Dokumentationsrichtlinien

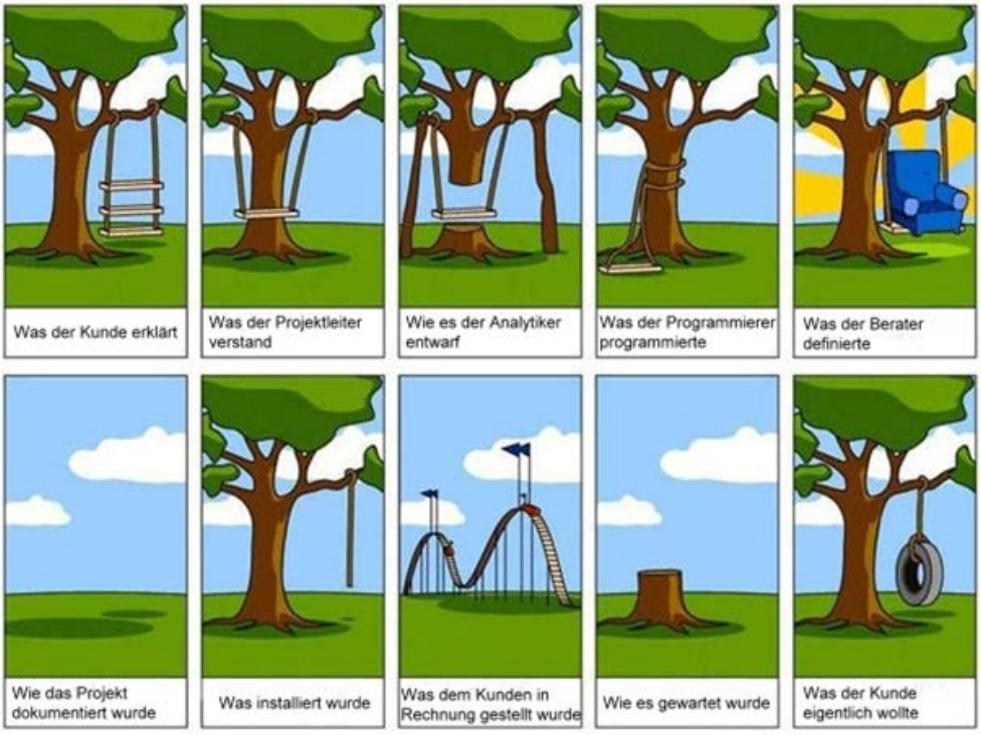
im Rahmen der Projektarbeit

für Informatik-Projekte

am

bib International College

Paderborn und Bielefeld



Version 2.4

Inhalt

1 Erweiterungen für IT-Dokumentationen

Die bereits beschriebenen Grundlagen gelten unverändert. Die in diesem Kapitel dargestellten Sachverhalte sind als ***Erweiterung der Grundlagen*** zu verstehen.

1.1 Layout der Dokumentation

Halten Sie sich beim Layout der Dokumentation bitte an die folgenden Punkte:

- Das Seitenformat ist generell DIN-A4.
- Sollte ein größeres Format zwingend sein (beispielsweise für Klassendiagramme), ist ein auszufaltendes Format zulässig.
- Der Quell-Listing-Teil wird auf einem Datenträger dokumentiert (CD oder DVD). Er muss unproblematisch auch in Teilen ausgedruckt werden können.
- Bei der Erstellung aller Unterlagen ist darauf zu achten, dass ein Bundsteg (Seitenrand, der beim Binden verloren geht) beachtet wird, damit die gebundene Dokumentation auch noch lesbar ist. Empfohlener Bundsteg: mindestens 2 Zentimeter.
- Jeder Dokumentationsunterlage ist ein Deckblatt voranzustellen.
- Zum schnellen Auffinden der Texte der einzelnen Kapitel bzw. Dokumentationsteile empfiehlt es sich, diese durch Einlegen eines farbigen Blattes voneinander zu trennen.

1.2 Umfang der Dokumentation

Die Dokumentation eines Softwareproduktes (Projektarbeit) besteht aus 6 Teilen:

1. Pflichtenheft
2. Entwurfsdokumentation
3. Implementierungsdokumentation
4. Testdokumentation
5. Benutzerdokumentation
6. Installationshandbuch

Alle Teile der Dokumentation sind ein integraler Bestandteil der Gesamtdokumentation und keine Einzeldokumentationen. Das Pflichtenheft wird kurz nach Beginn der Projektphase beim projektbetreuenden Fachdozenten abgegeben (den aktuellen Termin entnehmen Sie bitte der Terminliste), findet sich aber dennoch auch in der Gesamtdokumentation noch einmal wieder. Aus dem Pflichtenheft sollten sich der Entwurf und das Benutzerhandbuch ableiten lassen, aus dem Entwurf die

Implementation und das Installationshandbuch. Wichtig ist die gute Nachvollziehbarkeit der gesamten Dokumentation. Nicht die Darstellungsmethode ist entscheidend, sondern die Problemangemessenheit.

2 Pflichtenheft

Das Pflichtenheft beschreibt die für den Benutzer sichtbaren Leistungen des zu erstellenden Produktes und bildet damit die verbindliche Grundlage für Softwareentwickler und Auftraggeber während der weiteren Entwicklungsphasen. Im Pflichtenheft wird also das "was" beschrieben, ohne eine Lösung vorwegzunehmen. Das Pflichtenheft ist Bestandteil des Vertrages zwischen Softwareentwickler und Auftraggeber und muss daher ausreichend präzise beschrieben sein, um bei Streitfällen eine Entscheidung zu ermöglichen. Das Pflichtenheft besteht aus 9 bzw. 10 Teilen (s. Inhaltsverzeichnis), die in der angegebenen Reihenfolge nacheinander bearbeitet werden. Zu allen Punkten müssen Aussagen gemacht werden. Ist dieses nicht möglich, erscheint der betreffende Punkt im Inhaltsverzeichnis mit dem Zusatz "entfällt".

2.1 Vorgehensweise

Aus den Anforderungen des Benutzers wird das Produktmodell in UML erstellt. Zu diesem Zweck bieten sich das Umweltdiagramm als grober Überblick und das Use-Case-Diagramm an. Abläufe und Aktivitäten werden durch Aktivitätendiagramme beschrieben, die gleichzeitig zur Verfeinerung und Strukturierung der Use-Case-Diagramme dienen.

Das Produktmodell dient der Konkretisierung der Aufgabenstellung und fördert die Kommunikation zwischen Anwender und Entwickler. Es zwingt den Entwickler, die Aufgabenstellung vollständig zu durchdenken und auszuformulieren. Das Pflichtenheft bildet damit die verbindliche Grundlage für Softwareentwickler und Auftraggeber während der weiteren Entwicklungsphasen.

2.2 Aufbau des Pflichtenheftes

Da das Pflichtenheft sich an einen breiten Leserkreis richtet (Entwickler, Auftraggeber, Benutzer, ...), muss jede Lesergruppe den Inhalt des Pflichtenheftes verstehen können. Die Hauptabschnitte sind hier abgebildet. Selbstverständlich kann im Projekt das Inhaltsverzeichnis und damit auch der Inhalt des Pflichtenheftes um weitere Punkte ergänzt werden.

Zum Inhaltsverzeichnis

Das Inhaltsverzeichnis ist wie eine "Hauptsteuerleiste" aufzufassen. Es sorgt dafür, dass sowohl beim Lesen wie auch beim Schreiben die logische Struktur des Ganzen jederzeit erkennbar bleibt. Die Nummerierung sollte deshalb in Abschnitten, Unterabschnitten, usw. erfolgen. Die Abschnittsüberschriften sollen möglichst aussagekräftig formuliert sein. Eine Dokumentation ohne Inhaltsverzeichnis ist nicht annehmbar. Die Gliederungspunkte erscheinen mit Angabe der Seitennummer. Ihnen stehen zwei Möglichkeiten der Gestaltung offen:

1. Das Inhaltsverzeichnis ist vollständig, es enthält also alle Gliederungspunkte der Dokumentation.
2. Das Inhaltsverzeichnis (es folgt dem Deckblatt) enthält nur die Hauptgliederungspunkte, die Unterpunkte erscheinen am Anfang des jeweiligen Kapitels in dem Kapitelinhaltsverzeichnis.

Zur Nummerierung der Seiten:

- Alle Seiten werden fortlaufend nummeriert, beginnend ab Inhaltsverzeichnis.

