Softwaredokumentation im Praktikumseinsatz der Abteilung BIS

Simon Johanning, Hans-Gert Gräbe

Version vom 24. November 2016

Zusammenfassung

Dieses Dokument gibt einen Überblick über die Anfoderungen an die Softwaredokumentation im Rahmen der Praktika der Abteilung BIS und ist als Handreichung für Teilnehmer und Tutoren dieser Praktika gedacht.

Inhaltsverzeichnis

1 Wozu dokumentieren? 2

2 Dokumentation und Software-Einsatz 3

2.1 IT-Basis eines Unternehmens und Serviceschichten 3

2.2 Schichtenmodell konzeptionell 4

2.3 Beispiel: Wordpress 4

2.4 Konsequenzen für die Dokumentation 5

2.5 Zielgruppen und Zielstellung der Dokumentation 6

3 Dokumentation im Entwicklungsprozess 7

3.1 Dokumentation der Produktanforderungen 8

3.2 Beschreibung der Softwarearchitektur 9

3.3 Technische Dokumentation 10

3.4 Dokumentation von Softwarefeatures und Anwendungsfällen 11

3.5 Dokumentation im SWT-Praktikum 12

4 Praktische Hinweise 15

1 Wozu dokumentieren?

Das Ziel einer Softwaredokumentation ist es, den erfolgreichen Einsatz von Software zu unterstützen. Das ist im umfassenden Sinne zu verstehen und betrifft nicht nur funktionale Aspekte, sondern auch den Einsatz sowie die Administration und (Weiter-)Entwicklung der Software.

Darunter fällt unter anderem

1. die Darstellung und Bedienung der Features bzw. Funktionalitäten der Software aus der Sicht der *Endnutzer*, um bestimmte Prozesse innerhalb von Anwendungsfällen durchzuführen,

2. Aspekte der Installation und Konfiguration, die zum Ausrollen der Software den *Systemadministratoren* bekannt sein müssen,

3. eine geeignete Darstellung von Informationen zu Komponenten der Software bzw. Funktionen, Klassen etc. aus Sicht der *Entwickler* auf verschiedenen Detaillierungsstufen, um effektiv mit bereits vorhandenen Programmteilen arbeiten oder diese nachnutzen und weiterentwickeln zu können, sowie

4. die Dokumentation von Anforderungen, Entscheidungen und Modellierungen (bspw. Architektur) während der Entwicklung, um Informationen über Softwarearchitektur und Designentscheidungen auch über den Entwicklungskontext hinaus nachhaltig zu kommunizieren.

Schwerpunkt der Dokumentation in den SWT-Praktika sind die Punkte 3. und 4., da die zu entwickelnden Software-Lösungen oft prototypisch sind und über die konkrete Anwenderzielgruppe meist nur spekuliert werden kann. Die Informationen zu den Punkten 3. und 4. werden in der *technischen Dokumentation* zusammengefasst. Wichtig ist dabei vor allem Verständlichkeit auch über den Entwicklungskontext hinaus, wozu es erforderlich ist, im Entwicklungskontext vorhandenes implizites Wissen angemessen zu explizieren.

Der Fokus der Dokumentation liegt daher auf der Weitergabe impliziten Wissens zu Architektur-, Entwurfs- und Design-Fragen sowie zur Benutzung der Software an zukünftige Anwender. Das Projektteam sollte die Dokumente aus diesen Sichten gegenlesen und sich bei jedem Dokument klar werden, für welche Zielgruppe es genau geschrieben ist. Verständlichkeit, Korrektheit und Vollständigkeit stehen im Vordergrund.

Auch wenn die Dokumentation erst zur Auslieferung der Software (also zum Projektabschluss) fertiggestellt sein muss, ist es aus verschiedenen Gründen angebracht, diese entwicklungsnah zu verfassen und fortzuschreiben. So können die Dokumente auch frühzeitig mit Anwendern auf praktische Relevanz und Verständlichkeit abgeglichen, getestet und überarbeitet werden. Es ist für die Qualität der Dokumentation ausschlaggebend, diese evolutionär zu entwickeln und mehrfach (am Feedback der Anwender orientiert) zu überarbeiten.

Die Parallelität von Entwicklung, Dokumentation und Tests wird in den Praktika durch ein iteratives Entwicklungsmodell unterstützt, in dem die Arbeitsergebnisse regelmäßig in Releasebündeln zu integrieren sind.

2 Dokumentation und Software-Einsatz

Im ersten Abschnitt wurde als Ziel einer Softwaredokumentation formuliert, „den erfolgreichen Einsatz von Software zu unterstützen“. In diesem Abschnitt sollen einige Charakteristika des Einsatzes von Software zusammengetragen und zu Dokumentationsaspekten relatiert werden.

2.1 IT-Basis eines Unternehmens und Serviceschichten

Software tritt in den meisten Unternehmen als unterstützende Infrastrukturfunktion für die Kerngeschäftsprozesse in Erscheinung. Es ist heute üblich und aus Kostengründen erforderlich, dazu für einzelne Aufgaben- und Problembereiche die Dienste entsprechend qualifizierter externer Dritter in Anspruch zu nehmen. Der Einsatz von Software wird damit von unterschiedlichen Akteuren auf verschiedenen Serviceschichten verantwortet, die in der Regel unterschiedliche technische Kenntnisse und einen unterschiedlichen Zugang zur Software haben.

Es ist hilfreich, (mindestens) die folgenden drei Serviceschichten zu unterscheiden:

- Nutzung der Software durch Anwender,
- Konfiguration und Anpassung der Software durch Systemadministratoren,
- Softwarewartung, -installation und -erweiterung durch technisch qualifizierte Dritte.

Dies bedeutet, dass Akteure der ersten Schicht mit einer Softwareinstanz arbeiten, die von Akteuren der zweiten Schicht zur Verfügung gestellt wird. Im Vordergrund steht hier die Fachanwendung selbst und damit die *Weiterentwicklung des in der Datenschicht gespeicherten Unternehmens-Zustands*. Dieser Zustand wird von den Akteuren der ersten Schicht verantwortet, wobei diese sich nur längs des *firmenspezifischen Regelwerks* bewegen können, das von Akteuren der zweiten Schicht durch spezielle Konfiguration eingestellt wurde. Akteure der dritten Schicht verwenden und verändern die Software, um den Akteuren der zweiten Schicht konkrete Instanzen zur Verfügung zu stellen, und *verändern damit die Einstellungsmöglichkeiten im firmenspezifischen Regelwerk*.

