Qualitätsmanagement von Applikationsschnittstellenspezifikationen

Diplomarbeit im Fach Informatik

vorgelegt von Artan Kurtisi
Tetovo, Mazedonien

Matrikel-Nr. 98-726-557

Institut für Informatik
Universität Zürich
Prof. Harald C. Gall

Betreuer Prof. Harald C. Gall, Patrick Knab (IFI)
Boris Wedl (Swiss Re)

Abgabe der Arbeit 16. April 2006

Zusammenfassung

Der grösste Anteil von Software-Entwicklung entfällt auf Systeme, welche betriebliche oder technische Probleme lösen. Bei der Spezifikation der Anforderungen an solche Systeme ist es von grosser Bedeutung, festzustellen, wie ein solches System in seine Umgebung integriert sein soll und wo die Systemgrenzen liegen. Schwierigkeiten wie organisatorische Zuständigkeiten, Anpassungen der Nachbarapplikationen, Datentransformationen machen deren Identifizierung nicht einfach. Daher ist es wichtig, dass auch an diese Phase Qualitätsanforderungen gestellt werden und diese am Ende auf Erfüllung überprüft werden. Ziel der vorliegenden Diplomarbeit ist die Entwicklung eines Qualitätsmanagementmodells, das bei der Schnittstellenspezifikationen bzw. der Integration von (unternehmensübergreifenden) Applikationen innerhalb einer komplexen IT–Landschaft einsetzbar ist. Die Anwendung des Modells soll Qualitätsaussagen schaffen und kann heran gezogen werden, um die entwickelte Schnittstellen-Qualität abzuschätzen und gegebenenfalls Massnahmen zur Qualitätsverbesserung zu ermöglichen.

Abstract

The biggest part of software development applies on systems which are of operational or technical nature. During the specification of the requirements at such systems it is very important to determine how such a system should be embedded into its environment and where the system boundaries lie. Difficulties like organizational responsibilities, adaptations of the neighbour applications, data transformations don't make their identification simple. Therefore, it is important to set quality requirements also in this software development phase and they have to be checked for their satisfaction at the end. Goal of this thesis is the development of a quality management model that is applicable to the interface specifications, respectively by the integration of a global application within a complex IT landscape. The application of the model should be able stating the quality improvement of interfaces and it should enable measures for the quality improvement.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

1.1 Motivation

Der grösste Anteil von Software-Entwicklung entfällt auf Systeme, welche betriebliche oder technische Probleme lösen. Bei der Spezifikation der Anforderungen an solche Systeme ist es von grosser Bedeutung, festzustellen, wie ein solches System in seine Umgebung integriert werden soll und wo die Systemgrenzen liegen. Ein Software-System kann als eine Problemlösungsmaschine verstanden werden, welche über Schnittstellen mit ihrer Umwelt (Benutzer, Nachbarsysteme) interagiert. Die Identifizierung dieser Nachbarsysteme ist erstaunlicherweise nicht immer einfach. Oft werden sie erst spät oder sogar nach der Inbetriebnahme entdeckt [Siedersleben 2003]. Schwierigkeiten, wie organisatorische Zuständigkeiten (die Nachbarsysteme liegen in anderen Organisationseinheiten), Anpassungen der Nachbarsysteme, Datentransformationen (in der Regel hat jedes System seine eigene Vorstellung von dem Format der Daten) sind mögliche Gründe für ihre aufwendige Bestimmung. Wegen dieser Hindernisse - trotz ihrer erheblichen Bedeutung - werden sie meistens vernachlässigt.

Ein weiteres Problem der Software-Entwicklung, welchem auch zu wenig Beachtung geschenkt wird, ist die Qualität bzw. die Qualitätssicherung. Diese geschieht meistens nur beim Endprodukt und zwar mit Hilfe von Tests, Reviews, etc. (mit den sogenannten analytischen Qualitätsmassnahmen). Wenn man sich nicht nur mit der Bewertung von fertigen Software-Produkten zufrieden gibt, sondern Qualität auch konstruktiv realisieren will, muss ein Bewertungsansatz für den Entwicklungs- und Pflegeprozess aufgestellt werden [Wallmüller 2001]. Das bedeutet, dass Merkmale und Bewertungsmassstäbe auch für Zwischen- und Endprodukte sowie für deren Entwicklungsaktivitäten in allen Phasen des Prozesses benötigt werden. Es ist wichtig, dass an jede Phase Anforderungen gestellt werden und diese am Ende der Phase auf Erfüllung überprüft werden. Schliesslich entstehen Fehler in alle Phasen. Nach Mizumo [Mizumo 1983] kommt es beim Entstehen von Fehlern zu einem Summationseffekt. Normalerweise wird bei einem Projekt mit der Erfassung, Analyse und Definition der Anforderungen begonnen. Üblicherweise ist ein Teil dieser Anforderungen korrekt, ein anderer Teil ist mit Fehler behaftet.

In der nächsten Phase wird das System inklusive deren Schnittstellen spezifiziert. Das Ergebnis sind die Systemspezifikation und Schnittstellenspezifikationen[1]. Ein Teil der Spezifikationen ist nun korrekt. Der andere Teil beinhaltet Fehler, die bei den Spezifikationen entstanden sind, und ein dritter Teil der Spezifikationen basiert auf den fehlerhaften Anforderungen.

Dasselbe Szenario spielt sich auch bei den nächsten Phasen, nämlich Entwurf und Programmierung, ab. In der letzten Integrations- und Testphase entsteht dann folgende Situation: Ein Teil der Programme funktioniert korrekt. Ein anderer Teil der Programme enthält korrigierbare Fehler. Ein weiterer Teil enthält nicht korrigierbare Fehler und wiederum ein letzter Teil ist mit versteckten Fehler behaftet. Somit liegt ein unvollständiges Software-Produkt vor. Diese Unvollständigkeit ist durch einen Summationseffekt der aufgetretenen Fehler entstanden.

Durch Einsatz von Qualität in allen Phasen bzw. bei allen Zwischenprodukten, wie Systemspezifikation und Schnittstellenspezifikation, Entwurf, Implementierung sowie Integration und Test, kann dieser Effekt verringert werden.

"The study of products is vastly more important than the study of production, even for understanding production and its methods."

Karl Popper

1.2 Zielsetzung

Die vorliegende Arbeit wird im Rahmen des Projektes „Global Facultative System“ (GFS) der Swiss Re Group erstellt. Das GFS-Projekt beinhaltet die Entwicklung einer weltweiten Webapplikation für die Abwicklung und Verwaltung von Verträgen innerhalb des Geschäftsbereichs Property & Casualty. GFS unterstützt eine prozessorientierte Erfassung von Risiken und Prämien. Diese Informationen können global zwischen sämtlichen Swiss Re –Divisionen ausgetauscht werden.

Wie viele andere Applikationen hat sich auch GFS mit den im Abschnitt 1.1 beschriebenen Problemen auseinanderzusetzen. Deshalb ist in Anbetracht der geschilderten Problematik diese Arbeit entstanden.

[1] Damit sind die Schnittstellenspezifikationen der Nachbarsysteme gemeint.

Ziel der Diplomarbeit ist die Entwicklung eines Qualitätsmanagementmodels (QM), das bei den Schnittstellenspezifikationen bzw. der Integration von (unternehmensübergreifenden) Applikationen innerhalb einer komplexen IT–Landschaft einsetzbar ist. Die Anwendung des Modells soll Qualitätsaussagen schaffen und erlauben, die sich entwickelnde Schnittstellen-Qualität abzuschätzen und gegebenenfalls Massnahmen zur Qualitätsverbesserung zu ermöglichen.

Dazu werden sowohl die organisatorischen Prozesse als auch die technischen Probleme, die bei der Integration einer unternehmensübergreifenden Applikation in das bestehende IT-System eintreten, systematisch erfasst. Darauf aufbauend werden die schnittstellenspezifischen Qualitätseigenschaften definiert. Jede dieser Eigenschaften wird durch Merkmale näher bestimmt. Schliesslich werden für jedes Merkmal Qualitätskenngrössen zur quantitativen Bewertung angegeben.

Ein weiteres Ziel dieser Diplomarbeit ist die Umsetzung des erarbeiteten QM-Konzeptes mittels Anwendung des Modells im Umfeld des GFS Projekts sowie der Dokumentation und Bewertung der gewonnen Ergebnisse, aus welchen anschliessend konkret formulierte Empfehlungen zur Qualitäts-Lenkung und –Sicherung in der Swiss Re abgeleitet werden können.

1.3 Inhaltsübersicht

Die vorliegende Arbeit wird in fünf thematische Bereiche untergliedert. Kapitel 2 behandelt die Grundlagen und die Terminologie, die für ein besseres Verständnis dieser Arbeit notwendig sind.

Im Kapitel 3 werden ausführlich drei verwandte Qualitätsmodelle präsentiert. Alle drei Modelle befassen sich mit der Produktqualität. Im letzten Abschnitt dieses Kapitels sind sie kurz zusammengefasst.

Im Kapitel 4 wird die Anwendungsdomäne bzw. der Problembereich der vorliegenden Arbeit geschildert. Zunächst wird der Problembereich eingegrenzt und in den nächsten Abschnitten werden Aufbau der Schnittstellen und deren Typen behandelt. Der letzte Abschnitt ist der Beschreibung der GFS-Applikation gewidmet.

Das Kapitel 5 befasst sich mit dem Qualitätsmodell für die Spezifikation der Applikationsschnittstellen. In einem ersten Schritt wird gezeigt, welches Qualitätsmodell aus Kapitel 3 ausgewählt wurde und weshalb man auf diese Wahl kam. Weil das Qualitätsmodell sich stark auf den ISO/IEC Standard stützt bzw. dieser Standard als Basis dieses Modells dient, wird er in einem eigenen Unterkapitel ausführlicher behandelt.

Im Kapitel 6 wird dann das eigentliche Thema der vorliegenden Arbeit behandelt, nämlich die Anwendung des ISO/IEC Standards bei der Spezifikation der Schnittstellen am Beispiel der GFS-Applikation. Im ersten Teil werden zunächst alle möglichen Qualitätsmerkmale definiert. Danach folgt die Identifizierung der Beziehungen zwischen den Qualitätsmerkmalen. Und zuletzt werden für die Merkmale Metriken definiert, die die Qualität quantifizieren und somit mess- und prüfbar machen. Im zweiten Teil dieses Kapitels wird die Bewertung dieses Modells behandelt. Dabei werden Schlüsse gezogen, ob das Modell die gewünschten Ergebnisse erzielen konnte.

2 Einführung

2.1 Qualitätsbegriff und Terminologie

Der Ausdruck Qualität ist nicht einfach zu definieren. Im allgemeinen Sprachgebrauch wird von guter oder schlechter Qualität geredet. In dieser Hinsicht ist Qualität als Zweckeignung oder Kundenzufriedenheit definiert. Falls das gekaufte Produkt oder die Dienstleistung die Kundenwünsche erfüllt, hat sie im allgemeinen Sprachgebrauch eine gute Qualität. In Zusammenhang mit dem Begriff der Qualität werden auch folgende Attribute verwendet: Zuverlässigkeit, Werte, Vorzüge, Stärken, Fähigkeiten, Brauchbarkeit, usw.

Die Internationale Organisation für Normung (ISO) definiert in [ISO 9000:2000] die Qualität als „den Grad, in dem ein Satz inhärenter Merkmale Anforderungen erfüllt“. Inhärentes Merkmal bedeutet eine kennzeichnende Eigenschaft einer Einheit (Produkt, Prozess, Dienstleistung, etc), welche diese aus sich selbst heraus hat und die ihr nicht explizit zugeordnet ist.

Grundlegende Arbeiten zu der Software-Qualität hat D. A. Garvin [Garvin 1984] geliefert. Er unterscheidet fünf Ansätze, um zu einer Qualitätsvorstellung zu kommen. Diese Ansätze werden auch bei [Bächle 1996], [Kitchenham und Pfleeger 1996] und [Wallmüller 2001] beschrieben:

- *Transzendenter Ansatz*: Diese Sicht entspricht dem umgangssprachlichen Verständnis von Qualität. Die Qualität ist nicht präzise zu definieren und wird nur durch Erfahrungen wahrgenommen.
- *Produktbezogener Ansatz*: Qualität stellt eine objektive Grösse dar, die exakt bestimmt werden kann. Qualitätsunterschiede können durch quantitative Ausprägungen erfasst und somit verglichen werden.
- *Anwenderbezogener Ansatz:* Qualität wird vom Anwender bzw. Kunden subjektiv beurteilt. Die Güter, die am besten deren Bedürfnisse erfüllen, werden als qualitativ hochstehend angesehen.
- *Prozessbezogener Ansatz:* Entsprechend dieser Sichtweise ist die Qualität durch die Einhaltung vorgegebener Spezifikationen bestimmt. Gute Qualität entsteht demnach, wenn eine Tätigkeit zur Produkterstellung gleich das erste Mal richtig ausgeführt wird.

- *Wertbezogener Ansatz:* Qualität wird durch das Preis-Leistungsverhältnis ausgedrückt. Ein Qualitätsprodukt erbringt einen bestimmten Nutzen zu einem realistischen Preis oder die Übereinstimmung mit Spezifikationen erfolgt zu akzeptablen Kosten.

Qualitätsmodelle tragen wesentlich zur Vereinheitlichung der verschiedenen Vorstellungen über Software-Qualität bei. Der nebelhafte Begriff Software-Qualität wird durch deren strukturelle Zerlegungssystematik konkretisiert (vgl. Abschnitt 2.3).

Wenn über die Qualität geredet wird, dann ist der Begriff Fehler auch unvermeidbar. Da oft mit diesem Begriff nicht immer das gleiche bezeichnet wird, ist eine saubere Terminologie notwendig. Insbesondere müssen die Begriffe *Fehler, Fehlerursachen* und *Ausfälle* auseinander gehalten werden. Diese Begriffe werden in [IEEE 1988] wie folgt definiert (vgl. auch die Beschreibungen von [Glinz 2004]):

Ein *Irrtum (mistake)* wird von einer Person begangen. Als mögliche Folge davon enthält die Software einen *Defekt (defect, fault)*. Falls der Defekt beim Testen der Software gefunden wird, so ergibt dass einen *Befund (finding)*. Die Ausführung einer Software, die einen Defekt hat, führt zu einem *Fehler (error)*, d.h. die tatsächlichen Ergebnisse weichen von den erwarteten ab. Ein (oder mehrere) Fehler kann ein software-basiertes System zum *Ausfall (failure)* bringen. Wird ein Fehler festgestellt, so muss die Fehlerursache gefunden und behoben werden (*Fehlerbeseitigung, Debugging*).

Umgangssprachlich wird in allen obengenanten Fällen häufig nur von *Fehler (error, bug)* gesprochen. Im Qualitätsmanagement ist die genaue Unterscheidung oft sehr wichtig:

- Ein Defekt hat häufig mehrere Fehler zu Folge
- Ein Defekt führt nicht immer zu einem Fehler
- Nicht jeder Fehler hat einen Defekt als Ursache
- Nicht jeder Fehler hat einen Ausfall zu Folge

Folglich heisst es: Defekte sind nicht gleich Fehler und beide sind nicht gleich Ausfälle!

2.2 Qualitätsmodell

Im vorherigen Abschnitt war festzustellen, dass die Auffassung der Qualität nur als Zweckeignung oder Kundenzufriedenheit zu kurz greift. Qualität ist nicht das absolute

Mass für die Güte eines Gegenstands und wird nicht von selbst erzeugt, sondern die Qualität muss definiert und geschaffen werden. Zur Definition bzw. Spezifikation der Qualität aber auch zur Prüfung des Erfüllungsgrades von Qualitätsanforderungen werden Hilfsmittel benötigt, die sich sowohl auf den Entwicklungsprozess als auch auf das Produkt beziehen. Die nötige Hilfe, die einerseits den Qualitätsbegriff eindeutig definiert und andererseits die Möglichkeit zur Spezifikation von Qualitätsanforderungen bietet, schaffen die Qualitätsmodelle.

Mit Hilfe eines Qualitätsmodells wird der allgemeine Qualitätsbegriff durch Ableiten von Unterbegriffen operationalisiert (vgl. Abbildung 2.1) [Wallmüller 2001]. Die einzelnen Unterbegriffe bzw. Basiseigenschaften werden durch Festlegen von Merkmalen mess- und bewertbar gemacht.

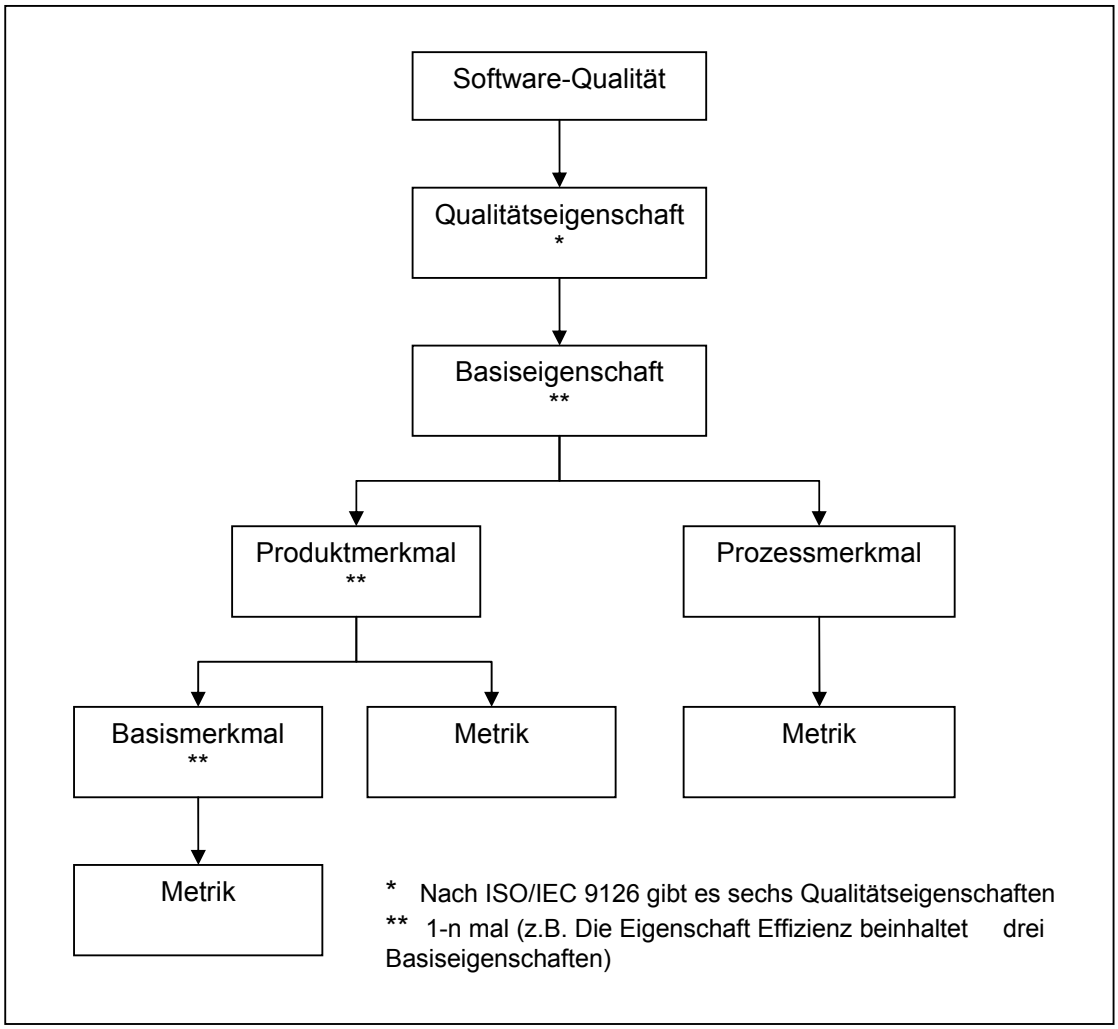


Abbildung 2.1: Struktur eines Qualitätsmodells (in Anlehnung an [Wallmüller 2001])

Ein elementarer Aspekt der Qualitätsmodelle ist die Zerlegungssystematik von Qualitätseigenschaften, die sich auf der untersten Ebene auf Qualitätskenngrössen abstützt.

In Abhängigkeit von dem Software-Produkt (Spezifikationen, Design, Implementierung) und den spezifischen Produktanforderungen (Anwendungsdomäne) wird für jedes Projekt bzw. Produkt eine spezifische Menge von Eigenschaften, Merkmalen und Qualitätsmetriken festgelegt, wobei deren Anzahl stark variieren kann. Für das Produkt Spezifikationen werden beispielsweise nicht die gleichen Eigenschaften benutzt wie bei der Implementierung. Für die Spezifikationen sind die Verständlichkeit und Umsetzbarkeit besonders wichtig, für die Implementierung steht hingegen die Funktionalität und Zuverlässigkeit im Vordergrund.

In vielen Projekten werden meistens nur funktionelle Anforderungen, Leistungs-Anforderungen und Anforderungen an die Benutzerschnittstelle spezifiziert. Hingegen sind die Anforderungen hinsichtlich der Wartbarkeit und der Portabilität nicht an der Tagesordnung. In vielen Fällen wird auch nur in der Implementierungsphase des Projekts Qualitätsmanagement betrieben. Die Anforderungsspezifikations- (inklusive der Schnittstellenspezifikationen) und Designphase werden gar nicht berücksichtigt. Der Grund dafür liegt darin, dass die für die Spezifikation Verantwortlichen keine Hilfsmittel oder kein nötiges Know-How besitzen, um diese Qualitätsanforderungen zu formulieren. Qualitätsmodelle bieten nun die Chance, diese Anforderungen zu spezifizieren [Wallmüller 2001].

Ein weiteres Ziel von Qualitätsmodellen ist, dass bei deren Anwendung die verschiedenen Benutzerperspektiven berücksichtigt werden müssen. Typische Benutzergruppen bei einer Eigenentwicklung sind die Projektleitung, die Analytiker, die Programmierer und die Tester.

Zusammengefasst wird folgender Zweck durch den Einsatz von Qualitätsmodellen erreicht [Wallmüller 2001]:

- Vereinheitlichung der verschiedenen Vorstellungen über die Software-Qualität
- Einheitliche Kommunikation
- Qualität wird konkretisiert, d.h. sie ist definier- und planbar.

Ein Nachteil ist, dass die Zusammenhänge zwischen Qualitätsmetriken, -merkmalen und –eigenschaften noch nicht theoretisch bewiesen, sondern blosse Hypothesen sind.

In den vorherigen Abschnitten wurde der Begriff Qualitätsmodell definiert sowie dessen Aufbau und Struktur gezeigt. Das Qualitätsmodell kann auch die Qualitätsprüfung unterstützen bzw. es kann auch als Hilfsmittel für das Qualitätsmanagement eingesetzt werden.

2.3 Qualitätsmanagement

Nach der Norm [ISO 9000:2000] wird Qualitätsmanagement folgendermassen definiert: “Alle Tätigkeiten der Gesamtführungsaufgabe, welche die Qualitätspolitik, Ziele und Verantwortungen festlegen, sowie diese durch Mittel der Qualitätsplanung, Qualitätslenkung, Qualitätssicherung und Qualitätsverbesserung im Rahmen des Qualitätsmanagementsystems verwirklichen.“

Das Qualitätsmanagement besteht aus folgenden Elementen:

- *Qualitätspolitik*: Darunter werden die übergeordneten Absichten und Ausrichtung einer Organisation in Hinblick auf die Qualität, wie sie von ihrer Leitung formell ausgedrückt wurden. Beispiele dafür sind: Kundenorientierung, leistungsstarke Kundendienstleistung etc.
- *Qualitätsmanagementsystem* (QMS): QMS ist definiert als ein Managementsystem zum führen und lenken einer Organisation hinsichtlich der Qualität. In Zusammenhang zu QMS steht das Totale Qualitätsmanagement (TQM). TQM ist eine Führungsmethode, bei der die Kundenzufriedenheit im Mittelpunkt steht. Qualität steht somit im Zentrum.
- *Qualitätsplanung*: Bei diesem Teil des Qualitätsmanagements geht es darum, zu definieren, welche Anforderungen an den Prozess und das Produkt in welchem Umfang erreicht werden sollen. Qualitätsplanung im Speziellen ist die Bestimmung der Ziele für ein Projekt bzw. für ein Zwischen- und Endprodukt. Qualitätsziele können mit Hilfe der Qualitätsmodelle, beispielsweise das ISO/IEC 9126 [ISO/IEC 9126-1 2001], ausgewählt und ergänzt werden.
- *Qualitätslenkung*: Darunter wird die Steuerung, die Überwachung und Korrektur der Realisierung einer Einheit mit dem Ziel, die festgelegten Anforderungen zu erfüllen. Es handelt sich dabei um die Realisierung konstruktiver Massnahmen, welche den Entwicklungsprozess präventiv lenken sollten. Das Ziel ist die Feststellung der fehlerverhindernden und –vermeidenden Prozesse und die Integration der Prüf- oder Korrekturmassnahmen in die Prozesse.

- *Qualitätsprüfung*: Bei der Qualitätsprüfung geht es darum, festzustellen, inwieweit eine Einheit (Prüfobjekt) die vorgegebenen Anforderungen erfüllt. Dabei handelt es sich um die analytischen Massnahmen – welche Fehler werden nachträglich durch Prüfung erkannt. Es wird zwischen statischen (Reviews, statische Analyse, formale Programmverifikation) und dynamischen (Testen, Simulieren) Prüfungen unterschieden. Die Prüfung sollte nicht bei dem Endprodukt stattfinden, sondern sie muss eine begleitende Aufgabe während des ganzen Entwicklungsprozess sein.

Die oben genannten Elemente des Qualitätsmanagements können ernsthaft nur durch quantitative Kenngrössen gelöst werden. Qualitätsmodelle bieten dazu die geeignete Grundlage.

3 Bestehende Qualitätsmodelle

Zur Spezifikation aber auch zur Prüfung des Erfüllungsgrades von Qualitätsanforderungen werden Hilfsmittel benötigt, die sich einerseits auf den Entwicklungsprozess, andererseits auf das Produkt beziehen. Differenzierte und den Erfordernissen von Software-Produkten angepasste Möglichkeiten zur Spezifikation von Qualitätsanforderungen, welche auch die Anforderungen und den Prozess berücksichtigen, bieten Qualitätsmodelle.

3.1 Bewertungskriterien

Wie in der Einleitung bereits angedeutet wurde, handelt es sich in der vorliegenden Arbeit um einen Teil des Software-Produktes, nämlich um die *Spezifikation von Applikationsschnittstellen*. Ein ganz wichtiges Kriterium bei der Auswahl von Qualitätsmodellen ist, dass sie sich vor allem auf das ganze Produkt und nicht nur auf die Prozesse beziehen. Obwohl die Prozessmodelle nicht Bestandteil dieser Arbeit sind, sollten sie aufgrund ihrer Wichtigkeit mindestens erwähnt werden. Das Ziel von Prozessqualitätsmodellen[2] ist die Verbesserung von Entwicklungsprozessen nach dem Motto „Nur gute Prozesse gewährleisten gute Produkte zu wettbewerbsfähigen Kosten“ [Bergbauer 2004] S. 1. Die Bedeutung der Prozesse wird auch von dem Software Engineering Institute[3] besonderes betont: „The quality of a product is largely determined by the quality of the process that is used to develop and maintain it” [SEI 2004] S. 6. Die bekanntesten Modelle sind: ISO 9001, CMM, Six Sigma, etc.

ISO 9001 stellt innerhalb der Normenreihe für Softwareentwicklungsprozesse die wichtigste Norm dar [Bächle 1996].

Das Prozessreifemodell (Capability Maturity Model - CMM) wird definiert als „a desription of the stages through which software organizations evolve as they define, implement, measure, control and improve their software process“ [Bächle 1996] S. 46. CMM dient als Anleitung zur Auswahl von Verbesserungsmassnahmen, indem es dabei hilft, die Stärken und die Schwächen des zu analysierenden Entwicklungsprozesses bzw. Softwareunternehmens zu identifizieren [Baker 2001], [SEI 2004]. Unter einem Prozess wird eine Se-

[2] Sie messen beispielsweise den Aufwand, Dauer, Fehlerkosten, etc. eines Entwicklungsprozesses bzw. eines Projekts

[3] Das Software Engineering Institute (SEI) ist ein Forschungs- und Entwicklungszentrum an der Carnegie-Mellon Universität in Pittsburg. Bekannt ist das SEI unter anderem durch das Capability Maturity Model Integration (CMMI), früher CMM

quenz von Aufgaben verstanden, die bei richtiger Durchführung zum gewünschten Ergebnis führen. Als Prozessreife wird dabei „the extent to which a specific process is explicitly defined, managed, measured, controlled, and effective“[Bächle 1996] S. 46 bezeichnet.

Six Sigma ist eine Methode zur Optimierung von Prozessketten. Sie ist gekennzeichnet durch die Kombination eines sehr systematischen phasenweisen Vorgehens mit der Erledigung der Arbeit in Teams, in denen Methoden- und Prozesskenner zusammengebracht werden [Bergbauer 2004].

Folglich werden Qualitätsmodelle, welche sich auf das Software-Produkt beziehen, der Schwerpunkt von Kapitel 3 sein. In der Vergangenheit wurden verschiedene Qualitätsmodelle entwickelt, wie beispielsweise von Goeff R. Dromey [Dromey 1995] und [Dromey 1996], B. Kitchenham [Kitchenham *et al.* 1999], X. Franch [Franch und Carvallo 2003] und Wallmüller [Wallmüller 2001]. Die ersten drei Modelle werden in den nächsten Unterkapiteln näher beschrieben. Im Jahr 1992 (bearbeitet im 2000) wurde ein Standard für Software-Produktmerkmale definiert und verabschiedet, nämlich das ISO/IEC 9126-1 Modell [ISO/IEC 9126-1 2001]. Dieses Modell wird im Abschnitt 5.1 genauer behandelt.

3.2 Beschreibung von Qualitätsmodellen

3.2.1 Franch’s Qualitätsmodell

Das Framework (Qualitätsmodell) von Franch ist eine Methodologie, die die Konstruktion von domäne-spezifischen Qualitätsmodellen im Hinblick auf das ISO/IEC 9126-1 Qualitätsstandard als Ziel hat. Solche Modelle können benutzt werden, um einerseits die Qualitätsfaktoren der Softwarepakete gleichmässig und umfassend zu beschreiben und andererseits Anforderungen anzugeben, wenn man ein Softwarepaket in einem bestimmten Kontext einführen will.

Die Methodologie von Franch besteht aus sechs Schritten (Abbildung 3.1). Neben diesen Schritten zieht er zudem eine einleitende Aktivität in Betracht, nämlich den Schritt 0 für die Analyse der Softwarepaket-Domäne. Diese Schritte können selbstverständlich auch mehrmals wiederholt werden oder sich verflechten, aber wegen besserer Übersicht werden sie in den nächsten Abschnitten sequentiell erklärt. Die folgende Beschreibung der Methodologie von Franch basiert auf den Ausführungen von [Franch und Carvallo 2003].