- Die Seitennummerierung beginnt in jedem Kapitel neu. Dann ist es jedoch zweckmäßig, die Kapitelnummer in die Seitennummer mit einzubeziehen. (z.B. 4-12 =Kapitel 4, lfd. Seite 12)
- Die Nummerierung beginnt in jedem Dokument neu. (Pflichtenheft, Entwurfsdokumentation, Installationshandbuch)

Achten Sie darauf, dass nach der Nummer „1“ auch eine Nummer „2“ kommen muss; ansonsten stimmt Ihre Logik der Kapitelbildung nicht!

Hinweis:

Nutzen Sie die Hilfestellung von Word, Ihnen aus den Überschriftenformaten (die Sie natürlich einsetzen müssen), automatisch ein Inhaltverzeichnis zu generieren.

Es folgt nun eine kurze Beschreibung der einzelnen Teile des Pflichtenhefts.

1 Aufgabenstellung und Einführung

Zuerst ist die Aufgabenstellung des Auftraggebers abzulegen. Handelt es sich um ein Folgeprojekt, sind die Änderungen/Erweiterungen gegenüber dem Vorgänger genau zu beschreiben. Anschließend wird in die Thematik eingeführt. Das Softwareprodukt steht im Mittelpunkt der Beschreibung. Der fachliche Grund für die Entwicklung, sowie die zu erreichenden Vorteile werden ebenso herausgearbeitet wie die noch fehlenden Teile. Die Einführung gibt dem Leser die Möglichkeit, sich einen ersten Überblick über die Aufgaben und Leistungen des Produktes zu verschaffen.

2 Produktumgebung

2.1 Anwendungsbereiche

Um einen falschen Eindruck zu vermeiden, müssen die geplanten Anwendungsbereiche soweit wie möglich eingeschränkt werden. Darüber hinausgehende Einsatzmöglichkeiten, die eventuell eine Anpassung des Produkts oder der Organisation der Produktumgebung erfordern, sollen auf jeden Fall genannt werden.

2.2 Anwendergruppen

Es werden die zukünftigen Anwender des Systems beschrieben. Gibt es mehrere Anwender-(Benutzer-) gruppen, so wird für jede dieser Gruppen eine eigene Beschreibung erstellt (Benutzerprofil). Konkret bedeutet dies:

- die Aufgaben der Benutzer
- seine problem- und fachbezogenen Kenntnisse
- seine Wünsche und Anforderungen
- von ihm zu beachtende Vorschriften und Einschränkungen (ges.Vorschriften, Zugriffsbeschränkung, Datenschutzbestimmungen, ...)

2.3 Basismaschine

Die Basismaschine beschreibt die HW- und SW-Umgebung unter denen das Produkt an seinem Einsatzort lauffähig ist. Dabei sind die Minimalkonfiguration und die wünschenswerte Konfiguration deutlich zu unterscheiden. Dazu zählen sowohl der Rechner als auch Peripheriegeräte. Die SW-Umgebung beschreibt alle zum Betrieb des Produktes benötigte Software. Zur Darstellung eignet sich eine Tabelle. Für die Gestaltung der Tabelle orientieren Sie sich bitte an gängigen Softwareprodukten. Sie finden auf jeder Herstellerseite und auf jeder Verpackung die Systemanforderungen an die Software. Meist besteht die Auflistung aus zwei Spalten: minimale Systemanforderungen und optimale Systemanforderungen. Diese Auflistung ist wichtig, damit der Auftragnehmer entscheiden kann, ob die erforderlichen Systemanforderungen gegeben sind oder ob er gegebenenfalls seine Infrastruktur ausbauen muss (bzw. kann oder will).

2.4 Mengengerüst

Das Mengengerüst beschreibt Mengenangaben zu den Daten, anfallenden Belegen etc. Das Mengengerüst liefert Kenndaten zum Speicherplatzverbrauch und zu den Peripheriegeräten. Bedenken Sie hier, dass in der elektronisch erfasste Daten zehn Jahre gespeichert werden müssen. Planen Sie in Ihre Kalkulation also nicht nur die Größe eines Datensatzes ein, sondern extrapolieren Sie auf voraussichtliche Datensätze pro Jahr multipliziert mit zehn. Der Auftragnehmer muss die Möglichkeit haben, die Daten auch speichern zu können.

3 Produktmodell

Das Produktmodell bildet den Schwerpunkt des Pflichtenhefts. Es beschreibt das System aus Benutzersicht. Es wird das „**was**“ beschrieben und keinesfalls das softwaretechnische „**wie**“. An dieser Stelle ist auch eine Aufteilung in Muss- und Kannkriterien zu treffen. Überlegen Sie sich, welche Funktionalität das Produkt

besitzen **muss**, um lauffähig zu sein und welche Funktionalität das Produkt zusätzlich besitzen **kann**.

Zur Darstellung des Produktmodells eignen sich folgende Darstellungsmethoden der UML:

- Use-Case-Diagramme (wer tut was)
- Aktivitätsdiagramme (welche Abläufe existieren im Produkt)
- Entity-Relationship-Model (natürlich nur für Datenbankprojekte)

Die Auswahl sollte problemgerecht vorgenommen werden. Eine Kombination der Darstellungsmittel ist erlaubt, wenn es für die Aufgabenstellung sinnvoll erscheint. Zu jedem Datenbankproblem ist zwingend ein ERM zu erstellen. Zu einem ERM gehört immer auch ein Mengengerüst, damit die voraussichtliche Tabellengröße und damit der Platzbedarf für die Speicherung (und Sicherung) der Daten berechnet werden kann.

3.1 Anwendungsfälle der Software

Das Use Case Diagramm ist eine erste, grafische Strukturierung des gesamten Systems. Es beschreibt die Hauptaufgaben, die Benutzergruppen (Akteure) mit dem System erledigenmüssen. Aus den Use-Cases und deren Verfeinerung können Sie die Muss-Kriterien ableiten. Externe Schnittstellen zu existierenden oder geplanten SW-Systemen werden hier ebenfalls mit Hilfe von Akteuren beschrieben. Im Anhang finden Sie eine Schablone, die Ihnen bei der Beschreibung der einzelnen Use Cases helfen wird.

3.2 Ablaufanalyse

Aktivitätsdiagramme formulieren die Abläufe des Systems auf hohem abstrakten Niveau. Hier sind noch keine feinen algorithmischen Beschreibungen gefordert, sondern die Abfolge der Tätigkeiten und Prozesse müssen beschrieben werden. Typischerweise wird für jeden Use Case mindestens ein Aktivitätsdiagramm angelegt. Sehen Sie daher Aktivitätsdiagramme als Verfeinerung der Use Cases. Mit Aktivitätsdiagrammen stellen Sie die Interna der Use Cases differenziert dar.

3.2 Strukturelle Datenanalyse

Anhand der identifizierten Aktivitäten und deren Gruppierung (beispielsweise mit Swimlanes) können Klassen für ein erstes Klassendiagramm identifiziert werden. Das erste Klassendiagramm beschreibt die fachlichen Klassen des Systems und ihre Zusammenhänge. Hier werden alle Informationen über eine Klasse (d.h. ihre Attribute und Methoden) zusammengetragen. Der fachliche Inhalt, für den eine Klasse im Modell steht, wird für jede Klasse dokumentiert. Diese Dokumentation kann beispielsweise tabellarisch für jede Klasse erfolgen. Für jede Klasse sind die bereits identifizierten Attribute und Methoden inklusive einer informalen Funktionsbeschreibung anzugeben. Programmiersprachenspezifische Klassen (wie beispielsweise Forms in .NET oder ActionListener in Java) brauchen an dieser Stelle noch nicht berücksichtigt zu werden. Das findet an der Schnittstelle Entwurf/Implementierung statt.

4 Qualitätsanforderungen

Hier werden alle Anforderungen des Auftraggebers zusammengefasst, die sich auf die Qualität des gesamten Produktes wie auch auf Teile des Produktes beziehen können.

Beispiel :
Benutzerfreundlichkeit
Datenschutz
Wartbarkeit
Kompatibel zu ...
Datenintegrität (speziell bei DB-Anwendungen)
usw.

5 Benutzerschnittstelle

Unter diesem Stichwort wird die Entwicklung der Oberfläche für menschliche Benutzer und der Schnittstellen zu externen Systemen beschrieben. Für Multimediaklassen gilt: Die Dokumentation des Faches „Entwicklung von Internetanwendungen“ sollte hier eingesetzt werden. Bei der Implementierung eines Softwareprojekts können an dieser Stelle Oberflächenprototypen erstellt und mit dem Auftraggeber diskutiert werden (denken Sie dabei an das iterative Wasserfallmodell). Zur Erstellung der Prototypen eignet sich beispielsweise der Form-Designer des Microsoft Visual Studio. Hier können Sie mit geringem Zeitaufwand Prototypen „zusammenklicken“.

