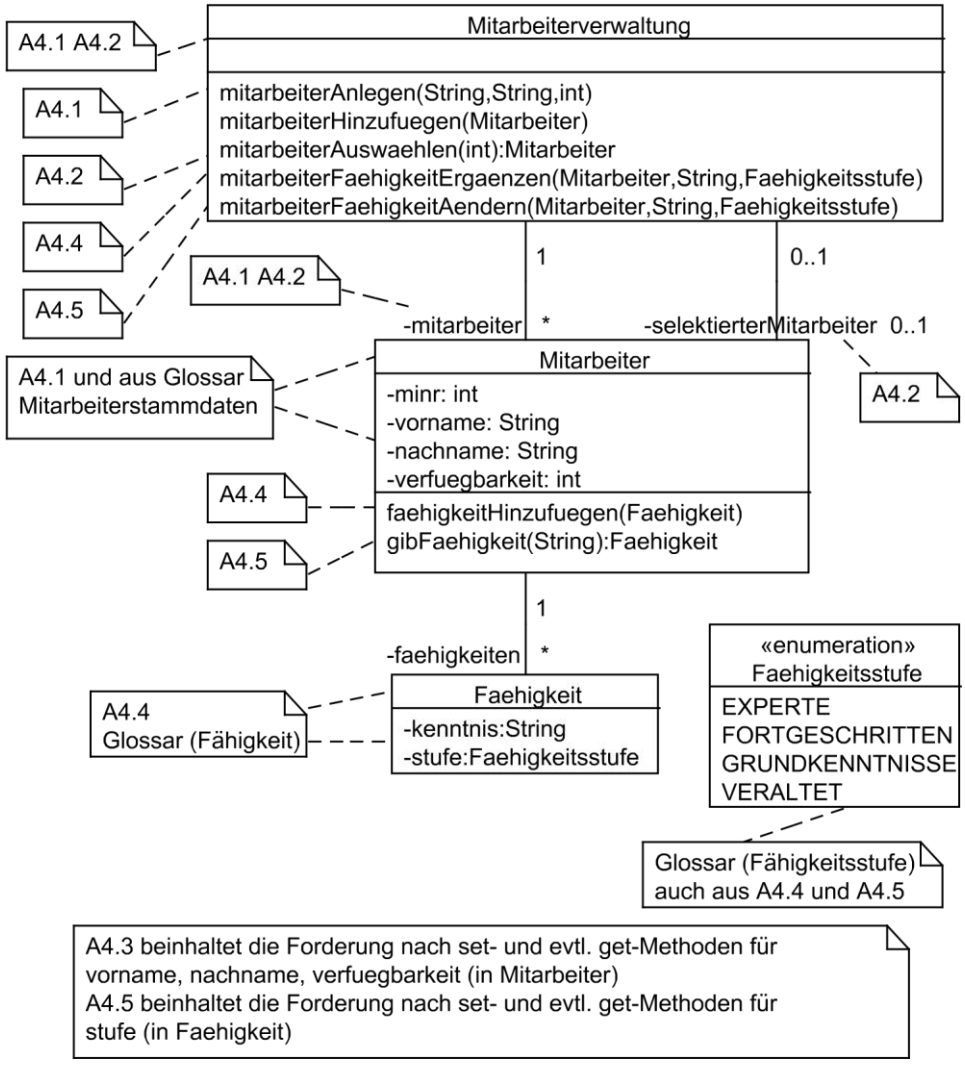


Abb. 11: Klassendiagramm zu Aufgabe 1

Die Methode „mitarbeiterHinzufuegen()“ wird hier erst bei der Sequenzdiagrammvalidierung gefunden.

Anmerkung: Bei folgenden Diagrammen wäre es auch denkbar, die Aufgaben dem Mitarbeiter zu übertragen, d.h. er erhält die Teilinformationen über die Fähigkeit und ändert seine Fähigkeit selber. Weiterhin könnte genauso gut dire Methode mitarbeiterFaehigkeitErgaenzen statt des Mitarbeiterobjekts die Id übergeben werden.

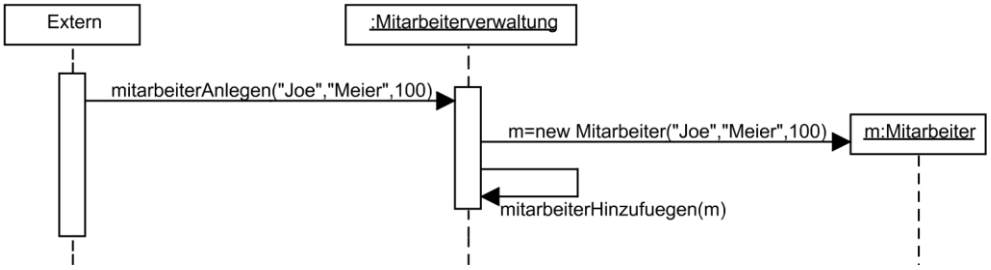


Abb. 12: Aufgabe 1 „Mitarbeiter anlegen“

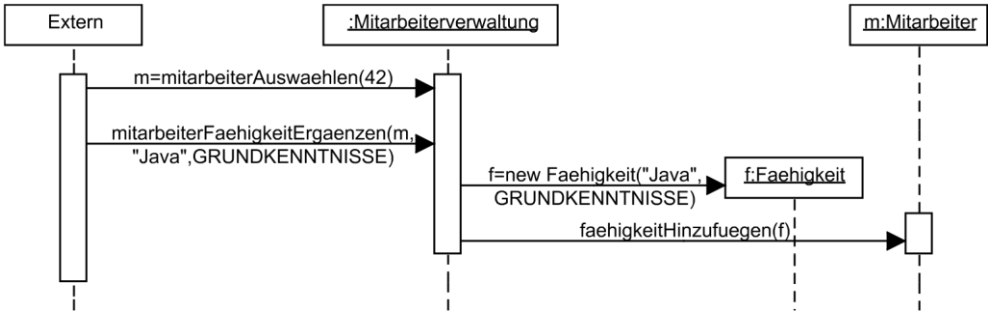


Abb. 13: Aufgabe 1 „neue Fähigkeit hinzufügen“

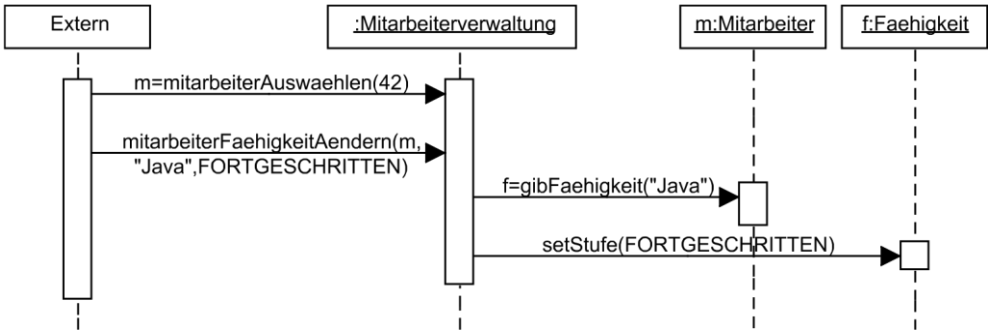


Abb. 14: Aufgabe 1 „Fähigkeit ändern“

zu 2)

Eine Klasse Klassendiagrammbearbeitung mit Verbindung zu den Klassendiagrammelementen ist eine sinnvolle Ergänzung.

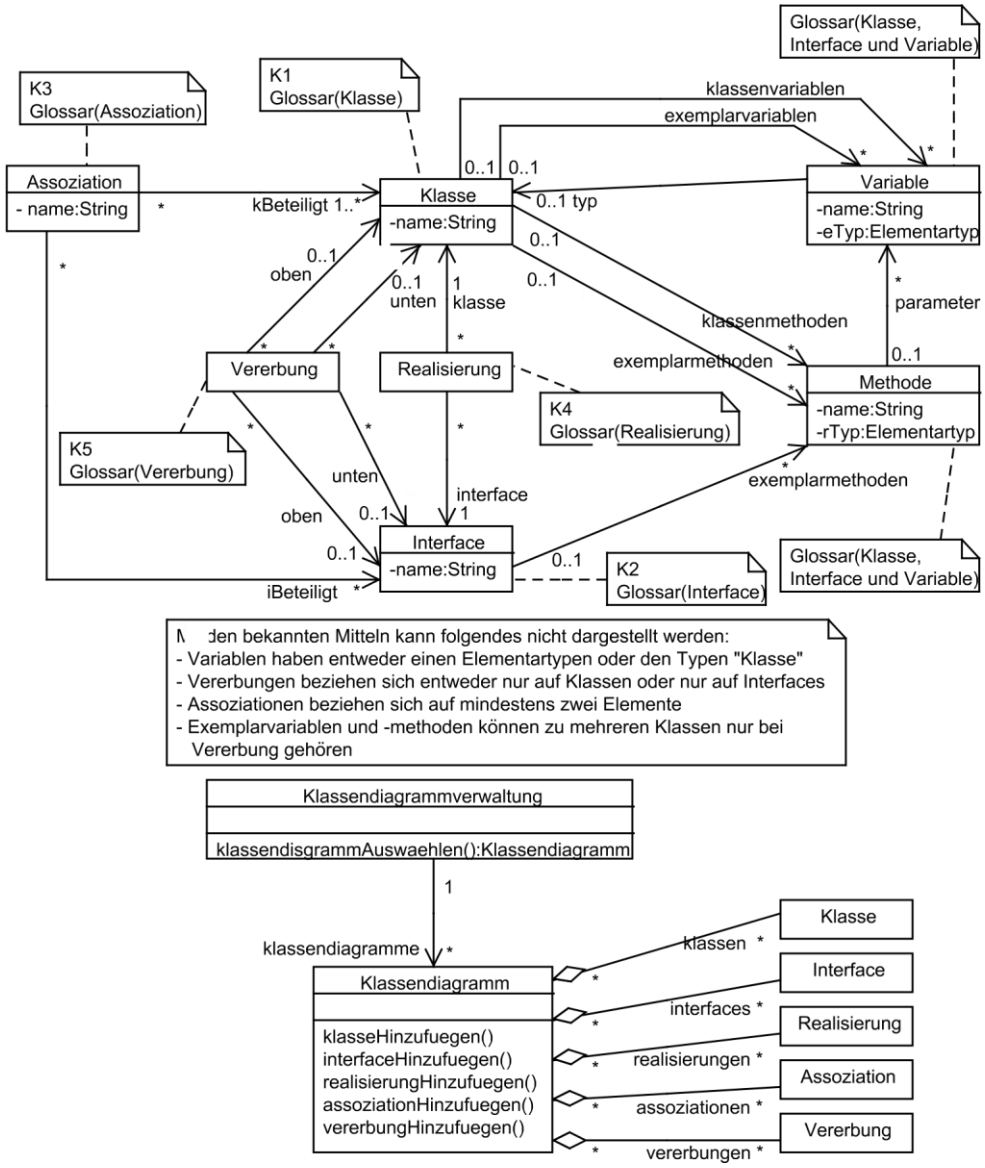


Abb. 15: Lösung Aufgabe 2

Die Anmerkungen im Kommentarkasten könnten durch OCL-Constraints beschrieben werden. Ein solcher Ansatz wird von der OMG bei der Formalisierung von Klassendiagrammen genutzt.

