Projekthandbuch
für die Durchführung
von SW-Projekten

1 Einleitung

Ziel dieses Projekts ist die Realisierung eines Systems

- (in einem Team),
- in einer praxisorientierten Entwicklungsumgebung,
- unter Verwendung moderner Entwurfsmethoden.

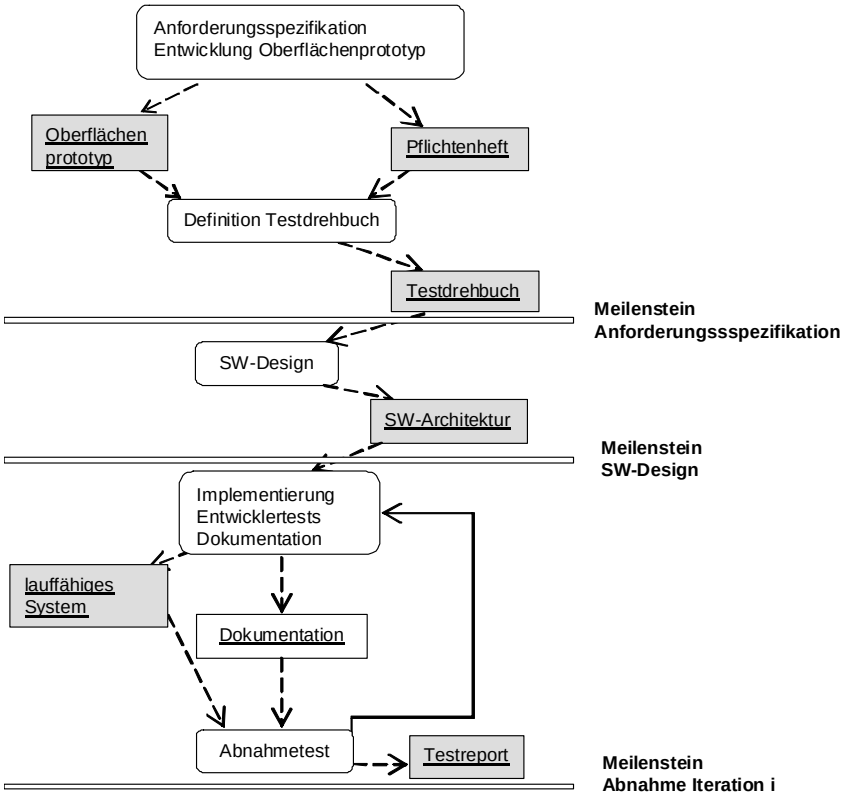
Verstehen Sie die geforderten Dokumente und Vorgehensweisen deshalb nicht als unnötige Bürokratie, sondern als Chance, diese Techniken in einem überschaubaren Kontext einüben zu können. Es ist klar, dass sich der Nutzen erst voll entfalten kann, wenn

- große Teams (mit wechselnden Mitgliedern) komplexe Systeme entwickeln,
- das System eine langen Lebenszyklus hat (also modifiziert und weiterentwickelt wird),
- Risiken bei der Realisierung auftreten (Technologiewechsel, instabile Kundenwünsche, kritische Anforderungen an das System,…).

Im nächsten Kapitel wird zunächst das Vorgehen im Projekt (der *Entwicklungsprozess*) definiert. Daran anschließend werden die einzelnen Entwicklungsaktivitäten mit den zu erstellenden Dokumenten genauer beschrieben.

2 Der Entwicklungsprozess

Die Graphik unten zeigt die zentralen Entwurfsaktivitäten und ihre zeitliche Abhängigkeit, die zu erstellenden Dokumente sowie die Meilensteine. Für die Entwicklung von Systemen im Rahmen von Praktika wählen wir dabei ein Vorgehen, bei dem die Implementierung iterativ durchgeführt wird, d.h. der Funktionsumfang des Systems sukzessive erweitert wird. Der Implementierung voran geht eine Spezifikationsphase (Anforderungsspezifikation, SW-Design), bei der die Anforderungen an das System und die Systemarchitektur festgelegt werden.



Begleitend dazu wird im Rahmen des Projektmanagements der *Projektplan* erstellt und fortgeschrieben (Kapitel 3).

In der *Anforderungsspezifikation* werden die fachlichen Anforderungen an das System und der Systemumfang identifiziert, mit dem Projektverantwortlichen und ggf. Kunden diskutiert und im Pflichtenheft festgehalten. Der Umfang der Arbeit wird geprüft und freigegeben. Die Entwicklung eines *Oberflächenprototyps* soll diesen Prozess unterstützen (Kapitel 4). Am Ende der Anforderungsspezifikation müssen die Meilensteine und die Tasks mit ihren Aufwendungen im Projektplan erstellt sein.

