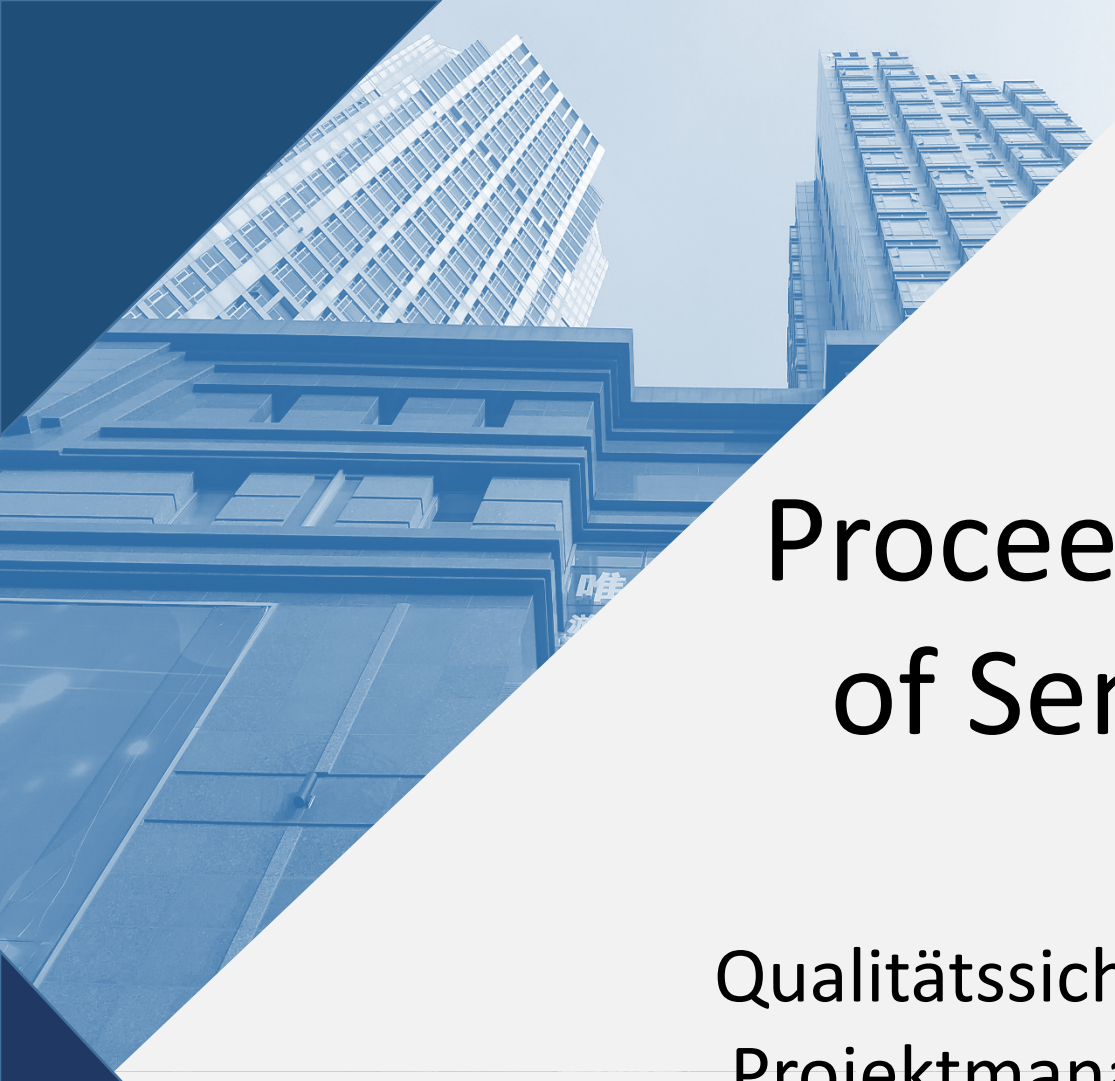

Proceedings of Seminar

Qualitätssicherung & Projektmanagement

2004

Editors: Horst Lichter

SWC Software Construction | RWTH AACHEN UNIVERSITY

Inhaltsverzeichnis

Teil 1

Prozesse und deren Bewertung

Capability Maturity Model Integration (CMMI) - Modell

Thomas Wilms

Inhaltsverzeichnis

1 Einführung

„Die Qualität eines Systems oder Produkts wird hochgradig beeinflusst von der Qualität der Prozesse, die für die Entwicklung dieses Systems oder Produkts notwendig sind.“ [3] Diese Prämisse des Prozessmanagements bildet die Grundlage für Capability Maturity Modelle (CMMs), deren Ziel es ist, Organisationsprozesse zu verbessern und zu optimieren. Seit 1991 wurden verschiedene CMMs für unzählige Disziplinen entwickelt; u.a. das „Capability Maturity Model for Software“ (SW-CMM), das „Systems Engineering Capability Modell“ (SECM) oder das „Integrated Product Development Capability Maturity Model“ (IPD-CMM). Obwohl sich diese Modelle als nützlich für viele Organisationen erwiesen haben, kam doch mit der Zeit der Wunsch auf, ein Modell zu entwickeln, das disziplinübergreifende Prozessverbesserung ermöglicht. Das Capability Maturity Model Integrated (CMMI) wurde vom Software Engineering Institute (SEI) entwickelt, und geht aus den oben genannten Modellen hervor. Es hat dabei die Aufgabe, die bisherigen disziplinspezifischen Modelle zu einem neuen Modell zu vereinen und dann abzulösen. Ziel dieser Ausarbeitung ist es, das CMMI Modell näher zu beleuchten.