Weiterhin kann als Übungsaufgabe überlegt werden, wie Sichtbarkeiten und das Überschreiben von Methoden modelliert werden können.

Lösungen zu Kapitel 6

zu 1)

```
package kapitel06_Aufgabe1;

public class Main {

  public static void main(String[] args) {
    Reiseabrechnung r= new Reiseabrechnung(42,4242);
    r.belegHinzufuegen(new Ticket("Anreise"
        ,2300,"Flug","Frankfurt","Honolulu"));
    r.belegHinzufuegen(new Hotel("Unterbringung"
        ,3000,"Palm Resort","1.11.06","15.11.06"));
    r.belegHinzufuegen((new Ticket("Rueckreise"
        ,1050,"Flug","Honolulu","Hahn")));
    r.ausgeben();
    System.out.println("Gesamtkosten: "+r.gesamtsumme());
  }
}
```

```
package kapitel06_Aufgabe1;

import java.util.ArrayList;
import java.util.List;

public class Reiseabrechnung {
  private int mitarbeiterNr;
  private int reiseNr;
  private List<Beleg> belege;

  public Reiseabrechnung(int mitarbeiterNr, int reiseNr) {
    this.mitarbeiterNr = mitarbeiterNr;
    this.reiseNr = reiseNr;
    this.belege = new ArrayList<Beleg>();
  }

  public void belegHinzufuegen(Beleg b) {
    this.belege.add(b);
  }
```

```
  public void ausgeben() {
    System.out.println("Reise Nr." + reiseNr
        + " von Mitarbeiter " + mitarbeiterNr);
    for (Beleg b : this.belege)
      b.ausgeben();
  }

  public int gesamtsumme() {
    int ergebnis = 0;
    for (Beleg b : this.belege)
      ergebnis += b.getKosten();
    return ergebnis;
  }
}
```

```
package kapitel06_Aufgabe1;

public class Beleg {

  private String beschreibung;
  private int kosten;

  public Beleg(String beschreibung, int kosten) {
    this.beschreibung = beschreibung;
    this.kosten = kosten;
  }

  public int getKosten() {
    return this.kosten;
  }

  public void ausgeben(){
    System.out.print(beschreibung+" "+ this.kosten);
  }
}
```

```java
package kapitel06_Aufgabe1;

public class Hotel extends Beleg {

  private String hotelname;
  private String von;
  private String bis;

  public Hotel(String beschreibung, int kosten
      , String hotelname, String von, String bis) {
    super(beschreibung, kosten);
    this.hotelname = hotelname;
    this.von = von;
    this.bis = bis;
  }

  @Override
  public void ausgeben(){
    super.ausgeben();
    System.out.println(" [Details:" + this.hotelname
        + ", von:" + this.von + ", bis:" + this.bis + "]");
  }
}
```

```java
package kapitel06_Aufgabe1;

public class Ticket extends Beleg {

  private String typ;
  private String von;
  private String nach;

  public Ticket(String beschreibung, int kosten
      , String typ, String von, String nach) {
    super(beschreibung, kosten);
    this.typ = typ;
    this.von = von;
    this.nach = nach;
  }
```

```
  @Override
  public void ausgeben(){
    super.ausgeben();
    System.out.println(" [Details:" + this.typ
        + ", von:" + this.von + ", nach:" + this.nach + "]");
  }
}
```

zu 2) a)

```
package kapitel06_Aufgabe2a;

import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

import kapitel06_Aufgabe2a.projekte.Projekt;
import kapitel06_Aufgabe2a.projektmitarbeiter.Mitarbeiter;

public class Main {

  private List<Projekt> projekte= new ArrayList<Projekt>();
  private List<Mitarbeiter> mitarbeiter
      = new ArrayList<Mitarbeiter>();

  public Main(){
    int ein=-1;
    while(ein!=0){
      System.out.println("Was wollen Sie machen?\n"
        +"(0) Programm beenden\n"
        +"(1) neuen Mitarbeiter anlegen\n"
        +"(2) neues Projekt anlegen\n"
        +"(3) einem Mitarbeiter ein Projekt zuordnen\n"
        +"(4) einem Projekt einen Mitarbeiter zuordnen\n"
        +"(5) einen Mitarbeiter von einem Projekt entfernen\n"
        +"(6) ein Projekt von einem Mitarbeiter entfernen\n"
        +"(7) Mitarbeiter ausgeben\n"
        +"(8) Projekte ausgeben");
      ein=new Scanner(System.in).nextInt();
```

```
        switch(ein){
        case 1:{
          this.mitarbeiter.add(new Mitarbeiter());
          break;
        }
        case 2:{
          this.projekte.add(new Projekt());
          break;
        }
        case 3:{
          Mitarbeiter mi=zuMinr();
          Projekt pr=zuPrnr();
          if(mi!=null && pr!=null)
            pr.bearbeiterHinzufuegen(mi);
          break;
        }
        case 4:{
          Projekt pr=zuPrnr();
          Mitarbeiter mi=zuMinr();
          if(mi!=null && pr!=null)
            mi.projektHinzufuegen(pr);
          break;
        }
        case 5:{
          Mitarbeiter mi=zuMinr();
          Projekt pr=zuPrnr();
          if(mi!=null && pr!=null)
            pr.bearbeiterLoeschen(mi);
          break;
        }
        case 6:{
          Projekt pr=zuPrnr();
          Mitarbeiter mi=zuMinr();
          if(mi!=null && pr!=null)
            mi.projektLoeschen(pr);
          break;
        }
        case 7:{
          System.out.println("Mitarbeiter:");
          for(Mitarbeiter m: this.mitarbeiter)
```

```
          System.out.println(" "+m);
        break;
      }
      case 8:{
        System.out.println("Projekte:");
        for(Projekt p: this.projekte)
          System.out.println(" "+p);
        break;
      }
      }
    }
  }

  public Mitarbeiter zuMinr(){
    System.out.print("Welcher Mitarbeiter (MiNr)? ");
    int minr=new Scanner(System.in).nextInt();
    Mitarbeiter ergebnis=null;
    for(Mitarbeiter m: this.mitarbeiter)
      if (m.getMinr()==minr)
        ergebnis=m;
    return ergebnis;
  }

  public Projekt zuPrnr(){
    System.out.print("Welches Projekt (PrNr)? ");
    int prnr=new Scanner(System.in).nextInt();
    Projekt ergebnis=null;
    for(Projekt p: this.projekte)
      if (p.getPrnr()==prnr)
        ergebnis=p;
    return ergebnis;
  }

  public static void main(String[] args) {
    new Main();
  }
}
```

```java
package kapitel06_Aufgabe2a.projektmitarbeiter;

import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

import kapitel06_Aufgabe2a.projekte.Projekt;

public class Mitarbeiter {

  private int minr;
  private String name;
  private List<Projekt> projekte;

  public int getMinr() {
    return this.minr;
  }

  public void setMinr(int minr) {
    this.minr = minr;
  }

  public String getName() {
    return this.name;
  }

  public void setName(String name) {
    this.name = name;
  }

  public Mitarbeiter(){
    Scanner eingabe= new Scanner(System.in);
    System.out.print("Mitarbeiternummer: ");
    this.minr=eingabe.nextInt();
    System.out.print("Name: ");
    this.name=eingabe.next();
    this.projekte= new ArrayList<Projekt>();
  }
```

```
  public Mitarbeiter(int minr, String name) {
    this.minr = minr;
    this.name = name;
    this.projekte= new ArrayList<Projekt>();
  }

  public void projektHinzufuegen (Projekt p){
    this.projekte.add(p); //alternativ dies nur,
                          // wenn p in projekte nicht existiert
    p.bearbeiterHinzufuegen2(this);
  }

  public void projektHinzufuegen2 (Projekt p){
    this.projekte.add(p);
  }

  public void projektLoeschen (Projekt p){
    this.projekte.remove(p); //alternativ dies nur,
                             // wenn p in projekte existiert
    p.bearbeiterLoeschen2(this);
  }

  public void projektLoeschen2 (Projekt p){
    this.projekte.remove(p);
  }

  @Override
  public String toString(){
    StringBuffer s= new StringBuffer(this.minr+": "
        + this.name+ "[ ");
    for(Projekt p:projekte){
      s.append(p.getName());
      s.append("(");
      s.append(""+p.getPrnr());
      s.append(") ");
    }
    s.append("]");
    return s.toString();
  }
}
```

```java
package kapitel06_Aufgabe2a.projekte;

import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

import kapitel06_Aufgabe2a.projektmitarbeiter.Mitarbeiter;

public class Projekt {

  private int prnr;
  private String name = "";
  private List<Mitarbeiter> bearbeiter;

  public int getPrnr() {
    return this.prnr;
  }

  public void setPrnr(int prnr) {
    this.prnr = prnr;
  }

  public String getName() {
    return this.name;
  }

  public void setName(String name) {
    this.name = name;
  }

  public Projekt(){
    Scanner eingabe= new Scanner(System.in);
    System.out.print("Projektnummer: ");
    this.prnr=eingabe.nextInt();
    System.out.print("Projektname: ");
    this.name=eingabe.next();
    this.bearbeiter= new ArrayList<Mitarbeiter>();
  }
```

```
  public Projekt(int prnr, String name) {
    super();
    this.prnr = prnr;
    this.name = name;
    this.bearbeiter= new ArrayList<Mitarbeiter>();
  }

  public void bearbeiterHinzufuegen (Mitarbeiter m){
    this.bearbeiter.add(m);
    m.projektHinzufuegen2(this);
  }

  public void bearbeiterHinzufuegen2 (Mitarbeiter m){
    this.bearbeiter.add(m);
  }

  public void bearbeiterLoeschen (Mitarbeiter m){
    this.bearbeiter.remove(m);
    m.projektLoeschen2(this);
  }

  public void bearbeiterLoeschen2 (Mitarbeiter m){
    this.bearbeiter.remove(m);
  }

  @Override
  public String toString(){
    StringBuffer s= new StringBuffer(this.prnr+": "
        + this.name+ "[ ");
    for(Mitarbeiter m: this.bearbeiter){
      s.append(m.getName());
      s.append("(");
      s.append(""+m.getMinr());
      s.append(") ");
    }
    s.append("]");
    return s.toString();
  }
}
```

zu b)

```
package kapitel06_Aufgabe2b.projekte;

public interface MitarbeiterInterface {
  public int getMinr();
  public String getName();
  public void projektHinzufuegen2(Projekt p);
  public void projektLoeschen2(Projekt p);
}
```

```
package kapitel06_Aufgabe2b.projekte;

import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

public class Projekt {

  private int prnr;
  private String name = "";
  private List<MitarbeiterInterface> bearbeiter;

  public int getPrnr() {
    return this.prnr;
  }

  public void setPrnr(int prnr) {
    this.prnr = prnr;
  }

  public String getName() {
    return this.name;
  }

  public void setName(String name) {
    this.name = name;
```

```
  }

  public Projekt(){
    Scanner eingabe= new Scanner(System.in);
    System.out.print("Projektnummer: ");
    this.prnr=eingabe.nextInt();
    System.out.print("Projektname: ");
    this.name=eingabe.next();
    this.bearbeiter= new ArrayList<MitarbeiterInterface>();
  }

  public Projekt(int prnr, String name) {
    super();
    this.prnr = prnr;
    this.name = name;
    this.bearbeiter= new ArrayList<MitarbeiterInterface>();
  }

  public void bearbeiterHinzufuegen (MitarbeiterInterface m){
    this.bearbeiter.add(m);
    m.projektHinzufuegen2(this);
  }

  public void bearbeiterHinzufuegen2 (MitarbeiterInterface m){
    this.bearbeiter.add(m);
  }

  public void bearbeiterLoeschen (MitarbeiterInterface m){
    this.bearbeiter.remove(m);
    m.projektLoeschen2(this);
  }

  public void bearbeiterLoeschen2 (MitarbeiterInterface m){
    this.bearbeiter.remove(m);
  }

  @Override
  public String toString(){
    StringBuffer s= new StringBuffer(this.prnr+": "
        + this.name+ "[ ");
```

```
    for(MitarbeiterInterface m: this.bearbeiter){
      s.append(m.getName());
      s.append("(");
      s.append(""+m.getMinr());
      s.append(") ");
    }
    s.append("]");
    return s.toString();
  }

}
```

in Mitarbeiter ändert sich nur der Anfang:

```
public class Mitarbeiter implements MitarbeiterInterface {
```