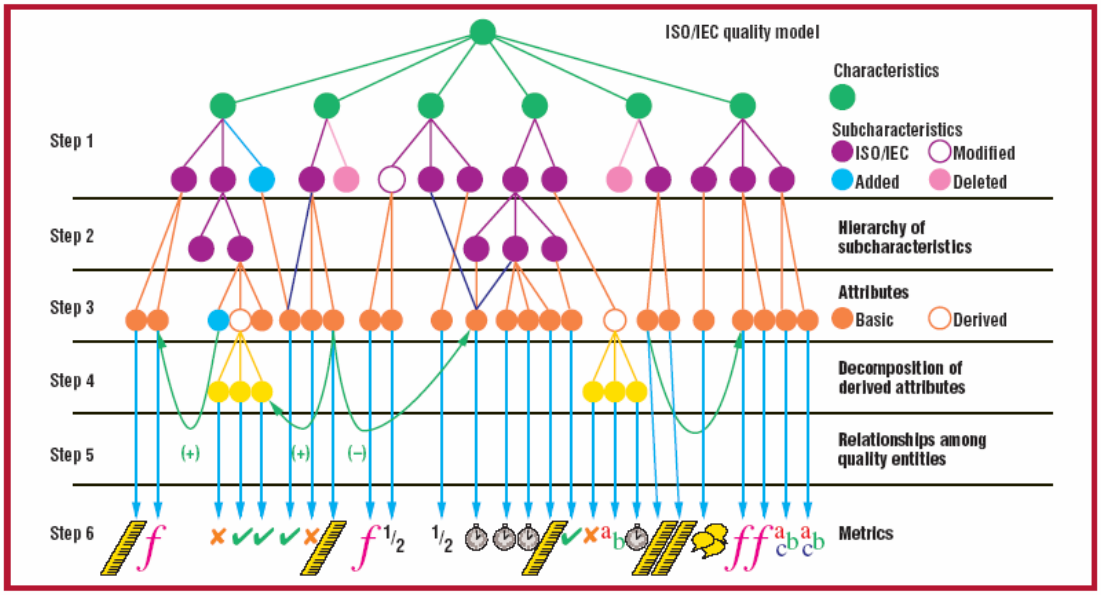


Abbildung 3.1: Das sechs-stufen Qualitätsframework (aus [Franch und Carvallo 2003])

Schritt 0: Definition der Domäne

Zuerst muss mit Hilfe von Experten das Gebiet bzw. die Domäne sorgfältig überprüft und beschrieben werden. Um das Gebiet zu beschreiben, empfiehlt Franch eine konzeptionelle Modellierung. Somit können die relevanten Konzepte besser verfolgt werden.

Eine der grössten Probleme bei der Auswahl eines Softwarepakets ist der Mangel an Standardterminologie. Unterschiedliche Verkäufer beziehen sich zwar auf das gleiche Konzept aber verwenden dafür unterschiedliche Namen, oder der gleiche Name kann unterschiedliche Konzepte bei unterschiedlichen Paketen bezeichnen. Solche semantische Konflikte, welche während der Auswahl von Softwarepaketen vorkommen, sollten während des einleitenden Schrittes 0 entdecket werden.

Schritt 1: Festlegung der Qualitätsbasiseigenschaften

Die von ISO/IEC 9126-1 vorgeschlagene Aufspaltung der Eigenschaften in Basiseigenschaften sollte hier auch gemacht werden, weil spätestens bei der Domäne-Analyse eine Aufteilung stattfinden muss. Möglicherweise wird auch nötig sein, neue Basiseigenschaften hinzuzufügen, die Definition von Bestehenden zu verfeinern, oder sogar einige zu beseitigen. z.B.:

- Bei der Domäne von ERP Systemen könnte für die Qualitätseigenschaft *Funktionalität* eine Basiseigenschaft *Überprüfung* (*Wird überprüft, ob die Geschäftsbereiche - Finanzierung oder Personal - gedeckt sind)* eingefügt werden.

- In der Domäne „data structure libraries“ könnte die Basiseigenschaft *Zeitverhalten* als “execution time of the methods provided by the classes inside the library” [Franch und Carvallo 2003] S. 36 verfeinert werden.
- In den Domänen, wo z.B. eine Applikation nur für die Integration von Systemen benötigt wird, kann die Basiseigenschaft *Attraktivität* weggelassen werden, weil sie nicht angewendet wird. Die Attraktivität ist standardmässig definiert als ein Merkmal, das den Qualitätsattributen *Farbe* oder *graphische Erscheinung* nachgeht.

Schritt 2: Definieren einer Hierarchie von Basiseigenschaften

In Bezug auf einige Faktoren, können die Eigenschaften weiter zerlegt werden; so entsteht eine Hierarchie.

Eine häufige Situation erscheint bei der Basiseigenschaft *Eignung*. Beispielsweise erfolgreiche Softwarepakete neigen dazu, fremde Anwendungen bzw. Plug-Ins einzufügen, die sie nicht selber entwickelt haben. Ein üblicher Grund dafür ist oft, dass viele Produkt-Lieferanten versuchen, Merkmale einzubinden, damit sich ihre Produkte von denen, der anderen Mitbewerber unterscheiden. Diese zusätzlichen Anwendungen werden normalerweise nicht mit den originalen Paketen verkauft, sondern sie werden getrennt und als Erweiterungen angeboten. Oft sie sind als erforderlich zu den Funktionalitäten, die das Paket bereitstellt, referenziert (vielfach sind einige Zusatzapplikationen notwendig). Demzufolge kann die Eigenschaft *Eignung* in zwei Untereigenschaften aufgeteilt werden, nämlich in *Grundeignung* und *hinzugefügte Eignung*. Diese Hierarchie wird in Abbildung 3.2 dargestellt:

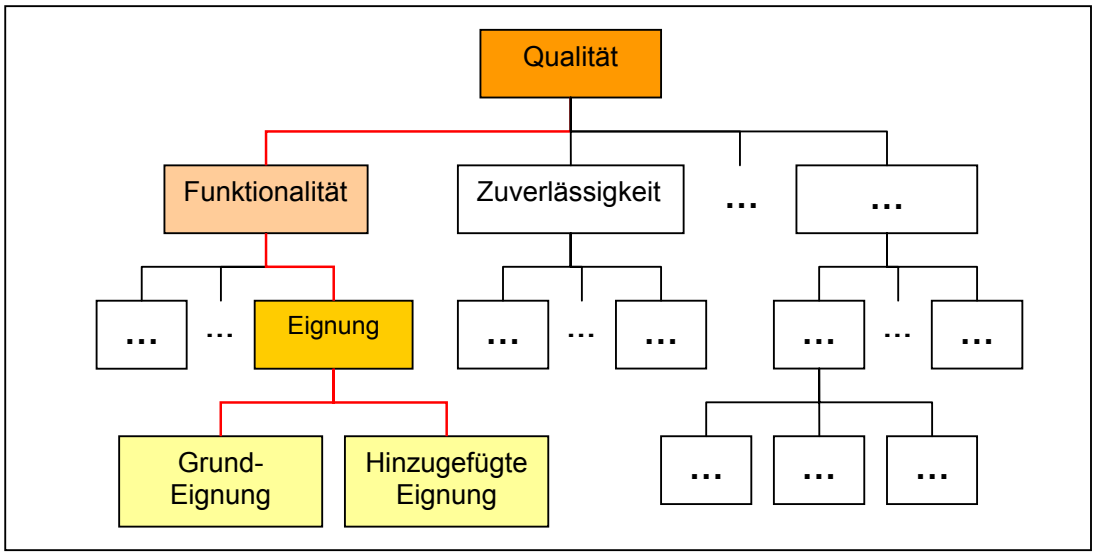


Abbildung 3.2: Hierarchische Aufteilung der Basiseigenschaft *Eignung*

Schritt 3: Spaltung der Basiseigenschaften in Merkmale

Qualitätsbasiseigenschaften bieten zwar eine nachvollziehbare abstrakte Sicht des Qualitätsmodells, aber diese abstrakten Konzepte müssen in konkretere Qualitätsmerkmale zerlegt werden. Die Basiseigenschaft *Erlernbarkeit* könnte zum Beispiel die Merkmale *Qualität der graphischen Benutzeroberfläche des Produktes, Zahl der unterstützten Sprachen* und *Qualität der verfügbaren Dokumentation* beinhalten. Die Qualitätsmerkmale müssen genau definiert werden, um die grundlegenden Qualitätskonzepte, welche sie verkörpern, zu verdeutlichen und sie mit den geeigneten Basiseigenschaften zu verlinken.

Das Qualitätsmodell von Franch ist für die Auswahl von Softwarepaketen und nicht für deren Entwicklung geeignet (der Softwarecode wird sowieso nicht freigegeben). Daher verwendet er eher die externen als die internen Qualitätsmerkmale.

Bei Aufspaltung der Basiseigenschaften kann es bei einigen Merkmalen vorkommen, dass sie zu mehr als zu einer Basiseigenschaft passen. Beispielsweise ein Merkmal für die Basiseigenschaft *Fehlertoleranz* in der Domäne von Datenstruktur-Bibliotheken könnte *der Mechanismustyp der Fehlerbehebung* sein. Aber dieses Merkmal könnte in der Tat auch ein Bestandteil der Basiseigenschaft *Prüfbarkeit* sein (ein mächtiger Mechanismustyp für die Fehlerbehebung macht die Prüfung der Bibliotheken leichter). Franch hält nicht an der Regel „Jedes Merkmal gehört nur einer Basiseigenschaft“ fest, ganz im Gegenteil, die Merkmale können Bestandteil von mehreren Basiseigenschaften sein. Der ISO/IEC Standard erlaubt schlussendlich solche Situationen.

Schritt 4: Aufspaltung der Merkmale in Basismerkmale

Einige von den aus Schritt 3 entstandenen Merkmale (z.B. *Anzahl der unterstützten Sprachen*) können bei einem gegebenen Produkt direkt gemessen werden. Andere hingegen könnten noch dermassen abstrakt sein, dass noch eine weitere Aufspaltung erforderlich ist. Dies ist der Fall mit dem Merkmal *Qualität der graphischen Benutzeroberfläche*, das im vorherigen Schritt bereits erwähnt wurde; die Qualität könnte von solchen Faktoren wie z.B. *Benutzerfreundlichkeit*, *Tiefe der längsten Pfades in einem durchblätternden Prozess* und *Typen der unterstützenden Schnittstellen* abhängen. Deshalb wird zwischen **Grund-** und **abgeleiteten Eigenschaften** unterschieden. Die abgeleiteten Merkmale sollten solange zerlegt werden, bis sie in Form von Basismerkmalen ausgedrückt werden.

Schritt 5: Festlegung der Beziehungen zwischen Qualitätseigenschaften

Um ein vollständiges Qualitätsmodell zu erzielen, müssen die Beziehungen zwischen Qualitätseigenschaften auch explizit angegeben werden. Das Modell wird gründlicher und die Auswirkungen auf die Qualität der Benutzeranforderungen könnten klarer werden. Zwischen zwei Qualitätseigenschaften A und B können verschiedene Arten von Beziehungen identifiziert werden:

- ***Zusammenarbeit***: Die Qualitätserhöhung von A impliziert die Erhöhung von B, z.B. die Basiseigenschaft *Sicherheit* kollaboriert mit der Basiseigenschaft *Reife*.
- ***Beeinträchtigung***: Die Erhöhung von A bewirkt die Qualitätsreduzierung von B, z.B. das Merkmal *Mechanismustyp der Fehlerbehebung* kollidiert mit der Basiseigenschaft *Zeitverhalten*, d.h. je mächtiger der Mechanismus für die Fehlerbehebung ist, desto langsamer werden die Programmläufe.
- ***Abhängigkeit***: Einige Werte von A erfordern B, um einige Bedingungen zu erfüllen. Z.B. um einen Ausnahme-basierten Fehlerbehebung-Mechanismus zu haben, muss die Programmiersprache Ausnahmekonstrukte anbieten.

Schritt 6: Festlegung der Metriken

Die Identifizierung von Merkmalen ist zwar notwendig aber nicht ausreichend, wenn sie nicht gemessen werden. Daher müssen in diesem Schritt Metriken für alle Basis- und abgeleiteten Merkmale ausgewählt werden.

Die Metriken für Basiseigenschaften sind quantitativ, z.B. Existenz von irgendeiner Art von Datenverschlüsselung, Tiefe des längsten Pfades in einem durchblätternden Prozess, unterstützte Protokolle für die Datenübermittlung, usw. Die abgeleiteten Merkmale können entweder quantitativ oder qualitativ, d,h mit expliziten Formeln, die die Werte von ihren Merkmalekomponenten berechnen, sein.

Einige Merkmale erfordern eine komplexere Darstellung, die zu strukturierten Metriken führen. Beispiele sind die Mengen (z.B. ein Satz von Sprachen, die von der Schnittstelle unterstützt werden) und Funktionen. Funktionen sind besonders nützlich für Merkmale, die von der darunterliegenden Plattform abhängen. So könnten sich viele Merkmale, die in Beziehung zu der Basiseigenschaft *Zeitverhalten* stehen, in dieser Kategorie gliedern.

Für einige Qualitätsmerkmale kann die Bestimmung von Metriken sehr schwierig sein. Dennoch ist das Vorhandensein von strengen Metriken der einzige Weg, um ein Qualitätsmodell nützlich für vertrauenswürdige Vergleiche zu machen.

Fallstudie: Mailserver

Franch hat sein Modell für die Auswahl von Mailservern angewendet. Er begründet die Wichtigkeit der Mailservern in einer Unternehmung wie folgt: „As electronic mail services have grown in importance, companies are increasingly using them to improve inside and outside communication and coordination. An overwhelming number of mail-related products are available, and organizations face the problem of choosing among them the ones that best fit their needs. For some companies, an inappropriate selection would compromise their success. Selection relies on manufacturing documentation, public evaluation reports, others' experience, and hands-on experimentation. These information sources can be inaccurate, not fully trustable, and costly." [Franch und Carvallo 2003] S. 38. Daher ist ein solches Qualitätsmodell für diese Domäne – Auswahl von Mailserverpaketen - von ganz grosser Bedeutung. Für eine detaillierte Beschreibung der Fallstudie sei auf [Franch und Carvallo 2002] verwiesen.

3.2.2 Das Squid Qualitätsmodell

Squid [Kitchenham *et al.* 1999] ist ein kontextabhängiges Qualitätsmodell wie das Modell von Franch. Das Modell heisst kontextabhängig, weil dessen Bestandteil die Metriken (und somit die Messungen) sind, d.h. die Messungen können nicht unabhängig vom Kontext definiert werden. Ein Satz von Metriken, der in einem Projekt als erfolgreich bewertet wurde, kann zu einer schlechten Qualität oder höheren Entwicklungskosten führen, wenn er in einem anderen Projekt angewendet wird [Kitchenham *et al.* 1999].

Squid wurde von dem Projekt „The Software Quality in Development", das vom Programm „Erupean Commission's Espirit" unterstützt wurde, entwickelt. Squid ist eine Methode für die Qualitätssicherung und –kontrolle. Eine Organisation für Softwareentwicklung, die Squid anwendet, kann die Messung benutzen, um

- die Produktqualität während der Entwicklung zu planen und kontrollieren,
- tiefer und systematischer über das Softwareprodukt und den Softwareprozess zu lernen, sowie die gewonnen Erfahrungen bei den aktuellen und zukünftigen Projekten wieder zu verwenden und
- die Qualität des Endproduktes zu bewerten.

Das Squid Modell wurde im Softwareentwicklungsprojekt „Telescience"[4] angewendet. Die folgende Beschreibungen basieren auf den Ausführungen von [Kitchenham *et al.* 1999]

[4] "This project aims to automate a scientific station in Antarctica, to monitor and control scientific experiments remotely during the Antarctic winter." [Kitchenham *et al.* 1999] S. 70

3.2.2.1 Der Squid-Ansatz

Squid definiert Qualität als funktionsfähiges Verhalten (wie es von seinen Benutzern verlangt wird) eines Produktes. Dieses Modell ist gleich bzw. basiert auf dem ISO 9126 Standardmodell, das die Qualität als eine Menge von Produkteigenschaften definiert. Eigenschaften, die die Arbeitsweise des Produkts in seiner Umgebung beeinflussen, werden **externe** Qualitätseigenschaften genannt, z.B. *Brauchbarkeit* und *Zuverlässigkeit*. Eigenschaften, die sich auf die Vorgehensweise der Produktentwicklung beziehen, werden **interne** Qualitätseigenschaften benannt. Sie beinhalten z.B. strukturelle Komplexität, Grösse, Testdeckung und Fehlerrate [Kitchenham *et al.* 1999]. Beispiele und Beschreibungen dieser Eigenschaften können in [ISO/IEC 9126-1 2001] gefunden werden.

Der Squid-Ansatz hilft auch bei der Festlegung der Beziehungen zwischen den internen und externen Qualitätseigenschaften. Bei deren Festlegung wird auch die gesammelte Erfahrung von ähnlich aufgebauten Software-Projekten genutzt. Diese vergangenen Erfahrungen werden auch verwendet, um die Machbarkeit der Qualitätsanforderungen vor dem Projektbeginn abzuschätzen, sowie um die internen Qualitätseigenschaften zu identifizieren und deren Ziele festzulegen. Damit der Vergleich quer durch verschiedene Projekte überhaupt möglich ist, sollten diese Erfahrungsdaten konsistent gespeichert werden. Das Squid-Datenmodell und der Qualitätsprozess stellen eine strenge und organisierte Methode für die Identifizierung, Definition, Zuordnung und den Vergleich von Qualitätsmassen zur Verfügung.

3.2.2.2 Das Squid-Datenmodell

Entwicklungsprozesse unterscheiden sich von Firma zu Firma, und sogar eine einzelne Firma kann häufig diverse unterschiedliche Entwicklungsmodelle benutzen. Diese Unterschiede begrenzen die Möglichkeit, die gleichen Menge von Metriken über unterschiedlichen Projekten zu verwenden. So macht es wenig Sinn, die Zahl der Zeilen eines Codes (LOC[5]), welcher mit strukturierten Techniken (z.B. mit der Programmiersprache C) entwickelt wurde, mit einem objektorientierten Code (z.B. Java, C++, etc.) zu vergleichen. Prozessunterschiede beeinflussen auch die Relation zwischen Massen und Qualitätseigenschaften. Der Einfluss "der Grösse" (gemessen als LOC) auf die Wartung einer Software, die in einem objektorientierten Entwicklungsprozess entwickelt wird, unterscheidet sich von einer, die das Wasserfall-Modell verwendet.

[5] Lines of Code

Eine auf das Squid-Modell basierte Organisation spezifiziert die Projektgegenstände (durchzuführende Arbeiten[6], Meilensteine, Module, usw.), die durch den Entwicklungsprozess entstehen. Projektgegenstände sind die Elemente, auf denen die Messungen durchgeführt werden. Squid erfordert genau so wie ISO 9126 das Mapping von Qualitätsanforderungen auf die Qualitätseigenschaften und folglich auf die Qualitätsmerkmale (d.h. die messbaren Eigenschaften). Qualitätsanforderungen dienen dann als Zielmenge für die externen Qualitätseigenschaften. Das Qualitätsmodell identifiziert auch die internen Qualitätseigenschaften, die die externen beeinflussen. Die Organisation überwacht und steuert dann die interne Qualitätseigenschaften während der Projektmessungen, um die Qualitätsanforderungen zu erzielen.

Darstellung des Entwicklungsprozesses

Das Squid-Datenmodell bestimmt die wesentlichen Eigenschaften des Entwicklungsprozesses, wie z.B. die Art des Entwicklungsmodells und die verwendete Sprache. Squid modelliert auch die Projektgegenstände (*durchzuführende Arbeiten, Entwicklungsaktivitäten oder Review-Zeitpunkten)*:

- *Durchzuführende Arbeiten* (wie z.B. die Spezifikationen und Codes) werden während der Lebenszykluszeit des Projekts erzeugt.
- *Die Aktivitäten* bereiten die durchzuführenden Arbeiten auf.
- *Review-Zeitpunkten* sind Kontrollinterwalen, die in einem entsprechenden Entwicklungsmodell verwendet werden.

Qualitätsmassnahmen werden bei den durchzuführenden Arbeiten während den festgelegten Review-Zeitpunkten ergriffen.

Darstellung des Qualitätsmodells

Squid spezifiziert das Qualitätsmodell in Form von Qualitätseigenschaften. Sie werden so lange verfeinert, bis sie direkt messbar sind und werden als Qualitätsmerkmale bezeichnet. Qualitätseigenschaften und -merkmale können entweder intern oder extern sein, d.h. der Benutzer bestimmt, wie interne Eigenschaften die externen beeinflussen, indem sie im Datenmodell verlinkt werden. Abbildung 3.3 zeigt zwei in Beziehung stehende Qualitätseigenschaften und wie sie im Qualitätsmodell verfeinert werden.

[6] Deliverables

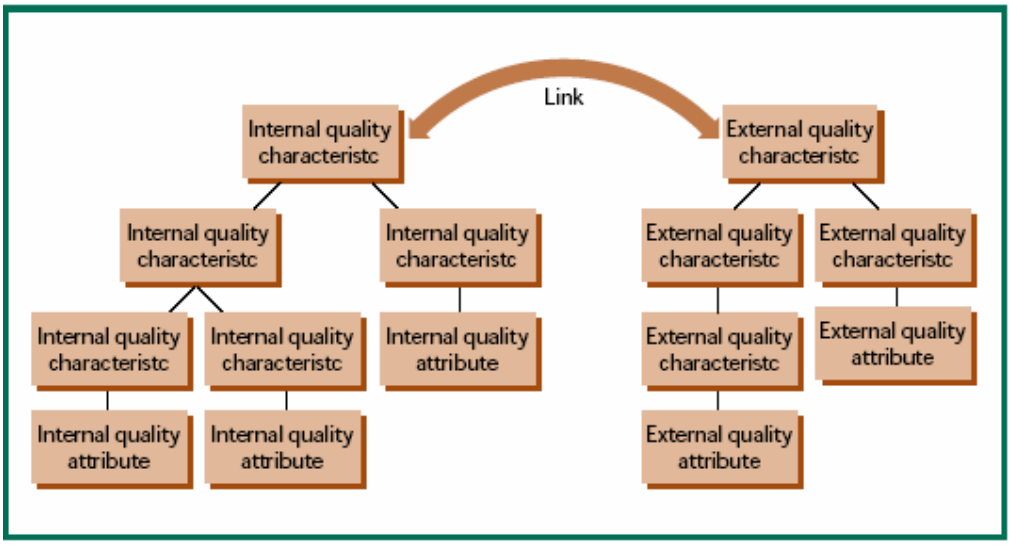


Abbildung 3.3: Eine mögliche Verfeinerung sowie eine Verlinkung der internen und externen Qualitätseigenschaft *Wartung* in das Squid-Qualitätsmodell (aus [Kitchenham *et al.* 1999])

Ausser bei extrem einfachen Projekten haben unterschiedliche Teile eines Software-Produktes normalerweise sehr unterschiedliche Verhaltensanforderungen. So hat die Schnittstellen-Komponente hohe Brauchbarkeitsanforderungen, während die Datenbank-Komponente hohe Vollständigkeitsanforderungen hat. Squid differenziert solche unterschiedliche Teile, indem das Konzept der *Produktaufteilung* verwendet wird, d.h. ein einzelnes Produkt beinhaltet eine Menge von Produktteilen und unterschiedliche Produktteile haben unterschiedliche Qualitätsanforderungen. Dieses Konzept kann helfen, um Konflikte zwischen unterschiedlichen Qualitätsanforderungen zu verringern. Während für das Produkt als Ganzes die Lieferung von höheren funktionsfähigen Anforderungen unmöglich sein kann, ist es für die einzelnen Produktteile sehr einfach, um anspruchsvolle Qualitätsanforderungen anzubieten.

Darstellung der Messungen

Die Messungen, die durch Squid gesammelt werden, können tatsächliche Werte, Schätzungen oder Zielwerte sein. Ein *tatsächlicher Wert* ist ein momentaner Wert, der für ein spezifisches Qualitätsmerkmal gemessen wird. Eine *Schätzung* kann entweder von den vergangenen Daten abgeleitet oder durch das Schätzung-Tool von Squid erreicht werden. *Zielwerte* werden während der Qualitätsspezifikationsphase festgelegt. Messungen können in unterschiedlichen Masseinheiten ausgedrückt werden, z.B. die *Zeilenanzahl des Codes* ist eine Masseinheit für die Länge eines Produktcodes; *Abgelaufene Tage* ist eine Masseinheit für die Projektdauer und *Arbeitsjahre* misst die Personalerfahrung.

Der Squid-Benutzer definiert selbst, wie die Messungen unter Rücksicht auf die *geltenden Regeln* durchgeführt werden. Die geltenden Regeln verdeutlichen und regulieren alle Bedingungen, welche die Objektivität der Messung beeinflussen können. Die Messung der Programmlänge kann erfasst werden als Codelinien, aber eine geltende Regel muss bestätigen, in welcher Entwicklungsphase diese Messung durchgeführt werden soll. Geltende Regeln stellen sicher, dass die Messungen wiederholbar und mit denen der historischen Datenbank vergleichbar sind.

3.2.2.3 Der Squid-Qualitätsprozess

Der Squid-Qualitätsprozess beinhaltet die Qualitätsspezifikation, -Planung, -Lenkung und -Auswertung. Squid bittet Benutzerinteraktion und -unterstützung in verschiedenen Formen.

Qualitätsspezifikation

Qualitätsspezifikation bedeutet die Festlegung von Qualitätsanforderungen für das Software-Produkt. Sie beinhaltet drei Hauptaktivitäten:

- *Auswahl des Qualitätsmodells*. Die Spezifikation der Qualitätsanforderungen findet innerhalb des Frameworks eines spezifischen Qualitätsmodells statt. Eine Organisation, welche den Squid-Ansatz verwendet, sollte die Qualitätsmodelle (z.B. einen internationalen Standard wie ISO 9126 oder ein firmen-spezifisches Modell) definieren, die bei der Entwicklung des Software-Produkts angewendet werden. Bei der Entwicklung eines neuen Projekts muss die Organisation festlegen, welches der vorhandenen Qualitätsmodelle und der Entwicklungsprozesse auf das gegenwärtige Projekt zutreffen.
- *Identifizierung der Produktteile, Qualitätsanforderungen und Ziele*. Das entwickelte Produkt muss in unabhängige Produktteile zerlegt werden, d.h. in Software-Produktteile, die verschiedene Qualitätsanforderungen haben. Jede geforderte Funktionalität wird dann mit dem Produktteil, auf das sie passt, verlinkt. Das Squid-Tool unterstützt die Verlinkung jeder Anforderung mit einer Qualitätseigenschaft. Sobald diese Verknüpfung gebildet wurde, identifiziert das Tool ein oder mehrere externe Merkmale (abgeleitet vom Qualitätsmodell), bei denen die Zielwerte festgesetzt werden.
- *Durchführung einer Machbarkeitsanalyse*. Squid hilft den Benutzer, die Machbarkeit der externen Attributziele auszuwerten. Die Erreichung der gegenwärtigen Ziele wird mit den Projektergebnissen von ähnlichen Projekten aus der Vergangenheit verglichen. Das Tool analysiert die historische Datenbank und sucht nach ähnlichen Projekten, um das Ähnlichkeitsniveau zu beurteilen. Als Überprüfungspunkte werden die externen Attributsziele und der Entwicklungsprozess der vergangenen Projekte verwen-

det. Der Benutzer kann Suchkriterien einschränken und bestimmte Projekte von der Betrachtung entfernen. Squid analysiert dann die aktuellsten Werte der externen Merkmale, die von den vergangenen Projekten erzielt wurden, und vergleicht diese anschliessend mit den Zielwerten des neuen Produktes. Wenn die aktuellen Werte von den geplanten abweichen, kann der normale Entwicklungsprozess möglicherweise nicht das Produkt liefern, das die festgelegten Qualitätsanforderungen erfüllt. Dieser Vergleich ist aussagekräftig, wenn die Datenbank mit Daten aus vielen vergangenen Projekten gefüllt ist.

Qualitätsplanung

Qualitätsplanung beinhaltet die Entscheidung für einen geeigneten Entwicklungs-Prozess mit sich und setzt Zielwerte für interne Eigenschaften.

- *Auswahl des Entwicklungsprozesses für jeden Produktteil*. Weil die Produktteile unterschiedliche Qualitätsanforderungen haben, wird das Entwicklungsteam möglicherweise verschiedene Prozesse adoptieren, um sie ausführen zu können.
- *Zuweisung der Zielwerte zu internen Qualitätseigenschaften*. Für die internen Qualitätseigenschaften, die im Qualitätsmodell identifiziert wurden, werden quantitative Ziele gesetzt. Die Eigenschaften werden dann mit Zwischenaktivitäten und fertigen Ergebnissen (Projekt-Objekten) assoziiert. Das Squid-Tool hilft Benutzern, die Zielwerte den internen Messungen zuzuweisen. Somit können sie die erreichten Software-Qualitätsziele aus ähnlichen vergangenen Projekten ansehen.

Qualitätskontrolle

Die Qualitätskontrolle beinhaltet folgende Aktivitäten:

- *Identifizierung von Abweichungen durch Vergleichen der Zielwerte mit Ist-Werten der internen Qualitätseigenschaften*. Bei jedem Projekt-Review kann der aktuelle Qualitätsstatus des Projektes bewertet werden, wenn bestimmte Qualitätsmerkmale gemessen worden sind.
- *Identifizierung der Software-Komponenten mit ungewöhnlichen Merkmale-Werten*. Solche Komponenten sollten vorsichtig analysiert werden, um eine möglichst frühe Erkennung eines Problemgebietes sicherzustellen. Das Squid-Tool automatisiert die Entdeckung der abweichenden Komponenten, um Projektobjekte mit ungewöhnlichen Kombinationen interner Merkmalwerten zu identifizieren.

Qualitätsauswertung

Die Qualitätsauswertung stellt sicher, dass ein Produkt seine festgelegten Anforderungen erfüllt. Das Entwicklungsteam kann auch die Qualitätsergebnisse eines Projektes analy-

sieren, um ein Feedback für den Entwicklungsprozess zu liefern. Squid bietet einen Grundauswertungsmechanismus, der auf

- die Messung von externen Merkmale-Werten bei jedem Produktteil,
- den Vergleich jedes eigentlichen Wertes mit seinem Zielwert und
- der Berichterstattung von Qualitätsdefiziten

basiert.

Die Abbildung 3.4 zeigt die Reihenfolge der Qualität-Prozessaktivitäten von Squid und ihre Hauptverbindungen zur Squid-Datenbank.

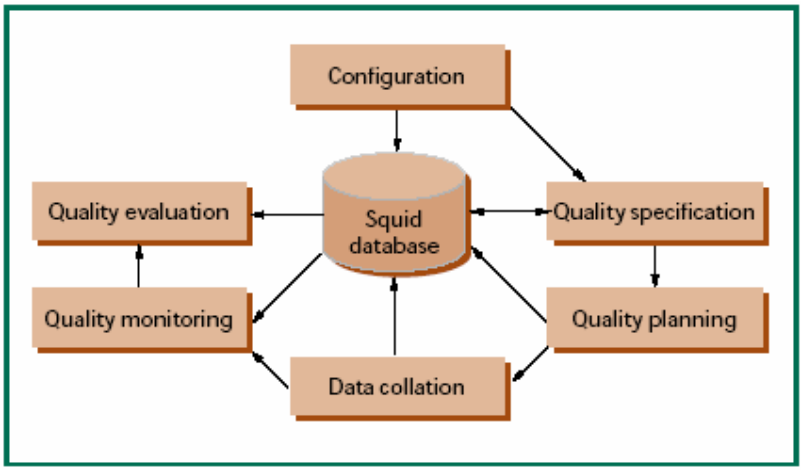


Abbildung 3.4: Qualität-Prozessaktivitäten und ihre Beziehungen zu der Squid-Datenbank (aus [Kitchenham *et al.* 1999] S.74)

Das Squid Qualitätsmodell wurde im Projekt „Telescience“ angewendet. Für eine ausführlich beschriebene Anwendung sei auf [Kitchenham *et al.* 1999] verwiesen.