Im *Testdrehbuch* werden die Testfälle für den Abnahmetest basierend auf dem Pflichtenheft beschrieben (Kapitel 5).

Pflichtenheft, Oberflächenprototyp, Testdrehbuch, sowie der aktuelle Projektplan werden dem Kunden (Betreuer) im Meilenstein *Anforderungsanalyse* präsentiert.

Im SW-Design wird die grobe Komponentenstruktur der Implementierung (d.h. die SW-Architektur) entworfen (Kapitel 6). Darauf aufbauend können Arbeitspakete für die Mitglieder des Teams definiert werden. Die SW-Architektur zusammen mit dem aktuellen Projektplan wird im Meilenstein *SW-Design* präsentiert.

In der darauf folgenden Phase wird das System in mehreren Iterationen (mindestens zwei) realisiert. Jede Iteration führt zu einem lauffähigen System, das in einem Meilenstein *Abnahme Iteration i* abgenommen wird. Die Auswahl der Funktionalität für eine Iteration orientiert sich dabei am Pflichtenheft. Während der Implementierung wird das System laufend getestet und dokumentiert. Am Ende des Projekts ist mit dem lauffähigen Code ein konsistenter Satz von Dokumenten (fachliche Dokumentation, SW-Architektur usw.) abzugeben (siehe Kapitel 7).

3 Projektmanagement

Das Projektmanagement sollte mindestens folgende leichtgewichtige Variante beinhalten:

- Definition von Meilensteinen (im PM-Werkzeug)
- Definition von Tasks (z.B. *Pflichtenheft erstellen*, *Implementierung Komponente XY*) und Zuordnung zu Meilensteinen (im PM-Werkzeug)
- Statuspflege der Tasks (im PM-Werkzeug)
- Schätzung des Arbeitsaufwands der Tasks mit Datum der Schätzung (z.B. in Spreadsheets)
- Aufzeichnung von Aufwänden (Arbeitsstunden) pro Teammitglied und Task (z.B. in Spreadsheets)

Der Abschlussbericht soll Ihr Team dazu anhalten, den Verlauf des Projektes zu analysieren, Ursachenforschung für entdeckte Probleme zu betreiben und Verbesserungsvorschläge für künftige Projekte zu machen. Außerdem sind wir an Feedback für die Durchführung künftiger Projekte interessiert. Die Vorlage für den Abschlussbericht mit einem Fragenkatalog finden Sie auf der Homepage.

4 Anforderungsspezifikation

Ziel der Anforderungsspezifikation ist die Identifikation der fachlichen Anforderungen an das System und die Abstimmung mit dem Kunden darüber. Ergebnis der Anforderungsspezifikation ist

- das Pflichtenheft und
- der Oberflächenprototyp.

Die Modelle des Pflichtenhefts werden im Modellierungstool erstellt. Dazu definieren Sie am besten ein Paket **Pflichtenheft** und ordnen diesem Paket alle erstellten Diagramme zu. Eine Vorlage für das Pflichtenheft selbst finden Sie auf der Homepage.

Am Ende der Anforderungsspezifikation präsentieren Sie dem Kunden (Betreuer) das Pflichtenheft zusammen mit dem Oberflächenprototyp und arbeiten evtl. Änderungswünsche ein.

4.1 Das Pflichtenheft

Das Pflichtenheft soll folgende Teile enthalten:

- Fachliches Klassendiagramm mit Invarianten
- Use Case-Diagramm mit textuell beschriebenen Use Cases
- evtl. Zustandsdiagramme

Die Entwicklung der Kernmodelle muss in enger Kooperation mit dem Kunden bzw. dem Betreuer erfolgen, um zu klären, was das System leisten soll oder welche Konzepte unterstützt werden sollen.

Fachliches Klassendiagramm und Zustandsdiagramme

Das fachliche Klassendiagramm beschreibt die statischen fachlichen Konzepte des Anwendungsbereichs (wie z.B. Kunde, Konto, Buchung in einem Bankkontext). Das Klassendiagramm soll folgende Elemente enthalten:

- Klassen
- Attribute (ohne Typ)
  Ausnahme: Attribute, die einen Status beschreiben; hier ist ein Enumerationstyp sinnvoll
- Assoziationen mit Multiplizitäten und Vererbungsbeziehungen
- eine textuelle Charakterisierung der Modellelemente, falls diese nicht selbsterklärend sind
- das fachliche Klassendiagramm muss keine Operationen enthalten

Zusätzlich sollen im Pflichtenheft *Invarianten* beschrieben werden.