Die Main-Klasse bleibt unverändert.

Lösungen zu Kapitel 7

zu 1) Anmerkung: Mit dem Wissen aus Kapitel 8 könnte das Programm unter Nutzung des State-Pattern noch etwas schöner gestaltet werden.

```
package kapitel07_Aufgabe1;

public enum Zustand {
  GEPLATZT, FETT, FIT, HUNGRIG, ABGEMAGERT, VERHUNGERT,
  WEGGELAUFEN,DEPREMIERT,TRAURIG,FROH,GLUECKLICH,
  LEBENDIG;
}
```

```
package kapitel07_Aufgabe1;

public class Wauzi {

  Zustand essen;
  Zustand spielen;
  Zustand gesamt;
```

```
  public Wauzi(){
    this.essen = Zustand.FIT;
    this.spielen = Zustand.FROH;
    this.gesamt = Zustand.LEBENDIG;
  }

  public void spielen(){
    switch(this.essen){
    case FETT:
      this.essen = Zustand.FIT;
      break;
    case FIT:
      this.essen = Zustand.HUNGRIG;
      break;
    case HUNGRIG:
      this.essen = Zustand.ABGEMAGERT;
      break;
    case ABGEMAGERT:
      this.gesamt = Zustand.VERHUNGERT;
      break;
    }
    switch(spielen){
    case DEPREMIERT:
      this.spielen = Zustand.TRAURIG;
      break;
    case TRAURIG:
      this.spielen = Zustand.FROH;
      break;
    case FROH:
      this.spielen = Zustand.GLUECKLICH;
      break;
    }
    this.auswerten();
  }

  public void gassi(){
    switch(essen){
    case FETT:
      this.essen = Zustand.FIT;
      break;
```

```
    case FIT:
      this.essen = Zustand.HUNGRIG;
      break;
    case HUNGRIG:
      this.essen = Zustand.ABGEMAGERT;
      break;
    case ABGEMAGERT:
      this.gesamt = Zustand.VERHUNGERT;
      break;
    }
    this.auswerten();
  }

  public void fuettern(){
    switch(essen){
    case FETT:
      this.gesamt = Zustand.GEPLATZT;
      break;
    case FIT:
      this.essen = Zustand.FETT;
      break;
    case HUNGRIG:
      this.essen = Zustand.FIT;
      break;
    case ABGEMAGERT:
      this.essen = Zustand.HUNGRIG;
      break;
    }
    this.auswerten();
  }

  public void ignorieren(){

    switch(spielen){
    case DEPREMIERT:
      this.gesamt = Zustand.WEGGELAUFEN;
      break;
    case TRAURIG:
      this.spielen = Zustand.DEPREMIERT;
      break;
```

```
    case FROH:
      this.spielen = Zustand.DEPREMIERT;
      break;
    case GLUECKLICH:
      this.spielen = Zustand.FROH;
      break;
    }
    this.auswerten();
  }

  private void auswerten(){
    if(this.lebt())
      System.out.println("Der Hund ist " + this.essen
          + " und " + this.spielen + ".");
    else
      System.out.println("Der Hund ist " + this.gesamt
                      + ".");
  }

  public boolean lebt(){
    return this.gesamt == Zustand.LEBENDIG;
  }
}


package kapitel07_Aufgabe1;

import java.util.Scanner;

public class Steuerung {
  private Wauzi wauzi;

  public Steuerung() {
    int eingabe = -1;
    this.wauzi = new Wauzi();
    while (eingabe != 0 && this.wauzi.lebt()) {
      System.out.println("Was wollen Sie mit Wauzi machen?\n"
          + " (0) verschenken (Programm beenden)\n"
          + " (1) mit ihm spielen\n"
          + " (2) mit ihm Gassi gehen\n"
```

```
            + " (3) ihn fuettern\n"
            + " (4) ihn ignorieren und lernen");
        eingabe = new Scanner(System.in).nextInt();
        switch (eingabe) {
          case 1:
            this.wauzi.spielen();
            break;
          case 2:
            this.wauzi.gassi();
            break;
          case 3:
            this.wauzi.fuettern();
            break;
          case 4:
            this.wauzi.ignorieren();
            break;
        }
      }
    }

    public static void main(String[] args) {
      new Steuerung();
    }
}
```

zu 2)

a) Das Klassendiagramm gibt genau vor, welche Klassen es mit welchen Exemplarvariablen und Exemplarmethoden es gibt. Die Assoziationen legen fest, wie viele Objekte einer Klasse Objekten einer anderen Klasse zugeordnet werden. So ist festgelegt, dass jedes Projekt immer genau einen leiter hat.

zu b)

i)

```
context Mitarbeiter
inv nichtInRente: self.geburtsjahr<1940
```

ii)

```
context Qualifikation
inv wertebereich: self.stufe>=0 and self.stufe <=4
```

iii)

```
   context Projekt::gibLeiternamen():String
   post: result=self.leiter.name
```

iv)

```
context Projekt::gibMitgliederanzahl():Integer
post: result=self.mitglieder->size()
```

v)

```
context Projekt
inv maximaleMitgliederzahl: self.mitglieder->size()<=10
```

vi)

```
context Mitarbeiter
inv jederKannJava: self.quali->exists(q | q.art='Java')
```

vii)

```
context Projekt
inv leiterHatQuali:
   self.leiter.quali->exists(q | q.art='Leitung' and q.stufe>2)
```

viii)

```
context p:Projekt
inv leiterNichtMitglied:
not p.mitglieder->exists(m | m=p.leiter)
```

ix)

```
context p:Projekt

inv qsInGroesseremProjekt:

   p.mitglieder.size()>3 implies
     p.mitglieder->exists(m |m.quali
                      ->exists(q | q.art='QS' and q.stufe>2))
```

Lösungen zu Kapitel 8

zu 1)

```
package kapitel08_Aufgabe1;

import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

public class Main {

  private Model model;
  private List<Controller> controller;

  public Main() {
    this.model = new Model(1);
    this.controller = new ArrayList<Controller>();
    int eingabe = -1;
    while (eingabe != 0) {
      System.out.println("Was wollen Sie?\n"
          + " (0) Programm beenden\n"
          + " (1) neuen View erstellen\n"
          + " (2) neuen Controller erstellen\n"
          + " (3) Controller zum aendern auswaehlen");
      eingabe = new Scanner(System.in).nextInt();
      switch (eingabe) {
        case 1:
          this.neuerView();
          break;
        case 2:
```

```
          this.neuerController();
          break;
        case 3:
          this.controllerNutzen();
          break;
      }
    }
  }

  public void neuerView() {
    Scanner sc = new Scanner(System.in);
    sc.useDelimiter("\n");
    System.out.print("Welches Ausgabezeichen? ");
    new View(this.model, sc.next().charAt(0));
  }

  public void neuerController() {
    Scanner sc = new Scanner(System.in);
    sc.useDelimiter("\n");
    System.out.print("Welche Eingabeaufforderung? ");
    String aufforderung = sc.next();
    // letztes Zeichen /n l?schen
    aufforderung = aufforderung.substring(0,
        aufforderung.length() - 1);
    this.controller.add(
        new Controller(this.model, aufforderung));
  }

  public void controllerNutzen() {
    if (!controller.isEmpty()) {
      System.out.print("Welchen der Controller (1-"
          + this.controller.size() + ")? ");
      this.controller
          .get((new Scanner(System.in).nextInt()) - 1)
          .eingabeSteuern();
    }
  }

  public static void main(String[] args) {
    new Main();
```

```
  }
}
```

```
package kapitel08_Aufgabe1;

public interface ModelListener {
  public void wertaenderung();
}
```

```
package kapitel08_Aufgabe1;

import java.util.ArrayList;
import java.util.List;

public class Model {
  private int wert;
  private List<ModelListener> listener;

  public Model(int wert) {
    this.wert = wert;
    this.listener = new ArrayList<ModelListener>();
  }

  public void listenerHinzufuegen(ModelListener m) {
    this.listener.add(m);
  }

  public int getwert() {
    return wert;
  }

  public void wertAendern(int wert) {
    this.wert = wert;
    for (ModelListener m : this.listener)
      m.wertaenderung();
```

```
  }
}
```

```
package kapitel08_Aufgabe1;

public class View implements ModelListener {

  private Model model;
  private char zeichen;

  public View(Model model, char zeichen) {
    this.model = model;
    this.zeichen = zeichen;
    this.model.listenerHinzufuegen(this);
  }

  public void wertaenderung() {
    int wert = this.model.getwert();
    for (int i = 0; i < wert; i++)
      System.out.print(this.zeichen);
    System.out.println();
  }
}
```

```
package kapitel08_Aufgabe1;

import java.util.Scanner;

public class Controller {

  private Model model;
  private String aendern;

  public Controller(Model model, String aendern) {
    this.model = model;
```

```
    this.aendern = aendern;
  }

  public void eingabeSteuern() {
    System.out.print(aendern + ": ");
    this.model.wertAendern(new Scanner(System.in).nextInt());
  }
}
```