5.1 Benutzermodell

Ein Benutzermodell gibt an, wie die menschlichen Benutzer und externen Systeme (Akteure der Use Cases) an den Use Cases teilnehmen. Es muss der Informationsfluss zwischen Akteur und System konkretisiert werden (Schnittstellenbeschreibung). Use Cases, die ein Benutzer aufruft, können durch einen Menüpunkt oder ein Dialogfenster in der Anwendung repräsentiert werden. Zusätzlich zum Prototypen der Benutzerschnittstelle können Sie sich an dieser Stelle Gedanken über zu übergebende Daten bzw. Datenstrukturen machen. Beispiel: Für einen Use Case „Kundendaten erfassen“ bietet sich sicherlich ein Dialogfeld mit Textfeldern an, in denen der Benutzer später die für den Vorgang relevanten Daten eingibt.

5.2 Kommunikationsstrategie

Sie stellt dar, in welcher Form und mit welchen E/A-Geräten die Benutzerkommunikation abläuft (z.b. Menü, Kommandosprache, Spracheingabe ...). Für jede Benutzergruppe muss die passende Strategie entschieden, festgelegt und beschrieben werden. Die Kommunikationsstrategie kann auch hilfreich bei der Erstellung des Benutzermodells sein. Daher sollten Sie den Punkt 5 eher parallel als sequentiell abarbeiten.

5.3 Kommunikationsaufbau

Hier wird das Konzept für die Gestaltung der Oberfläche beschrieben. Zu Beginn muss ein grundlegendes Konzept (grafischer Aufbau der Bildschirme, grundsätzliche Gestaltungselemente, Farbwahl) entwickelt werden (Stichwort: Screendesign). Diese Konzept und seine Konsequenzen wird an ausgewählten Beispielen verdeutlicht und erläutert. Die Abfolge der Bildschirmsequenzen wird durch Zustandsdiagramme oder Aktivitätsdiagramme beschrieben.

6 Entwicklungsumgebung

Hier wird die zu verwendende Programmierumgebung (Sprache, Compiler etc.) festgehalten. Sind Produkt- und Entwicklungsumgebung nicht identisch, dann müssen die Unterschiede präzise beschrieben werden, damit die Funktionalität und Robustheit des Programms gewährleistet sind.

7 Literaturverzeichnis

Das Literaturverzeichnis ist ein wichtiger Bestandteil einer Dokumentation. Es enthält die benutzte, bzw. notwendige System- und Fachliteratur. Die Nennung der Systemliteratur ist wichtig, weil späteres Arbeiten mit der Dokumentation nur korrekt wird, wenn man den zugrundeliegenden Stand der Systemsoftware berücksichtigt. Die Fachliteratur ist zu nennen, um sich nicht mit fremden Federn zu schmücken. Hierzu gehören:

- problembezogene und EDV-Fachaufsätze
- Lehrbücher über einzusetzende Methoden und Verfahren
- Herstellermanuals
- Richtlinien des Auftraggebers

8 Abkürzungsverzeichnis

Hier sollen alle im Pflichtenheft verwendeten projekt- und firmenspezifischen Abkürzungen und die ihnen entsprechenden vollständigen Bezeichnungen aufgelistet werden. Gängige Abkürzungen der deutschen Sprache, beispielsweise „usw.“ oder „z. B.“, werden nicht aufgenommen.

9 Sonstiges

Hier wird das aufgeführt, was sich keinem der Kapitel 1 bis 9 zuordnen lässt, bzw. die Teile, die nicht einem Kapitel der Dokumentation allein zuzuordnen sind, sondern für alle oder mehrere Kapitel gelten. Es ist im Einzelfall zu entscheiden, ob eine Redundanz von Information (also gleiches an verschiedenen Stellen der Dokumentation zu belassen = doppelt und dreifach) oder Unterbringung in Sonstiges der Lesbarkeit der Dokumentation dienlich sind.

2.3 Anhang zum Pflichtenheft

Schablone zur Erstellung von Use Cases

Geschäftsprozess:	Name (Was wird getan?)
Ziel:	Globale Zielsetzung bei erfolgreicher Ausführung des Geschäftsprozesses
Kategorie:	primär, sekundär oder optional
Vorbedingung:	Erwarteter Zustand, bevor der Geschäftsprozess beginnt

Nachbedingung bei Erfolg: Erwarteter Zustand nach erfolgreicher Ausführung des Geschäftsprozesses, d.h. Ergebnis des Geschäftsprozesses

Nachbed. Fehlschlag: Erwarteter Zustand, wenn das Ziel nicht erreicht werden kann

Akteure: Rollen von Personen oder andere Systeme, die den Geschäftsprozess auslösen oder daran beteiligt sind

Auslösendes Ereignis: Wenn diese Ereignis eintritt, dann wird der Geschäftsprozess initiiert

Beschreibung: **1** Erste Aktion
2 Zweite Aktion ...

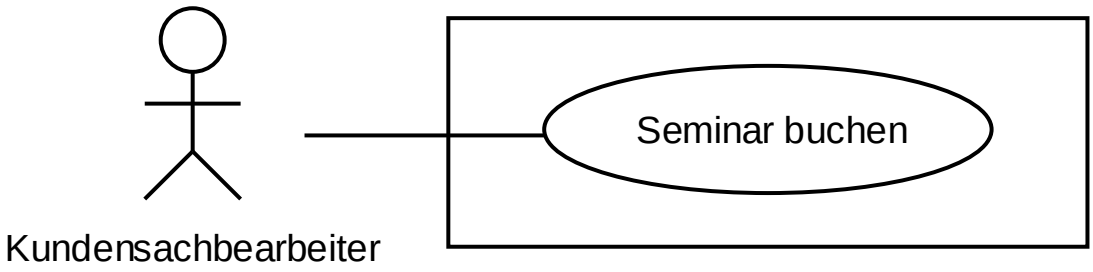
Erweiterungen: **1a** Erweiterung des Funktionsumfangs der ersten Aktion

Alternativen: **1a** Alternative Ausführung der ersten Aktion
1b Weitere Alternative zur ersten Aktion

Der betrachtete Geschäftsprozess kann nur ausgeführt werden, wenn die genannte **Vorbedingung** erfüllt ist. Die **Nachbedingung** eines Geschäftsprozesses A kann für einen Geschäftsprozess B eine Vorbedingung bilden. Diese Angaben bestimmen also, in welcher Reihenfolge Geschäftsprozesse ausgeführt werden können.
Unter **Beschreibung** erfolgt eine umgangssprachliche Spezifikation des Geschäftsprozesses. Die einzelnen Aufgaben werden der besseren Übersicht halber nummeriert. Wichtig ist, dass hier zunächst der **Standardfall**, d.h. der Fall, der am häufigsten auszuführen ist, beschrieben wird. Alle seltener eingesetzten Fälle werden unter **Erweiterungen** ausgeführt, wenn sie zusätzlich zu einer Aktion der Standardverarbeitung ausgeführt werden und **Alternativen**, wenn sie eine Aktion der Normalverarbeitung ersetzen.

Die Kategorie eines Geschäftsprozesses ist

- primär, wenn wer notwendiges Verhalten beschreibt, das häufig benötigt wird,
- sekundär, wenn er notwendiges Verhalten beschreibt, das selten benötigt wird,
- optional, wenn er ein Verhalten beschreibt, das für den Einsatz des Systems zwar nützlich, aber nicht unbedingt notwendig ist.