3.2.3 Dromey’s Qualitätsmodell

Im Gegensatz zu vielen anderen Modellen, welche Qualität für Prozesse messen, um somit ein qualitativ besseres Produkt zu erzeugen, ist Dromey’s Modell ein Produkt-basiertes Qualitätsmodell. Das Produkt-basierte Modell kann nicht alle Vorteile der Prozessabschätzung und –verbesserung erlangen und auch nicht die Prozessmodelle ersetzen, aber wenn die Ziele des Modells genau definiert sind, wird es viel einfacher die Prozesse entsprechend abzustimmen.

Dromey's Qualitätsmodell unterscheidet sich auch von anderen Produkt-basierten Modellen. Während die meisten Modelle den Top-Down Ansatz verwenden (d.h. sie identifizieren zuerst die Qualitätseigenschaften höchster Hierarchieebene wie z.B. Zuverlässigkeit oder Änderbarkeit. Sie verfeinern diese in Untereigenschaften und definieren anschliessend für jede Untereigenschaft Qualitätsmerkmale, die dann mit Hilfe von Metriken gemessen werden), geht Dromey anders vor, nämlich nach dem Button-Up Ansatz. Der wichtige Grundsatz für die Software-Produktqualität ist: „a product's tangible internal characteristics or properties determine its external quality attributes" [Dromey 1996] S. 34. D.h. man sollte diejenigen internen Eigenschaften in das Produkt einbauen, welche die gewünschten externen Qualitätseigenschaften einbringen. Daher muss das Produkt-basierte Qualitätsmodell übergreifend die internen (messbaren und / oder abschätzbaren) Produkteigenschaften identifizieren, die den signifikantesten Effekt auf die externen (subjektiv wahrnehmbare) Qualitätseigenschaften haben. Das produkt-basierte Qualitätsmodell muss zudem die Verbindungen zwischen diesen zwei festlegen.

Gemäss Dromey ist diese Vorgehensweise eine praktische Strategie für die Abwicklung von schwer fassbaren und komplexen Konzepten, wie die Qualität. Die folgende Beschreibung des Frameworks von Dromey basiert auf den Ausführungen von [Dromey 1995] und [Dromey 1996].

3.2.3.1 Framework für das Qualitätsmodell

Das Qualitätsmodell von Dromey baut auf folgende Fragestellung: Welche Art von Framework kann die objektiven[7] Produkteigenschaften mit den subjektiven[8] Qualitätseigenschaften verbinden? Dromey geht dieses Problem an, indem er folgende Streitpunkte berücksichtigt:

1. Viele Produkteigenschaften treten auf, welche die Softwarequalität beeinflussen
2. Ausser einigen empirischen Berichten gibt es eine kleine formale Basis, um festzustellen, welche Produkteigenschaft welche Qualitätseigenschaft (Funktionalität, Zuverlässigkeit, etc.) beeinflusst.

[7] **Objektive Produkteigenschaften** (eng. tangible product properties) sind die mess- oder abschätzbaren Eigenschaften. Z.B. die strukturelle Form ‚Ausdruck' eines Programmcodes besitzt die Eigenschaft ‚berechenbar', d.h. ein Ausdruck kann direkt gemessen werden, ob er berechenbar ist oder nicht. Beispielsweise ein durch Null dividierter Ausdruck verursacht einen Defekt. Dieser Defekt beeinflusst somit die subjektive Qualitätseigenschaft ‚Zuverlässigkeit'.

[8] **Subjektive Qualitätseigenschaften** (eng. intangible quality attribute) können nicht direkt gemessen werden, sondern sie werden von den objektiven Eigenschaften beeinflusst, bzw. abgeleitet. Z.B. eine Software ist zuverlässig, wenn keine Defekte auftreten, bzw. wenn kein Ausdruck durch Null dividiert wird.

Das Qualitätsmodell von Dormey und der Prozess für die Bildung solcher Modelle für verschiedene Software-Produkte hat sich die Lösung der obengenannten Probleme als Ziel gesetzt. Die Abbildung 3.5 zeigt die drei wichtigsten Elemente des generischen Qualitätsmodells: die Produkteigenschaften (welche die Qualität beeinflussen), der Satz der Qualitätseigenschaften (Funktionalität, Zuverlässigkeit, etc.) und ein Mittel zu deren Verlinkung.

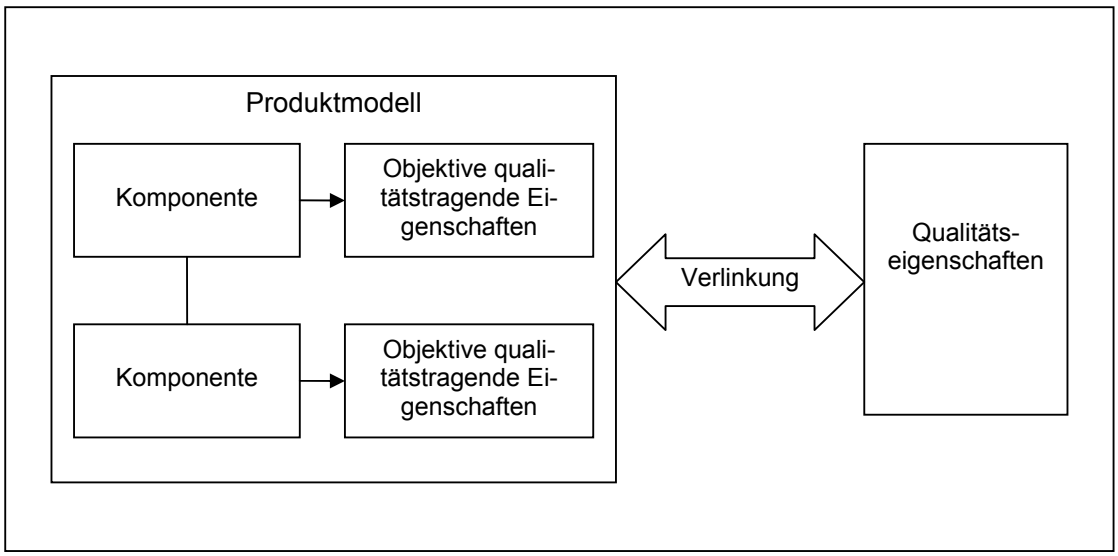


Abbildung 3.5: Die Komponenten des Produktqualitätsmodells (aus [Dromey 1996])

Produktmodell[9]

Die Produkte bestehen aus Komponenten[10]. Sie sind entweder atomar oder werden aus einfachen Komponenten zusammengesetzt. So sind die Komponenten des Produktes *Software-Implementierung* Variablen, Ausdrücken, Anweisungen, usw. Basierend auf diesem konzeptionellen Modell ist die Produktqualität zum grössten Teil durch:

- die Auswahl der Komponenten, die das Produkt bilden und wie sie implementiert sind
- die objektiven Eigenschaften einzelner Komponenten
- die Verknüpfung der objektiven Eigenschaften mit dem Komponentenaufbau

bestimmt.

[9] „Ein Produktmodell beschreibt die Struktur und die Eigenschaften von Komponenten eines Software- oder Informationssystems. Darunter wird nicht nur der Code, sondern sämtliche Arbeitsergebnisse, die bei der Entwicklung oder Pflege anfallen oder benötigt werden, verstanden.“ Wallmüller ff. 74

[10] Bestandteil eines Software-Produktes; z.B. *Variable* ist eine Komponente des Software-Produktes *Implementierung bzw. Programmcode*

Die Abbildung 3.6 illustriert diese Qualitätsanforderungen.

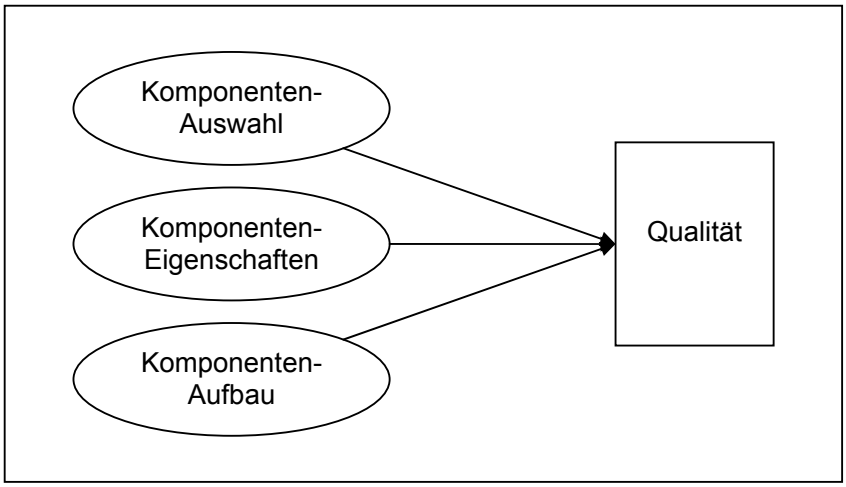


Abbildung 3.6: Einflussgrössen, die die Produktqualität bestimmen (aus [Dromey 1996])

Inhärente Formregeln (rules-of-form) bestimmen den Einsatz jedes Komponententyps. Eine Variable muss initialisiert werden, bevor sie benutzt wird. Und die Verknüpfungsregelen (rule-of-composition) bestimmen die Art und Weise, wie die Komponenten in Kontext anderer Komponenten eingesetzt werden. Die Verletzung einer dieser Regeln wirkt sich auf die Produktqualität aus. Einige Verletzungen sind stark genug, um die Funktionalität zu beeinflussen und somit das Produkt die vorgesehene Funktionalität nicht ausführen kann. Wenig schwere Verletzungen werden sich vielleicht nicht auf die Funktionalität auswirken, aber sie werden dann andere Qualitätseigenschaften wie die Effizienz und Wartbarkeit beeinflussen. Mögliche Ursachen für die Verletzung der Regeln sind:

- Benutzung der falschen Komponente in einem gegeben Kontext
- Falsche Implementierung einer Komponente
- Falsche Benutzung einer Komponente in Beziehung zu einer anderen

In diesem Abschnitt werden die Eigenschaften, die mit den Komponenten assoziiert werden können, beschrieben. Für die Auswahl der Eigenschaften ist nicht die Anzahl der Komponenten, sondern deren Qualität von Bedeutung: „Simply identifying large numbers of potential properties does not provide a basis for constructing a quality model.“ [Dromey 1996] S. 35. Die Abbildung 3.7 zeigt eine gute Basis für die Aufteilung der objektiven qualitätstragenden Eigenschaften der Komponenten.

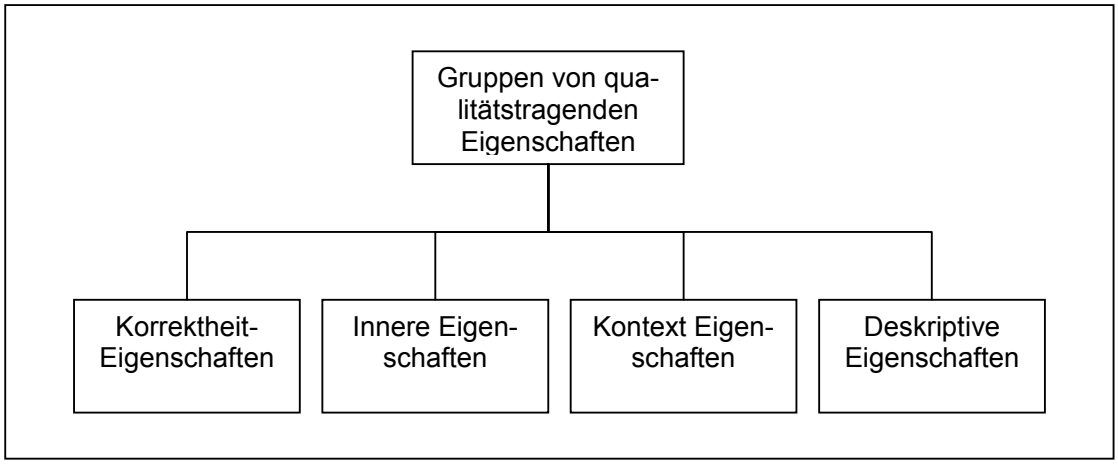


Abbildung 3.7: Produkteigenschaften, die die Qualität beeinflussen (aus [Dromey 1996])

Korrektheit-Eigenschaften

Fehler können entweder beim Einsatz der Komponente einzeln oder in einem Kontext auftreten. Einige Eigenschaften sind so signifikant, dass ihre Verletzung zum Fehlverhalten des Produktes führen kann. Solche Korrektheit-Eigenschaften brauchen eine besondere Betrachtung und deswegen sind sie separat klassifiziert. Sie können *intern* – assoziiert mit einer einzelnen Komponente – oder *kontextabhängig* – assoziiert mit dem wie die Komponenten in einem Kontext benutzt werden - sein.

Innere Eigenschaften

Jeder Komponente hat eine normale Form (bzw. ein normales Verhalten), die ihre innere „Richtigkeit“ definiert. Als Beispiel muss der Aufbau einer Programmschleife immer terminieren. Die normale Form einer Komponente sollte nicht verletzt werden (der Kontext wird nicht berücksichtigt). Die inneren Eigenschaften messen, wie gut eine Komponente, entsprechend ihrer vorgesehenen Verwendung oder ihren Implementierungsanforderungen, entwickelt wurde oder wie gut sie zusammengesetzt ist.

Kontextabhängige Eigenschaften

Die Produktqualität wird auch von der Art und Weise, wie die Komponenten zusammengesetzt sind, bzw. von den Beziehungen zwischen denen, beeinflusst. Die kontextabhängigen Eigenschaften beeinflussen selten die Korrektheit, hingegen öfter die wenig ernst genommenen Qualitätseigenschaften wie zum Beispiel die *Wartbarkeit*.

Deskriptive Eigenschaften

Damit eine Software nützlich ist, sollte sie verständlich sowie geeinget für die geplante Aufgabenbewältigung sein. Die deskriptiven Eigenschaften werden bei Anforderungsspe-

zifikationen, Design, Implementierung und Benutzerschnittstellen angewendet. Die Programmvariablen sollten zum Beispiel deskriptive Namen haben. Öfter spielen die deskriptiven Eigenschaften – entweder formell oder informell - eine wichtige Rolle bei der Spezifikation von Funktionalität und Randbedingungen oder bei der Identifizierung des Kontextes.

Qualitätseigenschaften

Software-Qualität ist oft in Hinsicht auf die Qualitätseigenschaften wie z.B. Funktionalität, Zuverlässigkeit und Wartbarkeit umstritten. Idealerweise sollte ein Satz von höheren Qualitätseigenschaften vollständig, kompatibel und nicht-überlappend sein. Aber oft hängt ein Qualitätsmerkmal von mehreren Produkteigenschaften ab, welche somit auch mehrere Qualitätseigenschaften beeinflussen. Beispielsweise beeinflussen verschiedene Redundanzformen sowohl die Effizienz, als auch die Wartbarkeit. Dromey löst dieses Problem wie folgt: Er verlinkt eine Produkteigenschaft als erstens mit derjenigen Qualitätseigenschaft, auf die sie den grössten Einfluss hat, und dann nacheinanderfolgend mit der zweitgrössten, usw.

Neben den sechs ISO 9126 Qualitätseigenschaften[11] verwendet Dromey noch zwei weitere Eigenschaften, nämlich *Wiederverwendbarkeit*[12] und *Prozessreife*[13]. *Prozessreife* ist besonderes wichtig, weil die Verbesserung des Qualitätsprozess zu höherer Produktqualität führt. Bergbauer sagt: „Nur gute Prozesse gewährleisten gute Produkte zu wettbewerbsfähigen Kosten“ [Bergbauer 2004] S. 1. Daher baut Dromey diese Eigenschaft bei jedem Produktqualitätsmodell ein.

Verlinkungen

Gemäss Dromey ist die Festlegung aller Verlinkungen zwischen Qualitätseigenschaften und Produktmerkmalen eine schwierige Aufgabe. Es ist ausserdem sehr entmutigend zu versuchen, alle diese Verlinkungen empirisch nachzuweisen. Daher sollte man nur für die vier bereits identifizierten Produkteigenschaften (korrekt, innerlich, kontextabhängig und

[11] Die sechs ISO 9126 Qualitätseigenschaften sind: Funktionalität, Zuverlässigkeit, Effizienz, Verwendbarkeit, Wartbarkeit und Portabilität. Weitere Details und deren Definitionen finden Sie in Kapitel 5.1

[12] Dromey definiert Wiederverwendbarkeit (reusability) wie folgt: „A structural form is *reusable* if it uses standard language features, it contains no machine dependencies and it implements a single, well-defined, encapsulated and precisely specified function whose computations are all fully adjustable and use no global variables or side-effects.” , wobei “structural form” ersetzt bei einer Software den Begriff Produkt-Komponente und ist definiert “structural form are the statement types and the statement components of the implementation language”

[13] „Die Reife eines Software-Prozesses ist das Ausmass, inwieweit ein konkreter Prozess explizit definiert, geführt, gemessen, kontrolliert und verbessert wird und effektiv ist“ [Wallmüller 2001] S. 82

deskriptiv) Verlinkungen festlegen. Ein Beispiel wäre die Verlinkung zwischen der Eigenschaft *Korrektheit* und den ISO 9126 Qualitätseigenschaften. Alle *Korrektheit* Eigenschaften unterstellen, dass eine Komponente – entweder einzeln oder in einem Kontext – nicht implementiert ist oder nicht wie vorgesehen funktioniert. Wenn diese Annahme richtig ist, dann ist weder die Funktionalität noch die Zuverlässigkeit dieses Produktes garantiert. Mit anderen Worten; damit das System sich zuverlässig verhält und seine funktionalen Anforderungen erfüllt, ist die Erfüllung der objektiven Korrektheitseigenschaft bei allen Produktkomponenten Voraussetzung. Daraus kann man schliessen, dass die *Korrektheit* die Qualitätseigenschaften *Funktionalität* und *Zuverlässigkeit* beeinflusst.

Modellkonstruktion

Die Konstruktion und Verfeinerung eines Produktqualitätsmodells beinhaltet fünf Schritte:

1. Identifizieren eines Satzes von Qualitätseigenschaften für das Produkt
2. Identifizieren der Produktkomponenten
3. Identifizieren die signifikantesten qualitätstragenden Eigenschaften für jede Komponente
4. Aufstellen eines Satzes von Axiomen für die Verbindung der Produkteigenschaften mit Qualitätseigenschaften.
5. Modellbewertung, Identifizierung seiner Schwächen und Verfeinerung oder Weglassen.

Auf diese Weise kann ein prüfbares, abschätzbares und redefinierbares Qualitätsmodell für die Schlüsselprodukte der Softwareentwicklung (Anforderungsspezifikationen, Design und Implementierung) konstruiert werden. Dromey empfiehlt drei Modelle, das Qualitätsmodell im Besonderen für das Produkt *Anforderungsspezifikationen, Design* und *Implementierung*. Weil meine Arbeit ein Bestandteil der Anforderungsspezifikationen ist, bzw. am meisten diesem Produkt ähnelt, wird hier nur dieses Qualitätsmodell beschrieben. Die Beschreibung der anderen zwei Modelle finden Sie in [Dromey 1996].

3.2.3.2 Qualitätsmodell für Anforderungsspezifikationen

In diesem Kapitel werden die Schlüsselaspekte des Qualitätsmodells für die Anforderungen behandelt; das Schwergewicht liegt jedoch eher auf den Anforderungen als den kompletten Anforderungsspezifikationen.

Identifizierung der Qualitätseigenschaften

Um die Qualitätseigenschaften zu identifizieren, sollte man sich folgende Frage stellen: Was will ich mit der Spezifikation machen? Hauptsächlich will man die Spezifikationen:

- für die Beschreibung der Anforderungsprobleme verwenden
- als Basis für das Design verwenden
- als Vertragsinstrument für ein gemeinsames Verständnis zwischen den Kunden, Benutzern und Entwicklern verwenden
- so ändern, um den neuen oder modifizierten Anforderungen nachzukommen
- wieder verwenden oder so anpassen, um andere Probleme lösen zu können

Wenn die Spezifikationen für die Problemanforderungen verwendet werden, sollte der Kunde in der Lage sein, sie zu verstehen und das Vertrauen haben, dass das Beschriebene tatsächlich seine Wünsche erfüllt. Um sie als Basis für das Design zu benutzen, sollte der Designer sie auch verstehen und auch sicher sein, dass sie alle wichtigen Informationen enthalten. Weil sich die Anforderungen während der Projektlebensdauer ändern, sollten die Spezifikationen auch diesen Modifikationen entgegenkommen. Daher muss das ergebende Qualitätsmodell verschiedene Bedürfnisse von unterschiedlichen Interessengruppen erfüllen.

Obwohl die Bedürfnisse der Interessengruppen bei allen Produkten gleich sind, haben die Qualitätseigenschaften unterschiedliche Gewichtungen. Beispielsweise ist die Eigenschaft *Verständlichkeit* beim Produkt *Anforderungsspezifikationen* viel wichtiger als *Funktionalität* und *Zuverlässigkeit*. Deshalb wählt Dromey andere Eigenschaften als ISO 9126, die die Qualitätsbedürfnisse und –erwartungen der Anforderungen besser erfüllen. Die Qualitätseigenschaften und deren Untereigenschaften können Sie der Tabelle 3.1 entnehmen.

Anforderungsspezifikationen: Qualitätseigenschaften und deren Untereigenschaften	
Eigenschaften	**Untereigenschaften**
Korrekt (accurate)	Regelrecht (conformant), funktional (functional), gültig (valid), abhängig (constrained)
Verständlich (understandable)	Motivierend (motivated), zusammenhängend (coherent), vollständig/in sich geschlossen (self-contained)
Praktisch anwendbar (implementable)	Ausführbar (achievable), realisierbar (viable), komplette Lösung (total-solution)
Anpassungsfähig (adaptable)	Veränderbar (modifiable), erweiterbar (extensible), wiederverwendbar (reusable)

Prozessreif (process-mature)	Kundenorientiert (client oriented), eindeutig definiert (well defined), sicher (assured), wirksam (effective)

Tabelle 3.1: Qualitätseigenschaften des Produktes Anforderungsspezifikationen (aus [Dromey 1996])

Identifizierung der Komponenten

Die Hauptursache vieler Probleme mit Anforderungen ist die Unklarheit darüber, was die Anforderungen genau sind und welche Form sie haben sollen. Die ausgewählte Form kann eine signifikante Wirkung auf die Qualität von Anforderungen haben. Die Anforderungen beginnen als eine Idee über ein wahrgenommenes Bedürfnis. Wie die Benutzer diese Bedürfnisse formulieren, hängt sehr vom Problemausmass ab. Die Anforderungen müssen sich von dieser Idee – die öfter sehr unklar und informell ist – zu etwas, das objektiv, korrekt, prüfbar, verifizierbar und implementierbar ist, entfalten.

Es gibt verschiedene Ansichten, wie die Anforderungen spezifiziert werden sollten aber es gibt kein universeller Ansatz, welcher bei jedem System anwendbar ist. Hier wird nicht darauf eingegangen, ob die Anforderungen formell oder informell sein sollten, sondern, ob die wesentlichen Grundformen, welche die Anforderungen unabhängig von ihrer Darstellung besitzen sollen, behandelt werden. Um gute Anforderungen zu schreiben, muss man über die Darstellungen hinaus die Grundformen sehen.

Ausgehend von obengenannten Überlegungen beinhalten die Anforderungsspezifikationen folgende Komponenten:

- Ein Satz von Anorderungen
- Einzelne Anforderungen
- Bedingungen
- Variablen
- Konstanten *und*
- Beziehungen

Identifizierung der qualitätstragenden Eigenschaften

Die Einzel-Anorderungen sind die Schlüsselkomponenten einer Anforderungsspezifikation. Bei Zuweisung der qualitätstragenden Eigenschaften zu den Einzel-Anforderungen, sollte man besonders vorsichtig sein. Beispielsweise sollte man vermeiden, dass die Eigenschaften, die eigentlich zu einer anderen Komponente (Variablen und Bedingungen) gehören, fälschlicherweise einer Einzel-Anforderung zugewiesen werden. Die meisten qualitätstragenden Eigenschaften beziehen sich auf die Relationen zwischen

den Anforderungskomponenten. So bezieht sich eine Eigenschaft auf die Form der Anforderungen. Alle Eingaben und Ausgaben, die mit dieser Anforderung assoziiert sind, müssen vorhanden sein. Wenn diese Anforderung sie nicht beinhaltet, dann ist sie nicht komplett.

Eine zweite qualitätstragende Eigenschaft von Anforderungen ist: Ein Satz von Bedingungen, die mit einer Anforderung assoziiert sind, müssen kompatibel und daher konsistent sein. Damit eine Anforderung, das was auch verlangt wird, erfassen kann, muss sie auch explizit, präzis und nicht-redundant sein.

Schliesslich ist es auch sehr wichtig, dass eine Spezifikation zu den Benutzeranforderungen zurückverfolgt werden kann und es sollte verifiziert werden, ob jede einzelne Anforderung erfüllt ist. In der Abbildung 3.8 werden die qualitätstragenden Eigenschaften von Einzel-Anforderungen gezeigt.

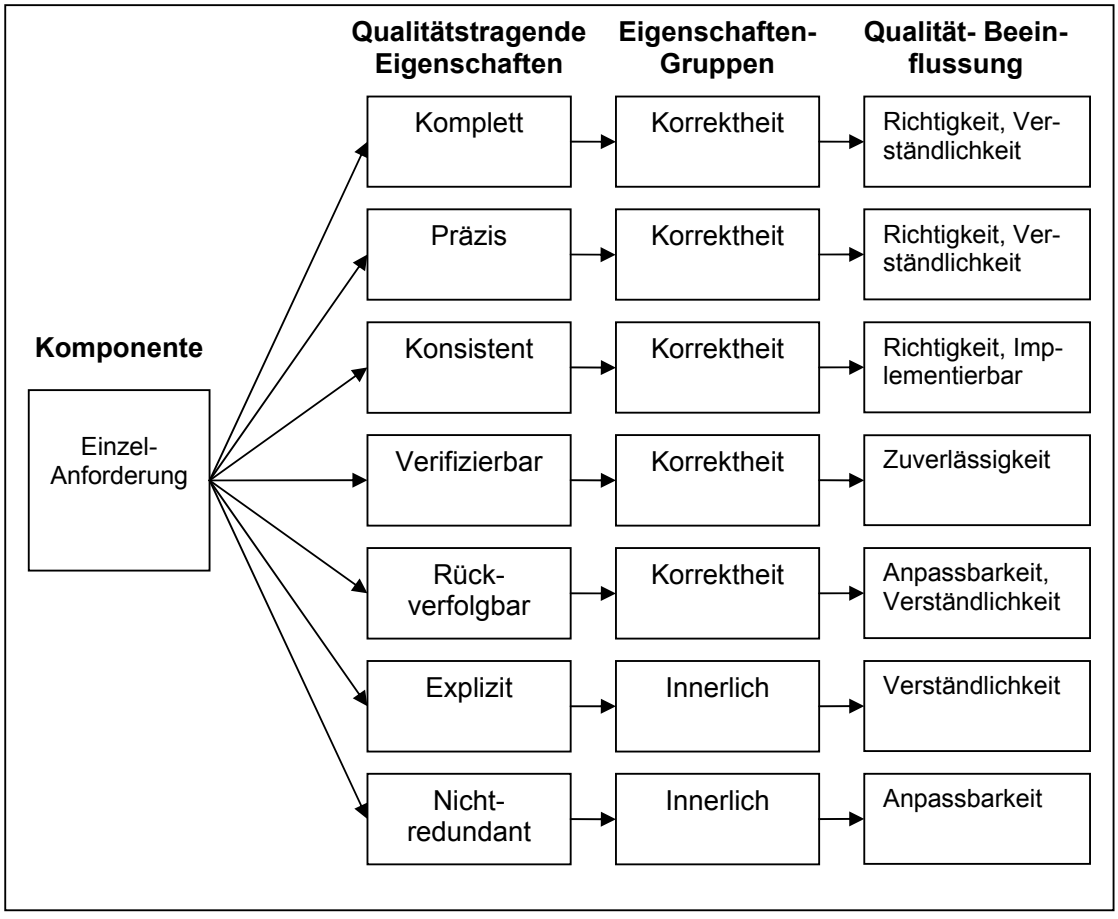


Abbildung 3.8: Verlinkung der Produkt- zu den Qualitätseigenschaften (aus [Dromey 1996])

Verlinkung der Produkt- mit den Qualitätseigenschaften

Schliesslich müssen die Produkteigenschaften erstens einer Gruppe von qualitätstragenden Eigenschaften zugeordnet und zweitens mit den Qualitätseigenschaften verlinkt werden. Die Produkteigenschaft *Komplett* gehört zu der Eigenschaftsgruppe *Korrektheit* und beeinflusst die Qualitätseigenschaften *Richtigkeit* und *Verständlichkeit*. Die anderen Verlinkungen können Sie der Abbildung 3.8 entnehmen.

3.3 Zusammenfassung

Viele Autoren haben verschiedene Qualitätsmodelle als Basis für die Software-Beurteilung vorgeschlagen, aber die meisten befassen sich mit der Messung von herkömmlicher Software [Kitchenham *et al.* 1999] und [Dromey 1996], anstatt mit der Auswahl von Software-Paketen wie bei Franch's Modell [Franch und Carvallo 2003] der Fall ist. Das Qualitätsmodell von Franch ist domäne-spezifisch (für die Auswahl von Softwarepaketen) und sich auf ISO/IEC 9126 Standards [ISO/IEC 9126-1 2001] stützt, d.h. als Basis seines Modells dienen, die von der ISO/IEC 9126 vorgeschlagenen Qualitätseigenschaften und –untereigenschaften. Die meisten Eigenschaften zerlegt er in Qualitätsmerkmale, die dann mittels Metriken gemessen werden können.

Die Squid-Methode konzentriert auf die firmeninterne Software-Qualität [Kitchenham *et al.* 1999]. Das Ziel von Squid ist breiter als der Ansatz von Franch; es deckt die Qualität bei der Planung, Lenkung und Auswertung sowie beim Modellieren. Aber Squid ist trotzdem verwandter mit Franch's Qualitätsmodell als mit dem von Dromey: die externen Eigenschaften stehen im Vordergrund und sind kontext-abhängig und beide Methoden forschen nach Wiederverwendbarkeit der Modelle, indem sie robuste Merkmale identifizieren. Auf der anderen Seite schlägt Squid kein besonderes Verfahren für die Erstellung des Qualitätsmodells vor und die ISO/IEC 9126 Standards sind nicht erforderlich. Der signifikanteste Unterschied ist, dass Squid nicht eindeutig zwischen dem Qualitätsmodell in sich und seinem Verwendungskontext unterscheidet (das Modell beinhaltet obere Zielwerte für die Merkmale). Somit wird die Wiederverwendung in anderen Kontexten beeinträchtigt.