zu 2)

```
package kapitel08_Aufgabe2;

import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

public class Main {

  List<Aktie> aktien = new ArrayList<Aktie>();
  List<Haendler> haendler = new ArrayList<Haendler>();

  public Main() {
    int eingabe = -1;
    while (eingabe != 0) {
      System.out.println("Was wollen Sie?\n"
          + " (0) Programm beenden\n"
          + " (1) neue Aktie erstellen\n"
          + " (2) neuen Haendler erstellen\n"
          + " (3) Haendler interessiert an Aktie\n"
          + " (4) Aktienwert aendern");
      eingabe = new Scanner(System.in).nextInt();
      switch (eingabe) {
        case 1:
          this.neueAktie();
          break;
        case 2:
          this.neuerHaendler();
          break;
        case 3:
```

```
            this.haendlerInteresseFuerAktie();
            break;
          case 4:
            this.aktienwertAendern();
            break;
        }
      }
    }

    public void neueAktie() {
      Scanner sc = new Scanner(System.in);
      sc.useDelimiter("\n");
      System.out.print("Aktienname? ");
      String name = sc.next();
      // letztes Zeichen /n loeschen
      name = name.substring(0, name.length() - 1);
      this.aktien.add(new Aktie(name));
    }

    public void neuerHaendler() {
      Scanner sc = new Scanner(System.in);
      sc.useDelimiter("\n");
      System.out.print("Haendlername? ");
      String name = sc.next();
      name = name.substring(0, name.length() - 1);
      this.haendler.add(new Haendler(name));
    }

    private int listenelementWaehlen(List<?> l) {
      // wird nicht mit leerer Liste aufgerufen
      int ergebnis = -1;
      while (ergebnis < 0 || ergebnis >= l.size()) {
        for (int i = 0; i < l.size(); i++)
          System.out.print((i + 1) + ":" + l.get(i) + "   ");
        ergebnis = new Scanner(System.in).nextInt() - 1;
      }
      return ergebnis;
    }

    public void haendlerInteresseFuerAktie() {
```

```
    if (this.aktien.size() > 0 && this.haendler.size() > 0) {
      System.out.println("Welcher Haendler?");
      int h = this.listenelementWaehlen(haendler);
      System.out.println("Welche Aktie?");
      int a = listenelementWaehlen(aktien);
      this.haendler.get(h).neueAktie(aktien.get(a));
    }
  }

  public void aktienwertAendern() {
    if (!this.aktien.isEmpty()) {
      System.out.println("Welche Aktie?");
      int a = listenelementWaehlen(aktien);
      System.out.print("Neuer Wert: ");
      this.aktien.get(a)
                 .setWert(new Scanner(System.in).nextInt());
    }
  }

  public static void main(String[] args) {
    new Main();
  }
}
```

```
package kapitel08_Aufgabe2;

public interface HaendlerInterface {
  public void aktualisieren(String aktienname);
}
```

```
package kapitel08_Aufgabe2;

import java.util.ArrayList;
import java.util.List;

public class Haendler implements HaendlerInterface {
```

```
  private String haendlername;
  private List<Aktie> aktien = new ArrayList<Aktie>();


  public Haendler(String haendlername) {
    this.haendlername = haendlername;
  }

  public void neueAktie(Aktie a){
    this.aktien.add(a);
    a.anmelden(this);
  }

  public void aktualisieren(String aktienname) {
    System.out.println(haendlername
        + " hat neuen Wert fuer " + aktienname + ": "
        +  this.holeAktienWert(aktienname));
  }

  private int holeAktienWert(String aktienname){
    for(Aktie a: this.aktien)
      if(a.getAktienname().equals(aktienname))
        return a.getWert();
    //nie erreichen
    assert(false);
    return 0;
  }

  @Override
  public String toString(){
    return this.haendlername;
  }
}



package kapitel08_Aufgabe2;

public class Aktie extends Aktienverwaltung {
```

```
  private int wert=42;

  public Aktie(String aktienname){
    super(aktienname);
  }

  public int getWert() {
    return wert;
  }

  public void setWert(int wert) {
    this.wert = wert;
    super.benachrichtigen();
  }

  @Override
  public String toString(){
    return this.getAktienname();
  }
}
```

```
package kapitel08_Aufgabe2;

import java.util.ArrayList;
import java.util.List;

public class Aktienverwaltung {
  private String aktienname;
  private List<HaendlerInterface> haendler
  = new ArrayList<HaendlerInterface>();

  protected Aktienverwaltung(String aktienname) {
    this.aktienname = aktienname;
  }

  public void anmelden(HaendlerInterface h) {
    this.haendler.add(h);
```

```
  }

  public void benachrichtigen() {
    for (HaendlerInterface h : this.haendler)
      h.aktualisieren(aktienname);
  }

  public String getAktienname() {
    return this.aktienname;
  }
}
```

zu 3)

Anmerkung: Man erkennt, dass die Klasse Kundenbewertung in dieser Form überflüssig ist und man direkt mit einem Zustand in TestKundenbewertung arbeiten könnte. Es gibt leichte Varianten, wichtig ist nur, dass ein konkreter Zustand auf eine Eingabe mit dem Folgezustand reagiert.

Die Aktion-Klassen wurden ergänzt, um mögliche Alternativen in der Ausführung zu verhindern und stattdessen dynamische Polymorphie zu nutzen (eher Spielerei).

```
package kapitel08_Aufgabe3;

import java.util.Scanner;

public class Main {

  public static void main(String[] args) {
    Kundenbewertung k = new Kundenbewertung();
    int eingabe = -1;
    while (eingabe != 0) {
      System.out.println("naechste Aktivitaet:\n"
          + "  (0) Programm beenden\n"
          + "  (1) letzte Rechnung puenklich\n"
          + "  (2) letzte Rechnung unpuenktlich\n"
          + "  (3) letzte Rechnung nicht gezahlt\n"
          + "  (4) Kundenstatus anzeigen\n"
          + "  (5) Kundenkreditrahmen anzeigen");
      eingabe = new Scanner(System.in).nextInt();
      switch (eingabe) {
```

```
          case 1:
            k.korrektGezahlt();
            break;
          case 2:
            k.verspaetetGezahlt();
            break;
          case 3:
            k.nichtGezahlt();
            break;
          case 4:
            System.out.println("Kunde " + k.auskunft()
                    + ".\n");
            break;
          case 5:
            System.out.println("maximaler Kredit: "
                    + k.kredit() + ".\n");
        }
      }
    }
}
```

```
package kapitel08_Aufgabe3;

public interface Zustand{
  public String auskunft();
  public int kredit();
  public Zustand folgezustand(AktionPuenktlich a);
  public Zustand folgezustand(AktionUnpuenktlich a);
  public Zustand folgezustand(AktionNichtBezahlt a);
}
```

```
package kapitel08_Aufgabe3;

public class ZustandBeobachtung implements Zustand {

  private int anzahl=0;

  public String auskunft() {
```

```
    return "wird noch bewertet";
  }

  public int kredit() {
    return 200;
  }

  public Zustand folgezustand(AktionPuenktlich a) {
    this.anzahl++;
    if(this.anzahl>2)
      return new ZustandZuverlaessig();
    else
      return this;
  }

  public Zustand folgezustand(AktionUnpuenktlich a) {
    return new ZustandKritisch();
  }

  public Zustand folgezustand(AktionNichtBezahlt a) {
    return new ZustandVorkasse();
  }
}
```

```
package kapitel08_Aufgabe3;

public class ZustandKritisch implements Zustand {

  public String auskunft() {
    return "zeigt Verhaltensdefizite";
  }

  public int kredit() {
    return 100;
  }

  public Zustand folgezustand(AktionPuenktlich a) {
    return new ZustandBeobachtung();
  }
```

```
  public Zustand folgezustand(AktionUnpuenktlich a){
    return new ZustandVorkasse();
  }

  public Zustand folgezustand(AktionNichtBezahlt a){
    return new ZustandVorkasse();
  }
}
```

```
package kapitel08_Aufgabe3;

public class ZustandVorkasse implements Zustand {

  public String auskunft() {
    return "ist kreditunwuerdig";
  }

  public int kredit() {
    return 0;
  }

  public Zustand folgezustand(AktionPuenktlich a){
    return this;
  }

  public Zustand folgezustand(AktionUnpuenktlich a) {
    return this;
  }

  public Zustand folgezustand(AktionNichtBezahlt a) {
    return this;
  }
}
```

```
package kapitel08_Aufgabe3;

public class ZustandZuverlaessig implements Zustand {
```

```
  public String auskunft() {
    return "ist zuverlaessig";
  }

  public int kredit() {
    return 500;
  }

  public Zustand folgezustand(AktionPuenktlich a) {
    return this;
  }

  public Zustand folgezustand(AktionUnpuenktlich a) {
    return new ZustandBeobachtung();
  }

  public Zustand folgezustand(AktionNichtBezahlt a) {
    return new ZustandVorkasse();
  }

}
```

```
package kapitel08_Aufgabe3;

public class Aktion {}
```

```
package kapitel08_Aufgabe3;

public class AktionNichtBezahlt extends Aktion {}
```

```
package kapitel08_Aufgabe3;

public class AktionPuenktlich extends Aktion {}
```

```
package kapitel08_Aufgabe3;
```

```
public class AktionUnpuenktlich extends Aktion {}


package kapitel08_Aufgabe3;

public class Kundenbewertung {
  private Zustand zustand;

  public Kundenbewertung(){
    this.zustand= new ZustandBeobachtung();
  }

  public void korrektGezahlt(){
    this.zustand= this.zustand.folgezustand(
                        new AktionPuenktlich());
  }

  public void verspaetetGezahlt(){
    this.zustand= this.zustand.folgezustand(
                        new AktionUnpuenktlich());
  }

  public void nichtGezahlt(){
    this.zustand= this.zustand.folgezustand(
                        new AktionNichtBezahlt());
}

  public int kredit(){
    return this.zustand.kredit();
  }

  public String auskunft(){
    return this.zustand.auskunft();
  }
}
```

zu 4)

Eine Komponente kann entweder ein Blatt oder ein Komposite sein. Ein Komposite kann wiederum aus beliebig vielen Komponenten bestehen. Insgesamt entsteht eine rekursive Struktur, bei der sich Komposite-Objekte aus Komposite-Objekten zusammensetzen. Die Rekursion endet, wenn man bei Blatt-Objekten ankommt. Insgesamt kann man so z. B. beliebige Baumstrukturen beschreiben. Durch die Multiplizität „1“ hat jedes Komposite-Objekte genau einen Nachfolger, der am Ende der so spezifizierten Liste ein Blatt ist.

Dadurch, dass nicht festgelegt ist, dass ein Komposite-Objekt nicht in irgendeiner Form wieder in einem kinder-Objekt auftaucht, ist es möglich, dass rekursive Strukturen entstehen, die nur aus Komposite-Objekten aufgebaut sind. Diese Eigenschaft kann leicht zu Endlosschleifen führen, kann aber auch für beliebige Graphstrukturen erwünscht sein.