Somit verwenden Akteure der ersten Serviceschicht eine konkrete Softwareinstanz, um einen Datenbestand über die Zeit weiterzuentwickeln. Für diese ist die Softwareinstanz von der Nutzung her etwas Konstantes, und die laufende Softwareinstanz wird als Werkzeug benutzt, um (mehr oder weniger standardisierte) Anwendungsfälle abzuarbeiten.

Akteure der zweiten Schicht modifizieren Softwareinstanzen, die sie mit verschiedenen Komponenten (mit einer potentiell großen Anzahl von Versionen) konfigurieren und (als etwas Konstantes) den Akteuren der ersten Schicht übergeben. Diese *im Betrieb garantierte Unveränderbarkeit der Regeln* wird oft durch einen entsprechenden Zustandsübergang in der Software selbst getriggert, wenn diese zwischen Konfigurations- und Betriebsmodus unterscheidet und damit eine Veränderung überhaupt nur möglich ist, nachdem die Software aus der Verfügbarkeit der ersten Schicht genommen wurde.

Akteure der dritten Schicht hingegen führen den Entwicklungsbedarf zusammen und erstellen instantiierbare Software, arbeiten aber in der Regel nicht mit diesen Instanzen. Hier wird Funktionalität zur Verfügung gestellt, die in anderen Kontexten genutzt wird.

2.2 Schichtenmodell konzeptionell

Von den oben genannten Schichten sind die ersten zwei Schichten eng mit dem operativen Kerngeschäft der Anwender der Software verbunden. In die Verantwortung dieser beiden Schichten fallen insbesondere das Anlegen und Pflegen der Daten, auf denen die Software operiert.

Unter dem Gesichtspunkt von Software als (zustandsloser) Funktionalität, die auf Daten (Zuständen) operiert, liegt die Verantwortung für die *Ergebnisse* der Anwendung der Software, die sich in gezielt und angemessen veränderten Zuständen der innerbetrieblichen Datenbasis manifestieren, beim Anwender der Software. Konzeptionell wird damit eine Trennung von Zustand und Verhalten erreicht, wobei der Anwender dafür verantwortlich ist, dass die Software gültige Zustände der Daten fortschreibt. Wirtschaftlich wird dies auch dadurch begründet, dass die Daten unmittelbar mit dem operativen Kerngeschäft des Anwenders verbunden sind.

Für Software als (zustandslose) Funktionalität ist – wie allgemein bei Funktionen in der Informatik – zwischen Definition und Anwendung zu unterscheiden. Setzt der Anwender auf eine externe Software-Entwicklung, so wird hier ein Geschäftsverhältnis begründet, bei dem sich der Anwender des Software-Entwicklers als Erfüllungsgehilfen bedient. Im Rahmen von Service Level Agreements verpflichten sich die Entwickler als externe Dritte, die Verfügbarkeit dieser Funktionalität einschließlich der Schulung von Personal des Anwenders zu sichern.

Die Differenz zwischen spezifizierter Funktionalität (Beschreibungsebene) und praktischer Ausführung (Anwendungsebene) lässt sich dabei aus prinzipiellen Gründen nicht vollständig überbrücken. Diese Differenz muss sich der Anwender grundsätzlich zunächst als Geschäftsrisiko zurechnen lassen, das er nur durch entsprechende Vertragsgestaltung bis zu einem gewissen Grad auf die Software-Entwickler abwälzen kann. Im Kontext einer IT-Landschaft mit einer Vielzahl von Anwendungen verschiedener Hersteller ist es allerdings bereits oft schwierig, Fehlfunktionen konkreten Anwendungen und damit konkreten Dritten zuzuordnen. Im Zeitalter von „Software aus Komponenten“ werden diese Abhängigkeitsstrukturen noch einmal deutlich komplexer.

Die Schicht der Softwarewartung und -installation repräsentiert das Betreiben der Software auf der Ebene der Funktionalität. Hierzu zählen Wartung, Pflege und Weiterentwicklung der Software. Neben der Weiterentwicklung von Funktionalität kann dies auch die *Weiterentwicklung des Datenmodells* beinhalten, womit auf das Servicelevel 2 durchgegriffen wird, da das Datenmodell zum Regelwerk gehört, welches die Strukturierung der Unternehmensdaten bestimmt. Wird auch diese Entwicklungsarbeit ausgelagert, so ergeben sich deutlich komplexere Abhängigkeitsverhältnisse zwischen dem Anwender als Auftraggegber und dem Software-Entwickler als Auftragnehmer. Die konkrete Ausgestaltung eines solchen Geschäftsverhältnisses ist abhängig von der Expertise des Anwenders und führt meist dazu, dass der Auftragnehmer auch einen Teil der Anforderungen auf dem Servicelevel 2 verantwortet.

2.3 Beispiel: Wordpress

Diese abstrakten Ausführungen zu Serviceleveln sollen am Beispiel eines Wordpress Systems erläutert werden. Die Schicht der *Nutzung der Software* wird hier von Autoren von Artikeln (i.d.R. Blogs) realisiert. Diese schreiben den tatsächlichen Inhalt und sind mit der Software in einem gewissen Zustand (ihrer Arbeitsumgebung) vertraut. Diese Inhalte sind in der Regel

rein redaktioneller Natur und folgen etablierten Workflows, mit denen *Zweck und Ziel* der entsprechenden Website realisiert werden sollen. Dies ist also die Ebene der täglichen Arbeit, die darin besteht, die Datenbasis mit Daten zu füttern.