Invarianten beschreiben Eigenschaften, die die Objekte des Klassendiagramms erfüllen. Mit den Invarianten lassen sich Abhängigkeiten im Klassendiagramm dokumentieren (z.B. Ab-

hängigkeiten von Attributen). Damit können z.B. fachliche Regeln ausgedrückt werden (Beispiel: *„Die Ausleihdauer von Büchern beträgt vier Wochen“*).

Die Invarianten können informell durch Text beschrieben werden.

Erstellen Sie außerdem für Klassen, deren Objekte relevante Zustände einnehmen, ein Zustandsdiagramm. Dieses Zustandsdiagramm enthält die Zustände und die Übergänge zwischen diesen Zuständen. Beschreiben Sie jeden Zustand dabei textuell durch einen kurzen Satz. Typische Klassen, für die die Entwicklung eines Zustandsdiagramms sinnvoll ist, sind Klassen, die eine Statusvariable enthalten.

Use Case Diagramm

Das Use Case Diagramm enthält die Use Cases des Systems zusammen mit den beteiligten Akteuren.

Dokumentieren Sie das Diagramm, indem Sie jeden Akteur kurz charakterisieren und jeden Use Case in der folgenden Form beschreiben.

1. Name des Use Cases
2. Auslösender Akteur
3. Vorbedingung: unter welcher Bedingung kann der Use Case ausgelöst werden?
4. Nachbedingung: welche Bedingung gilt nach Ablauf des Use Cases?
5. Basisablauf
6. alternative Abläufe, Ausnahmen
7. Welche Klassen des fachlichen Klassendiagramms sind in den Use Case involviert (d.h. werden gelesen, kreiert, modifiziert)?

Die Ablaufbeschreibung soll beinhalten, welche Informationen zwischen Akteur und System ausgetauscht werden und welche Aktionen das System vornimmt.

Use Cases, deren Bedeutung selbsterklärend ist, müssen nur kurz durch einen Satz beschrieben werden.

4.2 Oberflächenprototyp

Entwerfen Sie eine graphische Oberfläche, die die identifizierten Use Cases unterstützt. Die Oberfläche soll prototypisch in dem Sinne sein, dass

- noch keine fachliche Funktionalität hinterlegt ist bzw. nur eine Dummy-Funktionalität (z.B. Anzeige von konstanten Daten bei der Suche nach Personen)
- der Kern implementiert sein soll, aber einige Teile noch fehlen dürfen (von denen klar ist, was sie leisten sollen)

Entwerfen Sie die Oberfläche sorgfältig in der Weise, dass Sie sie im weiteren Projekt erweitern und mit Funktionalität versehen können. Achten Sie dabei auf eine saubere MVC-Struktur. Stellen Sie die wichtigsten Klassen der Oberfläche in einem Klassendiagramm dar (definieren Sie dazu ein weiteres Paket **SW-Architektur** im Modellierungstool).

5 SW-Design

Das zu implementierende System soll in einer 3-Schichten-Architektur realisiert werden, bestehend aus der graphischen Oberfläche, dem fachlichen Kern und der Persistenzschicht.

Im SW-Design wird die Implementierung des Systems geplant und Arbeitspakete für die Implementierung gebildet. Besonderes Augenmerk soll dabei auf die saubere Trennung der Schichten gelegt werden. Ergebnis des SW-Designs ist die *SW-Architektur*.

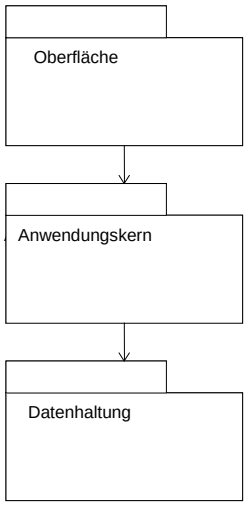
Als Ergebnis soll die SW-Architektur folgende Modelle enthalten:

- Ein Klassendiagramm, das die Pakete (*Komponenten*) und ihre Abhängigkeiten darstellt (*Logisches Komponentendiagramm*)
- Technische Klassendiagramme für alle Pakete
- Sequenzdiagramme (oder wahlweise Kollaborationsdiagramme), die die wesentlichen Abläufe im System als Nachrichtenfluss darstellen
- evtl. Zustandsdiagramme für Klassen mit relevanten Zuständen

Ein weiterer Bestandteil der SW-Architektur ist die Definition des Datenbankschemas.