Geschäftsprozess: Buchen: Von Anmeldung bis Buchung

Ziel: Anmeldebestätigung und Rechnung an Kunden geschickt

Kategorie: primär

Vorbedingung: keine

Nachbedingung bei Erfolg: Kunde ist angemeldet

Nachbed. bei Fehlschlag: Mitteilung an Kunden, dass Veranstaltung ausgebucht ist, ausfällt oder nicht existiert

Akteure: Kundensachbearbeiter

Auslösendes Ereignis: Anmeldung des Kunden liegt vor

Beschreibung:
1 Kundendaten abrufen
2 Veranstaltung prüfen
3 Anmeldebestätigung und Rechnung erstellen

Erweiterungen:
1a Kundendaten aktualisieren
1b Wenn Kunde Mitarbeiter einer Firma ist, dann Firmendaten erfassen bzw. wenn vorhanden, dann abrufen und aktualisieren
1c Zahlungsmoral überprüfen

Alternativen:
1a Neukunden erfassen
2a Auf alternative Veranstaltungen hinweisen, wenn ausgebucht
2b Mitteilung „falsche Veranstaltung“ ausgeben, wenn Veranstaltung nicht existiert

3 Entwurfsdokumentation

3.1 Allgemeines

Die Entwurfsphase schließt sich der Analysephase an. Der Input dieser Phase ist das Pflichtenheft. Während im Pflichtenheft das „WAS“ untersucht wurde, geht es bei der Entwurfsdokumentation um die Lösung des „WAS“, also dem „WIE“.
Die Aufgabe der Entwurfsphase ist die Erarbeitung eines detaillierten Modells, welches der späteren Realisierung der funktionalen und qualitativen Produktanforderungen, wie sie im Pflichtenheft beschrieben sind, dient. Das Entwurfsdokument ist damit also die Basis für das Entwicklungsteam während der Implementierungsphase und sollte alle für die Umsetzung relevanten Details enthalten. Das Autorenteam sollte sich den Adressaten immer vor Augen halten – es geht nicht um eine Fleißarbeit, sonder darum, Verständnis über das zu schaffende System zu dokumentieren. So reicht es oft aus, bei ähnlich aufgebauten Modulen nur eines detailliert zu beschreiben, bei den anderen dagegen nur auf die Spezifika einzugehen. Setzt das Projekt auf einem bestehenden System auf, so ist zunächst der Ist-Zustand zu beschreiben und deutlich abzugrenzen, welche Bestandteile durch das Projekt zu leisten sind.
Neben den funktionalen Anforderungen sind die nichtfunktionalen Anforderungen und Qualitätsmerkmale bestimmend für den Entwurf. So entstehen Qualitätsmerkmale wie Änderbarkeit, Wartbarkeit, Wiederverwendbarkeit oder Robustheit nicht automatisch, diese müssen im Entwurf mit berücksichtigt werden. Auch sollte im Entwurf auf Einfachheit geachtet („keep it simple“) und nicht jedes Problem verallgemeinert und generisch gelöst werden. Um das Rad nicht immer neu zu erfinden und auf bewährte Vorgehensweisen zurück zu greifen, sollte immer der Einsatz von Entwurfsmustern geprüft werden.

3.2 Das Vorgehen

In der Regel ist die Gesamtheit der Anforderungen sehr komplex. Um diese Komplexität strukturiert zu bewältigen, sind folgende Arbeitsschritte zu durchlaufen:

- Softwarearchitektur: Sinnvolle Zerlegung des Gesamtsystems in Teilsysteme / Schichten mit Beschreibung der Abhängigkeiten der Teilsysteme. Gängige Architekturmuster sind dabei hilfreich. . Eine gute Übersicht über Architekturmuster finden Sie in dem Buch „Design Patterns“ von Erich Gamma.

- Grobentwurf: Zerlegung der Teilsysteme in Module und Festlegung derer Schnittstellen. Auch an dieser Stelle bieten Architekturmuster einen guten Rahmen.
- Feinentwurf: Entwurf der Algorithmen, Datenstrukturen, ER-Diagramme und Tabellenbeschreibungen zur Bereitstellung der durch die Modulschnittstellen definierten Funktionalität. Bedenken Sie bitte, dass Sie die im Pflichtenheft erarbeiteten Modelle verfeinern und nicht „das Rad neu erfinden“.

Im Folgenden werden diese drei Arbeitsschritte und deren Dokumentation näher erläutert.

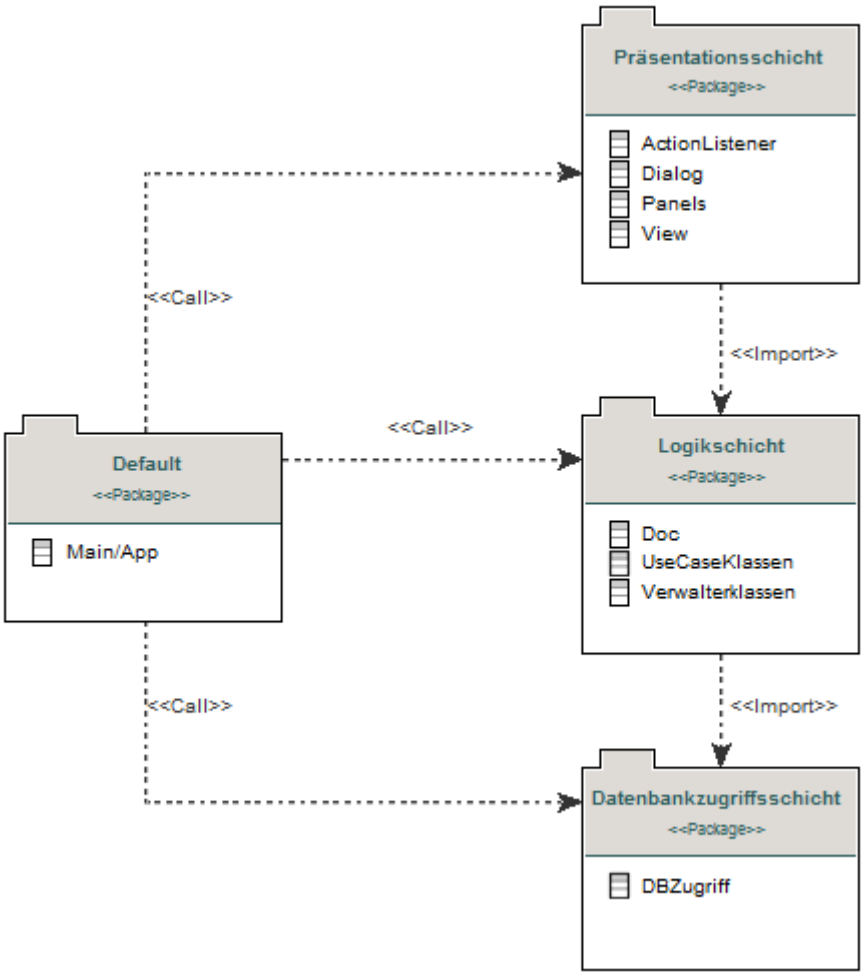
3.2.1 Die Softwarearchitektur

Um die Komplexität der Anforderungen zu reduzieren, wird zunächst eine Zerlegung in kleinere, überschaubare Einheiten erarbeitet, welche gut in die technische Umgebung (Hardware, Software) einzubetten sind. Sie haben diese Vorgehensweise im Softwareengineering als „Divide & Conquer“ kennen gelernt. In diesem Arbeitsschritt wird also die Softwarearchitektur bestimmt. Als gängige Hilfsmittel dienen Entwurfsmuster (bewährte Lösungsschemata für wiederkehrende Entwurfsaufgaben) wie zum Beispiel:

- Schichtenarchitektur
- MVC Muster
- Serviceorientierte Architektur (SOA)
- Einsatz von kommerziellen oder Open-Source Frameworks
- …

Die Softwarearchitektur beschreibt die grundlegenden Komponenten und deren Zusammenspiel innerhalb eines Softwaresystems. Im Rahmen der Entwicklung repräsentiert die Softwarearchitektur die früheste Design-Entscheidung. Eine einmal eingerichtete Softwarearchitektur ist später nur mit hohem Aufwand änderbar. Die Entscheidung über ihr Design ist somit eine der kritischsten und wichtigsten Punkte im Entwicklungsprozess einer Software.