Das folgende Programm nutzt „instanceof“, um die Klasse eines Objekts zu bestimmen. Dieser Befehl sollte in einer Anwendungsprogrammierung, die nicht ein Framework oder eine besondere Bibliothek werden soll, vermieden werden.

```
package kapitel08_Aufgabe4;

public class Main {

  public static void main(String[] args) {
    Blatt[] blaetter = new Blatt[9];
    for (int i = 0; i < blaetter.length; i++)
      blaetter[i] = new Blatt(i);
    Komposite[] k = new Komposite[3];
    for (int i = 0; i < k.length; i++) {
      k[i] = new Komposite();
      for (int j = 0; j < 3; j++)
        k[i].hinzufuegen(blaetter[i * 3 + j]);
    }
    Komposite gesamt = new Komposite();
    for (int j = 0; j < 3; j++)
      gesamt.hinzufuegen(k[j]);
```

```
    Komposite[] all = { k[0], k[1], k[2], gesamt };

    for (int i = 0; i < all.length; i++)
      System.out.println(all[i] + " :" + all[i].operation());
    System.out.println("---");
    gesamt.loeschen(blaetter[2]);
    gesamt.loeschen(blaetter[4]);
    gesamt.loeschen(blaetter[5]);
    for (int i = 0; i < all.length; i++)
      System.out.println(all[i] + " :" + all[i].operation());
    System.out.println("---");
    gesamt.loeschen(k[2]);
    System.out.println(gesamt + " :" + gesamt.operation());
  }
}
```

```
package kapitel08_Aufgabe4;

public interface Komponente {
  public int operation();
}
```

```
package kapitel08_Aufgabe4;

public class Blatt implements Komponente {

  private int wert;

  public Blatt(int wert) {
    this.wert = wert;
  }

  @Override
  public int operation() {
    return this.wert;
  }
```

```
  @Override
  public String toString() {
    return "[" + this.wert + "]";
  }
}
```

```
package kapitel08_Aufgabe4;

import java.util.ArrayList;
import java.util.List;

public class Komposite implements Komponente {

  private List<Komponente> kinder = new ArrayList<Komponente>();

  public void hinzufuegen(Komponente k) {
    // Alternativ erst eine Kopie von k erstellen
    this.kinder.add(k);
  }

  public void loeschen(Komponente l) {
    boolean gefunden = false;
    for (Komponente k : this.kinder) {
      if (k.equals(l)) // eguals und hashCode besser
implementieren
        gefunden = true;
      if (!(k instanceof Blatt)) // alternativ Flag zum Pruefen
einbauen
        ((Komposite) k).loeschen(l);
    }
    if (gefunden)
      this.kinder.remove(l);
  }

  public Komponente anPosition(int i) {
    // besser pruefen, ob Wert von i ok ist
    return this.kinder.get(i);
  }
```

```
  @Override
  public int operation() {
    int ergebnis = 0;
    for (Komponente k : this.kinder)
      ergebnis += k.operation();
    return ergebnis;
  }

  @Override
  public String toString() {
    String ergebnis = "[ ";
    for (Komponente k : this.kinder)
      ergebnis += k.toString() + " ";
    ergebnis += "]";
    return ergebnis;
  }
}
```

Lösungen zu Kapitel 9

zu 1)

Hinweis: Wie häufig bei XML-Dokumenten ist es diskutabel, was als Attribut und was als Elementinhalt modelliert wird.

```
<?xml version="1.0" encoding="UTF-8"?>
<klausur semester="SS07">
   <fach fachname="Datenbanken" lvnr="3112">
      <pruefer pruefername="Erna Meier"/>
      <teilnehmer>
         <student studiname ="Alexey Ako" matnr="346">
            <versuch> 3 </versuch>
            <note> 1.3 </note>
            </student>
         <student studiname ="Lisa Ott" matnr="347">
            <versuch> 1 </versuch>
            <note> 1.0 </note>
            </student>
      </teilnehmer>
 </fach>
```

```
    <fach fachname="Algorithmen und Datenstrukturen" lvnr="3113">
         <pruefer pruefername="Ali Meier"/>
         <teilnehmer>
            <student studiname ="Alexey Ako" matnr="346">
               <versuch> 3 </versuch>
               <note> 5.0 </note>
            </student>
            <student studiname ="Aische Schmidt" matnr="348">
               <versuch> 1 </versuch>
               <note> 3.3 </note>
            </student>
         </teilnehmer>
      </fach>
      <fach fachname="Java" lvnr="3114">
         <pruefer pruefername="Erna Meier"/>
         <pruefer pruefername="Ali Meier"/>
         <teilnehmer>
            <student studiname ="Arton Ode" matnr="349">
               <versuch> 2 </versuch>
               <note> 2.7 </note>
            </student>
            <student studiname ="Lisa Ott" matnr="347">
               <versuch> 1 </versuch>
               <note> 2.0 </note>
            </student>
        </teilnehmer>
     </fach>
 </klausur>
```

zu 2)

Hinweis: Es gibt einige Listener-Klassen, mit denen man auf Änderungen des Baumes reagieren kann und wodurch das folgende Programm noch „schöner“ werden könnte.

```
package kapitel09_Aufgabe2;

import java.awt.BorderLayout;
import java.awt.Dimension;
import java.awt.Toolkit;
import javax.swing.JFrame;
import javax.swing.JScrollPane;
```

```
import javax.swing.JTree;

public class ZahlenbaumGui extends JFrame {

  private static final long serialVersionUID = 1L;
  private Zahlenbaumverwaltung zbv;
  private JTree zahlenbaum;

  public JTree getZahlenbaum() {
    return this.zahlenbaum;
  }

  public void setZahlenbaum(JTree zahlenbaum) {
    this.zahlenbaum = zahlenbaum;
  }

  public ZahlenbaumGui() {
  }

  public Zahlenbaumverwaltung getZbv() {
    return this.zbv;
  }

  public void setZbv(Zahlenbaumverwaltung zbv) {
    this.zbv = zbv;
  }

  public ZahlenbaumGui(Zahlenbaumverwaltung zbv) {

    this.zbv = zbv;
    zahlenbaum = new JTree(this.zbv);
    JScrollPane hauptscroller
        = new JScrollPane(this.zahlenbaum);
    hauptscroller.setMinimumSize(new Dimension(600, 400));
    hauptscroller.setSize(new Dimension(600, 400));
    hauptscroller.setPreferredSize(new Dimension(600, 400));
    super.add(hauptscroller, BorderLayout.CENTER);

    super.setDefaultCloseOperation(EXIT_ON_CLOSE);
```

```
    super.pack();
    super.setLocation(0, 0);
    super.setVisible(true);

    Toolkit tk = Toolkit.getDefaultToolkit();
    tk.setDynamicLayout(true);
  }
}
```

```
package kapitel09_Aufgabe2;

public interface ZahlenbaumTreeInterface {
  public Zahlenbaum itesElement(int i);
  public int indexGeben(Zahlenbaum zb);
  public boolean istBlatt();
  public int gibAnzahl();
}
```

```
package kapitel09_Aufgabe2;

import java.util.ArrayList;

public class Zahlenbaum implements ZahlenbaumTreeInterface {

  private int wert;
  private ArrayList<Zahlenbaum> teiler;

  public Zahlenbaum() {
    this.wert = 0;
    this.teiler = new ArrayList<Zahlenbaum>();
  }

  public Zahlenbaum(int wert) {
    this.wert = wert;
    this.teiler = new ArrayList<Zahlenbaum>();
    for (int i = 2; i < wert; i++)
      if (wert % i == 0)
```

```
        this.teiler.add(new Zahlenbaum(i));
    }

    @Override
    public String toString() {
      return "" + this.wert;
    }

    public boolean istBlatt() {
      return this.teiler.isEmpty();
    }

    public Zahlenbaum itesElement(int i) {
      return this.teiler.get(i);
    }

    public int indexGeben(Zahlenbaum zb) {
      for (int i = 0; i < this.teiler.size(); i++)
        if (this.teiler.get(i).getWert() == zb.getWert())
          return i;
      return -1;
    }

    public int gibAnzahl() {
      return teiler.size();
    }

    public ArrayList<Zahlenbaum> getTeiler() {
      return this.teiler;
    }

    public void setTeiler(ArrayList<Zahlenbaum> teiler) {
      this.teiler = teiler;
    }

    public int getWert() {
      return this.wert;
    }

    public void setWert(int wert) {
```

```
    this.wert = wert;
  }
}
```

```
package kapitel09_Aufgabe2;

import java.beans.XMLDecoder;
import java.beans.XMLEncoder;
import java.io.BufferedInputStream;
import java.io.BufferedOutputStream;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.util.ArrayList;
import java.util.Scanner;

public class Zahlenbaumtext {

  private Zahlenbaumverwaltung verwaltung;
  private ZahlenbaumGui gui;
  private static String datei="Zahlenbaum.xml";


  public ZahlenbaumGui getGui() {
    return this.gui;
  }

  public void setGui(ZahlenbaumGui gui) {
    this.gui = this.gui;
  }

  public Zahlenbaumverwaltung getVerwaltung() {
    return this.verwaltung;
  }

  public void setVerwaltung(Zahlenbaumverwaltung verwaltung) {
    this.verwaltung = this.verwaltung;
  }
```

```
  public Zahlenbaumtext(){
  }

  public Zahlenbaumtext(Zahlenbaumverwaltung verwaltung){
    this.verwaltung=verwaltung;
    this.gui= new ZahlenbaumGui(this.verwaltung);
    menu();
  }

  @SuppressWarnings("unchecked")
  public void menu(){
    int wahl=-1;
    while(wahl!=0){
      System.out.println("(0) beenden\n"
          + "(1) Zahl ergaenzen\n"
          + "(2) speichern\n"
          + "(3) laden");
      wahl=new Scanner(System.in).nextInt();
      switch(wahl){
        case 1:
          System.out.print("Zahl eingeben: ");
          verwaltung.hinzufuegen(
              new Scanner(System.in).nextInt());
          break;
        case 2:
          try {
              XMLEncoder out= new XMLEncoder(
                new BufferedOutputStream(
                  new FileOutputStream(datei)));
              //out.writeObject(verwaltung);
              //um Exception mit der JTree-Behandlung zu
              // vermeiden
              out.writeObject(verwaltung.getBaeume());
              out.close();
            } catch (FileNotFoundException e) {} //schmutzig
            break;
        case 3:
          try {
              XMLDecoder in= new XMLDecoder(
```

```
                new BufferedInputStream(
                  new FileInputStream(datei)));
              this.verwaltung.setBaeume(
                  (ArrayList<Zahlenbaum>)in.readObject());
              this.gui.setZbv(this.verwaltung);
              in.close();
              } catch (FileNotFoundException e) {} //wegschauen
              break;
        }
      }
      System.exit(0);
    }

    public static void main(String[] s){
      new Zahlenbaumtext(new Zahlenbaumverwaltung());
    }
}
```

```
package kapitel09_Aufgabe2;

import java.util.ArrayList;
import javax.swing.event.TreeModelEvent;
import javax.swing.event.TreeModelListener;
import javax.swing.tree.TreeModel;
import javax.swing.tree.TreePath;

public class Zahlenbaumverwaltung
    implements TreeModel, ZahlenbaumTreeInterface {

  private static final long serialVersionUID = 1L;
  private ArrayList<Zahlenbaum> baeume;
  private ArrayList<TreeModelListener> listener

  public ArrayList<TreeModelListener> getListener() {
    return this.listener;
  }

  public void setListener(ArrayList<TreeModelListener> listener)
```

```
  {
    this.listener = listener;
  }

  public Zahlenbaumverwaltung() {
    this.baeume = new ArrayList<Zahlenbaum>();
    this.listener = new ArrayList<TreeModelListener>();
    // baeume.add(new Zahlenbaum(1));
    // baeume.add(new Zahlenbaum(1248));
  }

  @Override
  public String toString() {
    return "Zahlenbaumverwaltung";
  }

  public Object getRoot() {
    return this;
  }

  public Object getChild(Object arg0, int arg1) {
    return ((ZahlenbaumTreeInterface) arg0).itesElement(arg1);
  }

  public int getChildCount(Object arg0) {
    return ((ZahlenbaumTreeInterface) arg0).gibAnzahl();
  }

  public boolean isLeaf(Object arg0) {
    return ((ZahlenbaumTreeInterface) arg0).istBlatt();
  }

  public int getIndexOfChild(Object arg0, Object arg1) {
    return ((ZahlenbaumTreeInterface) arg0)
        .indexGeben((Zahlenbaum) arg1);
  }

  public ArrayList<Zahlenbaum> getBaeume() {
    return this.baeume;
  }
```

```
  public void setBaeume(ArrayList<Zahlenbaum> baeume) {
    this.baeume = baeume;
    Object pfad[] = { this };
    TreeModelEvent tme = new TreeModelEvent(this, pfad);
    for (TreeModelListener t : listener) {
      t.treeStructureChanged(tme);
    }
  }

  public void hinzufuegen(int i) {
    if (i > 0) {
      this.baeume.add(new Zahlenbaum(i));
      Object pfad[] = { this };
      TreeModelEvent tme = new TreeModelEvent(this, pfad);
      for (TreeModelListener t : this.listener) {
        t.treeStructureChanged(tme);
        // t.treeNodesChanged(tme);
        // t.treeNodesInserted(tme);
      }
    }
  }

  public Zahlenbaum itesElement(int i) {
    return this.baeume.get(i);
  }

  public int indexGeben(Zahlenbaum zb) {
    for (int i = 0; i < this.baeume.size(); i++)
      if (this.baeume.get(i).getWert()
              == ((Zahlenbaum) zb).getWert())
        return i;
    return -1;
  }

  public boolean istBlatt() {
    return false;
  }

  public int gibAnzahl() {
```

```
    return this.baeume.size();
  }

  public void valueForPathChanged(TreePath arg0, Object arg1) {
    System.out.println("valueForPathChanged");

  }

  public void addTreeModelListener(TreeModelListener arg0) {
    this.listener.add(arg0);
  }

  public void removeTreeModelListener(TreeModelListener arg0) {
    this.listener.remove(arg0);
  }
}
```