Dromey's Modell, im Gegensatz zu den anderen zwei, verwendet eine andere Methodik, nämlich den Buttom-Up Ansatz, d.h. er baut diejenigen internen Eigenschaften (messbaren und / oder abschätzbaren) in das Produkt ein, welche die gewünschten externen (subjektiv wahrnehmbaren) Qualitätseigenschaften einbringen, wie zum Beispiel die Funktio-

nalität, die Zuverlässigkeit, etc. Es gibt noch zwei wesentliche Unterschiede zwischen den zwei obengenanten Modellen und der Arbeit von Dromey: er bindet die Metriken nicht in das Modell ein und sein Qualitätskonzept ist kontext-unabhängig. Franch führt beispielsweise die Idee der kontext-abhängigen Qualitätsmerkmale ein als Mittel für die Qualitätsmessung; er berücksichtigt den Kontext ‚Softwarepaketauswahl'. Neben den Unterschieden gibt es auch Ähnlichkeiten: Dromey empfehlt die Verwendung der ISO/IEC 9126 Standards, obwohl er sie nicht voraussetzt.

4 Beschreibung des Anwendungsbereichs

Dieses Kapitel befasst sich mit der Beschreibung des Anwendungsbereichs. Dabei werden weder die Applikationsschnittstellen noch die betroffenen Technologien detailliert erläutert. Ziel ist es eine allgemeine Einführung und Überblick über die Schlüsselkonzepte, die bei der Spezifikation von Applikationsschnittstellen gehören, zu geben.

Der Begriff Schnittstelle ist in Software-Kontext einwenig irreführend. Bei einer Schnittstelle wird nichts durchgeschnitten, sondern im Gegenteil: Schnittstellen verbinden, und zwar entweder Applikationen, bzw. deren Komponenten untereinander (*Applikationsschnittstellen* oder kurz *Schnittstellen*[14]) oder Komponenten mit dem Benutzer (*Benutzerschnittstellen*) [Siedersleben 2004]. Thema der vorliegenden Arbeit sind die (Applikations-) Schnittstellen. Daher wird zunächst in diesem Kapitel das Problembereich definiert, nämlich die Schnittstellenspezifikation. Der zweite Abschnitt zeigt wie eine Schnittstelle identifiziert und aufgebaut wird. Und im letzten Abschnitt dieses Kapitels werden die möglichen Schnittstellentypen eines betrieblichen Informationssystems behandelt.

4.1 Spezifikation der Applikationsschnittstellen

Die Erstellung von Software im industriellen Massstab ist ein komplexer Prozess. Software umfasst erheblich mehr als nur Programme. Der Aufwand für das Schreiben der Programme beträgt in der Regel nur 10-20 Prozent des Gesamtaufwands für die Entwicklung von Software [Glinz 2005]. Software-Entwicklung umfasst alle Tätigkeiten und Ressourcen, die zur Herstellung von Software notwendig sind. Glinz definiert die Software-Entwicklung wie folgt: „Die Umsetzung der Bedürfnisse von Benutzern in Software umfasst: Spezifikation der Anforderungen, Konzept der Lösung, Entwurf und Programmierung der Komponenten, Zusammensetzung der Komponenten und ihre Einbindung in vorhandene Software, Inbetriebnahme der Software sowie Überprüfung des Entwickelten nach jedem Schritt.“ [Glinz 2005] S. 5. Jede systematische Entwicklung von Software sowie alle grösseren, abgrenzbaren Pflegevorhaben benötigen einen Abwicklungsrahmen, das in der Regel als Software-Projekt bezeichnet wird. Die bei der Definition von Glinz genannten Tätigkeiten werden in einem Projekt als Phasen bezeichnet. In der Abbildung 3.1 sind diese Phasen graphisch dargestellt.

[14] Es wird oft der Begriff *Schnittstelle* verwendet, wobei gemeint ist die *Applikationsschnittstelle*

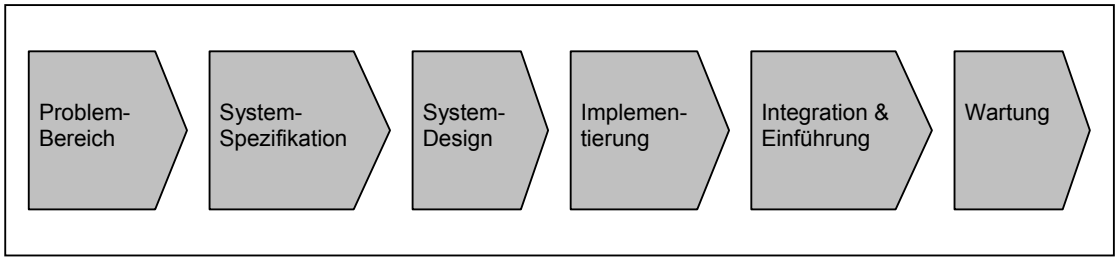


Abbildung 4.1: Die Phasen eines Software-Projektes (in Anlehnung an [Siedersleben 2003])

Die Phasen des Software-Projektes (Abbildung 4.1) stellen keine strikte zeitliche Reihenfolge dar, sondern sie überlappen sich und haben wechselseitige Einflüsse aufeinander [Siedersleben 2003]. Die Abbildung 4.2 skizziert eine typische Verzahnung und Wechselwirkung von der Spezifikations- bis zu der Integrationsphase. Dabei ist beispielsweise ersichtlich, dass mit dem Design (Systemkonstruktion) begonnen wird, bevor die Applikation fertig spezifiziert ist. Die Software muss auch nicht fertig implementiert sein, um sie in die IT-Landschaft integrieren zu beginnen.

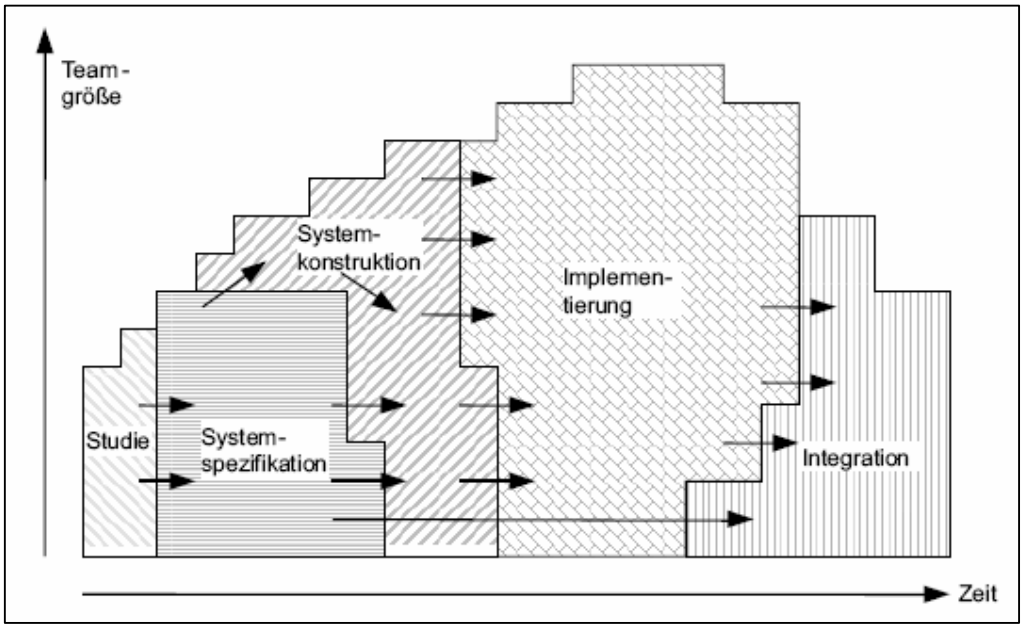


Abbildung 4.2: Verzahnung und Wechselwirkung der Phasen (aus [Siedersleben 2003])[15]

Zusätzlich zu den geschilderten Tätigkeiten gibt es auch phasenübergreifende Querschnittaufgaben: Projekt-, Qualitäts- und Konfigurationsmanagement.[16]

[15] Der Begriff ***Studie*** ist ein Synonym für *Problembereich*. Dabei geht es um folgende Punkte: Analyse der Ist-Situation, Anforderungen, Lösungsvarianten, Empfehlung und Vorschlag eines Projektplans [Siedersleben 2003]. Mit dem Begriff ***Systemkonstruktion*** ist *Design* gemeint.

Die Spezifikationsphase befasst sich mit den Leistungen, die ein System erbringen soll. Die Spezifikationen enthalten eine detaillierte, konsistente, vollständige und nachvollziehbare Beschreibung der fachlichen Anforderungen. Sie sind gleichzeitig auch eine verbindliche Vorgabe für alle folgenden Phasen [Siedersleben 2003]. Gemäss Siedersleben 2003 bestehen die Systemspezifikationen aus folgenden Hauptteilen:

- Geschäftsprozesse
- Anwendungsfälle (use cases)
- Benutzerschnittstelle – Fensterformulare, Dialogabläufe
- Objektmodell – Daten-/ Funktionenmodell
- Nachbarsysteme – Schnittstellen

Die Spezifikationen entstehen nicht sequentiell bzw. die obengenanten Bestandteile werden nicht sequentiell spezifiziert, sondern in mehreren Iterationen. Mehrfaches Durchdenken und Präzisieren der Anforderungen durch die beteiligten Fachbereiche führen letztlich zur Spezifikation. Die Abbildung 4.3 verdeutlicht diesen iterativen Prozess. Die horizontale Achse betont die verschiedenen Anforderungsmerkmale der Spezifikationsteilen. Auf der rechten Seite der Achse befinden sich die Teile, die der Präzision, Konsistenz und Vermeidung von Redundanz dienen. Hingegen sollen die auf der linken Seite der Achse aufgelisteten Bestandteile die Verständlichkeit und Lesbarkeit verbessern.

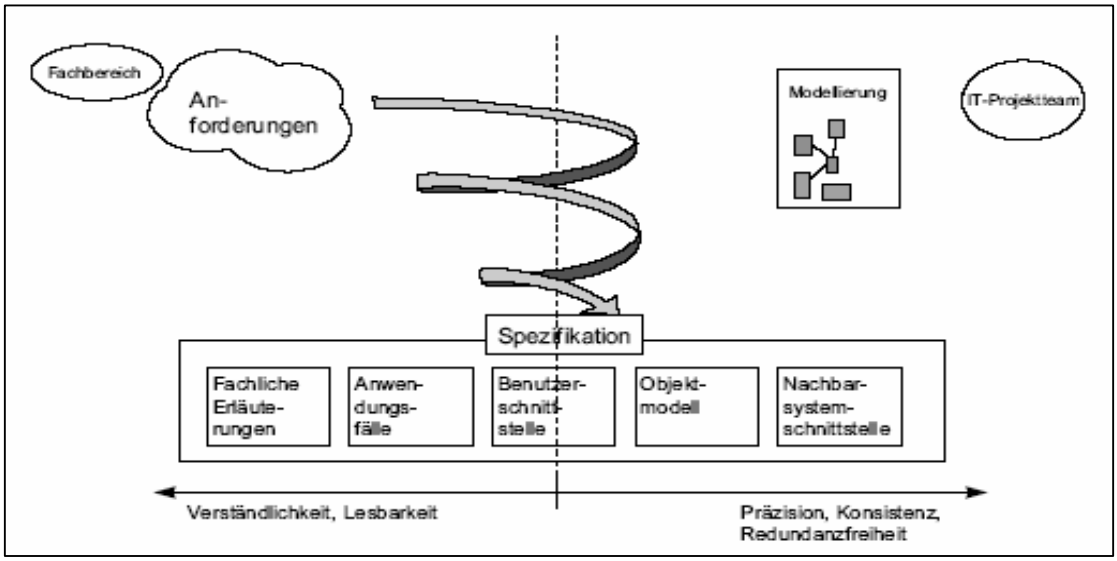


Abbildung 4.3: Prozess des Spezifizierens (aus [Siedersleben 2003])

[16] Die Querschnittsaufgaben Projekt- und Konfigurationsmanagement sowie die Tätigkeiten Problembereich, Design, Implementierung, Integration und Wartung sind nicht Bestandteil der vorliegenden Arbeit. Für nähere Informationen sei auf [Siedersleben 2003] verwiesen.

Die oben genannten Hauptteile der Systemspezifikationen bedürfen einer genaueren Aufteilung. [Siedersleben 2003] bezeichnet die Spezifikationsteile als Bausteine (insgesamt 17), die er sie auch als Fahrplan für die Erstellung von Spezifikationen benutzt. Dieser Bausteincharakter bzw. die Aufteilung der (betrieblichen) Informationssysteme in verschiedene Themen macht es möglich, die Domäne der vorliegenden Diplomarbeit besser einzugrenzen. Die Informationssysteme haben trotzt grosser Vielfalt doch immer wieder dieselbe Struktur (vgl. Abbildung 4.4). Dabei ist es ersichtlich, dass das Gesamtsystem aus dem Anwendungskern und einigen Subsystemen besteht, die mittels Schnittstellen mit vielen anderen Systemen (Nachbar- und Altsystemen) verbunden sind. Der Aufbau der Spezifikationen basiert sich an dieser Struktur. In der Abbildung 4.5 sind alle Bausteine in Übersicht gezeigt. Jeder Baustein dient als eine Anleitung für einen bestimmten Teil der Spezifikation.

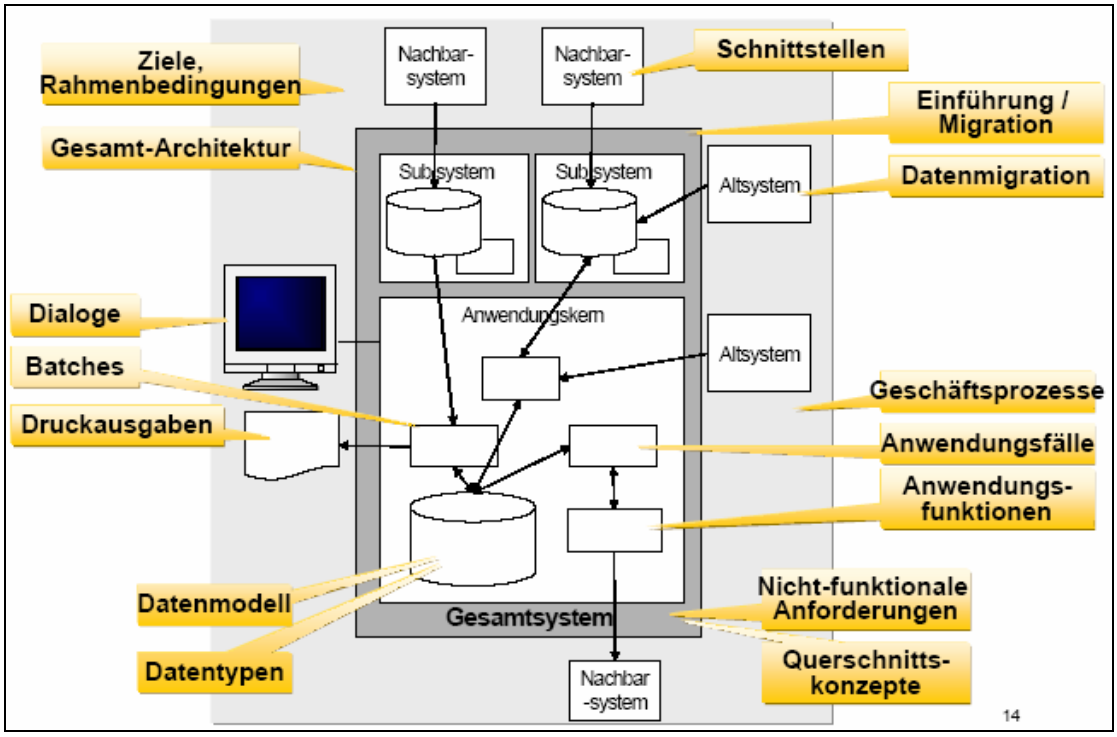


Abbildung 4.4: Aufbau betrieblicher Systeme (aus [Siedersleben 2003] S. 50)

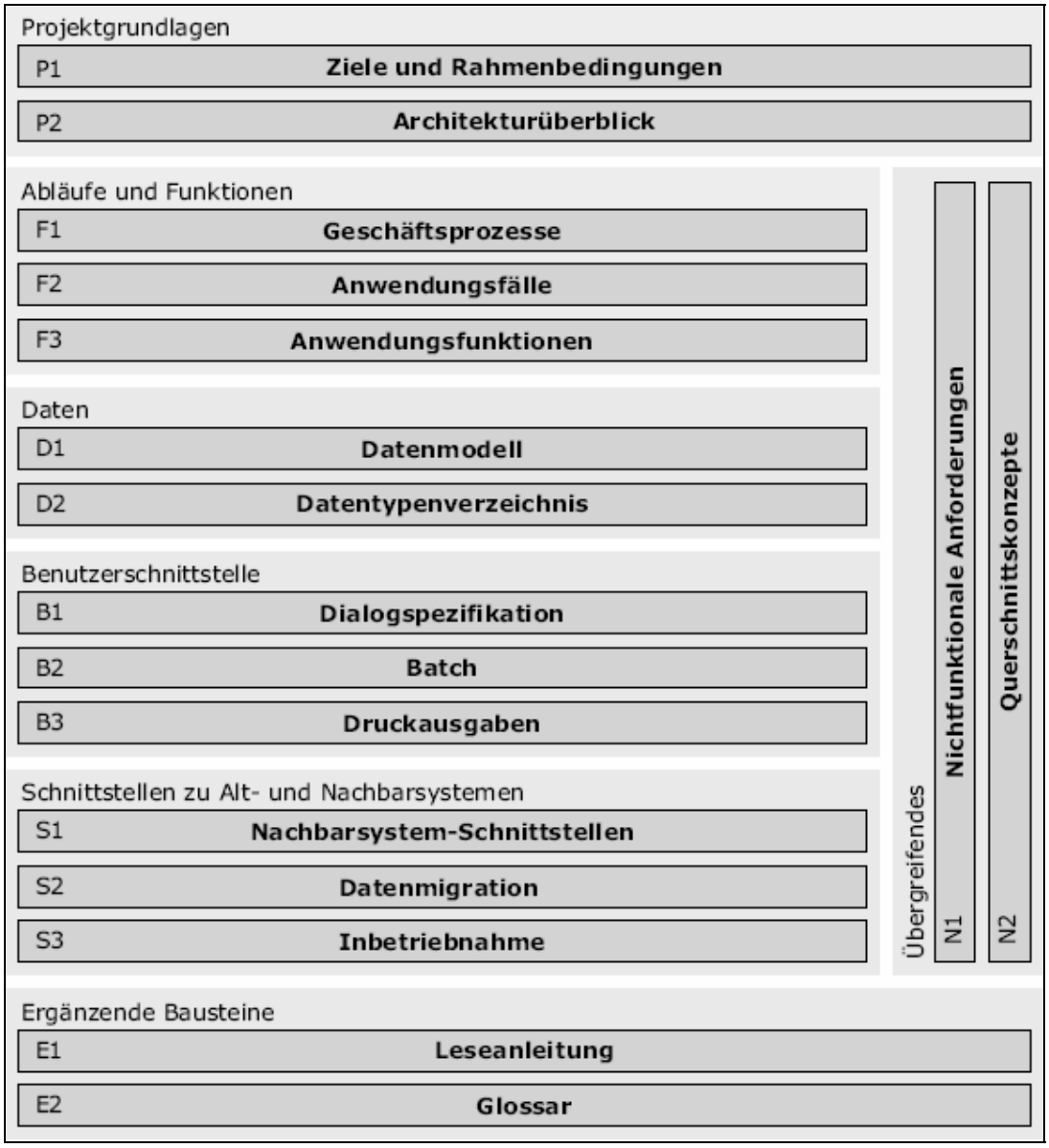


Abbildung 4.5: Die Bausteine der Spezifikation (aus [Siedersleben 2003] S. 51)

Gegenstand der vorliegenden Diplomarbeit ist die Bausteingruppe ***Schnittstellen zu Alt- und Nachbarsystemen.*** Alle anderen Bausteingruppen werden entweder als Unterstützungsaufgabe für die Schnittstellen verwendet oder erfahren hier keine weitere Behandlung.[17] Diese Bausteingruppe beinhaltet die Datenmigration, Inbetriebnahme und Nachbarsystem-Schnittstellen.

[17] Für nähere Informationen zu den nicht behandelten Bausteinen sei auf [Siedersleben 2003] verwiesen.

Datenmigration

Der Baustein Datenmigration beschreibt die Überführung der Daten in das neue System bei der Ablösung der Altsysteme. Die Aufgabe dieses Bausteins besteht in der Transformation der Daten. Dazu wird eine Abbildungstabelle (Mapping) – wie auch bei Nachbarsystem-Schnittstellen (vgl. Kapitel 4.2) – verwendet. Neben der Transformation geht es darum, die Datenkonsistenz beim Übergang zu gewährleisten. Dies ist gemäss Siedersleben aus folgendem Grund ein Problem: „Zum Übernahmezeitpunkt gibt es im Altsystem eine ganze Reihe von begonnen Anwendungsfällen, die im Neusystem zu einem guten Ende zu führen sind. Es ist überhaupt nicht klar, ob diese Halbfabrikate im Neusystem überhaupt unterzubringen sind. Manchmal stellt sich heraus, dass im Neusystem provisorische Anwendungsfälle erforderlich sind, um die Altlasten nach und nach abzuarbeiten.“ [Siedersleben 2003] S. 62.

Inbetriebnahme

Der Baustein Inbetriebnahme befasst sich mit dem Prozess der Ablösung von produktiven Altsystemen durch das Neusystem. In diesem Baustein werden gezeigt, wie dieser Übergang durchgeführt werden kann sowie der Ablauf und die Konsequenzen der gewählten Migrationsstrategie. In den meisten Fällen wird eine gestufte, schrittweise Einführungsstrategie gewählt. Im Rahmen der Inbetriebnahme werden alle möglichen Fehler gesammelt und analysiert. Dabei kann eine zu hohe Fehlerbehebungsmassnahme auch die Rückkehr zu Altsystem bedeuten [Siedersleben 2003].

Nachbarsystem-Schnittstellen

Bei der Spezifikation der Anforderungen an grosse Systeme (insbesondere Anwendungsprobleme betrieblicher Natur) ist es von grosser Bedeutung, sich darüber klar zu werden, wie ein solches System in sein Umfeld eingebettet sein soll und wo die Systemgrenzen liegen. Es gibt kaum ein System, das nicht in eine vorhandene Systemlandschaft eingebettet ist. Glinz beschreibt diesen Sachverhalt folgendermassen: „Ein Software-System kann man sich als eine spezialisierte Problemlösungsmaschine vorstellen, welche über Schnittstellen mit ihrer Umwelt interagiert.“ [Glinz 2005] S. 93. Jedes System kommuniziert mit einer Reihe von Partnern, die im eigenen Unternehmen oder auch ausserhalb liegen.

Bei den Nachbarsystem-Schnittstellen spielt die Tatsache welche Systeme dabei involviert sind eine grosse Rolle. Des Weiteren sind der Aufbau von Schnittstellen sowie auch die Kommunikation zwischen zwei Partnersysteme von Bedeutung.

4.2 Identifizierung und Aufbau der Schnittstellen

Die Spezifikation von Schnittstellen fordert eine Reihe von Aufgaben, bzw. Meilensteinen. In einem **ersten Schritt** werden alle Nachbarsysteme identifiziert und der grobe Datenfluss bestimmt. Die Bestimmung aller beteiligten Nachbarsysteme ist meistens nicht ganz einfach; oft wird ein Nachbarsystem erst spät, manchmal sogar nach Inbetriebnahme entdeckt [Siedersleben 2003]. Für jedes Nachbarsystem sind folgende Angaben erforderlich [Siedersleben 2003] und [PADi 2005]:

- *Organisatorische Zuständigkeiten*: Nachbarsysteme liegen in der Regel in andere Unternehmungsorganisationseinheiten. Deswegen kommt es oft zu Abstimmungsproblemen wie bei der Bereitstellung von Testdaten, Vergabe von Zugangsrechten und Anpassung des Nachbarsystems. Der letzte Punkt ist besonderes schwierig, da die Verantwortlichen des Nachbarsystems nicht dazu gebracht werden, Änderungen durchzuführen, die deren System nicht zu gute kommen. Es wird zwar versucht, Nachbarsysteme möglichst wenig zu ändern, manchmal sind aber Änderungen unausweichlich.
- *Version*: In der Regel werden die Systeme ständig weiterentwickelt, bzw. den veränderten Bedürfnisse angepasst. Dies bedeutet, dass ein Software-System aus mehreren Versionen (Releases) besteht. Deswegen ist es wichtig, die neueste Version des Nachbarsystems auszuwählen; damit werden mögliche Abbildungsunstimmigkeiten vermieden.
- *Art der Koppelung*: Die Systeme können eng oder lose gekoppelt sein. Enge Koppelung heisst, dass entweder alle Systeme laufen oder keines – jedes reisst bei einem möglichen Absturz alle anderen mit sich. Lose Koppelung bedeutet, dass jedes System wenigstens innerhalb gewisser Grenzen autonom ist, d.h. es kann ohne die anderen weiterlaufen. In Abschnitt 4.3 wird dieses Thema ausführlicher behandelt.
- *Datentransformation*: Jedes System hat seine eigene Vorstellung von dem Format der Daten und den Konsistenzbedingungen, denen sie genügen.

Für die Schnittstellen sind noch folgende Angaben erforderlich:

- *Frequenz und Art der Kommunikation*: Wie oft findet eine Interaktion statt, z.B. 5 Mal pro Stunde, 1500 Nachrichten pro Tag, etc.
- *Auslöser (Trigger)*: Welche Applikation ist der Anforderer oder Auslöser bei einer Schnittstelle. Grundsätzlich wird zwischen drei Auslösertypen unterschieden:
    - Data Push: Das Ursprungssystem initiiert den Datentransfer
    - Data Pull a): Das Zielsystem initiiert den Datentransfer

- Data Pull b): Das Zielsystem initiiert den Datentransfer durch ein Zwischensystem

In der Abbildung 4.6 sind die drei Typen graphisch dargestellt.

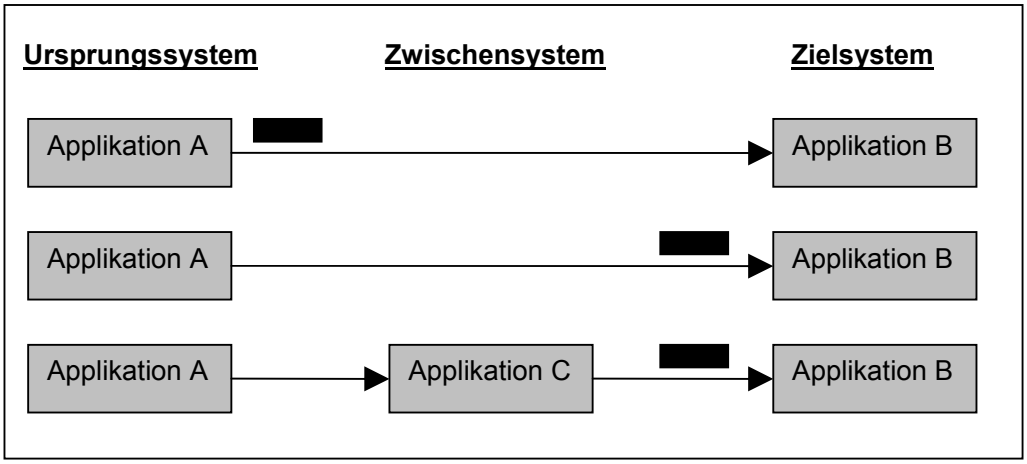


Abbildung 4.6: Trigger-Typen (aus [PADi 2005])

- *Technologie*: Die Technologie, die bei der Implementierung der Schnittstelle benötigt wird, z.B. EJB Services, Web Service, ETL (vgl. Abschnitt 4.3.3), JDBC, etc.

Nachdem die Nachbarsysteme definiert und der grobe Datenfluss bestimmt wurde, folgt in einem **zweiten Schritt** die Festlegung der Abbildungsattributen (Mapping). Dazu verwendet man eine Abbildungstabelle beispielsweise mit den folgenden sechs Spalten: Release Nummer, Attribut_Key, Attributname, Formattyp, Transformationsregeln und –anmerkungen (Mapping) und Notizen. Die folgende Tabelle zeigt exemplarisch die Abbildung des Attributs Vertragsabschlussdatum ins Data Warehouse.

Name der DWH-Tabelle (-View)					
Release	**Attribut_ID**	**Attributname**	**Formattyp**	**Abbildung (Mapping)**	**Notiz**
1.5	INCEPT_DT	Vertrags-abschluss	Datum (tt.mm.jjjj)	X.ACC.INCEPTION and X.ACC.INCTIME	Kurz be-schreiben

Tabelle 4.1: Abbildung des Attributs Vertragsabschlussdatum

Bei Swiss Re gibt es neben den zwei obengenanten Schritten noch zwei weitere, die sich mit der Implementierung bzw. Anpassung der Applikation befassen.

- *Implementierung der Mechanismen für die Prozesskontrolle*: Definition und Implementierung von bidirektionalen Kommunikationsmechanismen zwischen den Applikationen. Jede aktive Komponente (z.B. listener / demon) wird etabliert und die Datendienste (data services) werden angepasst – falls nötig.
- *Applikationen anpassen*: Die involvierten Zielapplikationen haben ihre Implementierungslogik für die Nutzung von neuen Schnittstellen bereits angepasst (eingerichtet)

Die Meilensteine (vier Schritte) für die Entwicklung von Schnittstellen sind in der Abbildung 4.7 dargestellt.

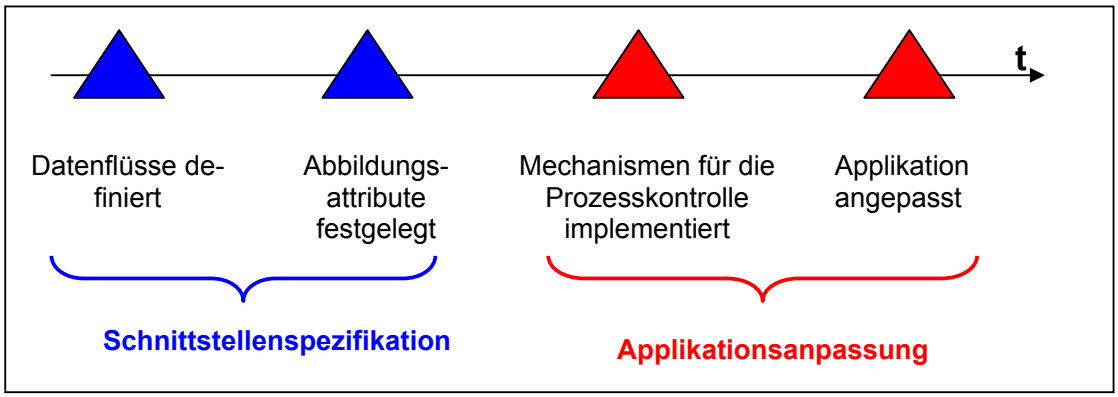


Abbildung 4.7: Die vier Meilensteine bei der Entwicklung einer Schnittstelle

4.3 Schnittstellentypen

4.3.1 Punkt-zu-Punkt Schnittstellen

Wie im vorherigen Abschnitt erläutert wurde, kommuniziert jedes System mit einer Reihe von Partnern, die im eigenen Unternehmen manchmal aber auch ausserhalb liegen. Daher stellt sich die Frage wie zwei Systeme miteinander kommunizieren. Die gebräuchlichsten Methoden für die Kommunikation zwischen verteilten Systemen sind die entfernten Prozeduraufrufe (RPC[18]) bzw. Objektaufrufe (RMI[19]) und die nachrichtenorientierte Kommunikation [Tanenbaum 2003].