2 CMMI Modell - ein erster Überblick

In der heutigen Zeit sollen Produkte immer besser, schneller und billiger gefertigt werden. Gleichzeitig steigt aber auch die Komplexität dieser Produkte; Einzelkomponenten werden nicht mehr nur firmenintern produziert, sondern auch von firmenexternen Lieferanten bezogen. Diese Einzelkomponenten werden dann zu einem fertigen Endprodukt zusammengesetzt. Heutige Organisationen müssen fähig sein, diese komplexe Produktentwicklung zu managen und zu kontrollieren. Außerdem finden sich viele Organisation im Software Business wieder, die ursprünglich keine typischen Softwarefirmen sind, wie z.B. Banken, Autohersteller oder Versicherungsgesellschaften, und die nun Software intern entwickeln wollen. Das Software- und Systems-Engineering werden mehr und mehr zu wichtigen Kernprozessen dieser Organisationen. Aus diesem Grunde werden Richtlinien in Form von CMMs, die solchen Organisation helfen, Ihre Geschäftsprozesse zu verbessern, immer wichtiger. Bisherige Modelle waren aber immer nur auf eine Disziplin beschränkt, wie z.B. das Software-Engineering, was zur Folge hatte, dass Prozessoptimierungen anderer Disziplinen nicht unterstützt wurden.

CMMI ist interdisziplinär und behandelt vier zentrale Disziplinen bzw. Wissensgebiete, die bei Bedarf um weitere ergänzt werden können:

- Systems-Engineering beinhaltet die Entwicklung ganzer Systeme, die Software beinhalten oder auch nicht.
- Software-Engineering beinhaltet die Entwicklung von Software Systemen.
- Intregrierte Produkt- und Prozessentwicklung beinhaltet Produktbetreuung und -pflege während einer Produktlebenszeit.

- Lieferantenfindung (Supplier Sourcing) beinhaltet die Suche, Analyse und das Managen von Lieferanten und die Zulieferung von Teilprodukten über diese.

Jeder Disziplin sind mehrere Prozessgebiete (Process Areas) zugeordnet, die jeweils für sich mehrere Praktiken beinhalten. Diese wiederum erfüllen, falls sie erfolgreich in die Abläufe einer Organisation implementiert werden, bestimmte Ziele (Goals), deren Erreichen eine Prozessverbesserung in diesem Gebiet darstellt.

Jedes Prozessgebiet wird einer gewissen Stufe (0-5) zugeordnet. Erreicht eine Organisation alle Ziele der Prozessgebiete einer Stufe, so erlangt sie nach einer Bewertung über Dritte einen bestimmten Reifegrad (Maturity Level). Dazu später mehr.

Die Stufen und zugehörigen Prozessgebiete der Systems-Engineering Disziplin sind die in Abb. 1 dargestellten.

Stufe	Prozessgebiet
2	Konfigurationsmanagement
2	Messung und Analyse
2	Projektverfolgung und -steuerung
2	Projektplanung
2	Qualitätssicherung von Prozessen und Produkten
2	Anforderungsmanagement
2	Management von Lieferantenvereinbarungen
3	Entscheidungsanalyse und -findung
3	Integriertes Projektmanagement
3	Integriertes Lieferanten Management
3	Integrierte Teamarbeit
3	Organisationsweites Integrationsumfeld
3	Organisationsweite Prozessdefinition
3	Organisationsweiter Prozessfokus
3	Organisationsweites Training
3	Produktintegration
3	Anforderungsentwicklung
3	Risiko Management
3	Technische Umsetzung
3	Validation
3	Verifikation
4	Organisationsweite Prozessperformanz
4	Quantitatives ProjektManagement
5	Ursachenanalyse und Problemlösung
5	Organisationsweite Innovation und Verbreitung

Abbildung 1: Prozessgebiete und ihre Stufen ([3])

Die Prozessgebiete und zug. Stufen der Software-Engineering Disziplin, der Disziplin der Integrierten Produkt- und Prozessentwicklung und der der Lieferantenfindung sind dieselben, wie für die erste Disziplin, mit dem Zusatz, dass die in jedem Prozessgebiet

existierenden Disziplinzusätze (Disciplin Amplifications) für eine gewählte Disziplin besondere Aufmerksamkeit erfahren müssen.