Die zweite Schicht (*Konfiguration der Software*) wird von den Administratoren realisiert. Diese sind nicht direkt für die Inhalte verantwortlich, sondern sorgen dafür, dass die Autoren die definierten Arbeitsabläufe durchführen können und dafür geeignete Konzepte und Werkzeuge zur Verfügung stehen. Dies umfasst in der Regel die Verwaltung der Datenbank wie z. B. das Anlegen von Benutzern, das Konfigurieren von Plugins und Themes, das Aktualisieren dieser und ähnliche Abläufe. Weiterhin sind die Akteure dieser Schicht dafür verantwortlich, den Anwendern in der ersten Schicht eine Softwareinstanz zur Verfügung zu stellen, die ihren Arbeitsabläufen entspricht. Das kann bspw. bedeuten, dass verschiedene Anwender verschiedene Plugins oder Pluginkonfigurationen haben oder die richtigen Themes (oder Childthemes) zur Verfügung gestellt bekommen. Selbstverständlich können zwischen diesen Serviceschichten personelle Überschneidungen auftreten. Es ist dennoch sinnvoll, diese konzeptionell zu trennen.

Unter die Schicht der *Softwarewartung und -installation* fallen alle Aufgabenbereiche, die damit zu tun haben, die Funktionalität selbst bereitzustellen und weiterzuentwickeln. Im Falle einer Wordpress-Instanz ist dies in der Regel die Einrichtung und Aktualisierung der Instanz selbst sowie Entwicklungs- oder Weiterentwicklungsarbeiten an Plugins oder Themes. Dieses geschieht selbstverständlich in Zusammenarbeit mit den Anwendern und den von ihnen *definierten* Anforderungen an die Software, um Arbeitsabläufe umzusetzen. Die Verantwortung für die *Umsetzung* dieser Anforderungen liegt jedoch beim Entwicklerteam.

2.4 Konsequenzen für die Dokumentation

Da auf Grund der Verantwortung für das operative Geschäft die Verantwortung für die ersten beiden Schichten in der Regel beim Anwender liegt, stehen diese Level bei der Softwaredokumentation nicht im Vordergrund. Zwar ist der Auftraggeber für die Organisation von Schulungen der Anwender der Software verantwortlich, vereinbart aber mit den Entwicklern der Software oft die Bereitstellung von Schulungsmaterialien oder die inhaltliche Absicherung der Schulungen durch kompetentes Personal. Da der Fokus unserer Praktika auf der Erstellung von Software liegt, bleiben derartige Aspekte bis auf die Frage der *Erstellung eines Anwenderhandbuchs* in der Regel außer Betracht.

Für die Entwickler von Software sind somit die Dokumente, die mit der dritten Schicht in Verbindung stehen, von zentraler Bedeutung. Insbesondere unterscheiden sich die Vermittlungsformen der Dokumentation für die dritte Schicht deutlich von denen der ersten beiden Schichten. In der Regel wird der Wissenstransfer (im Sinne der Nachhaltigkeit) über den Anwender an Dritte verlaufen. Dieses kann ein weiteres Entwicklerteam, das in-House Wartungspersonal des Anwenders oder ein dem ursprünglichem Entwicklungsteam ähnliches Team sein. Da sich oft auch personelle Zusammensetzungen während eines Projekts ändern, kann ein solcher Wissenstransferbedarf auch innerhalb eines Teams entstehen. Es ist also in jedem Fall angebracht, die Dokumentation für Dritte zu schreiben.

Werden Komponenten entwickelt, steht der Prozess der Informationsübermittlung im Vordergrund. Die Zielgruppe für Komponenten kann grob in *Komponentenentwickler* und *Komponentenmonteure* unterteilt werden. Hierbei sind die *Komponentenentwickler* diejenigen, wel-

che die Komponenten weiterentwickeln, dazu auf technische Detailkenntnisse zu Architektur und Entwurf der vorliegenden Komponente angewiesen sind und somit einer *technischen Dokumentation* bedürfen (siehe dort), während die *Komponentenmonteure* die Komponente in die Arbeitsumgebung des Anwenders deployen sowie das Zusammenspiel der Komponenten untereinander gewährleisten müssen. Daher sind diese stärker an Fragen der Konfiguration und Installation interessiert. Es empfiehlt sich, für diese Gruppen separate Dokumentationen oder Dokumente zu erstellen, wobei auch inhaltliche Überlappungen möglich sind.

2.5 Zielgruppen und Zielstellung der Dokumentation

Die Dokumentation setzt sich aus einer Reihe von Dokumenten unterschiedlicher Zielstellung, Detaillierungsgrad und Umfang zusammen. Wie bereits oben erwähnt wird häufig zwischen (mindestens) drei Zielgruppen für die Dokumentation unterschieden, wobei diese unterschiedliche Aufgaben, Interessen und technischen Hintergrund haben.

So steht bei Akteuren, welche der ersten Schicht zuzuordnen sind (Generierung und Editierung von Inhalten), die *Benutzung* der Software im Vordergrund, insbesondere die direkte Benutzung der Software und ihrer Features. Häufig ist bei diesen Akteuren wenig technische Expertise vorhanden. Weiterhin sind für diese Anwender nur die Features relevant, die für die Bewältigung von Arbeitsabläufen tatsächlich benötigt werden.

Akteure, welche der zweiten Schicht zuzuordnen sind (welche also die Software konfigurieren), benötigen ebenfalls keinen tieferen Einblick in die technischen Details der Software und sind auch vorrangig an den Features der Software interessiert. Im Gegensatz zu Akteuren der ersten Schicht sind diese jedoch mit technischeren Arbeitsabläufen befasst und sind damit potentiell an *allen* Features interessiert, welche die Software bietet, um den Leistungsumfang der Software zu überblicken und diese sachgerecht an den Anwendungskontext anpassen zu können. Weiterhin müssen sie in der Lage sein, die Software zu konfigurieren und damit einen Einblick in diese Prozesse haben. Somit sind Akteure dieser Schicht an Überblicksdokumenten zur Software (wie bspw. zur *Softwarearchitektur*) interessiert.

Akteure auf der dritten Schicht (*Wartung, Installation und Weiterentwicklung der Software*) benötigen die umfangreichste und detaillierteste Dokumentation. Für diese stehen die technischen Details der Software (wie *Architekturkonzepte, Design Pattern, Code, Algorithmen, Schnittstellen, APIs*) im Vorderung des Interesses. Sie benötigen einen genaueren Überblick über die Software und das Zusammenspiel der einzelnen Komponenten sowie Informationen über Details zu spezifischen Aspekten der Software. Daher ist es wichtig, dass in den Dokumenten verschiedene Abstraktionsebenen adressiert werden. Für das Navigieren zwischen verschiedenen Abstraktionsebenen (bspw. von der Softwarearchitektur zu einer bestimmten Funktion in einer Komponente) ist eine sinnvolle *Verlinkung* der Informationen hilfreich.