5.1 Logisches Komponentendiagramm

Das *Logische Komponentendiagramm* beschreibt die Komponenten des Systems und ihre Abhängigkeiten. Ihr Diagramm sollte mindestens die folgenden Pakete enthalten:



5.2 Technisches Klassendiagramm

Das technische Klassendiagramm enthält die Kernklassen, die in der Implementierung umgesetzt werden. Anders als das fachliche Klassendiagramm enthält das technische Klassendiagramm

- Typen für die Attribute
- Methoden mit Signatur und Sichtbarkeit

- Interfaces

Sie können das technische Klassendiagramm entweder als ein zentrales Klassendiagramm oder als mehrere Klassendiagramme, die den Paketen zugeordnet sind, darstellen.

Klassen, deren Objekte signifikante Zustände annehmen, sollten mit einem Zustandsdiagramm näher beschrieben werden. Die Übergänge zwischen den Zuständen sind dabei durch Methodenaufrufe getriggert.

5.3 Sequenzdiagramme

Stellen Sie zentrale Nachrichtenflüsse in Ihrem System als Sequenz- oder Kollaborationsdiagramme dar. Sinnvoll ist dabei die Darstellung

- des Ablaufs von Use Cases
- der Kollaboration von Objekten in unterschiedlichen Schichten (z.B. der Interface- und der fachlichen Schicht).

5.4 Methodisches Vorgehen

Für die Entwicklung der SW-Architektur schlagen wir folgendes Vorgehen vor:

Entwicklung von Sequenzdiagrammen auf der Basis der Use Cases

Gehen Sie die zentralen Use Cases durch und entwickeln Sie für jeden Use Case (mindestens) ein Sequenzdiagramm, das den Ablauf des Use Cases beschreibt. Die Beschreibung dieses Nachrichtenflusses wird zur Definition von neuen Klassen und Methoden führen. Integrieren Sie diese in das technische Klassendiagramm.

Konsolidierung des technischen Klassendiagramms

Überarbeiten Sie das so gewonnene technische Klassendiagramm. Achten Sie dabei insbesondere

- auf eine saubere Trennung der Schichten
- auf die richtige Granularität der Methoden (Methoden sollen nicht zu groß sein)
- auf sinnvolle Vererbungsstrukturen
- auf die Verwendung von Klassen aus der Java-Bibliothek (z.B. Containerklassen)
- auf die Verwendung von Interfaces, besonders zur Trennung der Schichten

Überlegen Sie sich außerdem ein Konzept

- für den Zugriff auf die Datenbank
- für die Behandlung von Fehlern (ein kommentarloses Abstürzen des Systems ist für den Benutzer nicht zumutbar)

Absprechen von Namenskonventionen

Vereinbaren Sie in Ihrem Team Namenskonventionen. Als Basiskonventionen schlagen wir vor:

- Klassennamen beginnen mit Großbuchstaben

- Methodennamen und Attributnamen sind kleingeschrieben
- Interfaces beginnen mit I

Einigen Sie sich auf eine Sprache (englisch oder deutsch), in der Sie benennen und dokumentieren (eine oftmals verwendete Variante ist auch die Benennungen in Englisch, Kommentare in der Team-/Muttersprache). Vereinbaren Sie außerdem Namenskonventionen, die die Rolle einer Klasse beschreiben (z.B. XXXController für Controllerklassen oder XXXDBAccess für Klassen, die auf die Datenbank zugreifen).

Entwicklung von Arbeitspaketen

Definieren Sie Arbeitspakete und achten Sie auf eine saubere und stabile Definition der Schnittstellen, um spätere Probleme zu vermeiden.

6 Test der Software

Im Rahmen des Praktikums sollen zwei Testverfahren angewendet werden:

1. Entwicklertests (Regressionstests mit JUnit)
2. Abnahmetests (mittels Testdrehbuch)

6.1 Entwicklertests

Testfälle für Entwicklertests werden während der Entwicklung des Systems festgelegt. Welche Testfälle sinnvoll sind, hängt natürlicherweise stark von der zu testenden Funktion und des zu betreibenden Aufwands ab.