3 Modellkomponenten

Zentraler Bestandteil des CMMI Modells sind die Prozessgebiete. Jedes Prozessgebiet bzw. das beschreibende Dokument setzt sich aus *geforderten* (Required Components), *erwarteten* (Expected Components) und *informativen* (Informative Components) Bestandteilen zusammen. Ein struktureller Aufbau des Modells soll mit Abb. 2 vermittelt werden.

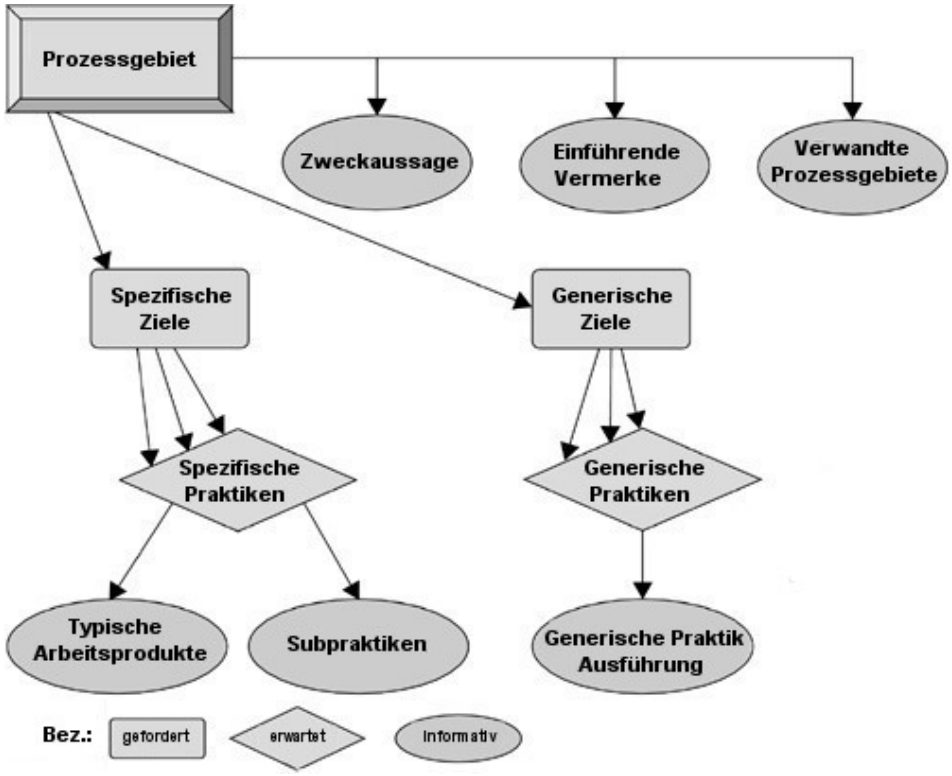

Abbildung 2: Komponenten des CMMI Modells ([3])

Geforderte Komponenten beschreiben in Form *spezifischer* und *generischer Ziele* (Generic Goals/Specific Goals), was eine Organisation erreichen muss, um eine Entwicklungsstufe des jeweiligen Prozessgebietes abzuschließen. Sie sind außerdem die Grundlage für Bewertungen (Appraisals), in denen Organisationen gewissen Reifegraden der Entwicklung (Maturity Levels) zugeordnet werden sollen. *Spezifische Ziele* definieren dabei die für ein Prozessgebiet speziellen minimalen Vorgaben, die erreicht werden müssen, um ein Prozessgebiet als durchgeführt anzusehen. *Generische Ziele* heißen generisch, weil sie nicht an ein bestimmtes Prozessgebiet gebunden sind und in mehreren Prozessgebieten erscheinen können. Diese Ziele definieren, welche ge-

nerellen Vorgaben erreicht werden müssen, um bestimmte Prozessentwicklungsstufen eines Gebiets abzuschließen. Sie bilden ein notwendiges Fundament, um die vorhandenen Teilprozesse eines Gebietes verbessert zu planen und zu kontrollieren. Außerdem stellen sie einen Indikator für die Wahrscheinlichkeit dar, ob ein Teilprozess effektiv, wiederholbar und stabil ist.

Erwartete Komponenten beschreiben in Form *spezifischer* und *generischer* Praktiken, welche Aktivitäten in einer bestimmten Art und Weise durchzuführen sind, um die geforderten Komponenten bzw. Ziele zu erlangen. Erwartete Komponenten dienen auch als Richtlinie für diejenigen, die Prozesse verbessern oder evaluieren wollen. Um Ziele als erreicht anzusehen, müssen die beschriebenen Praktiken in den gegenwärtigen und geplanten Aktivitäten bzw. Prozessen einer Organisation etabliert sein. *Spezifische Praktiken* sind diejenigen Aktivitäten, die als unabdingbar für das Erreichen der spezifischen Ziele eines Prozessgebietes gelten. Generische Praktiken sind Beschreibungen von Aktivitäten, die wichtig für das Erreichen generischer Ziele sind. Sie erscheinen ebenso wie die dazugehörenden Ziele in verschiedenen Prozessgebieten.