Zur Darstellung dieses Sachverhaltes eignen sich UML-Paketdiagramme, alternativ können Komponentendiagramme und Verteilungsdiagramme verwendet werden. In einem Paket werden Klassen oder auch selbst wieder Pakete zusammengefasst, die funktional oder logisch in einem engen Zusammenhang stehen. Pakete bilden Einheiten, die zusammenhängend entwickelt und implementiert werden sollten. Folgendes Beispiel stellt das Paketdiagramm einer typischen Schichtenarchitektur dar:



Die in der Präsentationsschicht verwendeten Klassen sind zum Teil stark programmiersprachenabhängig. Der in der vorangegangenen Abbildung dargestellte „ActionListener“ existiert beispielsweise in Java – nicht in .NET. Spätestens an dieser Stelle des Entwurfs muss Ihnen also klar sein, welche Entwicklungsumgebung eingesetzt wird. Die entworfenen Pakete des Systems werden dann anhand folgender Aspekte in der Dokumentation näher beschrieben:

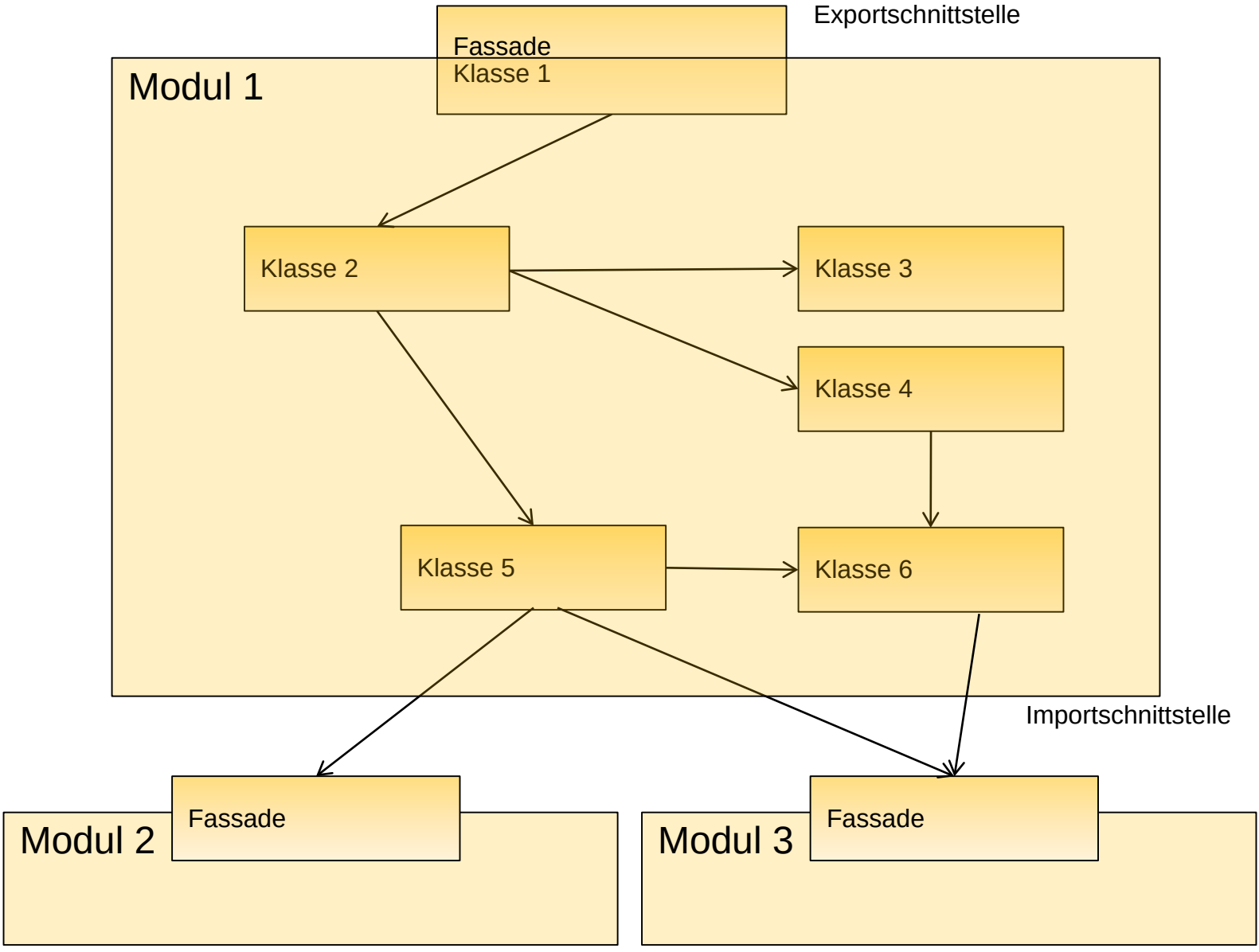
- Beziehungen der Pakte in einer Übersicht
- Aufgabenbereich des Pakets
- Schnittstelle zu anderen Paketen

Wird die zu erstellende Software in ein bestehendes System integriert, sollten die Schnittstellen zu den vorhandenen Soft- und Hardwarekomponenten aufgelistet und detailliert beschrieben werden.

3.2.2 Der Grobentwurf

Im nächsten Schritt werden die erarbeiteten Pakete näher untersucht und in sinnvoll geschnittene Module (nicht weiter unterteilte Komponenten) zerlegt. Die Kommunikation eines Moduls mit der Außenwelt erfolgt dabei über eindeutig spezifizierte Schnittstellen (Exportschnittstelle, Importschnittstelle). Eine solche Programmierschnittstelle definiert eine Menge von Methoden und sollte folgende Aspekte berücksichtigen:

- Rückgabewerte, Argumente, Datentypen, Ausnahmen
- Vorbedingungen und Nachbedingungen
- Beschreibung der Funktionalität und gegebenenfalls der nichtfunktionalen Eigenschaften



Als Zeichen für eine gute Modularisierung gilt eine geringe Modulkopplung, also möglichst wenig Schnittstellen zwischen den Modulen. Dieser Sachverhalt ist z.B. durch das Facade-Pattern zu erreichen. Die Fassade stellt die Schnittstelle des Moduls nach außen dar und delegiert die Methodenaufrufe an die internen Objekte. Dieser Sachverhalt trägt auch einer besseren Testbarkeit Rechnung. Überlegen Sie sich beim Entwurf direkt, welche Testfälle es für die jeweiligen Module geben wird: welche sinnigen Daten werden an die Module übergeben, welche Eingaben kann der Benutzer tatsächlich vornehmen. Sie erkennen so auch gleich, welche Fehlerfälle Sie abprüfen müssen. Neben der sehr wichtigen Beschreibung der Schnittstellen werden der Funktionsumfang jedes Moduls und deren Zusammenspiel exakt beschrieben. Bei Bedarf können Sequenz- und Zustandsdiagramme für einzelne Module erstellt werden, die dann in der nächsten Phase weiter verfeinert werden.

3.2.3 Der Feinentwurf

Jedes herausgearbeitete Modul wird während der Feinentwurfsphase näher untersucht und tiefergehend beschrieben. Dabei ist in dieser Phase, im Gegensatz zur Analyse, die Verwendung findende Programmiersprache von Bedeutung, da diese starken Einfluss auf das Feindesign hat (bestehende Bibliotheken, Datentypen...).

- Wichtige Abläufe und Algorithmen werden erarbeitet oder verfeinert, falls diese im Pflichtenheft bereits in einer ersten groben Form erstellt wurden. Zur anschaulichen Visualisierung dienen UML-Aktivitätsdiagramme, erweitert durch aussagekräftige textuelle Beschreibungen. Auch Pseudocode kann an dieser Stelle hilfreich sein, um Algorithmen zu beschreiben. Bereits bekannte Algorithmen müssen Sie nicht ausführlich beschreiben. Die Funktionsweise der Sortieralgorithmen „Quicksort“ oder „Bubblesort“ sind hinreichend bekannt. Ein einfacher Verweis, dass dieser Algorithmus eingesetzt wird, reicht aus. Wofür er eingesetzt wird (warum wird sortiert, was wird sortiert), das sind die Fragen, die einer Klärung bedürfen.
- Das in der Analysephase entstandene Klassendiagramm wird verfeinert, indem die bereits erarbeiteten Fachklassen durch weitere technische Klassen angereichert werden. Dies betrifft z.B. Klassen für das Userinterface (z.B. Klassen der Swing-Bibliothek) oder für interne Datenhaltung (Vektoren, Hashtables, Baumstrukturen, Anbindung an Persistenzschicht,..). Dazu wird der für dieses Modul relevante Teil des Klassendiagramms der Analyse in einem separatem Klassendiagramm weiter verfeinert und um die neu entworfenen Klassen angereichert.
- Falls für bestimmte Ablaufsequenzen das Zusammenspiel verschiedener Klassen oder Module von Bedeutung ist, sollte dieser Sachverhalt durch Sequenzdiagramme und textueller Beschreibung veranschaulicht werden.

- Durch Zustandsdiagramme kann das Verhalten von Klassen oder Modulen näher beschrieben werden.

3.3 Aufbau der Entwurfsdokumentation

Die Entwurfsdokumentation dient dem Softwareentwicklungsteam als Grundlage für seine Arbeit. Sie sollte so aufgebaut sein, dass die Softwareentwickler einen schnellen Überblick erhalten und alle relevanten Sachverhalte möglichst einfach finden. Sprachlich ist sie an diese Personengruppe anzupassen. Zunächst wird die erarbeitete Architektur beschrieben. Die Ergebnisse der Phasen Grob- und Feinentwurf werden in den darauf folgenden Kapiteln zusammen dokumentiert, indem jede Komponente detailliert erläutert wird.