Es kann sinnvoll sein, auch innerhalb dieser drei Gruppen weiter zu differenzieren, da verschiedene Benutzer in verschiedenen Situationen verschiedene Bedürfnisse haben. Ein wesentlicher Aspekt ist auch die *Zugänglichkeit* der Informationen, also die Möglichkeit, einfach und zielgruppenangemessen erschließen zu können, wo welche Informationen wie zu finden ist. Sehr hilfreich sind hierbei Querverweise innerhalb oder zwischen Dokumenten, am besten in Form von Hyperlinks. Die Art der Präsentation sowie der Detaillierungsgrad und die Wahl der Sprache sollten dem Benutzertyp angepasst sein. So sind beispielweise Video- oder integrier-

te Tutorials für Anwender in der ersten Schicht sinnvoll, um sich mit dem System vertraut zu machen, während detaillierte Referenzen eher von erfahrenen Benutzern oder Entwicklern genutzt werden. Zur Zielgruppenangemessenheit der Informationen gehört auch, dass die intendierte Zielgruppe (bzgl. Information und Informationspräsentation) für den Benutzer unmittelbar ersichtlich ist.

3 Dokumentation im Entwicklungsprozess

Wie bereits oben angemerkt, ist die Dokumentation über verschiedene Dokumente je nach Ziel, Zielgruppe und Detaillierungsgrad verteilt. Unterschiedliche Formen sind hierbei:

- Dokumentation der **Produktanforderungen**.
- Eine Beschreibung der **Softwarearchitektur**.
- Eine **technische Dokumentation** einschließlich der Darstellung und Begründung von Entwurfsentscheidungen.
- Die **Dokumentation von Softwarefeatures und typischer Anwendungsfälle** für Nutzer der Software.

Die Dokumentation besteht in den meisten Fällen aus einer größeren Anzahl von Artefakten in unterschiedlichen Formaten und verteilt sich bspw. auf *Hilfesystem*, *Codedokumentation*, *Anwenderhandbuch*, *Architekturdokumentation*, *Administrationshandbuch*, *Installationshandbuch*, *Entwurfsbeschreibung*, *Anforderungsanalyse*, verwendet verschiedene *Dokumentationswerkzeuge* (bspw. *JavaDoc*) und enthält zusätzliche Informationen wie *Protokolle von Arbeitstreffen* etc.

Für eine gute Erschließbarkeit dieser Artefakte ist oft ein **Dokumentationsguide** als Metadokument hilfreich, in dem diese einzelnen Bestandteile, deren Intention und Format genauer erläutert werden.

Im Regelfall sollten zu einem Softwaresystem wenigstens die folgenden Dokumentationen vorliegen:

- Ein **Anwenderhandbuch** und/oder **Hilfesystem** für Nutzer der Software im laufenden Betrieb.
- Ein **Administrationshandbuch** für Administratoren der betriebsbereiten Software.
- Ein **Installationshandbuch** für die lokale Vorbereitung der Software zum Betrieb in einer Systemumgebung mit den dafür erforderlichen Ressourcen.
- Eine **technische Dokumentation** für spätere Phasen, in denen Dritte die Software warten, anpassen oder weiterentwickeln sollen.

Mit Blick auf das Praktikumsziel spielt im SWT-Praktikum die *technische Dokumentation* eine zentrale Rolle, da dort genauer darzustellen ist, auf der Basis welcher software-technischer Prinzipien und Konzepte die Aufgabenstellung umgesetzt wurde.

3.1 Dokumentation der Produktanforderungen

Eine genauere Erfassung der *Produktanforderungen* spielt für die Entwicklung einer Software eine große Rolle, um die Richtung der Entwicklung festzulegen, Priorisierungen von Stories vorzunehmen sowie Abnahmekriterien der Komponenten bzw. Features zu spezifizieren. Die **Dokumentation der Produktanforderungen** soll auf strukturierte Weise deutlich machen, was die Software leisten soll bzw. leistet. Die Darstellung dieser Anforderungen ist in eine Darstellung des Anwendungskontexts der Software einzubetten, aus der Rahmenbedingungen und Motivation für die Entwicklung der Software deutlich werden.

Im SWT-Praktikum werden die Produktanforderungen im Zuge der Analyse der Aufgabenstellung erfasst und genauer spezifiziert. Dabei ist es erforderlich, sich mit der Anwendungsdomäne vertraut zu machen, die dort verwendeten Konzepte und Begrifflichkeiten zu erfassen und einen Übersetzungsprozess zwischen dieser domänenspezifischen und informationstechnischer Terminologie zu organisieren. Dabei gilt es, in der Anwendungsdomäne vorgegebene konzeptionelle Rahmenbedingungen einerseits und bestehende Gestaltungsspielräume andererseits zu identifizieren, um auf dieser Basis ein realistisches Portfolio von Produktanforderungen zu formulieren, das im Weiteren umgesetzt werden soll.

Im SWT-Praktikum ist diese Aufgabe Gegenstand des **Recherchеberichts**, mit dem neben einer ersten Annäherung an Rahmenbedingungen und Gestaltungsspielräume teamintern ein *gemeinsames Vokabular* fixiert werden soll, mit dem angemessen über die Produktanforderungen kommuniziert werden kann. Dieses *Glossar* dient dazu, Begriffe an vereinbarte Bedeutungen zu binden, und sollte besonders solche Begriffe erfassen, die projektrelevant sind, aber in ihrer Bedeutung teamintern missverständlich sind und einer Präzisierung bedürfen.