Lösungen zu Kapitel 10

zu 1)

Generell ist zu beachten, dass die Maßnahmen meist zur Erfüllung mehrerer Prinzipien beitragen können.

Aufgabenangemessenheit

1. Der Kunde kann Artikel anwählen und in einen Bestellkorb ablegen.
2. Erst wenn der Kunde bestellen will, werden personenkritische Daten abgefragt.

Es muss sichergestellt sein, dass die gewählten Artikel gemerkt werden, was z. B. durch das Verbot von Cookies erschwert werden kann. Die sichere Übertragung von Informationen im Web kann nicht vollständig gewährleistet werden.

Selbstbeschreibungsfähigkeit

1. Durch die Wahl typischer Symbole, wie Einkaufswagen und eine Kasse, wird deutlich, dass Waren genommen und bezahlt werden können.
2. Jedes graphische Symbol ist mit einem Hilfstext und einem Link auf die detaillierte Hilfe verknüpft.

Steuerbarkeit

1. Der Kunde kann den Ablauf ändern und erst seine persönlichen Daten und dann die Bestellung eingeben.
2. Zwischen der individuellen Produktwahl und der Übersicht über Produkte im Warenkorb kann beliebig gewechselt werden.

Erwartungskonformität

1. Der Kunde wird bei jedem Produkt informiert, ob es im Lager vorrätig ist.
2. Der Kunde erhält bei jedem Produkt eine Übersicht über mögliche Varianten, wie Größe und Farbe.

Fehlertoleranz

1. Die Eingabe unsinnig hoher Stückzahlen wird mit einem Hinweis verhindert.
2. Einzelne Bestellschritte für Artikel können rückgängig gemacht werden, ohne dass der gesamte Warenkorb geleert wird.

Individualisierbarkeit

1. Für erkannte Nutzer werden Angebote an seinen Interessen orientiert dargestellt.
2. Der Nutzer kann den Schrifttyp und die Größe der Darstellung frei wählen.

Dies ist abhängig von den Möglichkeiten, den Nutzer zu identifizieren.

Lernförderlichkeit

1. Die Seite bietet eine Hilfefunktion mit ausgefüllten Seiten für eine Beispielbestellung an.
2. Der Nutzer kann sich bei Bedarf vollständig vom System durch den Bestellprozess leiten lassen, die Schritte sind am Rand sichtbar.

Lösungen zu Kapitel 11

zu 1)

```
package kapitel11_Aufgabe1;

public class KontoException extends Exception {
```

```
  private static final long serialVersionUID
      = -2939631962750508971L;

  public KontoException(String text){
    super(text);
  }
}
```

```
package kapitel11_Aufgabe1;

public class Bankauskunft {

  public static boolean einzahlungPruefen(int kontoNr
                                          , int betrag) {
    return kontoNr==1;
  }

  public static boolean auszahlungPruefen(int kontoNr
                                          , int betrag) {
    return kontoNr==1;
  }

  public static boolean istKreditwuerdig(int kontoNr) {
    return kontoNr==1;
  }

  public static boolean istGuterKunde(int kontoNr) {
    return kontoNr==1;
  }
}
```

```
package kapitel11_Aufgabe1;

public class Kundenkonto {
  protected int geldwaschgrenze = 20000;
  protected int kontoNr;
  protected int guthaben = 0;
  protected int dispo = 0;
```

```
  public Kundenkonto(int kontoNr) {
    this.kontoNr = kontoNr;
  }

  public void setDispo(int dispo) {
    this.dispo = dispo;
  }

  public void einzahlen(int betrag) throws KontoException {
    if (betrag > this.getGeldwaschgrenze()
        && !Bankauskunft.einzahlungPruefen(
                this.kontoNr, betrag))
      throw new KontoException(
                "Geldwaescheverdacht bei Einzahlung");
    else
      this.guthaben = this.guthaben + betrag;
  }

  public void auszahlen(int betrag) throws KontoException {
    if (this.betrag > this.getGeldwaschgrenze()
        && !Bankauskunft.auszahlungPruefen(
                this.kontoNr, betrag))
      throw new KontoException(
                "Geldwaescheverdacht bei Auszahlung");
    else if (betrag <= guthaben)
      this.guthaben = this.guthaben - betrag;
    else if (betrag > this.guthaben
        && this.guthaben - betrag >= -this.dispo
        && Bankauskunft.istKreditwuerdig(this.kontoNr))
      this.guthaben = this.guthaben - betrag;
    else
      throw new KontoException("Auszahlung verweigert");
  }

  public int getGuthaben() {
    return this.guthaben;
  }

  public int getGeldwaschgrenze() {
```

```
    return this.geldwaschgrenze;
  }
}
```

```
package kapitel11_Aufgabe1;

import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

public class KundenkontoTest {
  Kundenkonto k1;
  Kundenkonto k2;

  @BeforeEach
  protected void setUp() throws Exception {
    this.k1 = new Kundenkonto(1);
    this.k1.setDispo(1000);
    this.k2 = new Kundenkonto(2);
    this.k2.setDispo(1000);
  }

  @Test
  public void testEinfacheEinzahlung() {
    try {
      this.k1.einzahlen(1000);
    } catch (KontoException e) {
      Assertions.fail();
    }
    Assertions.assertTrue(this.k1.getGuthaben() == 1000);
  }

  @Test
  public void testErfolgreicheHoheEinzahlung() {
    try {
      this.k1.einzahlen(30000);
    } catch (KontoException e) {
      Assertions.fail();
```

```
    }
    Assertions.assertTrue(this.k1.getGuthaben() == 30000);
  }

  @Test
  public void testVerboteneHoheEinzahlung() {
    try {
      this.k2.einzahlen(30000);
      Assertions.fail();
    } catch (KontoException e) {
    }
    Assertions.assertTrue(this.k2.getGuthaben() == 0);
  }

  @Test
  public void testErfolgreicheEinfacheAuszahlung() {
    try {
      this.k1.auszahlen(500);
    } catch (KontoException e) {
      Assertions.fail("" + this.k1.getGuthaben());
    }
    Assertions.assertTrue(this.k1.getGuthaben() == -500);
  }

  @Test
  public void testErfolgreicheHoheAuszahlung() {
    try {
      this.k1.einzahlen(30000);
      this.k1.auszahlen(30500);
    } catch (KontoException e) {
      Assertions.fail("" + this.k1.getGuthaben());
    }
    Assertions.assertTrue(this.k1.getGuthaben() == -500);
  }

  @Test
  public void testVerboteneEinfacheAuszahlung() {
    try {
      this.k2.auszahlen(3500);
      Assertions.fail();
```

```
    } catch (KontoException e) {
    }
    Assertions.assertTrue(this.k2.getGuthaben() == 0);
  }

  @Test
  public void testVerboteneHoheAuszahlung() {
    try {
      this.k2.einzahlen(10000);
      this.k2.einzahlen(10000);
      this.k2.auszahlen(20500);
      Assertions.fail();
    } catch (KontoException e) {
    }
    Assertions.assertTrue(this.k2.getGuthaben() == 20000);
  }
}
```

```
package kapitel11_Aufgabe1;

public class Grosskundenkonto extends Kundenkonto {

  public Grosskundenkonto(int kontoNr){
    super(kontoNr);
    super.geldwaschgrenze=1000000;
  }

  @Override
  public void auszahlen(int betrag) throws KontoException{
    try {
      super.auszahlen(betrag);
    } catch (KontoException e) {
      if (super.guthaben -betrag < -3 * super.dispo
          || !Bankauskunft.istGuterKunde(super.kontoNr))
        throw e;
      else
        super.guthaben=super.guthaben-betrag;
    }
```

```
  }
}
```

```
package kapitel11_Aufgabe1;

import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

public class GrosskundenkontoTest extends KundenkontoTest {

  @BeforeEach
  protected void setUp() throws Exception {
    //super.setUp();
    super.k1=new Grosskundenkonto(1);
    super.k1.setDispo(1000);
    super.k2=new Grosskundenkonto(2);
    super.k2.setDispo(1000);
  }

  @Test
  public void testUeberDispoGuterKunde(){
    try {
      super.k1.auszahlen(2500);
    } catch (KontoException e) {
      Assertions.fail();
    }
    Assertions.assertTrue(super.k1.getGuthaben() == -2500);
  }

  @Test
  public void testUeberDispoSchlechterKunde(){
    try {
      super.k2.auszahlen(2500);
      Assertions.fail();
    } catch (KontoException e) {
    }
    Assertions.assertTrue(super.k1.getGuthaben() == 0);
```

```
  }
}
```