Die Entwurfsdokumentation sollte nach folgendem Schema aufgebaut sein:

1. Softwarearchitektur
    1.1 Technische Infrastruktur
    1.2 Paketdiagramm
    1.3 Schnittstellen zu Nachbarsystemen
2. Paketbeschreibung
2.1. Paket 1
    2.1.1 Beschreibung der Exportschnittstelle
    2.1.2 Modul 1
2.1.2.1 Beschreibung der Exportschnittstelle
2.1.2.2 Entwurfsklassendiagramm
    2.1.2.2.1 Beschreibung Klasse 1
    2.1.2.2.2 Beschreibung Klasse 2
    2.1.2.2.3 …
2.1.2.3 Aktivitätsdiagramme
2.1.2.4 Sequenzdiagramme
2.1.2.5 Zustandsdiagramme
2.1.3 Modul 2
2.2. Paket 2
2.2.1 Beschreibung der Exportschnittstelle
2.2.2 Modul 1
…
3. Datenstrukturen
    3.1 Datenbankentwurf
        3.1.1 ER-Modell
        3.1.2 Tabellenbeschreibung
        3.1.3 SQL-Abfragen
    3.2 Dateibeschreibung
        3.2.1 Einsatzzweck
        3.2.2 Dateiaufbau

Das erste Kapitel „Softwarearchitektur“ beschreibt anhand eines Paketdiagramms die grobe Aufteilung in Teilsysteme und erläutert deren Aufgaben und Interaktion. Das Entwicklerteam soll dadurch einen schnellen Überblick über den grundsätzlichen Aufbau des Systems erhalten. An dieser Stelle ist für das Zusammenspiel der einzelnen Klassen untereinander ein Sequenzdiagramm hilfreich.
Kapitel zwei untersucht die einzelnen Pakete und deren Module näher. Folgende Inhalte sollten enthalten sein:

- Name des Moduls
- Aufgabe des Moduls
- Exportschnittstelle: Methoden mit Parametern, die von anderen Paketen benutzt werden. Sie legt fest, welche Methoden dieses Paket nach außen bekannt macht, d.h. exportiert, wo sie dann von anderen Paketen benutzt werden.
- Importschnittstelle: Methoden mit Parametern, die dieses Paket von anderen Paketen benutzt. Sie legt fest, welche Methoden dieses Paket aus anderen Paketen zur eigenen Verwendung benötigt und daher importieren muss.
- Funktionalitäten
- Normalverhalten
- Fehlerverhalten mit Liste der Fehler, die auftreten können.
- Integritätsbedingungen
- Klassendiagramm: Der Teil des Klassendiagramms aus dem Entwurf, der für das Paket notwendig ist, wird angegeben. Typischerweise umfasst dieses Klassendiagramm einen Ausschnitt aus dem gesamten Klassendiagramm des Entwurfs, das ggf. um weitere, pakettypische und für den Entwurf notwendige Klassen angereichert wurde.
- Verfeinerung von Abläufen: Weiterhin werden die Abläufe innerhalb des Paketes und komplexere Methoden von Klassen aus diesem Paket mit Hilfe von Aktivitäts-, Sequenz-, Zustandsdiagrammen oder Pseudocode weiter verfeinert und konkretisiert.

Kapitel drei befasst sich mit der Datenhaltung. Ausgehend vom Klassendiagramm oder einem daraus entwickelten ERM ist ein Entwurf der notwendigen Dateien, XML Strukturen und Tabellen vorzunehmen und zu dokumentieren. Alle Dateien und Tabellen mit ihren Feldern, Typen, Wertebereichen und evtl. Integritätsbedingungen (z.B. referentielle Integrität) sind zu beschreiben. Komplexe Datenbankabfragen sind mit SQL zu formulieren und zu beschreiben.

Beispiel einer Paketbeschreibung:

Mit Hilfe eines Paketes „Würfel“ sollen zufällige Augenzahlen von 1 bis 6 ermittelt und an die Oberfläche weitergegeben werden.

Paketname :	Würfel
Aufgabe des Paketes:	Das Paket dient dazu, ganze Zahlenwerte aus dem Intervall 1 bis 6 zufällig zu ermitteln und als Augenzahl eines Würfels an die Oberfläche weiterzugeben.
Exportschnittstelle:	int erzeugeAugenzahl ( ), Methode, mit der dieses Paket von anderen Paketen aufgerufen wird.
Importschnittstelle:	Methode aus einer Klasse zur Erzeugung von Zufallszahlen entweder aus der Java- oder aus der C++-Klassenbibliothek. Methode, die das betrachtete Paket in anderen, hier dem Bibliothekspaket, aufruft.
Integritätsbed.:	Die Ergebnisaugenzahl muss ganzzahlig sein.
Funktionalität:	Die Bibliotheksklasse aus dem Paket Bibliothek liefert eine rationale Zahl zwischen 0 und 1. Diese wird auf eine ganze Zahl zwischen 1 und 6 abgebildet und als Augenzahl an das Paket der Oberfläche weitergegeben.
Normalverhalten:	Weitergabe der Augenzahl an die Oberfläche.
Fehlerverhalten:	keine
benutzte Pakete:	Bibliotheken der Programmiersprache

4 Implementierungsdokumentation

4.1 Allgemeines

Die Implementierung des Softwareproduktes wird durch den Quell-Code selbst beschrieben. Dazu sollte der Quellcode ausreichend kommentiert sein, um sachverständigen Personen Wartungsarbeiten zu erleichtern. Besonderer Wert ist auf konsistente Namensvergabe in Analyse, Entwurf und Implementierung zu legen. Im Idealfall muss eine Klasse der Analyse bis in die Implementierung hinein nahtlos verfolgbar sein. Desweiteren werden grundlegende Informationen der Implementierung außerhalb des Quelltextes in einem separatem Kapitel erläutert.

4.2 Vorgehensweise

Aus der Entwurfsdokumentation wird die Implementierung abgeleitet. Es ist darauf zu achten, dass die Namen im Entwurf und der Implementierung eindeutig sind. Um den nachfolgenden Programmierer (Wartungsphase) zu unterstützen, sollte der Quellcode ausreichend mit Kommentaren versehen sein. Zusätzlich sollten für Attribut-, Variablen- und Methodennamen usw. aussagekräftige Bezeichner vergeben werden.

4.3 Dokumentationskommentare

Die meiste Dokumentation wird direkt durch den Entwickler fortgeschrieben. In wichtigen Programmiersprachen (wie z.B. Java, C#) gibt es auch Dokumentation, die aus dem Quellcode automatisch erzeugt wird [4, S. 75]. Dadurch wird die Programmstruktur in einer übersichtlichen Weise interaktiv abrufbar dargestellt. Die zu Grunde liegende Information lässt sich durch spezielle Kommentartypen (Javadoc für Java [3, S. 782f], XML-basiert für C# [2, S 359ff]) ergänzen.
Automatisch generierte Dokumentation kann am leichtesten mit der Entwicklung des Quellcodes aktuell gehalten werden. Sie sollte regelmäßig während der Entwicklung und nach Abschluss des Projekts erzeugt werden und dem Projekt auf einem Datenträger beigefügt werden.
Durch die neue Formatierung der Quellcodekommentare können auch Unzulänglichkeiten offensichtlich werden. Der Entwickler sollte die erzeugte Dokumentation auf angemessenen Umfang und Beschreibungsniveau, auf Gleichartigkeit in Form und Begrifflichkeit und auf Schreibfehler überprüfen.

Alle in anderen Klassen zugreifbaren Klassen, Methoden und Attribute müssen mit Dokumentationskommentaren kommentiert werden.

Kommentierung von Klassen [1, S. 819ff]

Beschreiben Sie die Zuständigkeit der Klasse innerhalb des Gesamtprojekts, d.h. welchen Zweck die Klasse innerhalb der Software erfüllt. Beschreiben Sie den Typ der Klasse (Fachklasse aus der OOA, Listenklasse zur Verwaltung von Objekten, Oberflächenklasse, algorithmisch geprägte Klasse, usw.). Welchen Einschränkungen unterliegt die Klasse in ihrer Verwendung? Verweisen Sie auch auf andere Klassen, die in ihrer Verwendung zur aktuellen Klasse in einem starken Zusammenhang stehen.

Zu jeder Klasse gehört die Angabe des Autor und der Versionsnummer.