Auch wenn Produktanforderungen grundsätzlich neutral gegenüber möglichen implementatorischen Details formuliert sein sollen, werden mit der Schärfung der sprachlichen Präzision dennoch erste, noch implizite Modellvorstellungen deutlich. Mit dem **Arbeitsplan** sind die Produktanforderungen weiter zu präzisieren, wobei darauf geachtet werden sollte, die Anforderungen unter Verwendung der im Glossar gegebenen Bedeutungen von Begrifflichkeiten sprachlich zu fixieren. Wenn erforderlich, sollte dabei auch das Glossar selbst weiterentwickelt werden. Dieser einer *expliziten Modellierung* vorgelagerte Aufwand ist eine bewährte Form der Präzisierung des Zielkorridors im Sinne der Scrum-Methodik, der im Gegensatz zu anderen Methoden, etwa einem *Softwareprototypen*, mit deutlich geringerem Ressourceneinsatz auskommt. Die Qualitätssicherung der Ergebnisse in einem Review als Umsetzung des *Prinzips der frühzeitigen Fehlersuche*, wie dies mit dem ersten Review vorgesehen ist, verstärkt einen solchen Effekt noch. Je genauer es gelingt, die Produktanforderungen sprachlich zu fassen, desto einfacher gestaltet sich die spätere Umsetzung.

Beim Erschließen des domänenspezifischen Vokabulars sind intensivere Kontakte mit dem Auftraggeber sinnvoll. Neben konzeptioneller Vermittlungsleistung in das Team hinein ist ein solcher Kommunikationsprozess oft auch hilfreich, um die Aufgabenstellung im Zuge der Fixierung der *Projektvision* weiter zu präzisieren und diese nicht nur unter dem Aspekt des Wünschenswerten zu sehen, sondern auch unter dem Aspekt des technisch und ressourcenmäßig Möglichen.

Gut spezifizierte Produktanforderungen können in einer späteren Phase zu einem **Anwenderhandbuch** weiterentwickelt werden.

3.2 Beschreibung der Softwarearchitektur

Neben konkreten Produktanforderungen aus der Zieldomäne ordnet sich der Einsatz von Software heute stets auch in einen übergreifenden informations-technischen Rahmen ein, der entweder durch den Projektkontext selbst vorgegeben ist oder zur Vereinfachung der Entwicklungsarbeiten im Rahmen der Entwurfsentscheidungen durch das Team bestimmt werden kann und sollte. Die Nutzung einer solchen *Softwarearchitektur* bringt eine Reihe von Vorteilen mit sich:

- bewährte und durchdachte Design Pattern,
- nachnutzbarer Code als Quellcode oder in Bibliotheken,
- etablierte Entwicklungswerkzeuge, -methoden und -standard.

Neben einer solchen Vereinfachung der eigenen Arbeit ergeben sich auch Synergien für die Weiterentwicklung der Software durch Dritte, denen die verwendeten Architekturkonzepte bereits bekannt sind oder in die sie sich an Hand öffentlich verfügbarer Materialien leicht einarbeiten können. Der Nachteil, dass sich das Team selbst in die Details der verwendeten Architektur einarbeiten muss, wird durch diese Vorteile mehr als wettgemacht.

Dieser vom Team zu organisierende Einarbeitungsprozess ist ebenfalls mit der Modellierung verzahnt. Während die Produktanforderungen den Weg zur Modellierung der Geschäftsanforderungen öffnen, öffnet die Auswahl der Softwarearchitektur den Weg zum *Modellierungsrahmen*, der durch konzeptionelle Vorgaben und klar strukturierte Erweiterungspunkte bereits ein bewährtes Grundgerüst für die Implementierung vorgibt. Dieser vorgegebene Rahmen ist projektspezifisch herunterzubrechen und dies in der **Beschreibung der allgemeinen Designentscheidungen** als Teil der **Entwurfsbeschreibung** zu dokumentieren. Hierbei sind so weit wie möglich die Konzepte und Begrifflichkeiten des jeweiligen Architekturkonzepts zu verwenden. Es kann sinnvoll sein, Teile dieser Begrifflichkeiten mit in das eigene *Glossar* zu übernehmen, um den genauen bedeutungsmäßigen Gebrauch dieser Begriffe für den Projektkontext zu fixieren.

Für Dritte ist die Dokumentation der Architektur besonders von Bedeutung, wenn Teile der Software verändert werden sollen und dabei auf APIs oder Protokolle zugegriffen wird. Zur Orientierung und Übersichtlichkeit der Ausführungen zur Softwarearchitektur können auch graphische Darstellungen und Verweise auf weiterführende Erläuterungen in der Literatur verwendet werden. Derartige Formfragen sind allerdings immer der angemessenen Vermittlung der Inhalte untergeordnet.

Es ist darauf zu achten, dass in der Dokumentation der Architektur ein einheitliches Abstraktionsniveau eingehalten wird. Beispielsweise sollte in der Architekturdokumentation eine Funktionalität weder direkt beschrieben werden noch erläutert sein, warum diese Funktionalität in dieser Form umgesetzt wurde. Die Dokumentation sollte sich auf die allgemeinen Anforderungen konzentrieren, welche diese Funktionalität motivieren.

3.3 Technische Dokumentation

Für einen neuen oder geänderten Anwendungskontext muss die Software entsprechend angepasst werden. Dies ist in einfachen Fällen allein mit Bezug auf die Spezifikation möglich, in den meisten Fällen, etwa bei der Bearbeitung von Problemen oder beim Bug Fixing, ist auch eine Weiterentwicklung auf Code-Ebene erforderlich. Hiermit werden meist entsprechend qualifizierte externe Dritte beauftragt, die sic h dazu umfassend in die Konzeption und den Entwurf der Software einarbeiten müssen. Die technische Dokumentation wird dabei als Referenz verwendet und dient technisch versierten Akteuren (Programmierer, Designer, Tester) dazu, die Software zu testen, zu bewerten, weiterzuentwickeln oder Teile der Software nachzunutzen.

Dazu muss die *technische Dokumentation* Sichten auf Konzeption und Entwurf der Software auf verschiedenen Abstraktionsebenen bereitstellen, dass sich ein einschlägig qualifizierter Entwickler eine Roadmap von Dokumenten zunehmender Detailliertheit zusammenstellen kann, um sich die projektspezifische Expertise anzueignen, die erforderlich ist, um einen vorgegebenen Auftrag umzusetzen.