Beim Grosskundenkonto können die Tests grundsätzlich übernommen werden, nur die überschriebene Methode ist besonders zu beachten. Dass dies nicht immer so einfach geht, sieht man an der Aufgabe 2).

zu 2)

Hinweis: Bei den Lösungen zu a) und b) wurde bereits über Lösungen zu b) und c) nachgedacht.

zu a)

```
package kapitel11_Aufgabe2;

public class X implements I{

  public int xx(int x, boolean b){
    if(b)
      return x+mogel()-1;
    else
      return x+mogel()+1;
  }

  public int mogel(){
    return 0;
  }
}
```

```
package kapitel11_Aufgabe2;

import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.Test;

public class XTest {

  @Test
  public void testTrue(){
    X x= new X();
    Assertions.assertTrue(x.xx(42, true) == 41);
```

```
  }

  @Test
  public void testFalse(){
    X x= new X();
    Assertions.assertTrue(x.xx(42, false) == 43);
  }
}
```

zu b)

```
package kapitel11_Aufgabe2;

public class Y extends X {

  @Override
  public int mogel(){
    return 1;
  }
}
```

```
package kapitel11_Aufgabe2;

import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.Test;

public class YTest {

  @Test
  public void testTrue(){
    X x= new Y();
    Assertions.assertTrue(x.xx(42, true) == 41);
  }

  @Test
  public void testFalse(){
    X x= new Y();
    Assertions.assertTrue(x.xx(42, false) == 43);
  }
}
```

zu c)

```
package kapitel11_Aufgabe2;

public interface I {
  public int xx(int x, boolean b);
}
```

```
package kapitel11_Aufgabe2;

public class A {
  private I i;

  public A(I i){
    this.i=i;
  }
  public int aa(int x){
    return this.i.xx(x,true);
  }
}
```

```
package kapitel11_Aufgabe2;

import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.Test;

public class ATest {

  @Test
  public void testX(){
    A a= new A(new X());
    Assertions.assertTrue(a.aa(42) == 41);
  }

  @Test
  public void testY(){
```

```
    A a= new A(new Y());
    Assertions.assertTrue(a.aa(42)==42);
  }
}
```

zu 3)

Äquivalenzklassen:

Alter:	(1) alter<14
	(2) alter>=14 && alter<=64
	(3) alter>64
Uhrzeit:	(4) uhr<9
	(5) uhr>=9 && uhr<15
	(6) uhr>=15
Betriebszugehörig:	(7) Ja
	(8) Nein

Zonenwert beliebig, oder gültig(>0) und ungültig(<=0), beim berechneten Wert 30 handelt es sich um Ausgabeäquivalenzklassen, die hier leider nicht betrachtet werden

Testfälle mit Grenzfällen:

Nummer	1	2	3	4
Klassen	(1o) (4o) (7)	(2u) (5u) (8)	(2o) (5o) (7)	(3u) (6u) (8)
Zonen	1	2	1	2
Alter	13	14	64	65
Uhrzeit	8	9	14	15
Betrieb	true	false	true	false
Ergebnis	30	130	30	110

Weiterhin fällt auf, dass man eigentlich alle Kombinationen von den drei genannten Äquivalenzklassenarten testen möchte, hier 3*3*2=18 Möglichkeiten, was im Ansatz wegen der kombinatorischen Explosionsmöglichkeit nicht vorgesehen ist.

```
package kapitel11_Aufgabe3;

public class Tarifrechner {
  public int preisBerechnen(int zonen, int alter, int uhr,
```

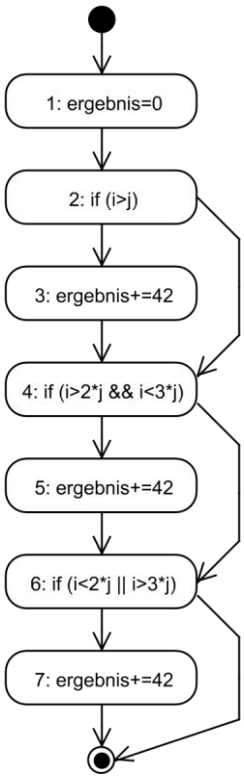
```
      boolean betrieb) {
    int ergebnis = zonen * 130;
    if (alter < 14 || alter > 64) {
      ergebnis -= 40;
    }
    if (betrieb && !(uhr >= 9 && uhr <= 15)) {
      ergebnis -= 150;
    } else {
      if (betrieb) {
        if ((ergebnis / 2) < (ergebnis - 150)) {
          ergebnis /= 2;
        } else {
          ergebnis -= 150;
        }
      } else {
        if ((uhr >= 9 && uhr <= 15)) {
          ergebnis /= 2;
        }
      }
    }
    if (ergebnis < 30) {
      ergebnis = 30;
    }
    return ergebnis;
  }

  public static void main(String[] args) {
    Tarifrechner t = new Tarifrechner();
    System.out.println(t.preisBerechnen(1, 13, 8, true));
    System.out.println(t.preisBerechnen(2, 14, 9, false));
    System.out.println(t.preisBerechnen(1, 64, 14, true));
    System.out.println(t.preisBerechnen(2, 65, 15, false));
  }
}
```

```
package kapitel11_Aufgabe3;

import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.BeforeEach;
```

```
import org.junit.jupiter.api.Test;

public class TarifrechnerTest {

  private Tarifrechner t;

  @BeforeEach
  protected void setUp() throws Exception {
    this.t= new Tarifrechner();
  }

  @Test
  public void test1(){
    Assertions.assertTrue(
        30 == this.t.preisBerechnen(1,13,8,true));
  }

  @Test
  public void test2(){
    Assertions.assertTrue(
        130 == this.t.preisBerechnen(2,14,9,false));
  }

  @Test
  public void test3(){
    Assertions.assertTrue(
        30 == this.t.preisBerechnen(1,64,14,true));
  }

  @Test
  public void test4(){
    Assertions.assertTrue(
        110 == this.t.preisBerechnen(2,65,15,false));
  }
}
```

zu 4)

zu a)



1,2,3,4,5,6,Ende
i=5, j=2
(i>j) :true
(i>2*j): true
(i<3*j):true
(i<2*j): false
(i>3*j):false

1,2,3,4,6,7,Ende
i=3, j=2
(i>j) :true
(i>2*j): false
(i<3*j):true
(i<2*j): true
(i>3*j):false
Kante (2,4) fehlt

zu b) zu a) noch zusätzlich

1,2,4,6,7,Ende
i=2, j=3
(i>j) :false
(i>2*j): false
(i<3*j):true
(i<2*j): true
(i>3*j):false

(i<3*j) ist immer true
(i>3*j) ist immer false

zu c) Testfall aus b)

1,2,4,6,7,Ende
i=2, j=3
(i>j) :false
(i>2*j): false
(i<3*j):true

(i<2*j): true
(i>3*j):false
ergänzt um
1,2,3,4,6,7,Ende
i=7 j=2
(i>j) :true
(i>2*j): true
(i<3*j): false
(i<2*j): false
(i>3*j): true

Es fehlen (4,5) und (5,6) für eine C1-Überdeckung (auch keine C0-Überdeckung).

zu d)

Erster Test aus a) ergibt für 4:true 6:false, zweiter Test aus a) 4:false, 6:true, letzter Testfall in c) rundet Überdeckung ab.

Lösungen zu Kapitel 12

zu 1)

Damit die Lösung in einem Programm implementiert werden kann, werden für A und B Interfaces definiert. Im Beispiel könnte AA von A und BB von B erben.

```
package kapitel12_Aufgabe1;

public interface AInterface {
  public int nutzeA();
}
```

```
package kapitel12_Aufgabe1;

public class A implements AInterface {

  public A(){}

  public A(BInterface b){
    try{
      if(b.nutzeB()>10)
```

```
        throw new Exception("Illegales B fuer A");
      System.out.println("A laeuft mit "
        + b.getClass().getSimpleName());
    }catch(Exception e){
      System.out.println(e.getMessage());
    }
  }

  @Override
  public int nutzeA() {
    return 42;
  }
}
```

```
package kapitel12_Aufgabe1;

public class AA implements AInterface {

  public AA() {
  }

  public AA(BInterface b) {
    try {
      if (b.nutzeB() > 10)
        throw new Exception("Illegales B fuer AA");
      System.out.println("AA laeuft mit "
          + b.getClass().getSimpleName());
    } catch (Exception e) {
      System.out.println(e.getMessage());
    }
  }

  @Override
  public int nutzeA() {
    return 2; // einzige Aenderung
  }
}
```

```java
package kapitel12_Aufgabe1;

public interface BInterface {
  public int nutzeB();
}
```

```java
package kapitel12_Aufgabe1;

public class B implements BInterface {

  public B(){}

  public B(AInterface a){
    try{
      if(a.nutzeA()<10)
        throw new Exception("Illegales A fuer B");
      System.out.println("B laeuft mit "
        + a.getClass().getSimpleName());
    }catch(Exception e){
      System.out.println(e.getMessage());
    }
  }

  @Override
  public int nutzeB() {
    return 1;
  }
}
```

```java
package kapitel12_Aufgabe1;

public class BB implements BInterface {

  public BB() {
  }
```

```
  public BB(AInterface a) {
    try {
      if (a.nutzeA() < 10) {
        throw new Exception("Illegales A fuer BB");
      }
      System.out.println("BB laeuft mit "
        + a.getClass().getSimpleName());
    } catch (Exception e) {
      System.out.println(e.getMessage());
    }
  }

  @Override
  public int nutzeB() {
    return 43; // einzige Aenderung
  }
}

package kapitel12_Aufgabe1;

public class Main {

  public static void main(String[] args) {
    new A(new B());
    new B(new A());
    new AA(new B());
    new BB(new A());
    new AA(new BB());
    new BB(new AA());
  }
}
```

Die Ausgabe lautet:

```
A laeuft mit B
B laeuft mit A
AA laeuft mit B
BB laeuft mit A
Illegales B für AA
Illegales A für BB
```

zu 2)

Der Plan hat eine einfache Ausgangslage, da es zentral nur einen Bearbeiter gibt, der zu 100% zur Verfügung steht. Die Arbeitspakete enthalten teilweise auch Wartezeiten, die nicht explizit als solche gekennzeichnet sind. Die Realisierung findet mit einem prototypischen Ansatz statt, wobei der Student für sich selbst testet, ob er mit der geforderten Technologie umgehen kann. Es gibt folgende Arbeitspakete:

A1: Aufgabe mit Unternehmen klären

A2: Aufgabe mit Professor absprechen

A3: Gemeinsame Detailklärung mit Betreuer im Unternehmen und dem Professor, mit erstem Inhaltverzeichnis und diesem Projektplan

M1: Diplomarbeit offiziell starten

A4: Prototyp mit zu nutzender Technologie (kann heimlich vor Start beginnen)

A5: Literaturrecherche (kann heimlich vor Start beginnen)