Für jede Methode einer Klasse sollten Testfälle definiert werden. Sinnvoll ist es, zum einen Standardfälle zu testen, zum anderen aber auch Grenzfälle von Parametern (extreme Werte, z.B. MAXINT, 0). Ein Sonderfall von Grenzfällen ist z.B. geplantes Fehlverhalten (z.B. das Werfen einer Java-Exception).

Für Klassen der Benutungsroberfläche (Fenster, Buttons, Labels) macht es im Allgemeinen wenig Sinn, automatisierbare Testfälle zu definieren. Deshalb sollten im Rahmen des Projekts diese Klassen nur manuell getestet werden (siehe Abnahmetests).

Im Allgemeinen muss für das Testen eine definierte Testumgebung (z.B. Datenbank mit Testdaten) aufgebaut werden.

In diesem Projekt soll JUnit als Testwerkzeug eingesetzt werden. JUnit erleichtert die Definition, die Organisation und den Ablauf von Testfällen. Insbesondere kann ein einmal entwickelter Satz von Testfällen immer wieder wiederholt werden (Regressionstests).

6.2 Abnahmetest

Der Abnahmetest definiert sich durch vom Benutzer/Abnehmer frei gestaltete Testabläufe. Hier beschränken wir uns auf Tests der Systemfunktionalität. Darüber hinaus könnten noch zahlreiche nicht-funktionale Aspekte (z.B. zugesicherter Durchsatz, Antwortzeiten, Lastverhalten, Konfiguration, Installation, ...) getestet werden.

6.2.1 Testdrehbuch

Die Abnahmetests werden nach einem Testdrehbuch durchgeführt. Das Testdrehbuch ist eine strukturierte Beschreibung von Testfällen, die von Testpersonen (über die Benutzungsoberfläche) durchgeführt werden. Die Testfälle orientieren sich am Use-Case-Modell des Pflichtenhefts. Typischerweise gibt es zu jedem Use-Case einen oder mehrere Testfälle.

Jeder Testfall ist beschrieben durch einen Namen, ggf. einen Bezug zum Use-Case-Modell, den Ausgangszustand, eine durchzuführende Aktion, und das erwartete Ergebnis. Im Allgemeinen ist das Testdrehbuch bereits so strukturiert, dass das Dokument als Testprotokoll mit der Eintragung der beobachteten Abweichungen benutzt werden kann.

Die Abweichung kann der Tester in eine Klassifikation einordnen: Eine mögliches Schema ist

Fehlerklasse	***Definition***
OK	Keine Abweichungen gefunden
kosmetische Abweichungen	Kleinere Layoutprobleme: z.B. Zeilenumbrüche im Text ungeschickt, Texte für Buttons zu lang, ...
mittlere Abweichungen	Die Funktionalität ist grundsätzlich vorhanden, kann aber nur eingeschränkt benutzt werden, z.B. einige erwartete Einträge in einer Drop-.Downliste fehlen, Datenänderungen sind erst nach Schließen und wieder Öffnen eines Fensters sichtbar, ...
große Abweichungen	Die Funktionalität ist nicht benutzbar, z.B. Buttons zeigen keine Reaktion, Daten werden nicht korrekt in die Datenbank geschrieben, …
System unbenutzbar	Die Durchführung dieses Tests hinterlässt das System in einem unbenutzbaren Zustand, z.B. System stürzt ab. Datenbank wird inkonsistent, Daten werden (ungeplant) gelöscht.

Beispiel für die Beschreibung eines Testfalls:

Testfall: unkorrekter Login	
Name:	Fehlerhafter Login
Use-Case:	<nummer> <name>
Ausgangszustand:	Die Anwendung zeigt nach dem Start die Loginmaske.
Aktion	Der Nutzer gibt den Loginnamen „meier“ und das Passwort „123456“ in die Loginmaske ein
erwarteter Ergebniszustand:	Die Anwendung zeigt erneut einen Loginbildschirm mit der Fehlermeldung „Login und Passwort stimmen nicht überein“, das Feld für den Loginnamen ist mit „meier“ vorbelegt,

beobachtete Abweichung	das Passwortfeld ist leer.
☐ OK ☐ kosmetische Abweichungen ☐ mittlere Abweichungen ☐ große Abweichungen ☐ System unbenutzbar	

Eventuell kann der Ausgangszustand bereits auf die Abarbeitung anderer Testfälle (z.B. die Eintragung eines Mitarbeiters in das System) Bezug nehmen.

6.2.2 Vorgehen

Das Testdrehbuch wird im Anschluss an die Anforderungsspezifikation erstellt. Das Testdrehbuch wird vom Projektteam erstellt und muss vom Projektverantwortlichen abgenommen werden. Im Allgemeinen könnte das Testdrehbuch auch vom Kunden selbst, oder einem unabhängigen Testteam erstellt werden.