Die technische Dokumentation liegt üblicherweise je nach Detaillierungsgrad in unterschiedlichen Formen vor. Neben einem eigenständigen Einstiegs- und Übersichtsdokument etwa im pdf-Format (im SWT-Praktikum die **Entwurfsbeschreibung**) können wesentliche Teile einer **strukturierten Dokumentation** mit einem *Dokumentationswerkzeug* wie z.B. JavaDoc erstellt und verwaltet werden. Derartige Systeme sind besonders geeignet, schnell und effektiv eine gesuchte konkrete Information zu finden (bspw. Parameter einer Funktion, Funktionalitäten einer Schnittstelle etc.).

Der größte Detailgrad der Dokumentation wird auf der Ebene des kommentierten Quellcodes erreicht. Neben der Anwendung allgemeiner Coding Standards trägt der gezielte Einsatz quelltextnaher **Inline-Kommentare** zu einer deutlichen Verbesserung der Verständlichkeit von Quellcode bei. Auch ist diese Form der Dokumentation gegenüber Code-Änderungen besonders robust, da Code und Dokumentation im selben Dokument stehen und damit Änderungen und Modifikationen entsprechend den im **Dokumentationskonzept** des Teams fixierten Vorgaben synchron an Code und Beschreibung vorgenommen werden können. Dokumentationswerkzeuge wie Javadoc bringen diese Vorteile einer quelltextnahen Dokumentation und eines strukturierten Dokumentationskonzepts zusammen.

Bei dieser Kommentierung ist darauf zu achten, dass sie für Außenstehende verständlich ist und nicht implizite Kenntnisse oder Konventionen voraussetzt, die nur im Kreis der aktuellen Entwickler bekannt oder an anderen Stellen der Dokumentation versteckt sind, die ein Dritter nur mit erheblichem Aufwand findet. Dieses Problem der „Betriebsblindheit“ kann die Verständlichkeit der Inline-Dokumentation deutlich erschweren.

Die verschiedenen Formen der technischen Dokumentation sollten mindestens die folgenden Aspekte erfassen:

- Eine Beschreibung der allgemeinen Prinzipien, die beim Entwurf der Software zur Anwendung kamen (Schichtenarchitektur, bekannte Design- oder Architektur-Pattern), von Strukturierungsaspekten (Aufteilung in Module und Pakete), den damit verfolgten Strukturierungsprinzipien und der strukturellen Bedeutung der einzelnen Einheiten.
- Eine Beschreibung der allgemeinen Prinzipien, die beim Entwurf der einzelnen Einheiten zur Anwendung kamen, und des innerer Aufbaus der Einheiten.
- Eine Beschreibung des Datenmodells der Anwendung auf einer angemessenen Datenab-

straktionsebene.

- Eine Übersicht der definierten Funktionen und Klassen, ihrer Signaturen und einer Beschreibung der bereitgestellten Funktionalität sowie wichtiger Variablen und Konstanten und eine Beschreibung ihrer Verwendung.

3.4 Dokumentation von Softwarefeatures und Anwendungsfällen

Die Erfassung von Softwarefeatures und Anwendungsfällen (Use Cases) dient in einer frühen Entwicklungsphase der Software dem genaueren Verständnis der Aufgabenstellung sowie der sprachlichen Schärfung konzeptioneller und begrifflicher Aspekte im Anfangsstadium der Modellierung. Diese in der Anforderungserhebung identifizierten Softwarefeatures und Anwendungsfälle sind auch für die Test- und Abnahmephase im Sinne des V-Modells interessant und sollten deshalb strukturiert dokumentiert werden. Später lassen sich diese Unterlagen auch zu Tutorien und Schritt-für-Schritt-Anleitungen für den Auftraggeber weiterentwickeln.

Man kann die Dokumentation von Softwarefeatures und Anwendungsfällen gleich als **Anwenderhandbuch** realisieren, welches nach Features, Komponenten oder Anwendungsfällen strukturiert ist. Ein ggf. automatisch erstelltes Inhaltsverzeichnis dient als Orientierung, um einzelne Features oder Anwendungsfälle schnell zu finden. Ein solches Anwenderhandbuch ist eine gute Basis für andere mediale Präsentationsformen wie Videotutorials, mit Screenshots angereicherte Textdokumente oder In-program-Komponenten, die auf einzelne Anwenderzielgruppen speziell zugeschnitten sind.

Für die Dokumentation ist Klarheit und Aktualität sowie Konsistenz und Einfachheit von großer Bedeutung. Es gibt drei wesentlich verschiedene Strukturierungsansätze für derartige Dokumentationen:

- In einer *Anleitung* oder einem *Tutorial* wird der Benutzer nach einem vorgegebenen Szenario Schritt für Schritt durch verschiedene Anwendungsfälle geführt.
- Im *thematischen Ansatz* werden die Informationen thematisch angeordnet. Der Benutzer hat die Möglichkeit, frei zwischen Themen zu wechseln, innerhalb eines Themas sind die Informationen nach einem vorgegebenen Szenario aufbereitet.
- Im *referentiellen Ansatz* sind die Informationseinheiten alphabetisch oder logisch gruppiert; der Benutzer hat umfassende Freiheiten, eigene Navigationswege zu definieren. Oft wird der Benutzer unterstützt, eigene Navigationswege oder Favoriten in einer personalisierten Übersicht zu speichern.

Die Wahl des Ansatzes wird größtenteils durch die Erfahrung des Benutzers bestimmt. Der anleitungsbasierte Ansatz richtet sich vor allem an Benutzer, welche noch keine oder wenig Erfahrungen mit der Software haben, der thematische oder referentielle Ansatz richtet sich dagegen an bereits erfahrenere Benutzer, die eine bestimmte Aufgabe mit der Software lösen möchten oder nach konkreter Information suchen. Strukturierte Informationen im thematischen oder referentiellen Ansatz lassen sich auch in kontextsensitive Hilfesysteme über *help-buttons* und *mouse-overs* einbinden.