A6: Anforderungsanalyse

M2: Anforderungen mit betrieblichem Betreuer klären

A7: Anforderungen in Arbeit einarbeiten

A8: Design durchführen

M3: Design mit betrieblichem Betreuer klären

A9: Design in Arbeit einarbeiten

A10: Implementierung und Tests durchführen

M4: Implementierung mit betrieblichem Betreuer klären

A11: Implementierung in Arbeit einarbeiten

A12: Theoretische Arbeiten auf Papier bringen

A13: erste Version abschließen

A14: Arbeit Korrektur lesen lassen

A15: Endversion erstellen

A16: Binden und abgeben

A17: regelmäßige Besprechungen mit betrieblichem Betreuer

A18: regelmäßige Besprechungen mit betreuendem Professor

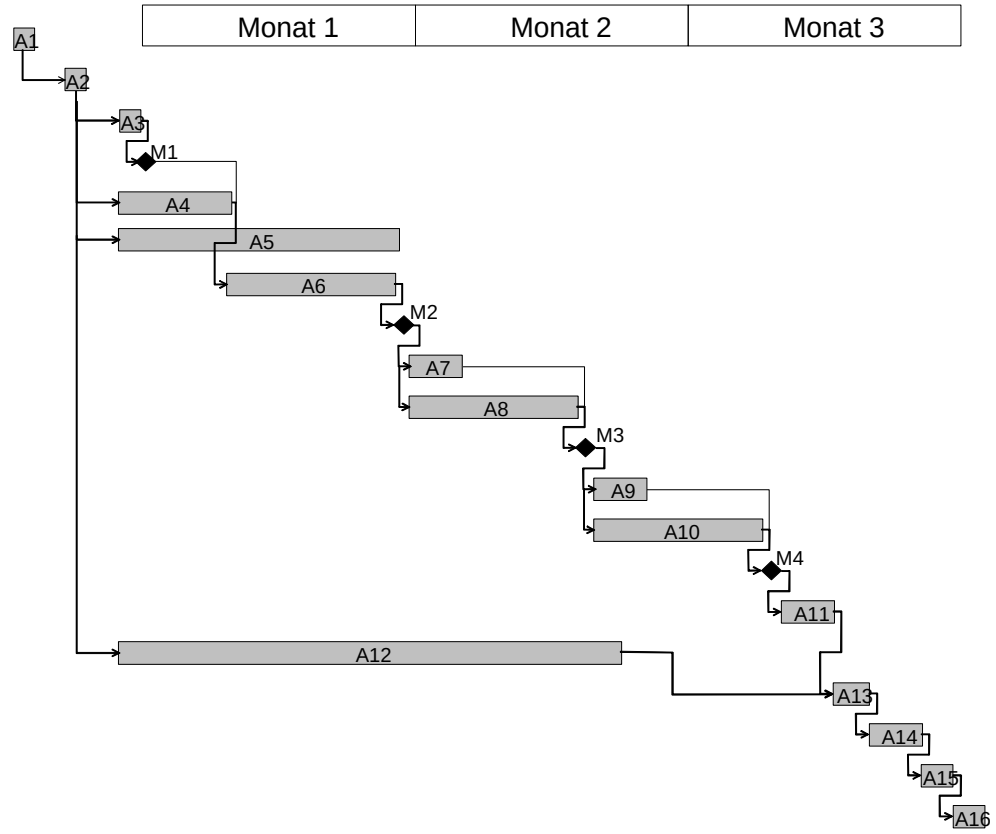


Abb. 16: Gantt-Diagramm zu Aufgabe 2

benötigte Ressourcen: Student, Unternehmensbetreuer, betreuender Professor, eventuell fachliche Ansprechpartner im Unternehmen, Korrektoren

Risiken (Auszug, kann sich durch spezielle Rahmenbedingungen noch verändern):

- Kann das Unternehmen einen kompetenten und meist kurzfristig ansprechbaren Betreuer zur Verfügung stellen?
- Hat das Unternehmen Erfahrungen mit Abschlussarbeiten?
- Hat der Betreuer des Unternehmens die formal notwendige Qualifikation zur Betreuung?

- Stehen die im Unternehmen benötigten Ansprechpartner im erwarteten Zeitrahmen zur Verfügung?
- Ist der Anfangstermin im Unternehmen geklärt, so dass keine Verschiebungen zu erwarten sind?
- Ist geklärt, wo die Arbeit geschrieben wird und welche Betriebsmittel (Schreibtisch, Rechner, Anschluss an Unternehmensnetzwerk) im Unternehmen in welcher Form ab wann zur Verfügung stehen?
- Kann der Professor fachlich, zeitlich und menschlich meine Arbeit betreuen? – Sehen alle Betreuer die Arbeit im kalkulierten Zeitrahmen als machbar an?
- Habe ich mich um die benötigten Unterlagen zur Einarbeitung, einschließlich des Zugangs zu verschiedenen Bibliotheken, gekümmert?
- Sind mir die Anmeldeprozedur und die Abgabeprozedur klar, sind alle Vorleistungen erbracht?
- Ist die Finanzierung während der Arbeit geregelt?

zu 3)

Verfahren	Daumen drauf	Function Point	CoCoMo II	vereinfachter Analogieschluss
Beschreibung	Experten schätzen anhand einer Projektbeschreibung den Aufwand	Für die SW wird die Anzahl von fünf verschiedenen Funktionsarten gezählt und mit Hilfe eines Komplexitätsfaktors eine Summe von unbewerteten Function Points (FP) berechnet. Diese FP werden mit einem Faktor für Randbedingungen mul- tipliziert. Für die erhaltenen bewerteten Function	Basierend auf einer KDSI oder unbewerteten FPSchätzung können eine große Menge von Projekteinflussfaktoren in die Schätzung eingebaut werden.	Eine Gruppe von Schätzern schätzt vier Projektgrößen für ein System, dessen Aufwand bekannt ist, und für das neue Projekt. Das Schätzergebnis wird als Analogieschluss berechnet.

		Points wird der Aufwand aus einer Tabelle abgelesen.		

Verfahren	Daumen drauf	Function Point	CoCoMo II	vereinfachter Analogieschluss
minimaler Aufwand	sehr gering	sehr aufwändig wegen detaillierter Zählung	aufwändig durch viele Parameter, allerdings toolgestützt	sehr aufwändig durch gemeinsamen Findungs- und Abstimmungsprozess aller Schätzer
wann frühestens nutzbar	sehr früh, mit geringem Kenntnisstand	eigentlich erst, wenn Anforderungen definiert	erste System-architektur muss vorliegen	früh, mit grober System-beschreibung
notwendige Schätz-erfahrung	extrem, da kaum Validierung mög-lich	relativ gering, da es Beschreibungen gibt, wie zu zählen und zu bewerten ist	relativ hoch, da man den Einfluss der Faktoren verstehen muss	gering, da nur fachliches Problem-verständnis gefordert wird

Aktualisierbarkeit	abhängig vom Dokumentationsniveau	relativ gut, durch Prozessstandardisierung	durch neue Modellvariante möglich	schwierig, nur mit gleichen Schätzern
nutzbar für Firmen-schätzdaten-bank	kaum, nur für Erfolgskontrolle des Schätzers	hoch, da Umrechnung FP in Aufwand firmenindividuell ist	eingeschränkt bis hoch, da Einschätzungen der Einfluss-faktoren verglichen werden können und Modell firmenspeziell angepasst werden kann	stark eingeschränkt, da abhängig vom Schätzteam, geschätzte und reale Größenverhältnisse zwischen Kompo-nenten interessant
Anwendungsgebiete	kleine Unternehmen, kleine Projekte, neues Geschäftsgebiet	viele gleichartige Projekte im Unternehmen zur Normierung	viele gleichartige Projekte im Unternehmen zur Normierung	Entwickler sollen schätzen, vorhandene ähnliche Projekte
Verfahren	Daumen drauf	Function Point	CoCoMo II	vereinfachter Analogieschluss
Vorteile	schnell, Schätzer fühlt sich verantwortlich	wissenschaftlich fundiert, hoher Verbreitungsgrad, genau dokumentiert	weit verbreitet, toolgestützt, anpassbar auf Firma	kein Schätzexperten-Know-how notwendig, gemeinsame Ergebnisfindung im geleiteten Gruppenprozess, Schätzer fühlt sich für Ergebnis verantwortlich

Nachteile	schwer nachvollziehbar, später kaum analysierbar	verlangt detailliertes Modell, muss auf Firma angepasst werden	Ausgangspunkt der Schätzung (KDSI oder FP) haben eigene Probleme, viele weiche Einfluss-faktoren	Findung des Schätzergebnisses später relativ schwer nachvollziehbar

zu 4)

AltS: A*B*C*D=36, ergibt für einen Punkt den Wert 120/36= 3 1/3

Neu1: A*B*C*D=15, ergibt den Aufwand 15* 3 1/3 = 50 Personentage

Neu2: A*B*C*D=57, ergibt den Aufwand 57* 3 1/3 = 190 Personentage

zu 5)

Problem auf Firmenumfeld: zu langsame Internet-Verbindung

- gescheiteter Versuch auf Prozessebene: bestimmte Abteilungen dürfen nur zu bestimmten Zeitpunkten das Internet nutzen (kann Arbeiten verzögern, zu Missstimmungen führen)

Problem auf Prozessebene: unklar, ob Qualitätssicherung Freigabe an Kunden verzögern kann

- gescheiteter Versuch auf sozialer Ebene: Mitarbeiter werden zum Team eingeschworen, man vertraut immer der Arbeit des Kollegen/Freundes, Freigabe ist aber kein Freundschaftsdienst
- gescheiteter Versuch auf Erfahrungsebene: Schulung der QSMitarbeiter (regelt nicht ihre Kompetenzen und ihr Verhalten, wenn Projektleiter Freigabe erzwingen will)
- gescheiteter Versuch im Firmenumfeld: räumliche Trennung von Entwicklung und QS um Distanz zu zeigen (es bleibt unklar, wann Informationen wie zwischen den Räumen fließen)

zu 6)

Dadurch, dass ein anderer eine Äußerung hören kann, wird er in dem Moment, in dem mindestens der Sender oder Empfänger die Anwesenheit der dritten Person erkennt,

automatisch zum Gesprächsteilnehmer. Für die vier Nachrichtenseiten ergeben sich bezüglich Sender, Empfänger und Zuhörer neue Interpretationsmöglichkeiten. Der Sachinhalt der Äußerung bleibt erhalten. Bei der Selbstkundgabe muss der Sender beachten, dass diese weiter nach außen getragen wird. Der Empfänger muss überlegen, ob die Form der Selbstkundgabe etwas mit der Anwesenheit des Zuhörers zu tun hat. Die Beziehungsebene wird ebenfalls dem Zuhörer offen gelegt und kann dadurch beeinflusst werden. Der Empfänger kann sich z. B. fragen, welches Bild seiner Person durch die Äußerung beim Zuhörer erweckt werden soll. Beim Appell stellt sich dann noch die Frage, ob er an alle zuhörenden Personen gerichtet ist. Theoretisch ist es dabei auch möglich, dass der Sender zwar zum Empfänger redet, eigentlich aber den Zuhörer zumindest mit ansprechen möchte.