[18] Remote Procedure Call (RPC)
[19] Remote Method Invocation (RMI)

Die folgende Beschreibung der entfernten Prozeduraufrufe und der nachrichtenorientierten Kommunikation sowie deren Aufteilung basiert auf den Ausführungen von [Tanenbaum 2003].

RPC

Wenn ein Prozess auf Maschine A eine Prozedur auf Maschine B aufruft, wird der aufrufende Prozess auf A unterbrochen, und die Ausführung der aufgerufenen Prozedur findet auf B statt. Informationen können in Parametern vom Aufrufer zum Aufgerufenen transportiert werden, um im Prozedurergebnis zurückzukommen. RPC bieten sowohl synchrone als auch asynchrone Kommunikationsfunktionen (vgl. Abbildung 4.8).

RMI

Ein RMI ist im Wesentlichen ein RPC, aber spezifisch für ein entferntes Objekt. Der grösste Unterschied ist, dass RMIs erlauben, systemübergreifende Objektreferenzen als Parameter zu übergeben.

Die nachrichtenorientierte Kommunikation wird in persistente und transiente Kommunikation aufgeteilt. Bei einer **persistenten Kommunikation** wird eine Nachricht, die für die Übertragung weitergegeben wurde, vom Kommunikationssystem solange gespeichert, wie es dauert, sie and den Empfänger auszuliefern. D.h. weder der Sender noch der Empfänger müssen aktiv sein und laufen, damit die Nachrichtenübertragung stattfinden kann; ein E-Mail-System ist ein typisches Beispiel. Im Gegensatz dazu wird eine Nachricht bei der **transienten Kommunikation** vom Kommunikationssystem nur solange gespeichert, bis sie ausgeliefert ist; die Router kommunizieren auf dieser Weise.

Kommunikation kann nicht nur persistent oder transient sein, sondern auch asynchron oder synchron. Bei der **asynchronen Kommunikation** kann der Sender unmittelbar nach der Weitergabe der Nachricht zur Übertragung fortgesetzt werden, möglicherweise sogar bevor sie überhaupt gesendet wurde. Das bedeutet, dass die Nachricht auf dem sendenden Host in einem lokalen Puffer gespeichert wird. Bei der **synchronen Kommunikation** wird der Sender blockiert, bis eine Nachricht in einem lokalen Puffer auf dem empfangenden Host abgelegt ist. Alternativ kann der Sender blockiert werden, bis die Nachrichtenauslieferung stattgefunden hat, oder sogar wie bei RPCs bis der Empfänger geantwortet hat. Abbildung 4.8 zeigt einen Überblick über Synchronität in der Kommunikation.

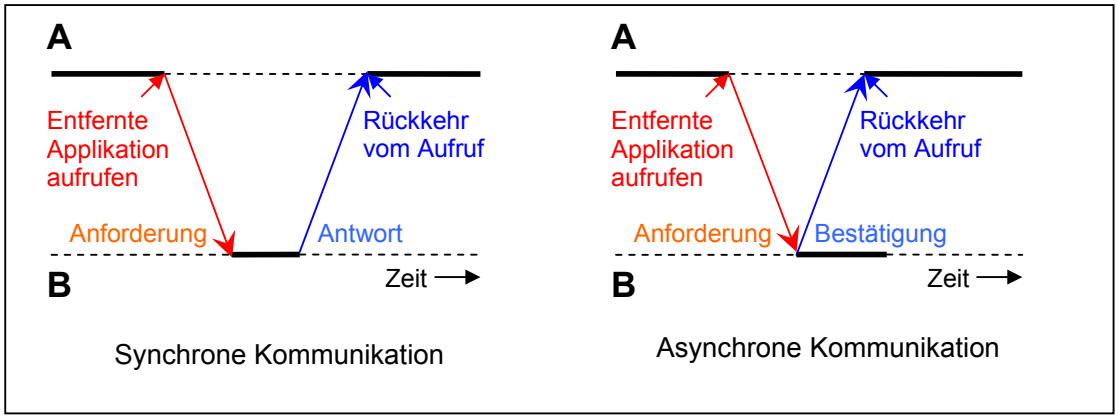


Abbildung 4.8: Synchrone und asynchrone Kommunikation (in Anlehnung an [Tanenbaum 2003])

4.3.2 Das Broker-Muster

Bei den oben beschriebenen Kommunikationstypen findet der Datenaustausch zwischen zwei Applikationen (Komponenten), d.h. die Applikationen verwenden eine Punkt-zu-Punkt Schnittstelle. Dieser Schnittstellentyp kann besonderes bei den grossen Unternehmungen, die eine Vielzahl von Applikationen besitzen, problematisch werden. Das führt zu den so genannten Spagettischnittstellen. Somit kann niemand ein Überblick über alle Schnittstellen haben und die Pflege der ganzen Architektur wird äusserst zeit- und kostenintensiv. Viele Unternehmungen lösen diese Probleme, indem sie EAI (Enterprise Application Integration) Produkte einführen. EAI ist die Integration heterogener Software-Welten unter Nutzung modernster Middleware-Tools von Dritt-Anbietern sowie Eigenentwicklungen mit dem Ziel Geschäftsprozesse systemübergreifend abzubilden. EAI spielt die Rolle eines Vermittlers (Message Broker), d.h. Applikationen tauschen die Daten nicht direkt miteinander, sondern durch diese Message Broker. Sie nehmen die Nachrichten von einer Applikation entgegen, routen, transfomieren in das Zielformat und liefern sie in die Zielapplikation ab. Die meisten EAI Werkzeuge verwenden XML für den Austausch von Daten.

Für die Integration von Applikation in EAI werden meistens gleiche Verfahren verwendet, d.h. für die Lösung eines aktuellen Problems wird die gleiche Grundidee verwendet, die auch bei ähnlichen Problemen aus der Vergangenheit eingesetzt wurde. Eine Lösung, die mehrmals verwendet werden kann, wird in der Software-Sprache als Muster bezeichnet. Buschmann definiert den Begriff des Muster folgendermassen: „Ein Software-Architektur-Muster beschreibt ein bestimmtes, in einem speziellen Entwurfskontext häufig auftretendes Entwurfsproblem und präsentiert ein erprobtes generisches Schema zu seiner Lösung. Dieses Lösungsschema spezifiziert die beteiligten Komponenten, ihre jeweiligen

Zuständigkeiten, ihre Beziehungen untereinander und die Art und Weise, wie sie kooperieren“ [Buschmann *at al*. 1998] S. 8.

Für die Spezifikation der Schnittstellen zu EAI kann das **Broker-Muster** von [Buschmann *at al*. 1998] verwendet werden. Die folgende Beschreibung der ausgewählten architektureller Muster basiert auf den Ausführungen von [Buschmann *at al*. 1998] und [Gall *at al.* 2003].[20]

„Das Broker-Muster kann für die Strukturierung verteilter Software-Systeme mit entkoppelten Komponenten benutzt werden, die durch das Aufrufen entfernter Dienste interagieren. Ein Vermittler (engl. Broker) ist für die Koordination der Kommunikation, zum Beispiel die Weiterleitung von Dienstanforderungen, sowie für die Übermittlung von Ergebnissen und Fehlermeldungen verantwortlich“ [Buschmann *at al*. 1998] S. 99. Buschmann gliedert seine Beschreibung der Mustern in folgende Teile:

Kontext: Es handelt sich um eine verteilte und eventuell heterogene Umgebung mit voneinander unabhängigen, miteinander kooperierenden Komponenten.

Problem: Die Erstellung eines komplexen Systems als eine Sammlung von entkoppelten, aber interagierenden Komponenten zu konstruieren führt zu einer grösseren Flexibilität, Wartbarkeit und Änderbarkeit. Durch die Aufteilung der Funktionalität in voneinander unabhängige Komponenten wird das System potentiell verteilbar und skalierbar.

Lösung: Durch Verwendung des Brokers kann eine Applikation auf verteilte Services zugreifen, indem das entsprechende Objekt durch einfaches Senden von Nachrichten angesprochen wird, anstatt sich auf Interprozesskommunikation zu fokussieren.

Varianten:

- Broker-System mit direkter Kommunikation
- Broker-System mit Nachrichtenvermittlung
- Trader-System
- Adapter-Broker-System
- Callback-Broker-System

Vorteile: Die Broker-Architektur bringt einige wesentliche Vorteile mit sich:

[20] Für eine detaillierte Beschreibung dieses und anderen Muster sei auf [Gall at al. 2003] und [Buschmann *at al*. 1998] verwiesen.

- Ortstransparenz – Die Clients müssen nicht wissen, wo sich die jeweiligen Server im Netzwerk befinden
- Änderbarkeit und Erweiterbarkeit von Komponenten
- Portabilität eines Broker-Systems
- Interoperabilität zwischen Broker-Systemen
- Wiederverwendbarkeit

4.3.3 Die Schnittstelle zu Data Warehousing

Ein Data Warehouse ist ein wichtiges System zur Integration von unterschiedlichen Informationsquellen innerhalb einer Unternehmung. Ein solches System beinhaltet typischerweise Daten, welche aus den unterschiedlichsten Quellen stammen. Es kann sich dabei um Daten aus Altsystemen, operativen Systemen, Web-Seiten, Text- oder anderen Dateien sowie auch um externe Daten handeln. Die gewünschten Daten müssen extrahiert, bereinigt, transformiert und in einer standardisierten Form innerhalb des Data Warehouse Systems gespeichert werden. [Scherz 2000]

Die Abbildung 4.9 stellt einen typischen Aufbau eines Data Warehouse Systems dar. Die wichtigsten Komponenten der Abbildung 4.9 werden nachfolgend erläutert. Es wird dabei den Definitionen aus [Rahm 2004], [Geppert 2002] und [Scherz 2000] gefolgt.

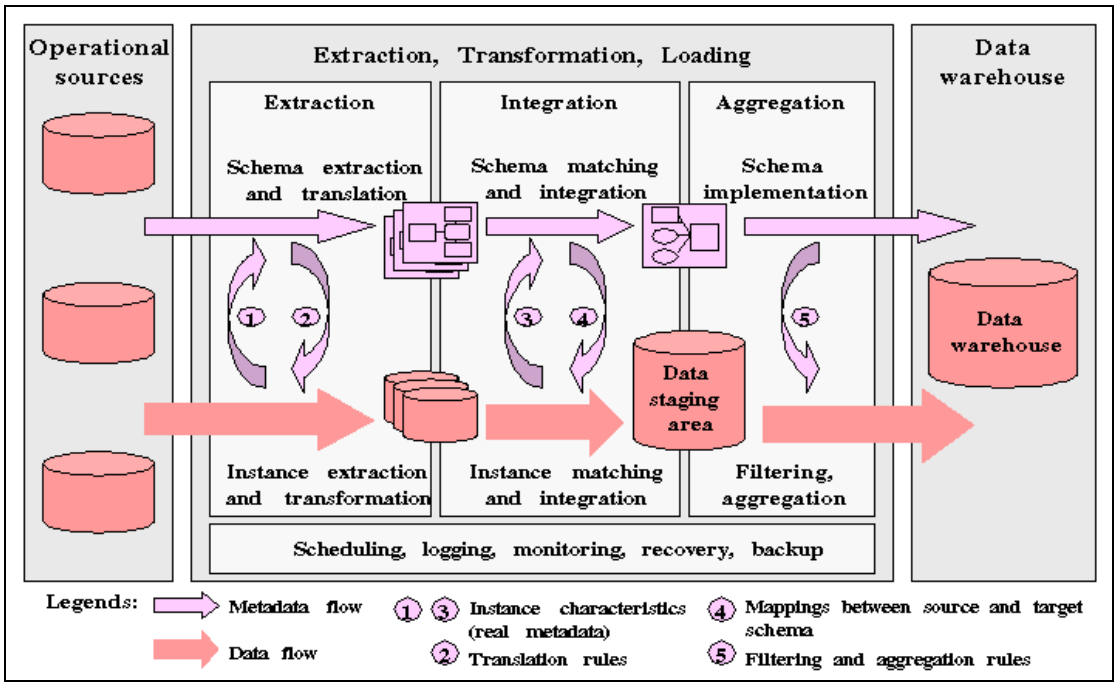


Abbildung 4.9: Aufbau eines Data Warehouse Systems (aus [Rahm 2004])

Datenquellen

Die Datenquellen beinhalten strukturierte, unstrukturierte oder halbstrukturierte Daten. Die Informationssysteme, welche die Daten speichern, sind nach unterschiedlichen Datenmodellen aufgebaut. Die Datenquellen werden oft nicht als Teil des Data Warehouse Systems betrachtet. Um den ETL-Prozess (vgl. nächster Abschnitt) spezifizieren zu können, ist es allerdings notwendig ebenfalls die Struktur und den Aufbau der Quelldateien mit einzubeziehen.

Extraktions-, Transformations-, und Ladekomponente (ETL)

Die ETL-Komponente bezeichnet einen wesentlichen Prozess beim Betrieb eines Data Warehouses, der sich aus drei Phasen zusammensetzt:

- **Extraktion:** Dabei werden die relevanten Daten aus verschiedenen Quellen extrahiert. Die Extraktion wird an den entsprechenden Quellen ausgeführt
- **Transformation**: Aufbereitung und Anpassung der Daten an vorgegebene Schema- und Qualitätsanforderungen. Bei diesem Vorgang können beispielsweise Duplikate eliminiert, Werte aus den Daten hergeleitet oder fehlende Werte mit Hilfe spezieller Regeln ergänzt werden. Dieser Prozess wird im temporären Arbeitsbereich (Data Staging Area) ausgeführt
- **Laden**: Physisches Einbringen der Daten aus dem Arbeitsbereich in das Data Warehouse einschliesslich eventueller Aggregationen. Diese Phase wird in der Data Warehouse-Datenbank ausgeführt.

Datenbanksystem

Das Datenbanksystem ist der zentrale Teil des Data Warehouse Systems. Es beinhaltet eine unternehmensweite Sammlung von integrierten Datenwerten. In der Regel erfolgt die Speicherung in einem relationalen Datenbanksystem.

Data Marts

Data Marts werden benötigt, um den speziellen Anforderungen bestimmter Anwendungen gerecht zu werden. Beispielsweise könnte ein Data Mart nur Daten über geographische Begebenheiten enthalten.

4.4 GFS und Swiss Re IT-Landschaft

Das Ziel des GFS (Global Facultative System) – Projekts ist die Entwicklung einer weltweiten Webapplikation für die Abwicklung und Verwaltung vom fakultativen Book of Business (einzelne Risiken) innerhalb des Geschäftsbereichs der Business Group ‚Property & Casualty' der Swiss Re Group. GFS unterstützt eine prozessorientierte Erfassung von Risiken und Prämien. Diese Informationen können dann global zwischen sämtlichen Swiss Re – Divisionen ausgetauscht werden.

Die GFS-Applikation ist ein Informationssystem, die wie viele andere Systeme auch, nicht allein existiert, sonder an eine IT-Landschaft integriert ist. Die Architektur der GFS ist grundsätzlich vergleichbar mit der aus der Abbildung 4.4. Die besteht auch aus dem Anwendungskern und einigen Subsystemen, die mittels Schnittstellen mit vielen anderen Nachbarsystemen verbunden sind. Dabei werden drei Gruppen von Schnittstellen unterschieden (vgl. auch Abschnitt 4.3): Punkt-zu-Punkt Schnittstellen, die Schnittstelle zu Data Warehouse und die Schnittstelle zu EAI, die die GFS-Applikation mit den Back-Office-Applikationen verbindet. Diese drei Gruppen werden auch als Muster bezeichnet.

Punkt-zu-Punkt Schnittstellen (Data Interaction Pattern)

Das „Data Interaction" Muster wird benutzt, wenn die Partner-Applikation eine Programmierschnittstelle (API)[21] für den Zugriff auf seine Geschäftslogik bietet. Der Datenfluss geht von der Partner-Applikation zu GFS, aber die Daten können auch in die andere Richtung fliessen (vgl. Abbildung 4.10)

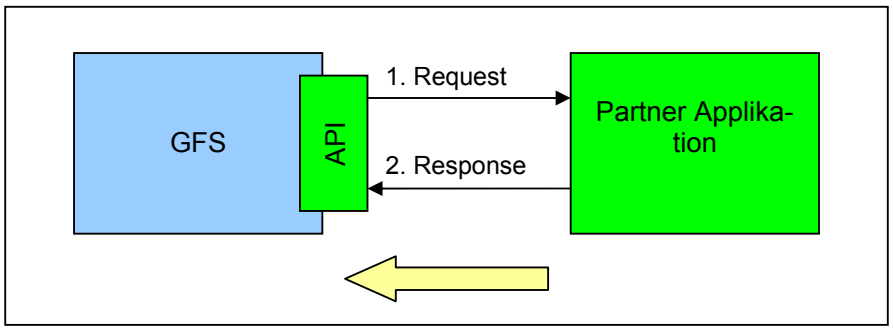


Abbildung 4.10: Data Interaction Pattern

[21] **A**pplication **P**rogramming **I**nterface ist die Schnittstelle, die von einem Betriebssystem oder von einem anderen Softwaresystem weiteren Programmen zur Verfügung gestellt wird

Die Kommunikation wird dabei wir folgt ausgeführt:

1. Baue die Verbindung zu der Partner-Applikation auf und schicke eine Anfrage
2. Empfange die Antwort, die einen Fehlercode enthält

Für die Implementierung von diesem Schnittstellentyp wird Java (RMI) verwendet. Dieses Muster wird beispielsweise für die Kommunikation zu der Partner-Applikation Währungsumrechnung angewendet.

Die Schnittstelle zu EAI (Inter Application Clearing Pattern)

Dieses Muster wird vor allem benutzt, um mit der Back-Ofiice-Applikationen Daten auszutauschen. Der Kommunikationsprozess besteht aus vier Komponenten (vgl. Abbildung 4.11):

- Die GFS-Applikation: Diese Front-Office-Applikation ist erweitert mit einem EAI - Adapter, der für die Interaktion mit dem EAI-Applikation zuständig ist. Dieser Teil hauptsächlich wandelt die Geschäftsobjekte in Nachrichten und schickt sie weiter an die EAI-Komponente und am Schluss nimmt sie die EAI-Antworten entgegen und übergibt sie weiter an die GFS-Applikation.
- Die EAI-Applikation: Dieser Teil ist zuständig für die Verwaltung des Informationsflusses, für die Umwandlung der Nachrichten und für die Interaktion mit verschieden Komponenten.
- Die Back-Office-Applikation: Diese Applikation ist in der Regel mit einer API Web Service erweitert, die als eine Eingangstür benutzt werden kann. Nachdem eine Transaktion ausgeführt ist, wird die Antwort (Nachricht) zurück an die EAI-Applikation gesendet, von wo dann die Antwort zu dem Aufrufer weitergeleitet wird.
- EAI-Monitor: Diese Applikation ist zuständig für die manuelle Bearbeitung der Nachrichten, die nicht automatisch weitergeleitet werden.

Für diesen Schnittstellentyp wird XML-Technologie verwendet. Eine typische Back-Office-Applikation ist Accounting.

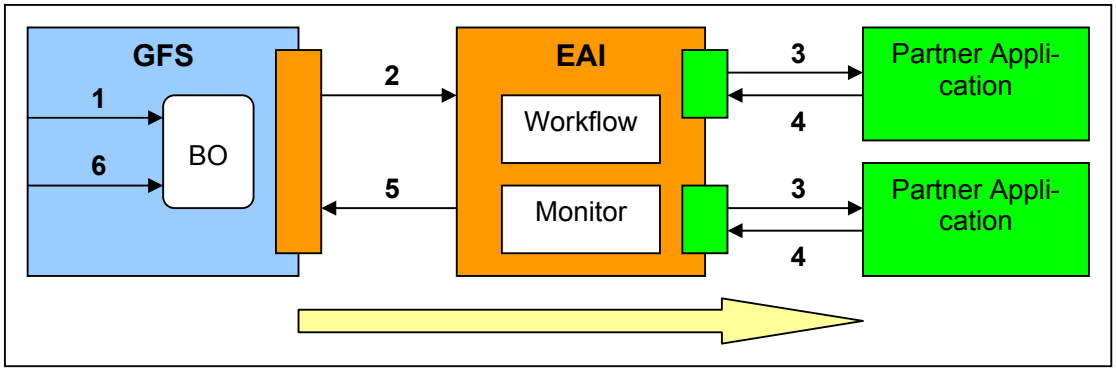


Abbildung 4.11: Inter Application Clearing Pattern

Die Schnittstelle zu Data Warehouse

Die Kommunikation zwischen GFS und Data Warehouse (DWH) ist bidirektional:

- Die Daten werden an die DWH in Form von Views exportiert. Dabei entsprechen die Views den Tabellen von Data Staging Area der DWH, von wo dann sie weiter an DWH geleitet werden.
- DWH bietet die Daten der GFS-Applikation mit Hilfe von DWH Services. Daher die Spezifikation der Schnittstellen zu DWH bedeutet die Spezifikation von DWH Services.

5 Das Qualitätsmodell

Qualitätsanforderungen sind oft schwierig zu bestimmen. Einerseits ihre subjektive Natur (Qualität wird subjektiv wahrgenommen) und die unterschiedlichen Bedürfnisse der Interessengruppen[22] machen ihre Bestimmung schwierig. Andererseits die Komplexität der IT-Landschaft, in der die Applikation integriert wird, schwächt deren Formulierung auch ab. Diese Schwierigkeiten erschweren die genaue Beschreibung von Schnittstellenspezifikationen und die präzise Darstellung von Qualitätsanforderungen.

Das Qualitätsmodell bietet die nötige Hilfe, um die oben genannten Probleme zu lösen. Das Modell für die Domäne der vorliegenden Arbeit liefert eine Taxonomie von Qualitätseigenschaften und Metriken für deren Wertberechnung; denn nur durch quantitative Kenngrössen können die zentralen Anliegen des Qualitätsmanagements, wie Qualitätsplanung, -lenkung, -prüfung und -verbesserung ernsthaft gelöst werden [Wallmüller 2001]. Nachdem das Modell eingebaut ist, können die Domäneanforderungen und die Schnittstelleneigenschaften sowie die Verhandlungen zwischen ihnen festgelegt werden. In der Abbildung 5.1 sind die Ziele des Qualitätsmodells graphisch dargestellt.

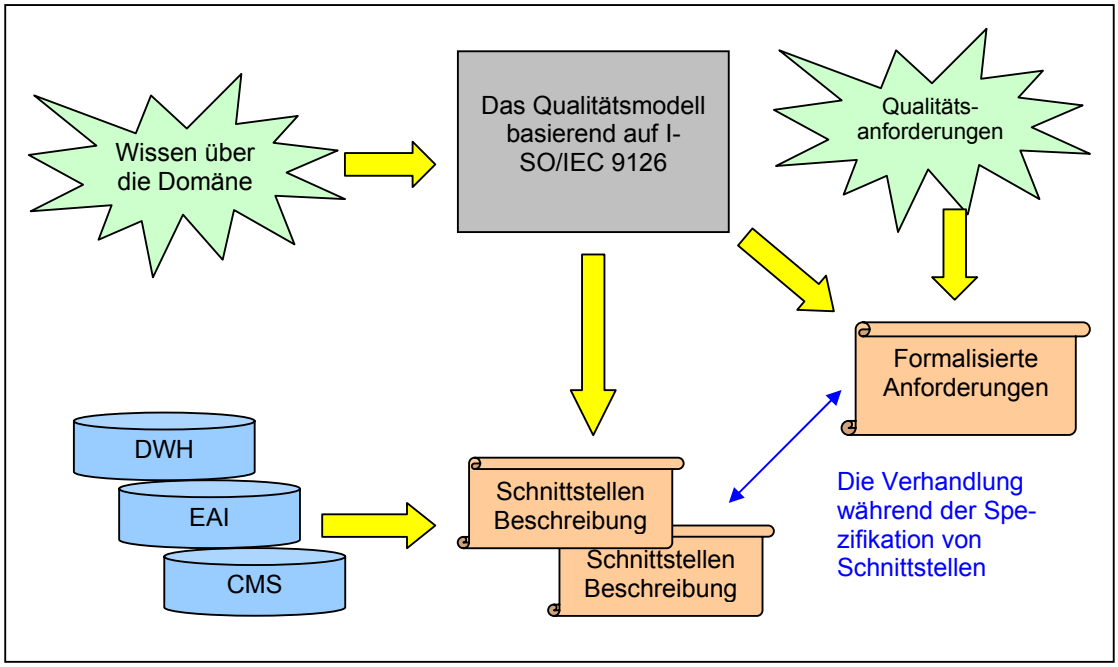


Abbildung 5.1: Verwendung des Qualitätsmodells bei der Spezifikation von Schnittstellen (in Anlehnung an [Franch und Carvallo 2003])

[22] Interessengruppen sind Benutzer, Betreiber, Designer und Programmierer, die unterschiedliche Qualitätsansprüche haben.

Für die Lösung der obengenanten Probleme ist das Qualitätsmodell von Xavier Franch [Franch und Carvallo 2002] (vgl. Abschnitt 3.2.1) am besten geeignet und zwar aus folgenden Gründen: Wie in der Einleitung und im Kapitel 4 bereits erläutert wurde, handelt es sich in der vorliegenden Arbeit um die Qualität des Software-Produktes *Spezifikation der Applikationsschnittstellen,* d.h. einer bestimmten Domäne. Deswegen kommen nur die domänespezifischen Produkt-Qualitätsmodelle in Betracht. Das Modell von Geoff Dromey (vgl. Abschnitt 3.2.3) ist im Gegensatz zu Franch's und Squid (vgl. Abschnitt 3.2.2) kontextfrei. Er bindet die Metriken nicht in das Modell ein – deshalb ist es für diesen Problembereich nicht gut geeignet. Das Squid-Modell ist breiter als das von Franch. Es deckt die Qualität bei der Planung, Lenkung und Auswertung sowie beim Modellieren. Dieses Modell übersteigt somit das Thema der vorliegenden Arbeit. Ein weiterer Grund für die Nicht-Auswahl von Squid ist, dass Squid das ISO/IEC Standard nicht erfordert.

Das Qualitätsmodell von Franch ist domänespezifisch und bindet die Metriken in das Modell ein. Seine Methodik basiert auf den ISO/IEC[23] 9126-1 Qualitätsstandard [ISO/IEC 9126-1 2001]. Der ISO/IEC Standard bittet folgende Vorteile:

- Wegen seinem generischen Charakter legt dieser Standard einige Qualitätsbegriffe auf höchster Ebene fest, anstatt konkreter Qualitätsmerkmale. Somit können die Qualitätsmodelle an die jeweiligen spezifischen Domänen angepasst werden. Das ist ein ganz wichtiger Punkt, weil die Qualitätsmodelle von einer Domäne zu einer anderen grundlegend unterscheiden können. Z.B. das Qualitätsmodell für die Spezifikationsanforderungen besitzt ganz andere Attribute als das für die Implementierung von Software.
- Der Standard ermöglicht die Hierarchiegenerierung von Qualitätseigenschaften, die unerlässlich für die Bildung von strukturierten Modellen sind
- Der Standard ist weit verbreitet.

Der ISO/IEC Standard – der als Basis für das vorliegende Qualitätsmodell dient - wird im nächsten Abschnitt ausführlicher behandelt. Dabei werden die vom Standard definierten sechs Qualitätseigenschaften und deren Untereigenschaften beschrieben. Das eigentliche Thema der vorliegenden Arbeit bzw. die Bildung des Qualitätsmodells für die Spezifikation der Schnittstellen wird im Unterkapitel 5.2 behandelt. Für dessen Bildung wird die Methodologie von [Franch und Carvallo 2003] verwendet.

[23] (ISO/IEC) International Organization for Standardization and International Electrotechnical Commission

5.1 Das ISO/IEC 9126 Standardqualitätsmodell

Die Internationale Organisation für Normung (kurz ISO) ist die internationale Vereinigung von Normungsorganisationen aus über 150 Ländern. Die Organisation wurde am 23. Februar 1947 in Genf gegründet. Die ISO erarbeitet internationale Normen (engl. standards) in allen Bereichen mit Ausnahme der Elektrik und der Elektronik, für die die IEC zuständig ist.

ISO/ IEC 9126 [ISO/IEC 9126-1 2001] stellt eins von vielen Modellen dar, um Softwarequalität sicherzustellen. Es bezieht sich ausschliesslich auf die Produktqualität und nicht die Prozessqualität. Das Modell ist auf jede Art von Software anwendbar. Der Standard definiert nicht die Attribute (weil sie zu jedem Kontext spezifisch sind), sondern einen Satz von sechs Software-Qualitätseigenschaften, denen die Attribute zugeordnet sind. Die Eigenschaften sind wie folgt definiert:

- *Funktionalität (Functionality)*: Die Fähigkeit eines Systems, die verlangte Funktionalität unter gegebenen Randbedingungen für eine gegebene Zeit zu erfüllen.[24] Die Funktion wird definiert als ein angegebenes oder implizites Bedürfnis, das erfüllt werden muss.
- *Zuverlässigkeit (Reliability)* ist der Satz der Attribute, die auf die Fähigkeit der Software auswirken, um die verlangte Funktionalität unter gegebenen Randbedingungen für eine gegebene Zeit zu erfüllen.
- *Verwendbarkeit (Usability)*: Die Leichtigkeit, mit der ein Benutzer die Bedienung eines Systems, die Vorbereitung von Daten dafür und die Interpretation seiner Ergebnisse lernen kann.
- *Effizienz (Efficiency):* Das Ausmass, in dem ein System seine Leistungen unter gegebenen Randbedingungen mit einem Minimum an Ressourcenverbrauch erbringt.
- *Wartbarkeit (Maintainability):* Die Leichtigkeit, mit der ein System geändert werden kann, um Fehler zu beheben, seine Fähigkeiten zu erhöhen oder es an eine veränderte Umgebung anzupassen.
- *Portabilität (Portability):* Die Leichtigkeit, mit der ein System von einer Hard- bzw. Software-Umgebung in eine andere transferiert werden kann.

[24] Diese Eigenschaft befasst sich mit dem ‚was' (was macht die Software, um Bedürfnisse zu erfüllen), wohingegen die anderen Merkmale befassen sich hauptsächlich mit ‚wann' und ‚wie' die Software die Bedürfnisse erfüllt.

Der Standard schlägt auch eine Gruppe nicht-zwingender Basiseigenschaften[25] vor. Der Benutzer kann in Abhängigkeit von den Bedürfnissen des spezifizierten Kontexts einige von ihnen beseitigen oder neue hinzufügen.

Das ISO/IEC Qualitätsmodell wird als Framework für diese Arbeit verwendet, weil es ein öffentlich-zugänglicher, offener und sehr weit verbreiteter internationaler Standard ist, der genug Flexibilität für die Auswahl von Basiseigenschaften und Attributen bietet. Somit können verschiedene Bedürfnisse erfüllt werden [Franch und Carvallo 2003].