Oftmals ist es hilfreich, zu einem Thema mehrere Dokumente für verschiedene Zielgruppen zu schreiben. Die Erstellung einer Dokumentation vollzieht sich im industriellen Kontext häufig in den folgenden Schritten:

- In einer *Benutzeranalyse* werden die Zielgruppen der Software genauer bestimmt und *use-cases* mit konkreten Arbeitsschritten für diese Zielgruppen modellieren.
- Diese Softwarefeatures und Anwendungsfälle bilden die Basis für die Entwicklung der Software. Im Zuge des Dokumentationsprozesses werden Unterlagen für die verschiedenen Zielgruppen *entworfen*.
- Diese Entwürfe werden bei konkreten Probanden des Auftraggebers (bei Auftragssoftware) bzw. im Zuge von Alpha- oder Beta-Tests (bei Produkten für einen anonymen Markt) in *Usability-Tests* eingesetzt und dazu *Feedback gesammelt*.
- Die Unterlagen werden auf Basis dieses Feedbacks weiterentwickelt und finalisiert.

3.5 Dokumentation im SWT-Praktikum

Im SWT-Praktikum spielt – wie oben bereits genauer ausgeführt – die *technische Dokumentation* die zentrale Rolle. Diese technische Dokumentation besteht aus drei Teilen:

1. In der **Entwurfsbeschreibung** als Textdokument mit vorgegebener Gliederung sind alle wichtigen Entwurfsentscheidungen dargestellt und begründet. Glossar und Datenmodell als Teile des Dokuments fixieren den begrifflichen Rahmen und strukturelle Momente der Modellierung, die der Anwendung und ihrer Beschreibung zu Grunde liegen.
2. Eine quelltextnahe **referentielle Dokumentation** kann mit einem Dokumentationswerkzeug in eine webgestützte strukturierte Dokumentation transformiert werden. Die **Dokumentationsstandards** des Projekts sowie die Quelltextnähe garantieren, dass der Quelltext und dieser Teil der Dokumentation synchron entwickelt werden.
3. Die **Inline-Dokumentation** erhöht die Verständlichkeit des Quelltexts.

Projektspezifisch kann die Erstellung weiterer Dokumente wie *Anwenderhandbuch*, *Administrationshandbuch* oder *Installationsanleitung* vereinbart werden.

Dokumentationsstandards

Zur Führung des Dokumentationsprozesses sind im Team **Dokumentationsstandards** verbindlich zu vereinbaren und praktisch anzuwenden. In den meisten Fällen enthalten die eingesetzten Architekturkonzepte oder Frameworks bereits umfangreiche Dokumentationsstandards, auf die sich die projektinternen Dokumentationsstandards beziehen sollten. Die Nachnutzung dieser vielfach bewährten Expertise erleichtert einerseits die Dokumentationsarbeit im Team und erhöht andererseits die Anschlussfähigkeit des im Projekt produzierten Codes.

Entwurfsbeschreibung

Die Entwurfsbeschreibung ist ein kompaktes Dokument, mit dem das Team darstellt, in welchem Umfang und auf welche Weise bei der Umsetzung der Aufgabenstellung *funktionale* software-technische Prinzipien und Standards zur Anwendung kamen. Zusammen mit der *Dokumentation des eigenen Vorgehens* als Umsetzung *organisatorischer* software-technischer

Prinzipien und Standards stellt das Team auf diese Weise dar, in welchem Umfang die *Ziele des Praktikums* erreicht wurden.

- Studierende sollen erkennen, dass man größere Projekte nur mit einer soliden Organisation durchführen kann ...
- Die Studierenden sollen die Bedeutung der Kommunikation (in allen Richtungen) erkennen und lernen, sich dieser Bedeutung entsprechend zu verhalten ...
- Es sind Fachkenntnisse zur Lösung der Probleme zu erwerben und anzuwenden.
- Die Arbeiten müssen systematisch und handwerklich ordentlich ausgeführt und termingerecht abgeschlossen werden ...

Im Rahmen des SWT-Praktikums gibt es deshalb allein für die *Entwurfsbeschreibung* allgemeine strukturelle Vorgaben (Aufgabenblatt 3), die hier weiter untersetzt sind. Einzelne Teile der Entwurfsbeschreibung werden im *Rechercheberich*t (Abgabetermin A2), im *Arbeitsplan* (Abgabetermin A3) sowie der *Modellierungsbeschreibung* (Abgabetermin A5) vorbereitet.

Die *Entwurfsbeschreibung* gliedert sich in die folgenden Teile:

1. Allgemeines

   In diesem Punkt ist der Kontext zu beschreiben, in dem sich die Aufgabenstellung bewegt. Es soll beschrieben werden, wozu die Software eingesetzt werden soll, welche in der Anwendungsdomäne vorgegebenen konzeptionellen Rahmenbedingungen zu berücksichtigen waren und welche Gestaltungsspielräume genutzt werden konnten.

   Vorarbeiten für diesen Abschnitt sind im *Rechercheberich*t zu finden.

2. Produktübersicht
   Beschreibung der äußerlichen Funktionsmerkmale des Systems.

   In diesem Punkt sollen die wesentlichen äußerlichen Funktionsmerkmale der Software in Bezug auf die Aufgabenstellung jenseits technischer Details beschrieben werden. Die Übersicht führt in den weiteren Text ein und setzt den projektinternen Rahmen, in welchem die weiteren Entwurfsentscheidungen erläutert und begründet werden.

   Vorarbeiten für diesen Abschnitt sind im *Arbeitsplan*, in der *Projektvision* sowie in den Produktanforderungen zu finden.

3. Grundsätzliche Struktur- und Entwurfsprinzipien
   Was sollte eine Informatikerin über das Gesamtsystem wissen, ehe sie sich allgemeinen Details zuwendet?

   In diesem Punkt sind die zentralen Entwurfsentscheidungen, insbesondere der Einsatz einer speziellen Architektur oder eines Frameworks, zu erläutern und zu begründen sowie übergreifende Aspekte der Modellierung und Strukturierung der Software darzustellen und zu motivieren, um damit einen Rahmen für die genaueren Ausführungen zu einzelnen Einheiten im nächsten Punkt zu setzen.

   Vorarbeiten für diesen Abschnitt sind im *Arbeitsplan* sowie in der *Modellbeschreibung* zu finden. Bei einem guten Zuschnitt des Arbeitsplans definieren die Arbeitspakete oft

bereits eine natürliche Strukturierung des zu entwickelnden Systems, indem sie zusammengehörende Funktionsbündel abstecken, die in einzelne Software-Einheiten gekapselt werden können.