Der Abnahmetest selbst wird jeweils von einem Testteam, das der Projektverantwortliche bestimmt, durchgeführt. Die Abnahmetests erfolgen nach Abschluss der Entwicklungsarbeiten und der Entwicklertests.

Jede Testperson notiert für jeden Testfall die beobachteten Abweichungen im Testdrehbuch. Das Ergebnis des Abnahmetests ist ein vollständig ausgefülltes und unterschriebenes Testprotokoll. Aufgrund des Testprotokolls kann der Kunde drei Entscheidungen treffen:

- Abnahme erteilt, oder
- die Überarbeitung der identifizierten Punkte ohne erneuten Abnahmetest, oder
- die Überarbeitung der identifizierten Abweichungen und ein erneuter Abnahmetest.

Anmerkung: Es wird generell erwartet, dass Software die in sie gestellten Erwartungen erfüllt. Der entwicklerseitige Test sollte daher weit über den Umfang des Testdrehbuchs hinausgehen, um eine möglichst hohe Fehlersicherheit zu gewährleisten.

Wichtiger als ein möglichst vollständiges Testdrehbuch ist der Test zum Beispiel bei

- nicht standardmäßiger oder komplexer Abläufe
- fehlerhafter Eingabe
- Randfällen von Use Cases
- Größeren Datensätzen

Der Test erfolgt auf einem Referenzsystem. Die Software muss dort (ohne Installation nichtvereinbarter Komponenten) ausführbar sein. Bitte achten Sie daher schon bei der Implementierung auf die notwendige Portierbarkeit!

6.2.3 Alternative

Alternativ zu den manuellen Abnahmetests kann das Framework Fit/Fitnesse (www.fitnesse.org) für ausführbare Akzeptanztests verwendet werden.

7 Dokumentation

Die Dokumentation ist für den Lebenszyklus eines realen Systems von großer Bedeutung (z.B. als Basis für Modifikationen und Erweiterungen und für die Einarbeitung neuer Teammitglieder). In den Praktika soll durch die Dokumentation die Arbeit im Team erleichtert und dem Betreuer der rasche Einstieg in das System ermöglicht werden. Wir möchten aber generell dazu raten, sich eine saubere Dokumentation anzugewöhnen.

Das System soll auf verschiedenen Abstraktionsebenen dokumentiert werden. Somit besteht die Dokumentation

- aus einer aktualisierten Version des Pflichtenhefts (*fachliche Dokumentation*)
- aus einer aktualisierten Version der SW-Architektur
- aus den JavaDoc-Seiten
- aus dem dokumentierten Code

7.1 Fachliche Dokumentation

Aktualisieren Sie das Pflichtenheft in der Weise, dass die Modelle die fachliche Sicht des realisierten Systems wiedergeben. Markieren Sie z.B. Use-Cases, die Sie nicht realisiert haben.

Halten Sie jedoch immer die abgenommene Version des Pflichtenhefts in unveränderter Form als Version 1.0.

7.2 SW-Architektur

Aktualisieren Sie die SW-Architektur ebenfalls in der Weise, dass die Diagramme die Kernkonzepte und Struktur der tatsächlichen Implementierung darstellen. Überarbeiten Sie vor allem die Sequenzdiagramme, so dass folgende Aspekte dargestellt werden:

- Ablauf der Use Cases
- Kollaboration von Objekten unterschiedlicher Schichten

7.3 JavaDoc-Seiten und Dokumentierung des Codes

Die JavaDoc-Seiten sollen mindestens folgende Elemente enthalten:

- Für jede Klasse bzw. Interface
    - Einen Autor
    - Eine ganz kurze textuelle Charakterisierung der Klasse (1 Satz)
- Methodenbeschreibungen für nicht-triviale Methoden:
    - Charakterisierung der Parameter und des Rückgabewerts der Methoden
    - Charakterisierung der Exceptions, die von dieser Methode geworfen werden

- Kurze Charakterisierung des Effekts der Methode. Werden Bedingungen an die Parameter gestellt? Wann werden Exceptions geworfen?

Nehmen Sie sich dabei die Java API Spezifikation zum Vorbild.

Der Code sollte außerdem Kommentare der Statements in den Methoden enthalten.

Wir wünschen viel Spaß und Erfolg bei der Entwicklung!

Ihre Forschungsgruppe Quality Engineering