Alle ISO/IEC Basiseigenschaften werden in der folgenden Tabelle gelistet. Deren Definitionen folgen den Ausführungen von [ISO/IEC 9126-1 2001]:

Eigenschaft	**Basiseigenschaft**	**Definition**
Funktionalität	Angemessenheit (suitability)	Die Fähigkeit des Software-Produktes, festgelegte Aufgaben und Benutzer-Zielsetzungen mit einem geeigneten Satz von Funktionen zu versorgen
	Richtigkeit (accuracy)	Die Fähigkeit des Software Produktes, die richtigen oder festgesetzten Ergebnisse oder Wirkungen mit dem nötigen Grad an Genauigkeit zu versorgen
	Verknüpfbarkeit (interoperability)	Die Leichtigkeit, mit der zwei oder mehrere Systeme Informationen austauschen und die ausgetauschten Informationen benutzen können
	Sicherheit (security)	Die Fähigkeit des Software Produktes, Informationen und Daten zu schützen, damit sie von unautorisierten Personen oder Systemen nicht gelesen oder modifiziert werden können. Hingegen sollte den ermächtigten Personen oder Systemen deren Zugang nicht verweigert werden
	Funktionalitätskonformität (functionality compliance)	Die Fähigkeit des Software-Produktes, zu den Standards, Richtlinien oder Gesetz-Regulierungen und ähnlichen Vorschriften, die in Bezug auf die Funktionalität stehen, festzuhalten.
Zuverlässigkeit	Reife (maturity)	Die Fähigkeit des Software-Produktes, Ausfall (als Folge von Fehlern in der Software) zu vermeiden.
	Fehlertoleranz (fault tolerance)	Die Fähigkeit des Software-Produktes, ein vorgegebenes Performance-Niveau im Falle von Software-Mangeln oder bei Verletzung seiner festgelegten Schnittstellen beizubehalten.

[25] Für die Unter-Eigenschaften (eng. subcharacteristic) wird hier der Begriff ‚Basiseigenschaft' verwendet. Z.B. die Qualitätseigenschaft *Effizienz* ist in zwei Basiseigenschaften unterteilt, nämlich in *Zeitverhalten* und *Verbrauchsverhalten*.

	Wiederherstellbarkeit (recoverability)	Die Fähigkeit des Software-Produktes, ein festgelegtes Performance-Niveau wiederherzustellen und im Falle eines Ausfalls die Daten zurückgewinnen zu können.
	Zuverlässigkeitskonformität (reliability compliance)	Die Fähigkeit des Software-Produktes, zu den Standards oder Richtlinien, die in Bezug auf die Zuverlässigkeit stehen, festzuhalten.
Verwendbarkeit	Verständlichkeit (understandability)	Die Fähigkeit des Software-Produktes, dem Benutzer zu verstehen ermöglichen, ob die Software geeignet ist und wie es für bestimmte Aufgaben und Einsatzbedingungen benutzt werden kann.
	Erlernbarkeit (learnability)	Die Fähigkeit des Software-Produktes, dem Benutzer zu ermöglichen, seine Anwendung zu lernen.
	Bedienbarkeit (operability)	Die Fähigkeit des Software-Produktes, dem Benutzer zu ermöglichen, sie zu bedienen und kontrollieren.
	Anziehungskraft (attractiveness)	Die Fähigkeit des Software-Produktes, attraktiv für den Benutzer zu sein
	Verwendbarkeitskonformität (usability compliance)	Die Fähigkeit des Software-Produktes, zu den Standards, Konventionen, Gestaltungsrichtlinien oder Regulierungen, die in Bezug auf die Verwendbarkeit stehen, festzuhalten.
Effizienz	Zeitverhalten (time behaviour)	Die Fähigkeit des Software-Produktes, passende Antwort sowie Abarbeitungszeiten und Durchsatzraten bereitzustellen, wenn die Software seine Funktionen unter festgelegten Zuständen durchführt.
	Verbrauchsverhalten (resource behaviour)	Die Fähigkeit des Software-Produktes, geeignete Mengen und Ressourcentypen zu benutzen, wenn die Software seine Funktion unter angegeben Zuständen ausführt.
	Effizienzkonformität (efficiency compliance)	Die Fähigkeit des Software-Produktes, zu den Standards oder Konventionen, die in Bezug auf die Effizienz stehen, festzuhalten.
Wartbarkeit	Analysierbarkeit (analyzability)	Die Fähigkeit des Software-Produktes, die Mängel oder Ursachen von Ausfällen in der Software zu diagnostizieren.
	Modifizierbarkeit (changeability)	Die Fähigkeit des Software Produktes, einer spezifizierten Modifikation die Umsetzbarkeit zu ermöglichen.
	Stabilität (stability)	Die Fähigkeit des Software Produktes, unerwartete Wirkungen von Modifikationen der Software zu vermeiden.
	Prüfbarkeit (testability)	Das Ausmass, in dem ein System das Erstellen von Testbedingungen sowie die Durchführung von Tests zur Feststellung, ob die Bedingungen erfüllt sind, erleichtert.

	Wartbarkeitskonformität (maintainability compliance)	Die Fähigkeit des Software-Produktes, zu den Standards, Geschäftsrichtlinien oder Konventionen, die in Bezug auf die Wartbarkeit stehen, festzuhalten.
Portabilität	Anpassbarkeit (adaptability)	Die Fähigkeit des Software-Produktes, bei anderen festgelegten Umgebungen unter Verwendung von gleichen Handlungen oder Mitteln angepasst zu werden.
	Installierbarkeit (installability)	Die Fähigkeit des Software-Produktes, in einer festgelegten Umgebung installiert zu werden.
	Koexistenz (co-existence)	Die Fähigkeit des Software-Produktes, mit anderen unabhängigen Softwaren in einer gemeinsamen Umgebung unter Nutzung von gemeinsamen Ressourcen zu koexistieren.
	Austauschbarkeit (replaceability)	Möglichkeit, diese Software anstelle einer spezifizierten anderen in der Umgebung jener Software zu verwenden, sowie der dafür notwendige Aufwand.
	Portabilitätskonformität (portability compliance)	Die Fähigkeit des Software-Produktes, zu den Standards oder Konventionen, die in Bezug auf die Portabilität stehen, festzuhalten.

Tabelle 5.1: Die ISO/IEC 9126-1 Basiseigenschaften und deren kurze Beschreibung

5.2 Erstellung eines ISO/IEC 9126 Qualitätsmodells für die Spezifikation von Applikationsschnittstellen

Ein wesentlicher Aspekt der Qualitätsmodelle ist die Zerlegungssystematik von Qualitätseigenschaften, die sich auf der untersten Ebene auf Qualitätskenngrössen abstützen. Basierend auf diesem Grundsatz ist auch das Qualitätsmodell von Xavier Franch aufgebaut. Seine Methodologie besteht aus sechs Schritten inklusive einer Vorstufe, nämlich die *Domänebeschreibung* (vgl. Abschnitt 3.2.1). In der vorliegenden Arbeit ist das Modell leicht modifiziert.

Der zweite Schritt ‚Aufteilung der Basiseigenschaften' wurde weggelassen, weil er bei der Domäne *Spezifikation der Schnittstellen* nicht zutrifft. Der dritte ‚Festlegung der Qualitätsmerkmale' und der vierte Schritt ‚Zerlegung der Merkmale in Basismerkmale' wurden zu einem neuen gruppiert, da für die vorliegende Arbeit diese Aufteilung überflüssig ist. Für beide Schritte werden schlussendlich die gleichen Überlegungen verwendet, nämlich wie die abstrakten Basiseigenschaften in solche Merkmale abgebildet werden können, um diese dann mittels Metriken messen zu können.

Das Qualitätsmodell besteht nun neu aus vier Schritten (vgl. Abbildung 5.2). Die Vorstufe für die Analyse des Domänebereichs wurde in Kapitel 4 bereits behandelt. Diese vier Schritte werden sequentiell erläutert, aber sie können selbstverständlich auch mehrmals wiederholt werden oder sich verflechten. In Abschnitt 5.3.1 wird überprüft, welche Eigenschaften bei der Spezifikation von Schnittstellen nicht passen, modifiziert oder weggelassen werden können. Abschnitt 5.3.2 behandelt die Abbildung der Basiseigenschaften in Qualitätsmerkmale sowie deren Zerlegung in Basismerkmale. Die Beziehungen zwischen Qualitätsmerkmalen sind Thema des Abschnitts 5.3.3. Schlussendlich werden im Abschnitt 5.3.4 die Metriken behandelt.

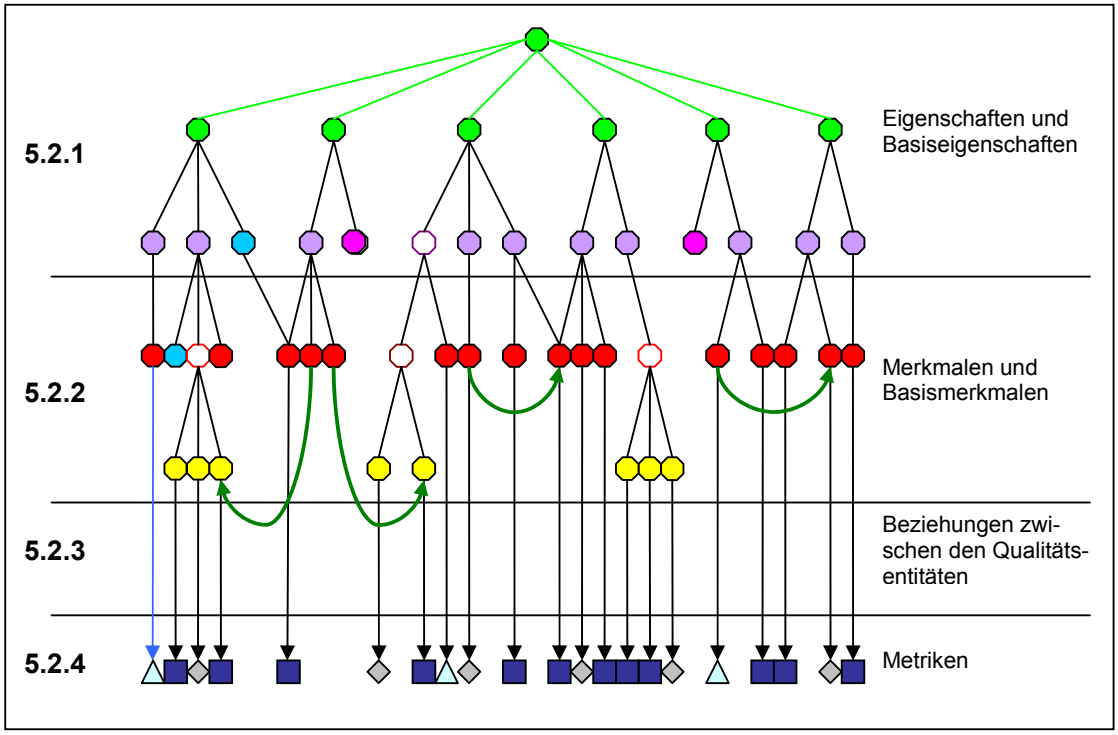


Abbildung 5.2: Aufbau des Qualitätsmodells (in Anlehnung an [Franch und Carvallo 2003])

5.2.1 Bestimmung der Qualitätsbasiseigenschaften

Im Unterkapitel 5.1 wurde erwähnt, dass ISO/IEC Standard einen Satz von (nicht obligatorischen) Basiseigenschaften für jede der sechs Eigenschaften des Modells vorschlägt. Somit könnten einige Basiseigenschaften, welche auf die festgelegte Domäne nicht zutreffen, weggelassen werden. Der Standard verbietet auch nicht das Hinzufügen neuer Eigenschaften.

Um herauszufinden welche ISO/IEC Eigenschaften für die *Spezifikation der Schnittstellen* verwendet werden, bzw. um die Qualitätseigenschaften zu identifizieren, wurde folgende Frage gestellt: Was will ich mit dieser Schnittstellenspezifikation machen?

Gemäss Geoff Dromey [Dromey 1996] will man die Spezifikationen hauptsächlich:

- für die Beschreibung der Anforderungsprobleme verwenden
- als Basis für das Design verwenden
- als Vertragsinstrument für ein gemeinsames Verständnis zwischen den Kunden, Benutzer und Entwickler verwenden
- so ändern, um den neuen oder modifizierten Anforderungen nachzukommen
- wieder verwenden oder so anpassen, um andere Probleme lösen zu können

5.2.2 Zuordnung der Qualitätsmerkmale zu den Basiseigenschaften und deren Zerlegung in Basismerkmale[26]

Die abstrakten Basiseigenschaften sollten weiter in konkretere Qualitätsmerkmale zerlegt werden, die zu einer besseren Auswertung der wahrnehmbaren Eigenschaften der Schnittstellenspezifikationen in der Domäne führen können.

Einige Qualitätsmerkmale können möglicherweise nicht direkt messbar sein und deswegen ist es erforderlich sie weiter in Basismerkmale zu unterteilen. Somit wird auch eine bessere Strukturiertheit des Modells erreicht.

5.2.3 Festlegung der Beziehungen zwischen den Qualitätseigenschaften

Das Qualitätsmodell sollte auch helfen, die Zusammenhänge zwischen den einzelnen Merkmalen[27] bzw. Eigenschaften besser zu verstehen und aufzuzeigen. Wallmüller [Wallmüller 2001] nennt folgende drei Klassen von Beziehungen zwischen Merkmalen:

- indifferent
- konkurrierend

[26] Basismerkmal bedeutet Submerkmal (eng. Subattribute). Es wurde dieser Begriff ausgewählt, da dieses Merkmal nicht mehr aufteilbar ist.
[27] Der Begriff Merkmal deckt in diesem Zusammenhang auch die Basismerkmale. D.h. es gibt nicht nur Beziehungen zwischen zwei Merkmalen, sondern auch zwischen einem Merkmal und einem Basismerkmal sowie zwischen Basismerkmalen – siehe die Abbildung

- verstärkend

Zwei Qualitätseigenschaften sind indifferent, wenn es keine sichtbare Wechselwirkung zwischen diesen gibt. Beispielsweise sind die Portabilität und die Korrektheit indifferent.

Zwei Qualitätseigenschaften sind konkurrierend, wenn eine Verbesserung der einen die Verschlechterung der anderen nach sich zieht bzw. ein Merkmal beeinflusst ein anderes negativ [Franch und Carvallo 2003]. Eine Erhöhung der Zuverlässigkeit beispielsweise führt in der Regel zu einer Verschlechterung der Performance.

Zwei Eigenschaften verstärken sich, wenn die Erhöhung der Qualität einer Eigenschaft die Erhöhung der Qualität der anderen nach sich zieht. Dies bedeutet, dass eine Eigenschaft eine andere positiv beeinflusst. Korrektheit und Zuverlässigkeit beispielsweise sind zwei sich verstärkende Eigenschaften.

5.2.4 Kenngrössen-Bestimmung der Merkmalen

Ein weiteres Problem bei der Definition des Qualitätsmodells ist die Auswahl der Metriken für jedes Merkmal. Die Problematik des Messens bringen folgende Zitate aus [Wallmüller 2001] S. 33 zum Ausdruck:

„When you can measure what you are speaking about and express it in numbers you know something about it, but when you cannot measure it, when you cannot express it in numbers, your knowledge is of a meager and unsatisfactory kind.“

Lord Kelvin

“To measure is to know.”

J. C. Maxwell

Es gibt verschiedene Verfahren, die für die Ableitung von Softwaremassen zur Verfügung stehen. Für die vorliegende Arbeit ist unter anderem der Goal-Question-Matric-Ansatz (GQM) von Basili [Basili 1992] verwendet. Für andere Ansätze wie GOALS[28] sei auf [Bächle 1996] verwiesen. Als Hilfsmittel für die Definition von Metriken wurde dabei [IEEE 1998] verwendet.

[28] Das von Boehm entwickelte Verfahren „Goal-Oriented Approach to Lifecycle Software“.

GQM ist ein zielorientierter Ansatz, der für den Ermittlungsprozess von Massen drei Stufen bzw. Ebenen vorsieht:

- *Goal*: Auf dieser (konzeptionellen) Ebene werden die Qualitätsziele festgelegt. Die Ziele werden für ein spezifiziertes Objekt definiert und hängen jeweils von der jeweiligen Interessengruppe ab. Ausserdem werden drei verschiedene Arten von Messobjekten unterschieden: Produkte, Prozess und Ressourcen. Die Spezifikation der Schnittstellen ist ein Produkt bzw. Zwischenprodukt der Softwareentwicklung.
- *Question*: In diesem Schritt (in [Basili 1992] als operationelle Ebene definiert) wird jedem Ziel ein Satz von Fragen zugeordnet. Somit können die abstrakten Ziele anhand konkreter Fragen operationalisiert werden.
- *Metric* (quantitative Ebene): Das Ziel dieser Ebene ist, die vorherigen Fragen quantitativ zu beantworten.

Mit Hilfe dieses Verfahrens wir das gemessen, was dazu beiträgt, die festgelegten Ziele zu erreichen.

6 Anwendung des Qualitätsmodells

Die Rationalisierung von Geschäftsprozessen, die Reduzierung von Prozesskomplexität, die Freisetzung von Management-Kapazitäten, die Flexibilisierung des Unternehmens und die Fokussierung auf das Kerngeschäft haben viele Unternehmungen veranlasst Teile der Aktivitäten auszulagern (Outsourcing). Aufgaben, die nicht zum Kerngeschäft der Firma gehören und deren Selbstausführung nicht effizient ist, werden an spezialisierte Dienstleister abgegeben. Beispielsweise wird die Programmierung einer Software an kompetente IT-Dienstleister übertragen. Hingegen werden die Anforderungsspezifikationen und die Integration der Applikation in die IT-Umgebung des Unternehmens selbst gemacht. Dieser Fall trifft auch bei der GFS-Applikation zu.

Die Anforderungsspezifikation und die Spezifikation der Schnittstellen insbesondere zu der Nachbarapplikationen ist eine der zentralen Aufgaben in dem GFS-Projekt. Sie sind einerseits als Basis für den Entwurf und Implementierung der Applikation wichtig (vgl. Abschnitt 4.1), andererseits dienen sie als eine Art Vertrag zu dem Outsorcing-Partner. Dies bedeutet, je qualitativ hochwertiger die Spezifikationen sind, welche an die Partner ausgeliefert werden, desto weniger Kommunikationsaufwand (weniger Rückfragen) ist nötig und somit werden auch weniger Kosten verursacht.

Die Spezifikationen bzw. die Schnittstellenspezifikationen stellen ein Zwischen-Produkt dar, das auch ein Qualitätsniveau anbieten sollte. Deswegen wurde für die vorliegende Arbeit ein Produktqualitätsmodell angewendet und zwar das Qualitätsmodell von Franch (vgl. Kapitel 5 für weitere Gründe).

Dieses Kapitel besteht aus zwei Teilen. Im ersten Teil wird das Qualitätsmodell mit Hilfe der Methodologie von Franch (vgl. Kapitel 5) erstellt. Die GFS-Applikation wird dabei als Beispiel verwendet. Das zweite Teil behandelt die Bewertung des Modells. Als Basis für dessen Bewertung dienen die ausgeführten Interviews.

6.1 Anwendung des Qualitätsmodells in GFS

Ausgehend von den genannten Punkten in Abschnitt 5.2.1 und mit Hilfe von den Interviews[29] und anderen Qualitätsmodellen, vor allem aber dem von Franch[30] und dem GQM-

[29] Die Interviews wurden in das Projekt GFS durchgeführt

Ansatz[31] wurden die Qualitätseigenschaften, deren Merkmale (vgl. Abschnitt 6.1.2) und Metriken für die quantitative Bestimmung der Qualität (vgl. Abschnitt 6.1.4) definiert.

Im Abschnitt 6.1.1 wird die Anwendbarkeit der ISO/IEC Eigenschaften und Basiseigenschaften bei der GFS-Schnittstellenspezifikationen überprüft. Diejenigen Eigenschaften, die bei der Spezifikation von Schnittstellen nicht passen, werden entweder modifiziert oder weggelassen. Abschnitt 6.1.2 behandelt die Abbildung der Basiseigenschaften in Qualitätsmerkmale sowie deren Zerlegung – wenn nötig – in Basismerkmale. Die Beziehungen zwischen Qualitätseigenschaften sind Thema des Abschnitts 6.1.3. Letztlich werden im Abschnitt 6.1.4 Metriken für die Qualitätsmerkmale festgelegt, um die Qualität quantitativ darzustellen.

6.1.1 Bestimmung der Qualitätsbasiseigenschaften

Das Ziel dieses Abschnitts ist die Festlegung welche vom ISO Standard (vgl. Abschnitt 5.1) vorgeschlagenen Eigenschaften und deren Basiseigenschaften für die Spezifikation von GFS-Schnittstellen besonderes relevant sind. Einige unwichtige Basiseigenschaften werden entweder weggelassen oder an diese Domäne angepasst bzw. modifiziert.

Für die Spezifikation der Schnittstellen sind die Basiseigenschaften *Verwendbarkeit* / *Bedienbarkeit* und *Verwendbarkeit* / *Attraktivität* beispielsweise nicht von Bedeutung. Sie sind typische Qualitätseigenschaften für die Benutzeroberfläche. Hier werden sie weggelassen.

Die Eigenschaft *Portabilität* ist auch irrelevant. Da eine Schnittstelle applikationsspezifisch ist, kann deren Übertragung in einer anderen Umgebung schwer realisiert werden. Stattdessen wird die Eigenschaft *Wiederverwendbarkeit* verwendet. Data Warehouse stellt beispielsweise den meisten Applikationen die gleichen Dienste zur Verfügung. Somit kann die Spezifikation der Dienste einer vorherigen Applikation verwendet werden.

[30] Die Methodologie von Franch (vgl. Abschnitt 3.2.1) wird schlussendlich für die vorliegende Arbeit eingesetzt

[31] Der Goal-Question-Metric-Ansatz (GQM) von V. R. Basili ist der bekannteste Vertreter aus der Klasse des zielorientierten Messens.

6.1.2 Zuordnung der Qualitätsmerkmale zu den Basiseigenschaften und deren Zerlegung in Basismerkmale

Die Liste der Qualitätsmerkmale bezogen auf die Schnittstellenspezifikationen der GFS-Applikation wird möglicherweise nicht ganz komplett sein. Es ist aber zweifellos möglich, eine Liste der relevantesten Merkmale zu erstellen.

Einige Qualitätsmerkmale können möglicherweise nicht direkt messbar sein und deswegen ist es erforderlich sie weiter in Basismerkmale zu unterteilen. Auf diese Weise können sie besser ausgewertet werden. In diesem Modell gibt es mehrere solcher Fälle, z.B. das Merkmal *Verwendbarkeit / Verständlichkeit / Übersichtlichkeit* wird zu den Basismerkmalen *Strukturiertheit* und *Knappheit* zerlegt. Es ist selbstverständlich möglich, dass die obengenanten Basismerkmale auch als separate Merkmale aufgelistet werden können. Es wurde aber dieser Ansatz gewählt, weil die Aufteilung der Merkmale in Basismerkmale eine bessere Strukturiertheit des Modells ermöglicht. Das ISO/IEC Modell verbietet deren Gebrauch auch nicht [ISO/IEC 9126-1 2001].

In den folgenden Abschnitten werden alle Merkmale aufgelistet, die während der Erstellung der vorliegenden Arbeit identifiziert wurden. Jedes definierte Merkmal bzw. Basismerkmal bezieht sich auf ein oder mehrere Objekte bzw. Bestandteile der Schnittstellenspezifikationen. Typische Bestandteile einer Spezifikation sind das Kontext-Diagramm der Applikation [K], Schnittstellendefinition [SD], Schnittstellenspezifikation [S] und die Abbildungstabellen [A]. Hinsichtlich der Überlegung vom Abschnitt 4.3 bezüglich der verschiedenen Schnittstellentypen werden drei Schnittstellenspezifikationen unterschieden. Data Warehouse - [SW], EAI – [SE] und andere Schnittstellen [SA]. Diese Aufteilung bzw. diese Bestandteile von Spezifikationen können auch als Bezugsobjekte der Merkmale verstanden werden. Beispielsweise das Bezugsobjekt des Merkmals *Attributformat* ist die Abbildungstabelle. Dieses Merkmal wird dann definiert als *Attributformat [A].*

6.1.2.1 Funktionalität

Die Funktionalität wird definiert als das Vorhandensein von Funktionen mit festgelegten Eigenschaften. Diese Funktionen erfüllen die definierten Anforderungen. Die Funktionalität befasst sich mit dem ‚was' (was macht die Software, um Bedürfnisse zu erfüllen), dahingegen befassen sich die anderen Merkmale hauptsächlich mit dem ‚wann' und ‚wie' die Software die Bedürfnisse erfüllt.

Funktionalität besteht aus folgenden Basiseigenschaften: Angemessenheit, Richtigkeit, Interoperabilität, Sicherheit und Konformität.

Angemessenheit

Angemessenheit ist definiert als die Eignung von Funktionen für spezifizierte Aufgaben. Im Kontext der Schnittstellenspezifikationen bedeutet dies, dass alle Funktionen (Anforderungen), die für die Integration der GFS-Applikation in die Swiss Re IT-Umgebung notwendig sind, spezifiziert wurden. Dazu gehört das Kontext-Diagramm, die Schnittstellen-Definitionen, das Schnittstellen-Spezifikationsdokument (zu Data Warehouse, EAI, und alle anderen Partnersysteme) wie auch die Abbildungstabellen.

- Das *Kontext-Diagramm* stellt die Applikation und seine Umgebung graphisch dar (vgl. Anhang A).
- Das *Schnittstellendefinition*-Dokument beinhaltet die relevantesten Eigenschaften, die für die Spezifikation der Schnittstellen erforderlich sind, z.B. Name der Partnerapplikation und deren kurze Beschreibung, Schnittstellentyp, Technologien, etc.
- Das Schnittstellenkonzept, die Beschreibung der Partnerapplikationen, die Identifizierung der Datenflüsse sowie die Bereitstellung der Dienste und die Erstellung von Datenbank-Views sind in das *Spezifikationsdokument*[32] definiert.
- Die *Mappingtabelle* beinhaltet alle Attribute, die in der Zielapplikation abgebildet sind.

In Tabelle 6.1 sind die relevantesten Merkmale definiert. Sie sind in vier Gruppen (Bezugsobjekten) aufgeteilt; die Gruppen entsprechen der oben genannten Aufteilung.

ANGEMESSENHEIT (SUITABILITY)		
Merkmal	Basismerkmal	Zusätzliche Beschreibung
1 Graphische Darstellung der Applikation [K][33]		Merkmale in Bezug auf die graphische Darstellung der Applikation in der Systemlandschaft
	1 Kontext-Diagramm Attribute (vollständig)	Beschreibung der Attribute des Kontext-Diagramms (automatische vs. manuelle Schnittstelle und Trigger)
	2 Ankommende Schnittstellen	Alle Applikationen, die Daten an die Ursprungsapplikation liefern

[32] Die Schnittstellendefinition ist eine Art Vertrag, die nur eine Aussage macht, was die Schnittstelle alles beinhaltet, hingegen das Spezifikationsdokument sagt wie die Schnittstellen spezifiziert sind
[33] Bezugsobjekt dieses Merkmals ist das Kontext-Diagramm

	3 Abgehende Schnittstellen	Alle Zielapplikationen, die mit Daten von der Ursprungsapplikation beliefert werden
	4 Auslöser (Trigger)	Welche Applikation löst eine Aktion (Datenaustausch) aus?
2 Schnittstellendefinition [SD]		Merkmale in Bezug auf die Definition einer entsprechenden Schnittstelle (z.B. Definition der Schnittstelle zu Data Warehousing – kurze Beschreibung der DWH, Typ der Schnittstelle, Beschreibung der Dienste etc.)
	1 Geschäftsanforderungen	Sind alle Geschäftsanforderungen vorhanden?
	2 Zielapplikation	Beschreibung der Zielapplikation (Applikation für Datenaustausch – EAI, Datenaufbewahrung – Data Warehousing, oder eine andere Applikation für die Datengewinnung – CMS)
	3 Schnittstellenbeschreibung	Kurz beschreiben, was für Geschäftsdaten über die Schnittstelle getauscht werden
	4 Typ der Schnittstelle	Sind folgende Elemente vorhanden: Mechanismus (Batch, Service), Automatisch/Manuell, Häufigkeit (z.B. täglich, oder bei Anfrage), Volumen (z.B. < 1MB pro Aufruf) und Push/Pull[34]
	5 Typ der ausgetauschten Daten	Daten die zwischen den Applikationen ausgetauscht werden, z.B. Schaden- und Prämieninformation bei einer Versicherung
	6 Mögliche Technologien	Die Technologien, die für die Implementierung der Schnittstelle gebraucht werden könnten, z.B. EJB Service[35], Web Service[36], ETL[37], JDBC[38], etc.

[34] **Data Push**: Die Ursprungsapplikation löst den Datentransfer aus; **Data Pull**: Der Datentransfer wird vom Zielsystem ausgelöst oder vom Zielsystem aber durch eine Zwischenapplikation
[35] **Enterprise Java Beans (EJB)** stellen wichtige Konzepte für Unternehmensanwendungen bereit, z.B. Transaktions-, Namens- oder Sicherheitsdienste, die für die „Geschäftslogik" einer Anwendung benötigt werden.

3 Schnittstellenspezifikation [S][39]		Bei diesem Teil geht es um das ‚Wie'. D.h. wie werden die Elemente von der Schnittstellendefinition spezifiziert.
	1 Beschreibung der Zielapplikation	Auf welcher Art und Weise kann auf die Daten der Zielapplikation zugegriffen werden?
	2 Schnittstellenkonzept	Ist der Schnittstellenentwurf vorhanden?
	3 Datenfluss	Sind alle ausgetauschten Daten definiert?
	4 Views (entprechend der DWH Landing Area) [SW]	Sind alle Datenbank-Views gemäss den Anoferderungen von Data Warehouse Landing Area definiert?
	5 Benötigte Dienste [SW]	Welche Dienste (Services) benötigt die Applikation? Z.B. Data Warehouse Services.
	6 Technologien	Mit welchen Technologien werden die Schnittstellenkonzepte, Views oder die Dienste implementiert?
4 Abbildungstabelle (Mapping) [A]		Die Abbildung der Attribute in die Zielapplikation
	1 Vollständigkeit der Attribute	Sind alle Attribute abgebildet?
	2 Eindeutigkeit der Attribute	Sind die Attribute eindeutig definiert?

Tabelle 6.1: Angemessenheit

Richtigkeit

Richtigkeit bedeutet im Kontext der Spezifikationen, dass alle graphische Darstellungen, Definitionen, Datenflüssen, Abbildungsattributen, etc. die korrekten oder vereinbarten Ergebnisse liefern. Bei dem Swiss Re wird zwischen einem *Vertragsabschlussdatum* und einem *Inkraftsetzungsdatum* unterschieden. Würden fälschlicherweise die beiden Daten nicht unterschieden, dann könnte das Attribut *Vertragsabschlussdatum* nicht korrekt in Data Warehouse abgebildet werden. Dieser Umstand könnte beispielsweise gie Auswer-

[36] Ein **Web Service** ist eine Software-Anwendung, die mit einem Uniform Resource Identifier (URI) eindeutig identifizierbar ist und deren Schnittstellen als XML-Artefakte definiert, beschrieben und gefunden werden können.
[37] vgl. Abschnitt 4.3.3
[38] **Java Database Connectivity (JDBC)** ist ein API der Java-Plattform, die eine einheitliche Schnittstelle zu Datenbanken verschiedener Hersteller bietet und speziell auf relationale Datenbanken ausgerichtet ist.
[39] Dieses Merkmal bezieht sich auf alle Schnittstellenspezifikationen (SW, SE und SA)

tung verfälschen oder zu falschen Berechnungen führen. Eine falsche Auswertung oder Berechnung würde auch vorkommen, wenn beispielsweise das Datumsformat nicht korrekt abgebildet wird. Z.B. *Vertragsabschlussdatum* (01.07.2005) wird in Data Warehouse als 07/01/2005 gespeichert.