4. Struktur- und Entwurfsprinzipien einzelner Pakete
   Was sollte ein Informatiker über 3. hinaus wissen, ehe er sich Details eines speziellen Pakets zuwendet?

   In diesem Punkt sind die Ausführungen unter 3. aufzunehmen und für die einzelnen Strukturierungseinheiten weiter zu verfeinern. Im Vordergrund stehen dabei *funktionale* Aspekte der Softwarelösung.

5. Datenmodell

   In diesem Punkt sind die Ausführungen unter 3. aufzunehmen und *strukturelle* Aspekte der Datenmodellierung der Softwarelösung darzustellen.

   In einer klassischen Drei-Schichten-Architektur (Präsentationsschicht, Modellschicht, Datenschicht), die von den meisten Architekturen und Frameworks verwendet wird, ist bei der Anbindung der Datenschicht zwischen einem *Datenmodell* als „Beschreibung der für die Erstellung eines Informationssystems relevanten Daten einschließlich ihrer Strukturen und Beziehungen“, den konzeptionell konkreteren *Datenstrukturen* als „Realisierung eines Datenmodells“ und einem praktischen *Binding* dieser Datenstrukturen an konkrete Persistenzstrukturen zu unterscheiden.

   Üblicherweise stellen Frameworks für den zweiten Schritt (Binding von Datenstrukturen) Implementierungen entsprechender Design Pattern wie etwa *Data Access Objects (DAO)* zur Verfügung, so dass die Darstellung der Anbindung eines projektspezifischen *Datenmodells* an diese architekturweit verwendeten *Datenstrukturen* ein wesentlicher Aspekt der Beschreibung struktureller Aspekte der Datenmodellierung der Softwarelösung ist.

   Dazu ist allerdings zunächst das *Datenmodell* auf einer (ontologischen) Abstraktionsebene zu entwickeln, die noch keinen Bezug auf konkrete Datenstrukturen nimmt. Bevor die „Realisierung eines Datenmodells“ beschrieben werden kann, muss dieses Datenmodell zunächst selbst beschrieben werden.

   Vorarbeiten für diesen Abschnitt sind in der *Modellbeschreibung* zu finden, wenn dabei bereits auf eine genauere Unterscheidung funktionaler und struktureller Aspekte des Modells geachtet wurde.

6. Glossar

   Das *Glossar* fasst die im Projekt entwickelten Bedeutungen von Begrifflichkeiten zusammen und erlaubt eine knappe und dennoch sprachliche präzise Darstellung in den verschiedenen Punkten der Entwurfsbeschreibung.

   Hier sind kaum weitere Präzisierungen erforderlich, wenn am Glossar über die ganze Projektlaufzeit intensiv gearbeitet wurde.

Die *Entwurfsbeschreibung* enthält somit alle wichtigen Informationen zu den Struktur- und Entwurfsentscheidungen, die im Zuge der Entwicklung der Anwendung getroffen wurden, die ein durchschnittlicher, bisher mit dem Projekt nicht befasster Programmierer kennen sollte,

bevor er Änderungen oder Ergänzungen des Quellcodes vornimmt und dabei die detailliertere, *referentielle Dokumentation* nutzt. Zur weiteren vertieften Einarbeitung kann schließlich die *Inline-Dokumentation* zu Rate gezogen werden.

4 Praktische Hinweise

- Eine Dokumentation wird immer für eine *spezifische Zielgruppe* geschrieben. Die sprachliche Darstellung muss sich an den Bedürfnissen und Erwartuingen dieser Zielgruppe orientieren und nicht an denen der Autoren der jeweiligen Dokumentation.

  Wichtige Fragen sind hierbei:

  - Welche Informationen könnten für die Zielgruppe nicht gleich klar bzw. nicht verständlich sein?
  - Welche referenzierten Dokumente könnten nicht (direkt) zugänglich sein?
  - Wie kann ein Benutzer Antwort auf eine konkrete Frage oder ein konkretes Problem finden?

- Die Zugänglichkeit der Dokumente wird meistens überschätzt. Es reicht nicht aus, die einzelnen Dokumente gesammelt aufzubewahren. Zur Zugänglichkeit gehört genauso, dass dem Benutzer sofort klar ist, wo er die Information für die Beantwortung einer konkreten Frage findet.

  Ebenso wichtig sind Querverweise zwischen den Dokumenten. Wenn auf ein Dokument verwiesen wird, sollte es (via Hyperlink) verlinkt sein, um die weiterführende Information für den Benutzer unmittelbar zugänglich zu machen. Ist das Dokument einer Kategorie zugeordnet, sollte diese angegeben werden. Hierbei sind auch Übersichtsseiten zu den Kategorien hilfreich, um verwandte Dokumente zu finden.

- Eine konsistente, verständliche und einheitliche logische Strukturierung der Dokumente ist wichtig für die Orientierung des Benutzers.

- Werden Abbildungen oder Tabellen innerhalb eines Textes referenziert, sollten diese so nah wie möglich an der referenzierenden Textstelle plaziert sein.

- Die Dokumentation sollte (insbesonders bei Handbüchern und Tutorials) kleinteilig und übersichtlich gehalten sein sowie auf konkrete Problemstellungen bezogen werden.

- Die Entwicklung der Dokumentation sollte (im besten Fall) feedbackorientiert erfolgen. Die Qualitätsziele *Angemessenheit* und *Benutzerfreundlichkeit* für die Dokumentation lassen sich am besten durch direkten Feedback der angesprochenen Zielgruppe sichern.

- Die *Vielfalt der Präsentationsformen* erhöht die Wirksamkeit der Dokumentation.

  Die Präsentationsform sollte dem Hintergrund und den Bedürfnissen der Zielgruppe angepasst sein. So sind beispielsweise Videotutorials für Anwender der Software relevanter, während eine umfassende, referentiell strukturierte technische Dokumentation für Entwickler oder Administratoren wichtig sein kann.

Die wichtigste Frage, die man sich beim Entwickeln der Dokumentation stellen sollte, lautet: „Hilft diese Art der Dokumentation dem Benutzer, für den sie gedacht ist?“