Kommentierung von Methoden [1, S. 814ff]

Beschreiben Sie die Aufgabe der Methode. Welchen Typ und welche Bedeutung besitzen die Parameter? Welche Einschränkungen im Wertebereich der Parameter gibt es, wenn der korrekte Ablauf des Algorithmus gewährleistet sein soll (Vorbedingungen)? Beschreiben Sie Bedeutung des Rückgabewerts und seinen Wertebereich bei korrektem Ablauf des Algorithmus (Nachbedingungen). Wie reagiert die Methode auf fehlerhafte Parameterwerte?

Gibt es auch eine Wertrückgabe über die Parameter oder andere Seiteneffekte der Methode? Gibt es Abhängigkeiten zu anderen Methoden derselben Klasse, z.B. eine geforderte Aufrufreihenfolge?

Triviale Methoden, wie z.B. get- und set-Methoden werden möglichst knapp kommentiert.

Kommentierung von Attributen

Die Bedeutung eines zugreifbaren Attributs sollte besonders dann in einem Kommentar erläutert werden, wenn der Name allein nicht genügend aussagekräftig ist. Außerdem sollten eventuelle Beschränkungen des Wertebereichs und gegebenenfalls die Einheit des Attributs angegeben werden.

4.4 Aufbau

1 Entwicklungsumgebung
- 1.1 Verzeichnisstruktur des Quellcodes
- 1.2 Konfigurationen
- 1.3 Bibliotheken

2 Programme
- 2.1 Allgemeines
- 2.1 Paket / Modul 1
- 2.2 Paket / Modul 2
- …
- 2.n Paket / Modul n

4.5 Quellen

[1] Eine ausführliche Anleitung zur Dokumentation von Quellcode findet sich in Steve McConnell, ‚Code Complete', Microsoft Press, Deutsche Ausgabe der Second Edition, 2004

[2] Jesse Liberty, ‚Programmieren mit C#', O'Reilly, Deutsche Ausgabe der 2. Auflage, 2002

[3] Michael Inden, ‚Der Weg zum Java-Profi', dpunkt.verlag, 1. Auflage, 2011

[4] Michael Hüttermann, ‚Agile Java-Entwicklung in der Praxis', O'Reilly, 1. Auflage, 2008

[5] Peter Rechenberg, Gustav Pomberger (Hrsg.), ‚Informatik-Handbuch', Hanser, 3. Auflage, 2002

5 Testdokumentation

5.1 Aufgabe

Die Testphase spielt in vielen Softwareentwicklungsprozessen eine große Rolle und trägt einen entscheidenden Beitrag zur Qualitätsverbesserung der entstehenden Software. Testaktivitäten sollten frühzeitig begonnen werden und laufen als paralleler Prozess während der gesamten Entwicklungszeit. Alle Testaktivitäten von der Planung bis zur Abnahme der Software werden in der Testdokumentation festgehalten.

5.2 Vorgehensweise

Das Testdokument sollte während oder kurz nach der Erstellung des Pflichtenhefts begonnen werden. Am Anfang stehen Überlegungen „was“ soll „wie“ getestet werden und welche Vorbereitungen müssen dafür getroffen werden. Die Testfälle für den Abnahmetest leiten sich zum großen Teil aus der Modellierung der Use Cases ab. Weitere Testfälle für den Komponententest können nach dem Entwurf der einzelnen Softwarekomponenten und ihren Schnittstellen spezifiziert werden. Zur Unterstützung des Testprozesses sollte über den Einsatz von Testwerkzeugen nachgedacht werden (z.B. JUnit, NUnit, …).
Während der Implementierung können bereits fertiggestellte Komponenten anhand der spezifizierten Testfälle getestet werden. Werden hierfür Testklassen programmiert, so sind sie Teil der Software und sollten ebenfalls dokumentiert werden. Erneute Tests sind nach Änderung der Software durch Testklassen schnell möglich. Es folgen mit sukzessiver Integration der einzelnen Komponenten der Integrationstest und schließlich am fertigen Produkt der Abnahmetest. Die Durchführung aller Tests sollte in Testprotokollen festgehalten werden, die auch Teil der Dokumentation sind.

5.3 Aufbau

1 Testplan
  1.1 Teststrategien der Teststufen
  1.2 Testumgebung
  1.3 Testwerkzeuge
  1.4 Testdaten
2 Testfälle

2.1 Funktionale Tests
2.2 Nichtfunktionale Tests
3 Testprotokolle

Zu 1: Testplan

An dieser Stelle sollte beschrieben werden, wie der Softwaretest angegangen und durchgeführt werden soll. Für jede geplante Teststufe (Komponententest, Integrationstest, Systemtest, Abnahmetest) muss entschieden werden, welche Strategie für den Test verfolgt werden soll und welche Vorbereitungen dafür nötig sind. Die Testumgebungen in denen die Tests stattfinden sollen müssen vorbereitet und deren Konfiguration dokumentiert werden. Dasselbe gilt für die verwendeten Testwerkzeuge. Für die Durchführung der Tests werden Testdaten benötigt. Hierfür können Datenbankskripte vorbereitet werden. Bei Tests mit Echtdaten ist aus Datenschutzgründen zu beachten, dass diese vor den Tests anonymisiert werden müssen.

Zu 2: Testfälle

Bei den funktionalen Tests handelt es sich um alle Testmethoden, die anhand des Ein-/Ausgabeverhaltens die Software testen. Die dafür benötigten Testfälle werden anhand der funktionalen Anforderungen an das System (siehe Pflichtenheft) bzw. der einzelnen Schnittstellen (siehe Entwurfsdokumentation) spezifiziert. Die Beschreibung eines Testfalls enthält:

- einen eindeutigen Bezeichner für den Testfall (z.B. Testfallnummer),
- die Anforderungen, die getestet werden,
- die Vorbedingungen für die Durchführung des Tests,
- die einzelnen Testschritte und Eingabedaten und
- das erwartete Ergebnis.

Beim Komponententest kann neben bzw. an Stelle von der tabellarischen Beschreibung der Testfälle, ein Entwurf der benötigten Testklassen stehen. Neben den funktionalen Tests ist auch der Test der nicht funktionalen Anforderungen wichtig. Sie testen die Qualität des Systems und beeinflussen stark die Zufriedenheit der Kunden und Anwender. Nicht funktionale Tests sind u.a. Lasttest, Performanztest und Test unterschiedlicher Systemkonfigurationen.

Zu 3: Testprotokoll

Alle Testaktivitäten sind zu protokollieren. Hierfür sollte eine einheitliche Protokollvorlage erstellt werden. Im Protokoll wird u. a. festgehalten, welche Tests, wann und von wem durchgeführt wurden. Für jeden Testfall ist das Ergebnis des Testdurchlaufs festzuhalten. Im Fehlerfall - das Testergebnis entspricht nicht dem erwarteten Ergebnis – muss der Fehler klassifiziert und weitere Maßnahmen beschrieben werden.

6 Benutzerdokumentation

6.1 Aufgabe

Das Benutzerhandbuch ist eine Aufbereitung des Pflichtenheftes mit dem Ziel, Gebrauch und Verhalten des Produktes den Benutzern zu erläutern. Ein Benutzerhandbuch ist genau an eine Benutzergruppe anzupassen. Existieren mehrere Benutzergruppen, so ist für jede ein gesondertes Handbuch zu erstellen. Ein Benutzerhandbuch ist in seinem Sprach- und Abstraktionsniveau auf die angesprochene Benutzergruppe zuzuschneiden. Die Fachsprache des Softwareentwicklers ist zu vermeiden.

Am Anfang eines Benutzerhandbuches steht immer eine Einleitung mit den Hinweisen für den Gebrauch des Handbuches. Diese umfasst Angaben über die Zielsetzung der Software, typische Anwendungsfälle, die benötigten Vorkenntnisse sowohl für die Benutzung des Produkts als auch zum Verstehen des Handbuches und eine Übersicht über Abhängigkeiten der Kapitel des Benutzerhandbuches untereinander.

Den Abschluss bildet ein Glossar, in dem die Einzelfunktionen des Produkts beschrieben sind (Nachschlagewerk). Graphische Darstellungen (z. B. Masken-hierarchie), Skizzen und Tabellen unterstützen die Beschreibung.

Prägnante Beispiele erhöhen die Verständlichkeit und die Akzeptanz des Benutzers. Vielfach ist es günstig, das Benutzerhandbuch auf der Grundlage eines Beispiels aufzubauen und so dem Leser die Möglichkeit zu geben, die Anwendung praktisch nachzuvollziehen. Warnungen, die auf mögliche Fehlbedienungen hinweisen, sind ebenfalls notwendiger Bestandteil des Benutzerhandbuches. Sie sollten optisch deutlich hervorgehoben werden. Häufige Wiederholungen gemeinsamer Abläufe sind unerwünscht. Auf diese Weise wird der Text gestrafft.