Informative Komponenten geben eine Anleitung, wie Organisationen am besten die geforderten und erwarteten Komponenten erreichen können:

- Jedes Dokument über ein Prozessgebiet beginnt mit einer *Zweckaussage* (Purpose), die den Sinn und Zweck des Prozessgebietes näher beschreibt (Bsp. aus dem Gebiet Anforderungsentwicklung: „Der Sinn und Zweck der Anforderungsentwicklung ist der, Kunden-, Produkt- und Komponentenanforderungen hervorzubringen und zu analysieren.“(Alle Beispiele aus [1]).
- In den darauf folgenden *Einführenden Vermerken* (Introductory Notes) werden die Hauptkonzepte des Prozessgebietes aufgeführt. (Bsp. aus dem Gebiet Projektplanung: „Die Planung beginnt mit den Anforderungen, die ein Produkt und ein Projekt definieren.“)
- Die *verwandten Prozessgebiete* (related Process Areas) beschreiben high-level Beziehungen zwischen den einzelnen Gebieten.(Bsp. aus der Projektplanung: „Betrachte das Risiko Management Prozessgebiet, um mehr Informationen über das Identifizieren und Managen von Risiken zu erlangen.“)
- *Typische Arbeitsprodukte* bilden den allgemeinen Output einer spezifischen Praktik. (Bsp. aus dem Gebiet der Prozess Verifikation: Aus der spezifischen Praktik „Etabliere und verfolge Verifikationsprozeduren und Kriterien für die gewählten Produkte“ geht das Arbeitsprodukt „Verifikationskriterium“ hervor).
- *Subpraktiken* geben Richtlinien vor, um eine spezifische Praktik zu interpretieren und zu implementieren. (Bsp. aus dem Gebiet Projektverfolgung und -steuerung: Eine Subpraktik für die spezifische Praktik „Leite korrigierende Maßnahmen bzgl. identifizierter Probleme ein.“ ist „Bestimme und dokumentiere geeignete Maßnahmen um das identifizierte Problem anzugehen“.)
- Die letzte der informativen Komponenten ist die *Generische-Praktik-Erweiterungs* (Generic Practic Elaboration) Komponente. Sie folgt einer generischen Praktik

und beschreibt, wie diese speziell auf das jeweilige Prozessgebiet angewendet werden soll. (Bsp. aus dem Gebiet Prozessverifikation: Eine Ausführung der generischen Praktik „Etabliere und verfolge eine Organisationsstrategie für das Planen und Durchführen von Verifikationsprozessen.“ ist „Diese Strategie etabliert Erwartungen der Organisation für das Etablieren und Erhalten von Verifikations Methoden, Prozeduren, Kriterien usw.“).

- Besondere Bedeutung haben auch noch die *Disziplinzusätze* (Discipline Amplifications), die gewisse Zusatzinformationen liefern, sofern man sich für die Verbesserung in der angegebenen Disziplin entschieden hat. (Bsp. aus dem Gebiet der Prozess Integration: Ein Disziplinzusatz der spezifischen Praktik „Etabliere und verfolge Prozeduren und Kriterien für die Integration von Produktkomponenten“ lautet „Zusatz für die Disziplin Lieferantenfindung: Sachbezogene Informationen des Entwurfplans und Kriterien für den Zusammenbau sollten mit den Lieferanten von Arbeitsprodukten geteilt werden, um das Auftreten von Verzögerungen und Komponentenversagen zu vermindern.“)

Abb. 3-5 im Folgenden sollen einen Eindruck vom strukturellen Aufbau eines Prozessgebietsdokuments vermitteln, welches die oben angesprochen Komponenten für das jeweilige Gebiet beinhaltet.