Ein weiteres Beispiel ist das Kontext-Diagramm. Die Partnerapplikationen müssen in ihrer letzten vorhandenen Version dargestellt werden, d.h. im Kontext-Diagramm könnte für Data Warehouse die falsche Version-Nummer, nämlich 5.5 anstatt 6.0 dargestellt werden. Folglich werden auch die GFS-Attribute zu der falschen Version der Data Warehosue abgebildet.

Die Basiseigenschaften *Angemessenheit* und *Richtigkeit* bestehen aus den gleichen Merkmalen, aber bei der Richtigkeit geht es im Gegensatz zu Angemessenheit nicht um die Vollständigkeit der Merkmale, sondern um deren Korrektheit.

In der Tabelle 6.2 werden alle Qualitätsmerkmale genannt, welche die Richtigkeit und somit auch die Qualität der Schnittstellenspezifikationen beeinflussen.

RICHTIGKEIT (ACCURACY)		
Merkmal	Basismerkmal	Zusätzliche Beschreibung
1 Graphische Darstellung der Applikation [K]		Ist die Graphik korrekt dargestellt?
	1 Typ der Schnittstelle	Ist der Schnittstellentyp richtig dargestellt (eine automatische Schnittstelle in der Graphik wird mit einem vollen Pfeil → die manuelle --> gemalt)
	2 Datenaustausch	Ist jede Verbindung (Pfeil) korrekt mit Daten, die durch diese Schnittstelle ausgetauscht werden, beschriftet. (z.B. zwischen CMS und der entwickelten Applikation wird die Adresse eines Klienten ausgetauscht)
2 Schnittstellendefinition [SD]		Die Beschreibung der Schnittstelle, Schnittstellentyp (Mechanismus, Auto/Manuell, Frequenz, Volumen, Push/Pull), Technologie, Datentypen und geforderte Dienste – vgl. mit dem Merkmal Schnittstellendefinition aus der Eigenschaft Angemessenheit.

3 Schnittstellenspezifikation [S]		Beschreibung der Zielapplikation, Schnittstellenkonzept, Datenflüsse, Views und Dienste – vgl. mit dem Merkmal Schnittstellenspezifikation aus der Eigenschaft Angemessenheit.
4 Abbildungstabelle (Mapping) [A]		Die Korrektheit der Abbildungstabelle wie das Abbildungsattribut, sein Format etc.
	1 Abbildungsattribut	Ist das Attribut korrekt abgebildet bzw. ist die Transformationsregel eines Abbildungsattributs korrekt?
	2 Attribut-Format	Das Attributformat bei der Zielapplikation (z.B. Datumsformat der Applikation ist TT.MM.JJJJ, hingegen Data Warehousing verwendet JJJJ/MM/TT)

Tabelle 6.2: Richtigkeit

Sicherheit

Die Eigenschaft Sicherheit beinhaltet alle Vorgänge zur Gewährleistung, dass nur die autorisierten Mitarbeiter Zugang zu den Daten erhalten. Sicherheitsaspekte werden zu einem grossen Teil von dem Swiss Re abgedeckt.[40] Für die Schnittstellen sind insbesondere folgende Merkmale wichtig: Datensicherheit, Zugriffsberechtigung und Informationsaustausch.

Die Tabelle 6.3 zeigt alle Sicherheitsmerkmale und deren Beschreibung.

SICHERHEIT (SECURITY)		
Merkmal	Basismerkmal	Zusätzliche Beschreibung
1 Datensicherheit		Bieten die Schnittstellen eine Wiederherstellungsfunktion (recovery)?
2 Zugriffberechtigung		Zugriffsrechte bezeichnen die Regeln, nach denen entschieden wird, ob und wie Benutzer, Programme oder Programmteile, Operationen auf Objekten ausführen dürfen.

[40] Die meisten Sicherheitseigenschaften sind von den Unternehmungsrichtlinien vorgegeben.

	1 Authentifizierung	Die Authentifizierung bezeichnet den Vorgang, die Identität einer Person oder eines Programms an Hand eines bestimmten Merkmals zu überprüfen. Dies kann zum Beispiel mit einem Passwort oder einem beliebigen anderen Berechtigungsnachweis geschehen.
	2 Autorisierung	Die Autorisierung bezeichnet die Zuweisung und Überprüfung von Zugriffsrechten auf Daten und Diensten an Systemnutzer. Die Autorisierung erfolgt meist nach einer erfolgreichen Authentifizierung. Beispielsweise: Hat die Partnerapplikation nur Leserechte (d.h. sie kann die Daten nicht modifizieren) oder sowohl Lese- als auch Schreibrechte?
3 Informationsaustausch		Ist der Informationsaustausch gesichert?
	1 Integrität	Unter Integrität wird die Eigenschaft nicht verändert worden zu sein verstanden.
	2 Non-Repudiation	Eine Leugnung ist das Bestreiten der Wahrheit von etwas, etwas für falsch erklären, etwas verneinen.
	3 Vertraulichkeit / Verschlüsselung	Ist die Nachricht verschlüsselt?
	4 Virenschutz	Ist die Applikation von Viren geschützt? Können Viren von anderen Applikationen ins GFS eingeschleust werden?

Tabelle 6.3: Sicherheit

Funktionalitätskonformität

Gerade bei grossen Unternehmungen bzw. derjenigen mit grossem IT wie Swiss Re ist die Eigenschaft Funktionalitätskonformität von ganz grosser Bedeutung. Sie haben stärkere Geschäftsrichtlinien, müssen mehr an internationale Normen halten, sie bieten ver-

schiedene Schablonen (Templates) und haben strenge Sicherheitsvorkehrungen (vgl. Tabelle 6.4).

FUNKTIONALITÄTSKONFORMITÄT (FUNCTIONALITY COMPLIANCE)	
Merkmal	Zusätzliche Beschreibung
1 Geschäftsrichtlinien, Normen (Governance)	Erfolgt die Spezifikation von GFS-Schnittstellen gemäss den Swiss Re Geschäftsrichtlinien oder / und internationalen Normen, falls es solche gibt?
2 Geschäftsschablonen (Templates)	Werden Geschäftsschablonen verwendet? Z.B. Kontext-Diagram Schablone, Schablonen für die Spezifikationsdokumente
3 Datenschutz	Ist das Autorisierungskonzept konform zu Swiss Re Richtlinien, um persönliche Daten zu schützen?
4 Netzwerksicherheit	Können die Schnittstellen von der Netzwerksicherheit, die Swiss Re anbietet, Gebrauch machen?
5 Datensicherung	Sind Backup- und Recovery-Funktionen vorhanden?
6 Autorisierung und Authentizität	Ist definiert, wer welche Rechte auf der Produktionsdatenbank der Applikation hat? Welche Einschränkungen müssen beispielsweise dem Outsourcing-Partner[41] gemacht werden?

Tabelle 6.4: Funktionalitätskonformität

6.1.2.2 Zuverlässigkeit

Zuverlässigkeit ist definiert als die Fähigkeit der Software, ihr Leistungsniveau unter festgelegten Bedingungen über einen festgelegten Zeitraum zu bewahren [ISO/IEC 9126-1 2001]. Für die Domäne der vorliegenden Arbeit werden folgende Basiseigenschaften gebraucht: Reife, Fehlertoleranz und Zuverlässigkeitskonformität.

Reife

Im GFS-Kontext ist die Basiseigenschaft Reife definiert als die Versagenshäufigkeit der Schnittstellen durch Fehlerzustände. In Tabelle 6.5 sind alle Merkmale aufgelistet.

[41] Ein Teil der GFS-Entwicklung bzw. die GFS-Implementierung ist ausgelagert.

REIFE (MATURITY)	
Merkmal	Zusätzliche Beschreibung
1 Produkteinführungszeit (Time to market)	Wie schnell wird die Spezifikation den Programmierer ausgeliefert, welche Versionen sind momentan verfügbar und wie viele Verbesserungen wurden pro Release angeboten. Dieses Merkmal kann sehr wichtig werden; z.B. um zu wissen, auf welchen Stand die Spezifikation befindet und wie erfolgreich Releases waren.
2 Reife der Anforderungsspezifikationen	Wie vollständig sind die Anforderungen für die Applikation spezifiziert. Z.B. Ein Attribut kann nicht abgebildet werden, falls es noch nicht definiert wurde und einheitlich ist.
3 Fehlerstatistik / Reklamationen	Unter einer Reklamation durch die Designer oder Programmierer wird das Vorliegen von Fehlern oder/und Programmiereranfragen verstanden. Z.B. wie viel Attribute wurden falsch abgebildet oder von den Programmierern nicht verstanden?

Tabelle 6.5: Reife

Fehlertoleranz

Fehlertoleranz ist definiert als die Fähigkeit der Software, ein spezifiziertes Leistungsniveau bei Nicht-Einhaltung ihrer spezifizierten Schnittstelle zu bewahren [ISO/IEC 9126-1 2001]. Ein typischer Vertreter dieser Eigenschaft ist das Merkmal Fähigkeit bei Ausfallsicherung (vgl. Tabelle 6.6).

FEHLERTOLERANZ (FAULT TOLERANCE)	
Merkmal	Zusätzliche Beschreibung
1 Fähigkeiten bei Ausfall einer Attribut	Angebotene Mechanismen, um die Verfügbarkeit und die Sicherung der anderen Attribute beizubehalten im Falle einer falsch abgebildeten Attribut oder beim Fehlen eines (z.B. beim Austausch von Kundeninformationen ist das Geburtsdatum falsch abgebildet bzw. das Format JJJJ.MM.TT wird bei der Zielapplikation nicht unterstützt. Damit der Datenaustausch trotzdem erfolgreich läuft, sollte das falsch abgebildete Attribut ignoriert werden, bzw. sollte eine Ausnahme spezifiziert werden)

2 Fähigkeiten bei Ausfall einer ganzen Komponente bzw. Applikation	Sind die Applikationen lose gekoppelt? Bedeutet der Ausfall einer Partner-Applikation auch den Ausfall der anderen?

Tabelle 6.6: Fehlertoleranz

Widerherstellbarkeit

Die Basiseigenschaft Widerherstellbarkeit ist bei der Spezifikation der Schnittstellen nicht relevant. Die Spezifikationsdokumente werden in der Regel in den Unternehmungsservern als Dateien gespeichert und somit sind sie von den Sicherheitsmassnahmen der Firma geschützt.

Zuverlässigkeitskonformität

Grad, in dem die Software Normen oder Vereinbarungen zur Zuverlässigkeit erfüllt [ISO/IEC 9126-1 2001]. Die möglichen Vereinbarungen sind vom Projekt bzw. Projektmanagement definiert. Angenommen das Projektmanagement hat die Applikationsverfügbarkeit auf 99% definiert. D.h. die Schnittstellen, die bei einem möglichen Ausfall auch die Applikation zum Absturz bzw. zum Nicht-Funktionieren bringen, müssten auch eine Verfügbarkeit von mindestens 99% gewährleisten. Nur dann ist die Applikation zuverlässig.

6.1.2.3 Verwendbarkeit

Die Schnittstellenspezifikationen sollten verständlich, lernbar und attraktiv für den Benutzer sein. [ISO/IEC 9126-1 2001]. Im Kontext der Spezifikationen sind folgende Benutzer gemeint: die Programmierer (sie sind verantwortlich für die Umsetzung der Spezifikationen), Geschäftsanalysten (sind diejenigen, die die Geschäftsbedürfnisse der Unternehmung analysieren, bzw. die Anforderungen spezifizieren) und die System-Tester.

Die Eigenschaft *Verwendbarkeit* ist gemäss ISO/IEC Standard [ISO/IEC 9126-1 2001] in fünf Basiseigenschaften aufgeteilt (vgl. Kapitel 5.1). Für die Spezifikation der Schnittstellen sind die Basiseigenschaften **Bedienbarkeit** und **Attraktivität** nicht von Bedeutung; daher werden sie hier nicht behandelt.

Verständlichkeit

Die Basiseigenschaft Verständlichkeit ist eine der wichtigsten Eigenschaften bei der Spezifikation von Schnittstellen. Um über eine verständliche Spezifikation sprechen zu kön-

nen, muss das Dokument einige Qualitätsmerkmale aufweisen. Es muss übersichtlich, einheitlich, widerspruchsfrei und lesbar sein. Das Merkmal *Übersichtlichkeit* ist immer noch ziemlich abstrakt; daher ist es nötig, es in konkretere Basismerkmale zu zerlegen: *Strukturiertheit (die Gliederung des Dokuments und die Art der Darstellung in den einzelnen Kapiteln)* und *Knappheit (redundanzfrei)*. Unter *Einheitlichkeit* wird die Eigenschaft des Produkts (Schnittstellenspezifikation) verstanden, die ein konsistentes Erscheinungsbild für alle Funktionen gewährleistet und alle definierten Standards berücksichtigt [Wallmüller 2001]. Beispielsweise müssen bei Swiss Re alle Abbildungsattribute den SDL[42] Standard berücksichtigen. Z.B. für das Attribut *Geschäftstyp* sollte nach Data Warehouse nicht sein Wert ‚Haftpflicht', sondern der entsprechende SDL-Code ‚167' exportiert werden. Das Merkmal *Widerspruchsfreiheit* heisst, dass zwei oder mehrere Funktionen einander nicht widersprechen dürfen. Z.B. Eine vollständige Liste aller Qualitätsmerkmale dieser Eigenschaft können Sie der Tabelle 6.7 entnehmen.

VERSTÄNDLICHKEIT (UNDERSTANDABILITY)		
Merkmal	Basismerkmal	Zusätzliche Beschreibung
1 Übersichtlichkeit		Das Dokument sollte strukturiert und knapp sein.
	1 Strukturiertheit	Hat das Spezifikationsdokument ein Inhaltsverzeichnis, ist in einzelne Kapitel gegliedert, sind die Abbildungstabellen übersichtlich, d.h. falls Excel benutzt wird, werden die Daten in mehrere Tabs gegliedert.
	2 Knappheit	Die Spezifikation muss knapp, präzis und redundanzfrei sein. Unverständliche Spezifikationen erhöhen die Gefahr der Missinterpretationen.
2 Einheitlichkeit		Berücksichtigung aller definierten Standards und ein konsistentes Erscheinungsbild für alle Funktionen.
	1 Standards und Normen	Wurden die firmenspezifischen Standards oder internationalen Normen für alle Schnittstellen eingehalten? Wird eine einheitliche Tabelle für alle verwendet? Wurde jedes Attribut gemäss dem Daten-Standard abgebildet?

[42] SDL (Sructure Data Language) ist ein Standard, der alle geschäftsrelevanten Daten in einheitliche und strukturierte Form darstellt.

	2 Einheitliches Erscheinungsbild	Ist der Aufbau bzw. die Gliederung des Spezifikationsdokumentes einheitlich?
3 Widerspruchsfreiheit		Gibt es Widersprüche zwischen Funktionen?
4 Lesbarkeit		Ist das Dokument für alle Benutzergruppen (Analytiker, Programmierer und Tester) lesbar? Sind die Sachverhalte anschaulich dargestellt (Graphiken, Beispiele)?

Tabelle 6.7: Verständlichkeit

Erlernbarkeit

Die Erlernbarkeit (einem Benutzer die Aneignung des Umgangs mit dem Produkt möglichst einfach zu gestalten [Wallmüller 2001]) trifft besonderes bei den Benutzerschnittstellen zu, aber für die Nachbarschnittstellen ist sie auch nicht ganz unwichtig. Insbesondere ist sie wichtig für die Analytiker; nachdem sie die Nachbarapplikationen besser kennen (beispielsweise Data Warehouse, die schlussendlich als Basis für verschiedene Auswertungen dient - Reporting), können sie die Applikationsanforderungen auch entsprechend anpassen.

ERLERNBARKEIT (LEARNABILITY)	
Merkmal	Zusätzliche Beschreibung
1 Benutzerhandbuch, Hilfe und Unterrichtskurse	Hintergrundmaterial, z.B. Was ist ein Servlet, bzw. wie ist es implementiert?
2 Referenz	Wo kommt dieser Begriff her?
3 Lernzeit	Durchschnittliche Zeit zum Erlernen einer Nachbarschnittstelle.

Tabelle 6.8: Erlernbarkeit

Verwendbarkeitskonformität

Grad, in dem die Software Normen oder Vereinbarungen zur Verwendbarkeit erfüllt [ISO/IEC 9126-1 2001]. In Hinsicht der Domäne dieser Arbeit wird diese Eigenschaft wie folgt definiert: In welchem Masse entsprechen die Schnittstellenspezifikationen den Normen oder Vereinbarungen zur Verwendbarkeit.

VERWENDBARKEITSKONFORMITÄT (USABILITY COMPLIANCE)	
Merkmal	Zusätzliche Beschreibung
1 Spezifikationsschablone (Templates)	Firmen Schablonen – die bieten bessere Strukturiertheit
2 Dateneinheitlichkeit	SDL (Structure Data Language) – unternehmensübergreifender Standard für alle Geschäftsdaten.

Tabelle 6.9: Verwendbarkeitseinhaltung

6.1.2.4 Effizienz

Verhältnis zwischen dem Leistungsniveau der Software und dem Umfang der eingesetzten Betriebsmittel unter festgelegten Bedingungen [ISO/IEC 9126-1 2001]. Für die Schnittstellen ist nur die Eigenschaft Zeitverhalten relevant.

Zeitverhalten

Zeitverhalten ist definiert als die Antwort- und Verarbeitungszeiten sowie Durchsatz bei der Funktionsausführung.

ZEITVERHALTEN (TIME BEHAVIOUR)	
Merkmal	Zusätzliche Beschreibung
1 Antwortszeit einer Nachbarapplikation	Wie schnell wir GFS von einer Nachbarapplikation mit Daten beliefert? Wie schnell liefert GFS die Daten?
2 Datenexport	Läuft der ganze Datenexport innerhalb von dem festgelegten Zeitpunkt erfolgreich?

Tabelle 6.10: Zeitverhalten

6.1.2.5 Wartbarkeit

Wartbarkeit ist definiert als Aufwand, der zur Durchführung vorgegebener Änderungen notwendig ist [ISO/IEC 9126-1 2001]. Änderungen der Schnittstellenspezifikationen können Korrekturen, Verbesserungen oder Anpassungen an Änderungen der Umgebung (z.B. neue Release für das Data Warehouse), der Anforderungen (z.B. zu einer Nachbarapplikation findet neu eine bidirektionale Kommunikation statt und nicht eine einseitige wie bis jetzt der Fall war) oder der funktionalen Spezifikationen einschliessen.

Analysierbarkeit

Wie gross ist der Aufwand, um Mängel oder Ursachen von Schnittstellen-Versagen zu diagnostizieren oder um änderungsbedürftige Teile zu bestimmen.

ANALYSIERBARKEIT (ANALYZABILITY)		
Merkmal	Basismerkmal	Zusätzliche Beschreibung
1 Verständlichkeit		Wie gross ist der Aufwand, um eine Schnittstellespezifikation zu verstehen und sie im Falle eines Versagens zu diagnostizieren.
	1 Übersichtlichkeit und Lesbarkeit	Ein gut übersichtliches und lesbares Dokument braucht in der Regel weniger Aufwand.
	2 Einheitlichkeit	Sind die Spezifikationen einheitlich? Zwei gut übersichtliche Dokumente, die aber nicht einheitlich sind, brauchen auch mehr Aufwand.
2 Komplexität der Schnittstellen		Wie komplex und verschachtelt sind die Schnittstellen spezifiziert?

Tabelle 6.11: Analysierbarkeit

Modifizierbarkeit

Hinsichtlich der Schnittstellenspezifikationen ist Modifizierbarkeit definiert als Aufwand zur Ausführung von Verbesserungen der Schnittstellen, zu deren Fehlerbeseitigung oder Anpassung an Umgebungsänderungen. Mit anderen Worten; was passiert bzw. wie gross ist der Aufwand, falls beispielsweise die CMS[43]-Applikation geändert wird.

MODIFIZIERBARKEIT (CHANGEABILITY)	
Merkmal	Zusätzliche Beschreibung
1 Versionierung	Sind alle Spezifikationsdokumente in Versionen gespeichert?
2 Strukturflexibilität	Kann die Schnittstelle geändert werden, ohne dass die Funktionalität des Programms beeinträchtigt wird? Ist die Behandlung von Ausnahmezuständen vorgesehen (d.h. was sind die Konsequenzen, falls eine Schnittstelle modifiziert wird)?

Tabelle 6.12: Modifizierbarkeit

[43] Client Management System

Prüfbarkeit

Prüfbarkeit ist definiert als Aufwand, der zur Prüfung der geänderten Schnittstelle notwendig ist.

PRÜFBARKEIT (TESTABILTIY)	
Merkmal	Zusätzliche Beschreibung
1 Analyseprotokolle / Traces (Log-File)	Log-Dateien verwalten, damit alle gesendeten oder/und empfangenen Nachrichten (messages) verfolgt werden können.
2 Zugänglichkeit	Ist eine Beschreibung aller Schnittstellen zu anderen Systemen vorhanden? Ist die Wartungsdokumentation vorhanden? Sind in der Wartungsdokumentation umgebungsspezifische Informationen verfügbar wie Version-Nummer einer Komponente, Serveradressen, etc.

Tabelle 6.13: Prüfbarkeit

6.1.2.6 Wiederverwendbarkeit

Unter Wiederverwendbarkeit wird die Eignung des Produktes oder einzelner Teile zur Übertragung in ein anderes Anwendungsgebiet bei gleich bleibender Systemumgebung verstanden.

Allgemeingültigkeit ist das wichtigste Merkmal der Wiederverwendbarkeit (vgl. Tabelle 6.14). [Wallmüller 2001] definiert dieses Merkmal als die Eigenschaft einer Komponente oder eines Moduls, von verschiedenen Applikationen unverändert benutzt zu werden.

WIEDERVERWENDBARKEIT (REUSABILITY)	
Merkmal	Zusätzliche Beschreibung
1 Allgemeingültigkeit der Schnittstelle	Kann eine ganze Schnittstelle (z.B. zu CMS) wiederverwendet werden?
2 Allgemeingültigkeit der verwendeten Module (Funktione)	Anzahl der Module, die von anderen Applikationen verwendet werden im Verhältnis zu der Gesamtanzahl der Applikation

Tabelle 6.14: Wiederverwendbarkeit

6.1.3 Festlegung der Beziehungen zwischen den Qualitätsmerkmalen

Wie bereits in Abschnitt 5.2.3 erwähnt wurde, gibt es folgende drei Klassen von Beziehungen zwischen Merkmalen: indifferent, konkurrierend und verstärkend.

Indifferente Merkmale beeinflussen sich nicht gegenseitig. Die Richtigkeitsmerkmale haben keine Wechselwirkung mit den Merkmalen der Eigenschaft Wiederverwendbarkeit.

Besondere Vorsicht sollte bei den konkurrierenden Merkmalen geboten werden, da die Erhöhung eines Merkmals die Verschlechterung des anderen bedeutet. Beispielsweise beeinflusst die Anforderung eines grossen Korrektheitsniveaus die Zuverlässigkeit negativ. Die Erhöhung des Merkmals Produkteinführungszeit wird mehrere Abbildungsfehler als Folge haben.

Weniger problematisch sind die Merkmale, die sich gegenseitig verstärken. Je höhere Qualität ein Merkmal aufweist, desto positiver wirkt es bei dem anderen. Ein typisches Beispiel ist das Merkmal *Verwendbarkeit / Verwendbarkeitseinhaltung / Dateneinheitlichkeit*, das das Basismerkmal *Funktionalität / Richtigkeit / Mapping / Abbildungsattribut* positiv beeinflussen kann. Falls die Abbildungsattribute basierend auf den Dateneinheitlichkeit-Standard spezifiziert werden, ist die Wahrscheinlichkeit viel kleiner, dass Fehler auftreten.

6.1.4 Kenngrössen-Bestimmung der Merkmale

Für mansche Merkmale ist es möglich, Metriken in Form von Formeln anzugeben. Das Basismerkmal *Richtigkeit / Mapping / Attributformat* wird mit Hilfe der folgenden Formel berechnet: Das Verhältnis zwischen der Anzahl der Attributen mit korrektem Format und der Gesamtanzahl der Attribute.

Die meisten Merkmale können mit Hilfe von einfachen booleschen Werten (Ja / Nein) bzw. Prüffragen bewertet werden (ob die Applikation mit denen übereinstimmt oder nicht). Die anderen werden in Form von Listen dargestellt. Dies bedeutet, dass für ein bestimmtes Merkmal alle möglichen Attribute aufgelistet werden.

Die komplette Liste mit allen Eigenschaften, Merkmalen (und deren Bezugsobjekten) und Metriken (Prüffragen, Kenngrössen und Listen) befindet sich im Anhang C.

6.2 Bewertung der Ergebnisse

In diesem Kapitel wird in einem ersten Teil anhand eines Rückblicks der Verlauf der Arbeit erläutert. In einem zweiten Teil wird die Auswertung der Befragung zusammengefasst und Schlussfolgerungen aus den gesammelten Erkenntnissen gezogen.

6.2.1 Rückblick

Die Zielsetzung bei der Entwicklung dieses Qualitätsmodells und insbesondere bei der Definition der Merkmale für jede ISO/IEC 9126 Basiseigenschaft lag darin, ein Bewertungssystem für Qualitätsaussagen zu schaffen, das in der (Schnittstellen-) Spezifikationsphase der Software-Entwicklung einsetzbar ist. Und es sollte als Hilfsmittel dienen, um Qualitätsziele und Qualitätsanforderungen auszuwählen und sie auch präzise und vollständig zu spezifizieren. Ein weiteres Ziel war, das Modell im Qualitätsmanagement für die Prüfung bzw. Sicherung der Qualität einzusetzen.

6.2.2 Befragung

Um die Anwendbarkeit und Effektivität des erstellten Qualitätsmodells bei der Spezifikation der GFS-Schnittstellen auszuwerten bzw. es für die Qualitätssicherung zu verwenden, gibt es grundsätzlich zwei Möglichkeiten: Das Modell wird entweder in Form einer Fallstudie in das GFS-Projekt eingesetzt oder es wird mit Hilfe von Befragungen oder / und Interviews bewertet. Die erste Möglichkeit beansprucht normalerweise viel Zeit. Das bedeutet, dass diese Form für die vorliegende Arbeit nicht anwendbar ist.

Für die Auswertung dieses Qualitätsmodell wurde ein Fragebogen erstellt (vgl. Anhang B), die aus zwei Gruppen von Fragen besteht.

Bei der ersten Gruppe geht es darum, festzustellen, ob das Qualitätsmodell für die Qualitätssicherung bzw. Qualitätsmanagement einsetzbar ist. Um die Verwendbarkeit des Qualitätsmodells für das Qualitätsmanagement sicherzustellen, müssen einige Anforderungen beachtet werden. Nach [Bächle 1996] werden sieben Anforderungen unterschieden:

- *Strukturiertheit*: Das Qualitätsmodell muss strukturiert sein, d.h. die Qualitätseigenschaften sind in Form einer Hierarchie definiert und sie sind bis auf eine operationalisierbare Ebene zerlegt worden.

- *Wiederspruchsfreiheit / Konsistenz*: Ein Qualitätsmerkmal darf nicht im Widerspruch zu einer Eigenschaft sein, der es zugeordnet ist. Ein Untermerkmal muss auch konsistent zu dem Merkmal höherer Hierarchiestufe sein.
- *Eindeutigkeit und Verständlichkeit*: Jedes Qualitätsmerkmal ist eindeutig und verständlich zu beschreiben.
- *Vollständigkeit*: Ein Qualitätsmodell ist vollständig, wenn es alle entscheidungsrelevanten Qualitätsmerkmale beinhaltet. Unter entscheidungsrelevante Merkmale werden dabei solche verstanden, die bei den beteiligten Interessengruppen (z.B. Analytiker, Programmierer, etc.) als wichtige Erfolgsfaktoren des entwickelten Systems identifiziert wurden.
- *Überschneidungsfreiheit*: Die Überschneidung der Qualitätsmerkmale darf nicht zugelassen werden. Dies bedeutet, dass die Merkmale unabhängig von einander sein müssen.
- *Operationalisierbarkeit*: Qualität ist Zielerfüllung. Die Erfüllung eines Ziels kann nur dann gemessen und geprüft werden, wenn das Ziel quantifizierbar ist. Dies bedeutet, dass die Qualitätsmerkmale eines Qualitätsmodells durch das Softwaremanagement nicht geplant, gesteuert und kontrolliert werden können, wenn sie nicht messbar sind. Die subjektiven Messungen bzw. Prüffragen sind selbstverständlich auch möglich, denn beispielsweise ist bei dem Produkt Anforderungsspezifikation die Quantifizierung aller Qualitätsmerkmale sehr schwierig oder unmöglich.
- *Erreichbarkeit*: Bei dem Qualitätsmodell geht es darum realistische Merkmale zu definieren. Das heisst die dargestellten Zielvorstellungen an die Prozess- bzw. Produktqualität, müssen den Realitätsbezug wahren. Beispielsweise stellt die Forderung der Fehlerfreiheit der Spezifikation von Schnittstellen zwar ein ideales, aber kein wirtschaftlich realisierbares Qualitätsmerkmal dar.

Ein Qualitätsmodell, das die oben genannten Anforderungen erfüllt, kann sinnvoll für die Qualitätssicherung eingesetzt werden. Dabei sind die Anforderungen als idealtypisch zu betrachten. In der Praxis kann man das Anforderungsniveau reduzieren lassen. Es könnten aber auch mehrere Niveaustufen gebildet werden. Für die Auswertung des Modells der vorliegenden Arbeit wurden diese Anforderungen als Fragen für die Befragung verwendet. Es gab für jede Anfrage fünf mögliche Antworten: Die Anforderung trifft sehr schlecht, schlecht, neutral, gut und sehr gut zu.

Die zweite Gruppe bilden Fragen der Art: Kann mit Hilfe des Qualitätsmodells der allgemeine Qualitätsbegriff durch hierarchische Ableitung von Unterbegriffen definitorisch operationalisiert werden? Kann das Modell für die Festlegung von Qualitätsanforderungen

verwendet werden? Können mit Hilfe des Qualitätsmodells die Schnittstellen beschrieben werden? Die Befragung bzw. die komplette Liste mit Fragen kann dem Anhang B entnommen werden.