6.2 Vorgehensweise

Die Benutzerdokumentation ist ein wichtiger Bestandteil der gesamten okumentation. Sie entscheidet häufig über den Einsatz der Software. Auch ein noch so gutes Programm ist nicht oder kaum benutzbar, wenn es an der Benutzerdokumentation mangelt. Was ein Handbuch "taugt", kann man bereits an seiner Gliederung erkennen.

Die Gliederung des Inhaltsverzeichnisses entscheidet über das Finden bzw. Nichtfinden von Informationen. Jede Überschrift erweckt beim Benutzer eine Erwartungshaltung, welche Art von Informationen ihn im zugehörigen Textabschnitt erwarten. Eine übersichtliche und vor allem klare Gliederung als schnelle Orientierungshilfe ist deshalb ein wichtiges Qualitätskriterium für die Benutzerdokumentation.
Allerdings sieht eine gute Gliederung nicht immer gleich aus. Je nach Aufgabenstellung, Funktionen und Zielgruppe (Benutzergruppen siehe Benutzermodell) arbeitet man mit unterschiedlichen Ansätzen:

- produktorientierter Ansatz
- funktionsbezogener Ansatz
- benutzerorientierter Ansatz
- lernorientierter Ansatz

Jede Gliederungstechnik präsentiert dem Benutzer das Produkt aus einem speziellen Blickwinkel.

6.2.1 Produktorientierter Ansatz

Der Blickwinkel dieser Gliederungstechnik liegt auf den Produktfunktionen. Beispiel : Stoppuhr (produktorientierte Gliederung)

- Knopf A
- Knopf B
- Knopf C
- Anzeige

Der Leser erfährt aus der Gliederung nichts über die verschiedenen Anwendungsmöglichkeiten der Stoppuhr, sondern nur etwas über die Komponenten des Produkts. Diese Art der Gliederung ist nicht grundsätzlich schlecht, sie hat ihre bestimmten Anwendungsgebiete. (Referenzhandbuch z.b. Cobol Sprachbeschreibung)

6.2.2 Funktionsorientierter Ansatz

Hier ergibt sich der Ansatz aus den Funktionen und den Anwendungen, die das Produkt ermöglicht und die der Anwender kennt. Beispiel : Stoppuhr (funktionsorientierte Gliederung)

- Zeit stoppen
- Zeit einstellen
- ....

Dieser Ansatz erscheint in der Praxis viel zu selten. Häufig verstecken sich hinter Komponentennamen Anwendungen. Wenn z.B. unter der produktorientierten Überschrift "Bedienelemente" die Funktion "Gerät ein- und ausschalten" beschrieben wird, muss der Benutzer erst kombinieren, bevor er einschalten kann.
Das, was er in der Bedienungsanleitung sucht, nämlich wie man das Gerät ein - und ausschaltet, findet er nur auf Umwegen und nicht mit Hilfe der Überschrift. Das bedeutet, die Informationen sind "versteckt" im Inhaltsverzeichnis (schlechte Qualität der Dokumentation). Der anwendungsorientierte Ansatz ist benutzerfreundlicher als der produktorientierte, er gibt jedoch kaum Hinweise über den zeitlichen Ablauf einer Handlung.

6.2.3 Benutzerorientierter Ansatz

Hier ergibt sich der Ansatz aus dem zeitlichen Verlauf der Anwendung. Dem Benutzer werden Handlungsketten chronologisch beschrieben. Beispiel : Stoppuhr (benutzerorientierte Gliederung)

- Zeit einstellen
- Monat einstellen
- Tag einstellen
- Stunden einstellen
- Minuten einstellen
- Sekunden einstellen

Der chronologische Ablauf bestimmt die Gliederung des Handbuches. Den benutzerorientierten Ansatz der Gliederung erkennt man an der Reihenfolge und der Beziehungen einzelner Überschriften der Gliederung zueinander.

6.2.4 Lernorientierter Ansatz

Hier ergibt sich der Ansatz aus der Erwartungshaltung des Anwenders. Die Produktlogik tritt hierbei in den Hintergrund. Den Aufbau bestimmt die zentrale Frage: Wo und in welchem Zusammenhang sucht der Benutzer die gewünschten Informationen? Mit welcher Erwartungshaltung tritt der Benutzer an das Produkt heran?
Der lernorientierte Ansatz bedeutet konkret:
Die wichtigsten und am häufigsten gebrauchten Funktionen stehen am Anfang der Anleitung. Wesentliches ist vom Unwesentlichen getrennt. Technische Daten und

Garantieerklärungen sind nach hinten verbannt. Der Lernteil kommt vor die Problembeschreibung und den speziellen Tricks und Tipps.
Diese Beschreibung kommt, wenn sie optimal gelöst ist, sicherlich dem Benutzer entgegen. Sie hat allerdings den Nachteil, dass die Beschreibung viel zu umfangreich wird und deshalb den Benutzer wieder abschreckt und so den Ansatz zunichte macht. Die Benutzerlogik ist kein Dogma. Ist beispielsweise bei einer Stoppuhr zuerst die Uhr- und Weckfunktion einzustellen, bevor die Zeit gestoppt werden kann, so erfordert die Produktlogik eine Beschreibung in dieser Reihenfolge, auch wenn in der Erwartungshaltung des Benutzers die Funktion Zeit stoppen an erster Stelle steht.
Mischformen der vier Gliederungstechniken in einem Handbuch sind oft sogar notwendig, um dem Anwender die Bedienung des Produktes klar zu machen. Es ist wichtig, dass diese Ansätze bewusst und funktional eingesetzt werden. Das kann man bereits erkennen, wenn man sich das Inhaltsverzeichnis eines Handbuches genauer anschaut.

6.3 Aufbau

Der Aufbau des Benutzerhandbuches richtet sich stark am Produkt selbst aus (siehe Vorgehensweise). Deshalb kann an dieser Stelle nur ein allgemeines Beispiel gegeben werden.

1. Über dieses Handbuch
   1.1 Gegenstand
   1.2 Zielgruppe
   1.3 Gebrauch des Handbuches
2. Einleitung
3. Produktbeschreibung
   (siehe oben: Vorgehensweise)
4. Bedienungsanleitung
5. Beispielanwendung
6. Glossar

6.4 Online-Dokumentation

Unter Benutzerdokumentation wird gewöhnlich nur das gedruckte Benutzerhandbuch verstanden. Als Ergänzung zu diesem sind das Hilfesystem und die Online-Dokumentation zu sehen. Man unterscheidet mehrere Arten von Online-Dokumentation:

- elektronische Bücher (Dokumentation auf CD, Diskette ...)

- Online-Hilfe (Help-Funktionen)
- Computer Based Training
- Einsatz von Hypertext- und Hypermedia

Wichtig ist die Konsistenz zwischen Software und Dokumentation und nicht die Form und das Medium. Die Merkmale für den Entwurf und die Gestaltung einer Online-Dokumentation können von den Dokumentationsrichtlinien übernommen werden. Die Verwendung von Zugriffen über Stichworte ist der große Vorteil der Online-Benutzerdokumentation.

7 Installationshandbuch

7.1 Aufgabe

Das Installationshandbuch enthält alle Angaben, die zur Inbetriebnahme des Produktes benötigt werden. Das Installationshandbuch ist ähnlich wie das Benutzerhandbuch an das Abstraktionsniveau und an die sprachlichen Mittel der Benutzer anzupassen.

7.2 Aufbau

1. Inhaltsverzeichnis
2. Beschreibung der Dateien
3. Beschreibung der Installation
4. Glossar

2 Beschreibung der Dateien

Für jede Datei sind folgende Angaben zu machen:

- Name der Datei
- Typ der Datei (Programm, Daten, Text)
- Zugriffsart (Lesen, Lesen/Schreiben, Ausführen)
- Aufgabe der Datei bei der Ausführung

3 Beschreibung der Installation

Viele Softwareprodukte erfordern das Einhalten einer bestimmten Vorgehensweise bei der Inbetriebnahme. z.B. Installationsprogramm

- Dateien müssen in einem bestimmten Pfad liegen;
- Betriebssystem-Parameter müssen gesetzt sein
- (z.B. Änderung der config.sys, autoexec.bat, ini-Dateien unter Windows, ...).

4 Glossar

Mögliche Installationsfehler (-meldungen) ...

Mögliche Reaktionen des Benutzers