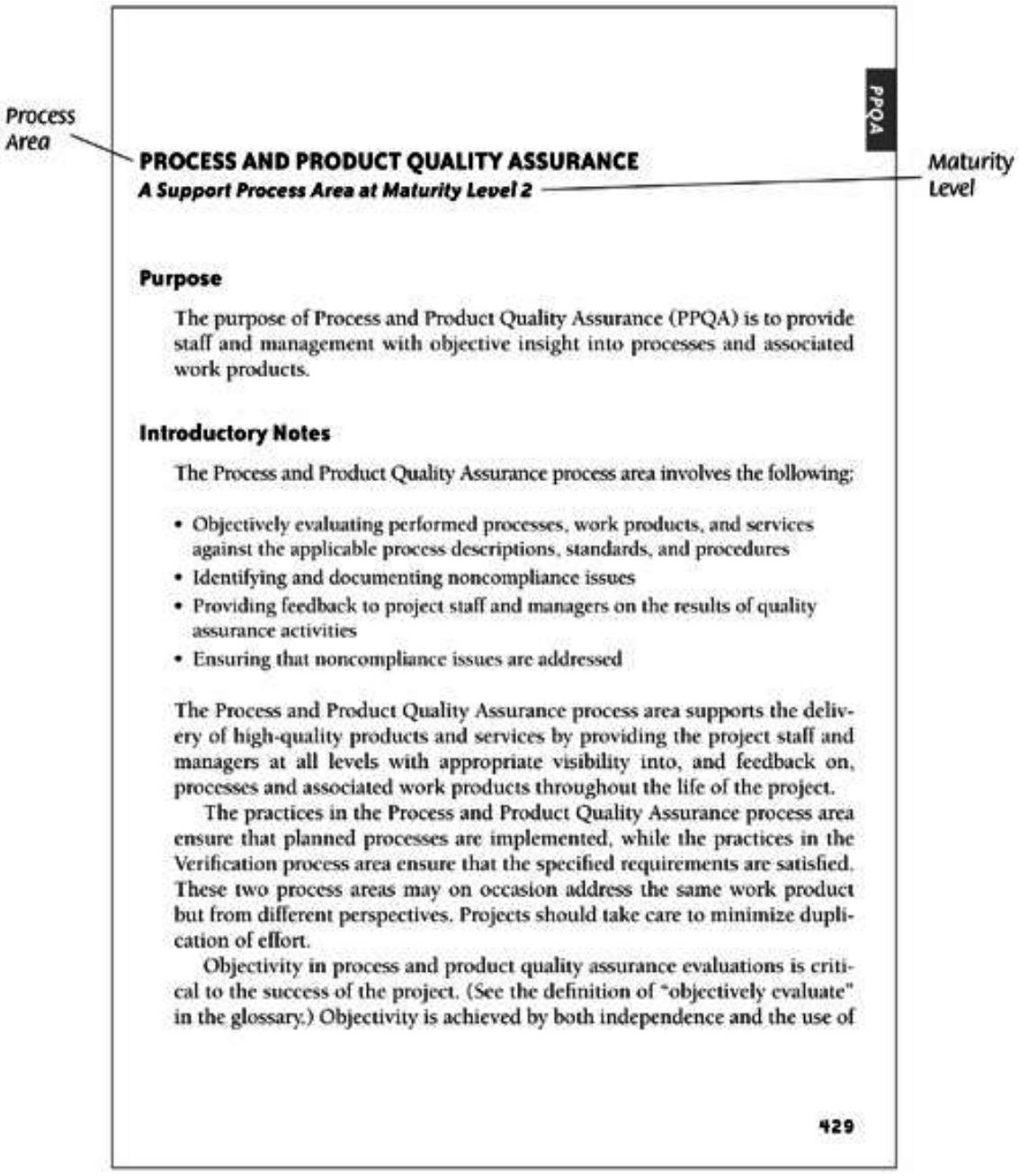

Abbildung 3: Dokument eines Prozessgebiets - 1

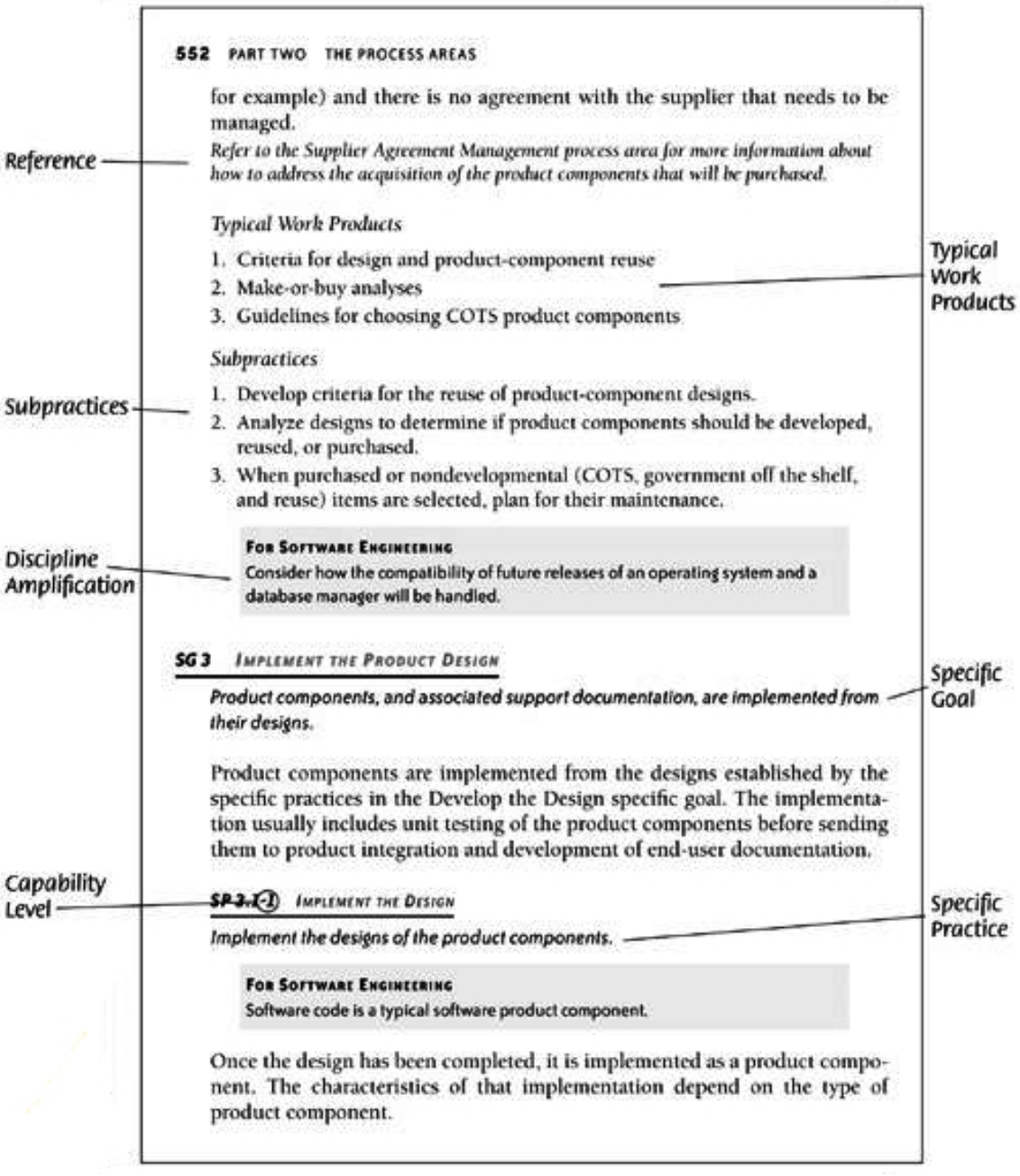

Abbildung 4: Dokument eines Prozessgebiets - 2

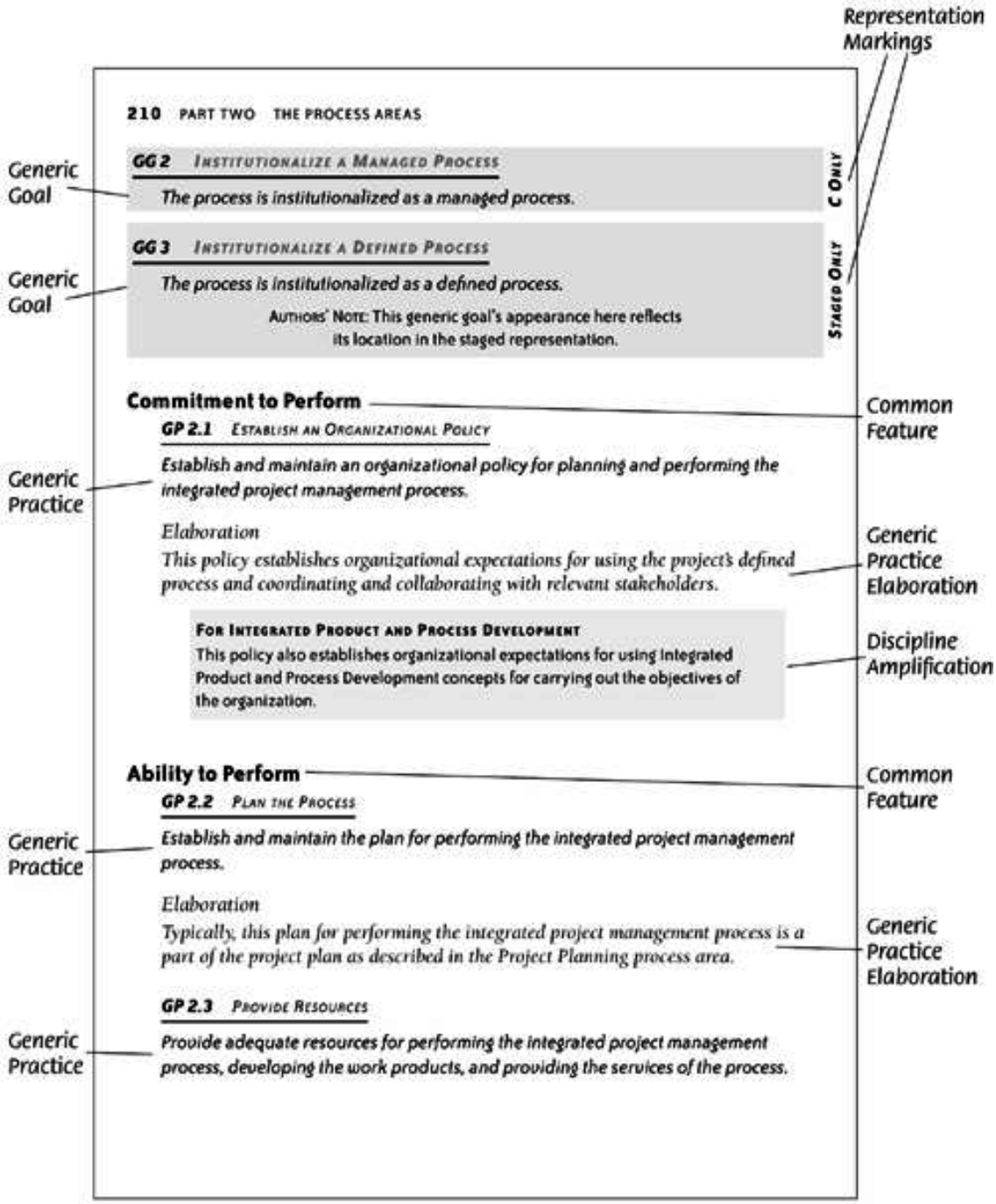

Abbildung 5: Dokument eines Prozessgebiets - 3

4 Prozesse und ihre Entwicklungsstufen

Um das Prinzip der Prozessoptimierung, das zentrales Thema des CMMI Modells ist, besser zu verstehen, ist es notwendig, die einzelnen Stufen der Entwicklung eines Prozesses näher zu beleuchten. Das CMMI Modell unterscheidet 5 Arten von Prozessentwicklungsstufen, die jeweils mit dem Erreichen des zugehörigen generischen Zieles abgeschlossen werden. Sie sind in Abb. 6 dargestellt und werden nachfolgend eingehender betrachtet.

Generisches Ziel (GZ)	Prozess Entwicklungsstufe
GZ 1	Ausgeführter Prozess (Performed)
GZ 2	Gemanagter Prozess (Managed)
GZ 3	Definierter Prozess (Defined)
GZ 4	Quantitativ gemanagter Prozess (Quantitativly Managed)
GZ 5	Optimierender Prozess (Optimizing)

Abbildung 6: Generische Ziele und Prozessentwicklungsgrade

- *Ausgeführter Prozess*

  Ein durchgeführter Prozess ist ein Prozess, der imstande ist, Arbeitsprodukte hervorzubringen.

- *Gemanagter Prozess*

  Ein gemanagter Prozess ist ein ausgeführter Prozess mit folgenden Eigenschaften:

  - er muss geplant und ausgeführt werden unter Zuhilfenahme geeigneter Arbeitskräfte, die mit geeigneten und angemessenen Ressourcen den gewünschten Output produzieren.
  - er involviert die am Abschluss des Prozesses interessierten Personen, wird überwacht, kontrolliert und geprüft.
  - der Prozess kann instantiiert werden durch ein Projekt, einer Gruppe oder einer organisationellen Funktion.
  - einen Prozess zu managen heißt das betreffende Prozessgebiet zu institutionalisieren und die damit verbundenen Faktoren wie Kosten, zeitlicher Ablauf und Qualitätsbedingungen zu betrachten und zu bestimmen.
  - die durch einen gemanagten Prozess erlangte Kontrolle über diesen hilft dabei, den Prozess in Stresszeiten zu stabilisieren. Die Anforderungen und Ziele des Prozesses werden festgelegt. Der Status von Arbeitsprodukten und Dienstleistungen ist jederzeit vom Management einsehbar (z.B. durch Milestones oder der Vollendung von Teilaufgaben).
  - Verantwortungen werden den an der Arbeit beteiligten Personen und beteiligten Geschäftspartnern zugeteilt und neu aufgeteilt falls nötig. Arbeitsprodukte werden mit den relevanten Geschäftspartnern überprüft und kontrolliert, außerdem müssen sie den Anforderungen genügen.