6.2.3 Schlussfolgerungen

Obwohl die Erstellung des Qualitätsmodells komplex ist, bietet deren Anwendbarkeit bei der Schnittstellenspezifikationen einige Vorteile bzw. Stärken an:

- Die Anwendung des Modells hat ein besseres Verständnis des Begriffs Qualität geschaffen
- Mit Hilfe des Qualitätsmodells können die Qualitätsziele und die Qualitätsanforderungen formalisierter definiert werden
- Das Modell kann als Hilfsmittel für die die Spezifikation der Schnittstellen benutzt werden, d.h. es dient somit als präventive Massnahme
- In Verbindung mit dem Qualitätsplan kann als aktives Hilfsmittel, um die Aufgaben des Qualitätsmanagements zu erfüllen, eingesetzt werden aber es muss laufend entsprechend den änderten Bedürfnissen erweitert und modifiziert werden

Den Vorteilen sind jedoch einige noch bestehende Probleme bzw. Schwächen gegenüberzustellen:

- Für die Qualitätssicherung ist das Qualitätsmodell noch nicht reif. Es weist zwar eine gute Strukturiertheit und ist relativ konsistent, aber die Merkmale sind nicht ganz eindeutig, nicht genügend operationalisierbar und sie überschneiden sich teilweise.
- Eine weitere Schwäche des Qualitätsmodells ist die Unvollständigkeit. Einerseits sind nicht alle wichtige Merkmale definiert und andererseits wurden die Interessengruppen nicht berücksichtigt bzw. die Merkmale wurden nicht nach einer bestimmten Interessengruppe (z.B. Programmierer) identifiziert.
- Es gibt noch einen weiteren Aspekt, der den Einsatz des Qualitätsmodells für die Qualitätssicherung beeinträchtigt. Es handelt sich hier um die Überprüfungshäufigkeit eines Merkmals. Es werden nicht alle Merkmale gleich häufig bzw. periodisch überprüft. Wenn beispielsweise das Kontext-Diagramm einmal korrekt war, müsste das Merkmal nicht oder nicht so häufig überprüft werden wie die andere.

Zusammenfassend lässt sich sagen, dass durch eine Berücksichtigung der verschiedenen Sichten die Anwendbarkeit eines Qualitätsmodells erleichtert und vereinfacht wird. Ein ideales Modell deckt alle Aspekte ab, bietet alle Sichten, ist quantitativ und anpassbar.

Ein ganz wichtiger Aspekt ist noch zu berücksichtigen, der die Anwendung des Qualitätsmodells bei der Schnittstellenspezifikationen erschwert, bzw. die Überprüfung der Qualität bei den Spezifikationen beeinträchtigt. Das Ziel ist immer die Erzielung einer erforderlichen und hinreichenden Qualität, um die eigentlichen Bedürfnüsse der Benutzer zu erfüllen. ISO 8402 definiert die Qualität als die Erfüllung von festgelegten und impliziten Bedürfnissen. Dennoch reflektieren die Bedürfnisse, die von den Benutzern festgelegt wurden, nicht immer die eigentlichen Benutzerbedürfnisse, weil:

- der Benutzer oft nicht seinen eigentlichen Bedürfnissen bewusst ist
- die Bedürfnisse geändert werden können, nachdem sie festgelegt wurden
- unterschiedliche Benutzer verschiedene Betriebsumgebungen haben können
- es unwahrscheinlich ist, alle mögliche Benutzergruppen zuzuziehen

Daher können die Qualitätsanforderungen auch nicht komplett definiert werden.

7 Zusammenfassung und Ausblick

7.1 Zusammenfassung

Ziel dieser Diplomarbeit war die Entwicklung eines Qualitätsmanagementmodells (QM), welches bei den Schnittstellenspezifikationen bzw. der Integration von (unternehmensübergreifenden) GFS-Applikationen innerhalb der IT–Umgebung von Swiss Re einsetzbar ist. Das Qualitätsmodell sollte einerseits für die präzise Darstellung von Qualitätsanforderungen und für die genaue Beschreibung von Schnittstellenspezifikationen anwendbar sein und andererseits in Verbindung mit dem Qualitätsplan als Hilfsmittel dem Qualitätsmanagement bzw. der Qualitätssicherung dienen.

Nachdem in einem ersten Schritt der Problembereich bzw. die Schnittstellenspezifikations-Domäne im Projekt GFS genau analysiert wurde, um alle mögliche Probleme zu erkennen, sind im Kapitel 3 einige bestehende Qualitätsmodelle behandelt worden, welche für diese Domäne anwendbar wären.

Im zweiten Teil der vorliegenden Arbeit (Kapitel 5) wurde zunächst begründet, weshalb das Qualitätsmodell von Franch ausgewählt wurde. Anschliessend folgte die Beschreibung dieser Methodologie. Da das Franchsche Qualitätsmodell auf dem ISO/IEC 9126 basiert, wurde dieser Standard auch behandelt.

Den Abschluss dieser Arbeit bildet die Anwendung des ISO/IEC 9126 Qualitätsmodells - mit Hilfe der Methodologie von Franch - bei der Spezifikation der Schnittstellen am Beispiel der GFS-Applikation. Zudem wurde in einem letzten Abschnitt die Anwendbarkeit dieses Qualitätsmodells ausgewertet.

7.2 Ausblick

Dieser Abschnitt zeigt Ideen für weitere Untersuchungen auf, die in Zukunft vorgenommen werden könnten. Sie konnten in dieser Arbeit keine Berücksichtigung finden, da ihre Einbeziehung einerseits den zeitlichen Rahmen dieser Arbeit gesprengt hätte und anderseits zusätzliche Versuche dafür notwendig gewesen wären. Hinzu kommt noch, dass diese Ideen teilweise erst aufgrund der Ergebnisse aus dieser Arbeit entstanden sind.

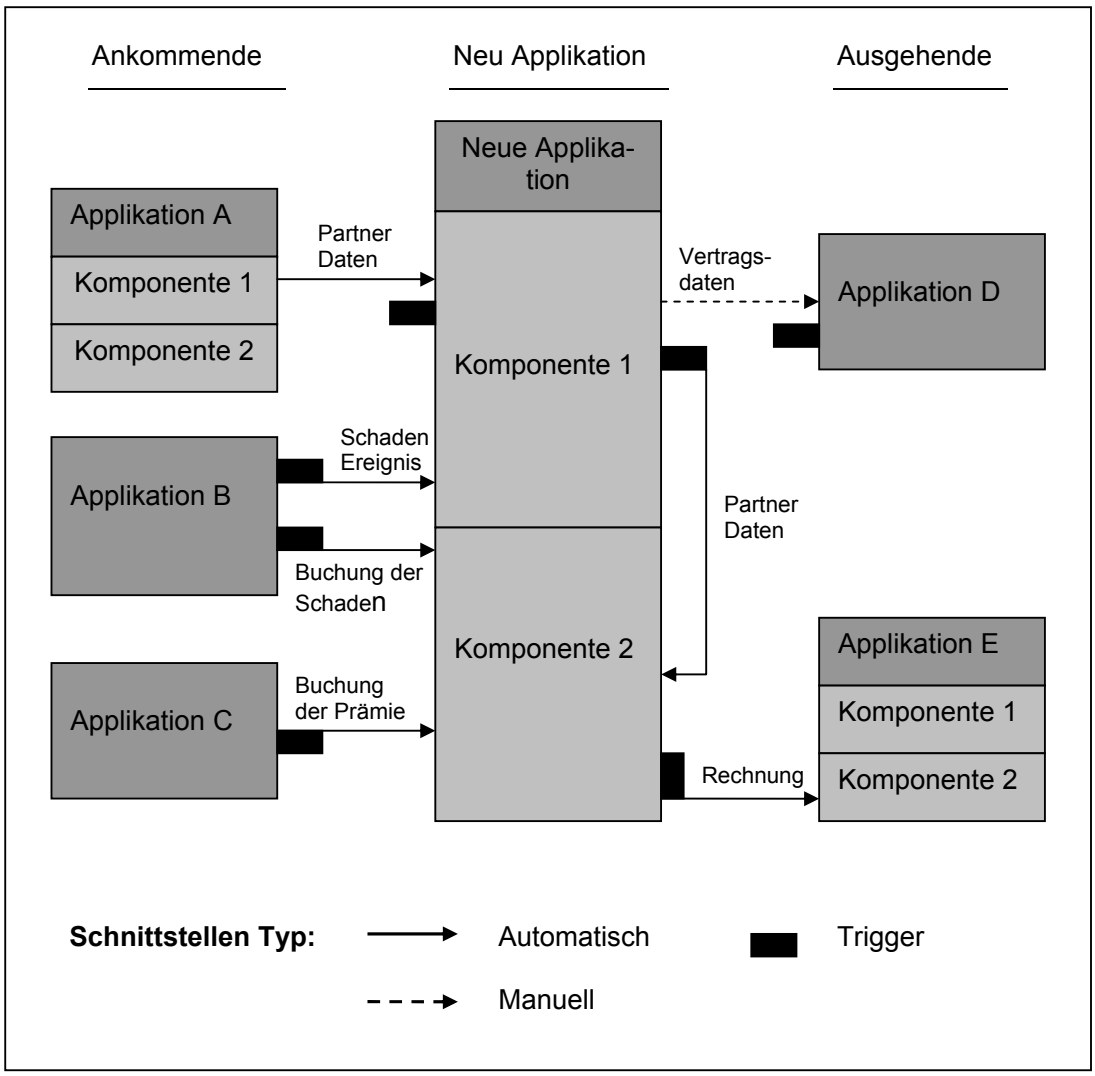
Sehr interessant wäre herauszufinden, ob das Qualitätsmodell nicht nur bei der Spezifikation von Schnittstellen anwendbar ist, sondern auch in der ganzen Anforderungsspezifikationsphase. Die Erweiterung des Modells sollte auch die verschiedenen Benutzergesichtspunkte (Analytiker, Entwickler und Benutzer) berücksichtigen.

Die Erstellung eines Qualitätsmodells, das sowohl die Produkt- als auch die Prozessqualität beinhaltet, wäre ein weiteres interessantes Thema für eine Diplomarbeit. Dabei sollte das Qualitätsmodell nicht nur die Qualitätsplanung decken, sondern auch die anderen Bestandteile des Qualitätsmanagementsystems wie die Qualitätskontrolle und Qualitätsverbesserung (vgl. das Squid-Qualitätsmodell im Abschnitt 3.2.2). Bei einem Projekt wie GFS, wo nur die Implementierung der Applikation ausgelagert wird, kann das Modell in Verbindung mit den Qualitätskosten als Hilfsmittel für das ganze Qualitätsmanagement eingesetzt werden.

Das Qualitätsmodell von Geoff Dromey verwendet (vgl. Abschnitt 3.2.3) im Gegensatz zu den anderen Modellen eine andere Methodik, nämlich den Buttom-Up Ansatz. Dieser baut diejenigen internen Eigenschaften (messbare und / oder abschätzbare) in das Produkt ein, welche die gewünschten externen (subjektiv wahrnehmbaren) Qualitätseigenschaften einbringen, wie zum Beispiel die Funktionalität, die Zuverlässigkeit, etc. In einer Arbeit könnte man die Anwendung des Qualitätsmodells von Dromey und ISO/IEC 9126 (als Top-Down Ansatz) bei den Spezifikationen empirisch untersuchen.

8 Anhang

Anhang A: Kontext-Diagramm



Anhang B: Befragung über Anwendbarkeit des Qualitätsmodells

Um die Verwendbarkeit eines Qualitätsmodells für das Qualitätsmanagement sicherzustellen, müssen folgende Anforderungen bei der Modellbildung beachtet werden [Bächle 1996]:

Verwendbarkeit des Qualitätsmodells		**Trifft gar nicht zu bis trifft voll und ganz zu**				
Eigenschaft	**Frage**	**--**	**-**	**o**	**+**	**++**
Strukturiertheit	Sind die Qualitätsmerkmale strukturiert angeordnet, d.h. in Form einer Hierarchie?					
Widerspruchsfreiheit / Konsistenz	Widersprechen die Qualitätsmerkmale den Qualitätseigenschaften oder sich gegenseitig?					
Eindeutigkeit und Verständlichkeit	Ist jedes Qualitätsmerkmal eindeutig und verständlich beschrieben?					
Vollständigkeit	Umfasst das Qualitätsmodell alle entscheidungsrelevanten Qualitätsmerkmale? Als entscheidungs-relevant zählen solche Merkmale, die von beteiligten Interessengruppen (Analytiker, Programmierer, Tester) als ein Erfolgsfaktor des zu entwickelten Informationssystems identifiziert wurden.					
Überschneidungsfreiheit	Überschneiden sich die Qualitätsmerkmale? Die Merkmale müssen abhängig voneinander sein.					
Operationalisierbarkeit	Sind alle Merkmale messbar (entweder mit Kenngrössen oder mittels Prüffragen)? Die Prüffragen müssen mit Hilfe von Reviews, Fragebogen und / oder Sitzungen. Falls die Qualitätsmerkmale nicht messbar sind, können sie durch das Softwaremanagement nicht geplant, gesteuert und kontrolliert werden.					
Erreichbarkeit	Qualitätsmerkmale stellen Zielvorstellungen an die Produktqualität dar. Sind diese Ziele realistisch?					

Zielereichung des Qualitätsmodells		Trifft gar nicht zu bis trifft voll und ganz zu				
Eigenschaft	**Frage**	**--**	**-**	**o**	**+**	**++**
Der Begriff Qualität	Konnte dieses Modell dein Verständnis für den Begriff Qualität verbessern?					
Qualitätsanforderungen	Kann man anhand von diesem Modell die Qualitätsanforderungen für die Schnittstellen formalisiert definieren? Z.B. das Modell					
Spezifikation der Schnittstellen	Kann man dieses Qualitätsmodell als Hilfsmittel (Leitfaden) für die Spezifikation der Schnittstellen einer Applikation (GFS) anwenden?					
Wissen über Domäne	Konnte ich mit Hilfe dieses Modell (weil es domänespezifisch ist) auch mehr über Schnittstellen lernen?					
Qualitätssicherung	Könnte man das Modell für die Qualitätssicherung einsetzen?					

Anhang C: Das Qualitätsmodell für die Schnittstellenspezifikationen

BASISEIGENSCHAFT	MERKMAL		BASISMERKKMAL	Bezugsobjekt	Metriktyp	Metrik / Prüffragen / Liste	Prüfung
FUNKTIONALITÄT (FUNCTIONALITY)							
Angemessenheit (Suitability)	1 Graphische Darstellung der Applikation	1	Kontext-Diagramm Attribute	A	Boolean	Sind alle Attribute des Kontext-Diagramms (automatisch vs. manuell, Trigger und Pfeile) beschrieben worden?	Review
		2	Ankommende Schnittstellen	A	Boolean / Funktion	Sind alle ankommende Schnittstellen dargestellt? Anzahl der vorhandenen im Verhältniss zu alle möglichen.	Review
		3	Abgehende Schnittstellen	A	Boolean	vgl. oben 1.2	Review
		4	Auslöser	A	Boolean	Sind alle Trigger vorhanden? (d.h. welche Applikation löst eine Aktion aus)	Review
	2 Schnittstellendefinition	1	Geschäftsanforderungen	SD	Funktion	Wie viel Prozent (%) der Geschäftsanforderungen von insgesamt sind vorhanden?	Review, Fragebogen
		2	Zielapplikation	SD	Boolean	Sind alle Zielapplikationen beschriben?	Review
		3	Schnittstellenbeschreibung	SD	Boolean	Für was brauchen wir diese Schnittstelle?	Review
		4	Typ der Schnittstelle	SD	Liste	Sind volgende Elemente definiert: Mechanismus (Batch vs. Service), Automatisch / Manuell, Häufigkeit, Volumen und Push / Pull	Review
		5	Typ der ausgetauschten Daten	SD	Liste	Daten, die ausgetausch werden. Schaden-, Prämien-, Partnerinformationen	Review
		6	Technologien	SD	Liste	Ist eine Liste mit möglichen Technologien vorhanden?	Review

	3	Schnittstellen-spezifikation	1	Beschreibung der Zielapplikation	S	Boolean	Auf welcher Art und Weise kann auf die Daten der Zielapplikation zugegriffen werden?	Review, Fragebogen
			2	Schnittstellenkonzept	S	Boolean	Ist der Schnittstellenentwurf vorhanden?	Review, Fragebogen
			3	Datenfluss	S	Boolean	Sind alle ausgetauschten Daten definiert?	Review, Fragebogen
			4	Views (entprechend der DWH Landing Area)	SW	Boolean	Sind alle Datenbank-Views gemäss den Anoferderungen von Data Warehouse Landing Area definiert?	Review, Fragebogen
			5	Benötigte Dienste	SW	Boolean	Welche Dienste (Services) benötigt die Applikation? Z.B. Data Warehouse Services.	Review, Fragebogen
			6	Technologien	S	Boolean	Mit welchen Technologien werden die Schnittstellenkonzepte, Views oder die Dienste implementiert?	Review, Fragebogen
	4	Mapping	1	Vollständigkeit der Attribute	A	Funktion	Verhältniss zwischen den vorhandeten und alle mögliche Attributen	Test, Review, Fragebogen
			2	Eindeutigkeit der Attribute	A	Funktion	Sind die Attribute eindeutig definiert?	Test, Review, Fragebogen

Richtigkeit (Accuracy)	1	Graphische Darstellung der Applikation	1	Typ der Schnittstelle	K	Funktion	Anzahl der korrekten Schnittstellentypen im Verhältniss zu der Gesamtanzahl	Test, Review
			2	Datenaustausch	K	Funktion	Anzahl der korrekten Schnittstellentypen im Verhältniss zu der Gesamtanzahl	Test, Review
	2	Schnittstellendefinition			SD	Funktion	# bereits definiert / # gesamt	Test, Review
	3	Schnittstellenspezifikation			S	Funktion	# bereits definiert / # gesamt	Test, Review

	4	Mapping	1	Abbildungsattribut	A	Funktion	# der korrekten Attributen oder Transformationsrelgeln / # gesamt	Test
			2	Attribut-Format	A	Funktion	# der Attributen mit korrektem Format / # gesamt	Test

Sicherheit (Security)	1	Datensicherheit			S	Boolean	Bieten die Schnittstellen eine Wiederherstellungsfunktion (recovery)?	Review
	2	Zugriffberechtigung	1	Authentifizierung	S	Boolean	Ist definiert?	Review
			2	Autorisierung	S	Boolean	Ist definiert?	Review
	3	Informationsaustausch	1	Integrität	S	Boolean	Wird die Nicht-Veränderbarkeit der Informationen spezifiziert?	Review
			2	Non-Repudiation	S	Boolean	Wird die Non-Repudiation der Informationen spezifiziert?	Review
			3	Vertraulichkeit / Verschlüsselung	S	Boolean	Wird die Verschüsselung der Daten spezifiziert?	Review
			4	Virenschutz	S	Boolean	Ist der Schutzt gegen Viren spezifiziert?	Review

Funktionalitätseinhaltung (Functionality compliance)	1	Geschäftsrichtlinien, Normen (Governance)			S	Boolean	Erfolgt die Spezifikation von GFS-Schnittstellen gemäss den Swiss Re Geschäftsrichtlinien oder / und internationalen Normen, falls es solche gibt?	Review
	2	Geschäftsschablonen (Templates)			S	Boolean	Werden Geschäftsschablonen verwendet? Z.B. Kontext-Diagram Schablone, Schablonen für die Spezifikationsdokumente	Review
	3	Datenschutz			S	Boolean	Ist das Autorisierungskonzept konform zu Swiss Re Richtlinien, um persönliche Daten zu schützen?	Review

	4	Netzwerksicherheit			S	Boolean	Können die Schnittstellen von der Netzwerksicherheit, die Swiss Re anbietet, gebrauch machen?	Review
	5	Datensicherung			S	Boolean	Sind Backup- und Recovery-Funktionen vorhanden?	Review
	6	Autorisierung und Authentizität			S	Boolean	Ist definiert, wer welche Rechte auf der Produktionsdatenbank der Applikation hat? Welche Einschränkungen müssen beispielsweise dem Outsourcing-Partner gemacht	Review

ZUVERLÄSSIGKEIT (RELIABILITY)								
Reife (Maturity)	1	Produkteinführungszeit	1	Fertigstellung der Spezifikation	S, A	Funktion	Wie schnell wird die Spezifikation den Programmierer ausgeliefert? Verzögerungszeit	Test
			2	Verbesserungen pro Release	S, A	Funktion	Anzahl der Verbesserungen pro Release	Test
	2	Reife der Anforderungsspezifikationen			U	Boolean	Wie vollständig sind die Anforderungen für die Applikation spezifiziert?	Review
	3	Fehlerstatistik / Reklamationen			S, A	Funktion	Wie viel Attribute wurden falsch abgebildet oder von den Programmierern nicht verstanden?	Test

Fehlertoleranz (Fault tolerance)	1	Fähigkeiten bei Ausfall einer Attribut	S	Boolean	Gibt es Mechanismen, um die Verfügbarkeit und die Sicherung von anderen Attributen beizubehalten im Falle einer falsch abgebildeten Attribut oder beim Fehlen vom einen?	Review

	2	Fähigkeiten bei Ausfall einer ganzen Komponente bzw. Applikation			S	Funktion	Wie oft bringt der Ausfall einer Partner-Applikation andere Applikationen zum Absturz?	Test

VERWENDBARKEIT (USABILITY)								
Verständlichkeit (Understandability)	1	Übersichtlichkeit	1	Strukturiertheit	S, A	Boolean	Hat das Spezifikationsdokument ein Inhaltsverzeichnis, ist in einzelne Kapitel gegliedert, sind die Abbildungstabellen übersichtlich, d.h. falls Excel benutzt wird, werden die Daten in mehrere Tabs gegliedert.	Review
			2	Knappheit	S, A	Boolean	Ist die Spezifikation knapp, präzis und redundanzfrei?	Review
	2	Einheitlichkeit	1	Standards und Normen	S, A	Boolean	Sind alle definierten Standards berücksichtigt?	Review
			2	Einheitliches Erscheinungsbild	S, A	Boolean	Ist ein konsistentes Erscheinungsbild für alle Funktionen berücksichtigt?	Review
	3	Widerspruchsfreiheit			S, A	Boolean	Gibt es Widersprüche zwischen Funktionen?	Review
	4	Lesbarkeit			S, A	Boolean	Ist das Dokument für alle Benutzergruppen (Analytiker, Programmierer und Tester) lesbar?	Review

Erlernbarkeit (Learnability)	1	Benutzerhandbuch, Hilfe und Unterrichtskurse			S	Boolean	Hintergrundmaterial, z.B. Was ist ein Servlet, bzw. wie ist es implementiert?	Review, Fragebogen
	2	Referenz			S	Boolean	Wo kommt dieser Begriff her?	Review, Fragebogen
	3	Lernzeit			K, SD, S, A	Funktion	Durchschnittliche Zeit zum Erlernen einer Nachbarschnittstelle.	Test, Fragebogen

Verwendbarkeitseinhaltung (Usability compliance)	1	Spezifikationsschablone (Templates)			SD, S	Boolean	Wurden Firmen-Schablonen verwendet? Die bieten bessere Strukturiertheit	Review
	2	Dateneinheitlichkeit			S	Boolean	Wurden die unternehmensübergreifende Geschäftsdaten-Standards einhgehalten? Z.B. SDL (Structure Data Language)	Review

EFFIZIENZ (EFFICIENCY)								
Zeitverhalten (Time behaviour)	1	Antwortszeit einer Nachbarapplikation			S	Funktion	Wie schnell wir GFS von einer Nachbarapplikation mit Daten beliefert? Wie schnell liefert GFS die Daten?	Test
	2	Datenexport			S	Funktion	Läuft der ganze Datenexport innerhalb von dem festgelegten Zeitpunkt erfolgreich?	Test

WARTBARKEIT (MAINTAINABILITY)								
Analysierbarkeit (Analyzability)	1	Verständlichkeit	1	Übersichtlichkeit und Lesbarkeit	S, A	Funkiton	Wie gross ist der Aufwand, um eine Schnittstellespezifikation zu verstehen und somit im Falle eines Versagens sie zu diagnostizieren.	Review, Befragung
			2	Einheitlichkeit	S, A	Funkiton	Sind die Spezifikationen einheitlich? Zwei gut übersichtliche Dokumente, die aber nicht einheitlich sind, brauchen auch mehr Aufwand.	Review, Befragung
	2	Komplexität der Schnittstellen			S, A	Boolean	Wie komplex sind die Schnittstellen spezifiziert? Wie verschachtelt sind sie?	Review, Befragung

Modifizierbarkeit (Changeability)	1	Versionierung	K, SD, S, A	Boolean	Sind alle Spezifikationsdokumente in Versionen gespeichert?	Test
	2	Strukturflexibilität	S	Boolean	Kann die Schnittstelle geändert werden, ohne dass die Funktionalität des Programms beeinträchtigt wird? Ist die Behandlung von Ausnahmezuständen vorgesehen (d.h. was sind die Konsequenzen, falls eine Schnittstelle modifiziert wird)?	Review

Prüfbarkeit (Testability)	2	Analyseprotokolle / Traces (Log-File)	S	Boolean	Gibt es Log-Dateien, damit alle gesendeten oder/und empfangenen Nachrichten (messages) verfolgt werden können?	Test
	3	Zugänglichkeit	S	Liste	Ist eine Beschreibung aller Schnittstellen zu anderen Systemen vorhanden? Ist die Wartungsdokumentation vorhanden? Sind in der Wartungsdokumentation umgebungsspezifische Informationen verfügbar wie Version-Nummer einer Komponente, Serveradressen, etc.	Test, Review

WIEDERVERWENDBARKEIT (REUSABILITY)						
	1	Allgemeingültigkeit der Schnittstelle	S	Boolean	Kann eine ganze Schnittstelle (z.B. zu CMS) wiederverwendet werden?	Review

Wiederverwendbarkeit	2	Allgemeingültigkeit der verwendeten Module (Funktione)	S	Funktion	Anzahl der Module, die von anderen Applikationen verwendet werden im Verhältnis zu der Gesamtanzahl der Applikation	Test

Legende:

Es werden folgende Bezugsobjekte verwendet:

Kontext-Diagramm der Applikation	**K**
Schnittstellendefinition	**SD**
Schnittstellenspezifikation	**S**
Schnittstellenspezifikation - Data Warahouse	**SW**
Schnittstellenspezifikation - EAI	**SE**
Schnittstellenspezifikation - Allgemein	**SA**
Abbildungstabelle	**A**
Unterstützungsfunktionen (z.B. die Anforderungen)	**U**

9 Literaturverzeichnis

[Bächle 1996] Bächle, M.: *Qualitätsmanagement der Softwareentwicklung: das QEG-Verfahren als Instrument des Total Quality Managements*. Deutscher Universitätsverlag, Wiesbaden, 1996

[Baker 2001] Baker, E.B.: *Which way, SQA?: Software quality assurance*. IEEE Software, Volume 18, Issue 1, 16 – 18, Jan.-Feb. 2001

[Basili 1992] Basili, V.: *Software Modelling and Measurement: The Goal-Question-Metric Paradigm*. Technical Report, University of Maryland, CS-TR-2956

[Bergbauer 2004] Axel, K. Bergbauer, K.A., mit Beiträgen von Kleemann, B. at al.: *Six Sigma in der Praxis : das Programm für nachhaltige Prozessverbesserungen und Ertragssteigerungen*. Expert-Verlag, Renningen, 2004

[Buschmann *at al*. 1998] Buschmann, F., Meunier, R., Rohnert, H., Sommerland, P., Stal, M.: *Pattern-orientierte Software-Architektur.* Addison-Wesley-Longman, 1998

[Dromey 1995] Dromey, R.G.: *A model for software product quality*. IEEE Transactions on Software Engineering, 21(2), 146-162, Feb. 1995

[Dromey 1996] Dromey, R.G.: *Concerning the Chimera.* IEEE Software, Volume 13, Issue 1, 33 – 43, Jan. 1996

[Franch und Carvallo 2003] Franch, X., Pablo Carvallo, J.: *Using quality models in software package selection.* IEEE Software, Volume 20, Issue 1, 34 – 41, Jan.-Feb. 2003

[Franch und Carvallo 2002]: *Towards the definition of a quality model for mail servers*. [Online Document] http://www.lsi.upc.edu/dept/techreps/llistat_detallat.php?id=593 (12.02.2006)

[Gall at al. 2003] Gall, H., Dustdar, S., Hauswirth, M.: *Software-Architekturen für Verteilte Systeme.* Springer Verlag, 2003

[Garvin 1984] Garvin, D. A.: *Product Quality: What does Product Quality really mean?*. Sloan Management Review, Vol. 26 (1984), H. 1, 25-43

[Geppert 2002] Geppert, A.: *Data Warehousing*. [Online Dokument] URL: http://www.ifi.unizh.ch/dbtg/Classes/DWH/ Skript, Universität Zürich, 2002

[Glinz 2004] Glinz M.: *KV Software Engineering*. [Online Dokument] URL: http://www.ifi.unizh.ch/req/courses/kvse/, Kapitel 7 – 10, SS 2005

[Glinz 2005] Glinz M.: *Software Engineering*. [Online Dokument] URL: http://www.ifi.unizh.ch/req/courses/se/, WS 2005/2006

[IEEE 1988] IEEE (1988): *Standard Dictionary of Measures to Produce Reliable Software.* IEEE Std 982.1 – 1988, IEEE Computer Society Press

[IEEE 1998] IEEE Std 1061 – 1998: *IEEE Standard for a Software Quality Metrics Methodology.* The Institute of Electrical and Electronics Engineers, Inc., New York, 1998

[ISO 9000:2000] *Qualitätsmanagementsysteme – Grundlagen und Begriffe.* Deutsche Fassung der Europäischen Norm EN ISO 9000 (deutsch/englisch/französisch)

[ISO/IEC 9126-1 2001] ISO/IEC Standard 9126-1 Software engineering: *Product quality - Part 1: Quality model.* ISO Copyright Office, Geneva, 2001

[Kitchenham und Pfleeger 1996] Kitchenham, B.; Pfleeger, S.L.: *Software quality: the elusive target.* Software, IEEE Volume 13, Issue 1, 12 – 21, Jan. 1996

[Kitchenham *et al.* 1999] Kitchenham B., Boegh J., Depanfilis S., Pasquini A.: *A Method for Software Quality Planning, Control, and Evaluation.* IEEE Software, Volume 23, No. 2, 69-77, Mar. / Apr. 1999.

[Mizumo 1983] Mizumo Y.: *Software Quality Improvement*. IEEE Computer, 66-72, März 1983

[PADi 2005] *P&C Application Directory*, [Internes Dokument] Swiss Re, 2005

[Rahm 2004] Rahm, E.: *Data Warehousing*. [Online Dokument] URL: http://dbs.uni-leipzig.de/de/skripte/DWDM/HTML/kap4-15.html SS 2004 (15.01.2006)

[Scherz 2000] Scherz R.: *Qualitätsaspekte für Data Warehouse Systeme*. Diplomarbeit, Universität Zürich, IFI, 2000

[SEI 2004] http://www.sei.cmu.edu/cmmi/presentations/euro-sepg-tutorial/sld006.htm Software Engineering Institute, CMMI Combined Tutorial, 2004 (22.03.2006)

[Siedersleben 2003] Siedersleben, J.: *Softwaretechnik : Praxiswissen für Software-Ingenieure.* Hanser Verlag, München, 2003

[Siedersleben 2004] Siederleben, J.: *Moderne Software-Architektur.* dpunkt.verlag, Heidelberg:, 2004

[Tanenbaum 2003] Tanenbaum, A., van Stehen, M.: *Verteilte Systeme : Grundlagen und Paradigmen.* Pearson Studium, München, 2003

[Wallmüller 2001] Wallmüller, E.: *Software- Qualitätsmanagement in der Praxis*. Hanser Verlag, München, 2001