Eine wichtige Unterscheidung zwischen einem ausgeführten Prozess und einem gemanagten Prozess ist das Ausmaß, in welchem der Prozess einer gewissen Planung unterliegt. Ein gemanagter Prozess ist ein geplanter Prozess, und sein Ablauf wird ständig anhand des Plans überprüft. Korrigierende Maßnahmen müssen ergriffen werden, wenn die realen Ergebnisse signifikant vom Plan abweichen. Ein gemanagter Prozess erfüllt die planmäßig vorgegeben Ziele und ist angelegt auf eine beständige Durchführung. Das Erreichen des generischen Ziels auf Stufe 2 eines Prozessgebietes heißt, dass die Prozesse im jeweiligen Gebiet von nun an gemanagte Prozesse sind.

- *Definierter Prozess*

  Ein definierter Prozess ist ein gemanagter Prozess, der aus dem Pool von Organisationprozessen gemäß entsprechender interner Richtlinien zugeschnitten wird. Er hat eine ständig zu aktualisierende Prozessbeschreibung, steuert Arbeitsprodukte, Maßregeln und andere prozessoptimierende Informationen dem Pool von Organisationsprozessen bei.

  Der Pool von Organisationsprozessen besteht aus Artefakten, die sich auf das Beschreiben, Implementieren und Optimieren von Prozessen beziehen. Diese Artefakte werden für die primären Geschäftsziele einer Organisation entwickelt oder angesammelt.

  Ein Set von Standardprozessen, die die Basis eines definierten Prozesses bilden, werden mit der Zeit etabliert und verbessert. Standardprozesse beschreiben die fundamentalen Prozesselemente, die in einem definierten Prozess erwartet werden. Sie beschreiben außerdem die Beziehungen zwischen diesen Prozesselementen. Die Infrastruktur einer Organisation, die den gegenwärtigen und zukünftigen Gebrauch von Standardprozessen unterstützt, wird etabliert und im Laufe der Zeit verbessert. Der einem Projekt zugehörige definierte Prozess oder auch mehrere derselben Art stellen die Basis für das Planen, Durchführen und Verbessern von Projektaufgaben dar. Ein definierter Prozess gibt folgende Punkte an:

  - Zweck
  - Eingaben
  - Eingangskriterien
  - Aktivitäten
  - Rollen
  - Maßregeln
  - Verifikationsschritte
  - Ausgaben
  - Ausgabekriterien

  Eine wichtige Unterscheidung zwischen einem gemanagten Prozess und einem definierten Prozess ist das Ausmaß der Anwendungen, auf die sich die Prozessbeschreibungen und -prozeduren beziehen. Bei einem gemanagten Prozess

beziehen sich die Prozessbeschreibungen und -beschreibungen auf ein spezielles Projekt, eine Gruppe oder organisationelle Funktion. Deshalb können sich die gemanagten Prozesse zweier Projekte einer Organisation voneinander unterscheiden. Eine weitere wichtige Unterscheidung ist, dass ein definierter Prozess detaillierter beschrieben und strenger durchgeführt wird als ein gemanagter Prozess. Dies bedeutet, dass eine Information über Verbesserungen leichter zu verstehen, zu analysieren und zu gebrauchen ist. Letztendlich basiert ein definierter Prozesses auf dem zusätzlichen Wissen, dass durch das Verstehen von Prozesszusammenhängen bzw. deren Aktivitäten erst entsteht. Das Erreichen des generischen Ziels auf Stufe 3 eines Prozessgebietes heißt, dass die Prozesse im jeweiligen Gebiet von nun an definierte Prozesse sind.

- *Quantitativ gemanagter Prozess*

  Ein quantitativ gemanagter Prozess ist ein definierter Prozess, der kontrolliert wird durch statistische und quantitative Techniken.

  - die Produktqualität, Servicequalität und Leistungseigenschaften des Prozesses sind messbar und werden im ganzen Verlauf des Projektes kontrolliert.
  - quantitative Ziele werden mit Blick auf die Fähigkeiten der jeweils vorhandenen Organisationprozesse, den Geschäftszielen und den Bedürfnissen von Kunden, Endanwendern und Prozessimplementierern festgelegt. Die den Prozess durchführenden Leute sind direkt daran beteiligt, den Prozess quantitativ zu managen.
  - quantitatives Management wird auf der Gesamtheit von Prozessen durchgeführt, die ein Produkt oder einen Service hervorbringen.
  - die Subprozesse, die entscheidend zur Gesamtperformance des Prozesses beitragen, werden statistisch gemanagt. Für ausgewählte Subprozesse werden detaillierte Messungen der Prozessleistung gesammelt und statistisch analysiert.
  - Spezialfälle und Schwankungen im Prozessablauf werden identifiziert und angesprochen, um eine erneute Variation zu verhindern.
  - die Qualität und die Messungen der Prozessleistung werden schriftlich festgehalten, um zukünftige Entscheidungen zu unterstützen.

  Aktivitäten, die quantitativ die Leistung eines Prozesses managen, beinhalten folgende Punkte:

  - Identifiziere Subprozesse und bringe sie unter statistisches Management
  - Identifiziere und messe Produkt- und Prozesseigenschaften, die wichtig für die Prozessleistung und -qualität sind
  - Identifiziere und adressiere Spezialfälle und Prozessschwankungen
  - Manage jeden gewählten Subprozess mit dem Ziel, seine Leistung statistisch stabil und vorhersagbar bzgl. der Prozess- und Produkteigenschaften zu machen.

- Schätze die Fähigkeit des Prozesses ein, die vorgegebenen Qualitäts- und Leistungsziele zu erreichen.

- Leite angemessene korrektive Maßnahmen ein, wenn abzusehen ist, dass diese Ziele gegenwärtig nicht erreicht werden können. Diese korrektiven Maßnahmen können auch das Ändern dieser Ziele beinhalten.

Eine wichtige Unterscheidung zwischen einem definierten Prozess und einem quantitativ gemanagten Prozess ist die Vorhersagbarkeit der Prozessleistung. Der Ausdruck „quantitativ gemanagt“ impliziert, dass angemessene statistische und andere quantitative Techniken ergriffen werden, um die Leistung von einem oder mehreren kritischen Subprozessen zu managen und so die Prozessgesamtleistung besser vorherzusehen. Ein definierter Prozess liefert nur eine qualitative Vorhersagbarkeit. Das Erreichen des generischen Ziels auf Stufe 4 eines Prozessgebietes heißt, dass die Prozesse im jeweiligen Gebiet von nun an quantitativ gemanagte Prozesse sind.

- *Optimierender Prozess*

  Ein optimierender Prozess ist ein quantitativ gemanagter Prozess, der verändert und gegenwärtigen und geplanten Geschäftszielen angepasst wird. Ein optimierender Prozess fokussiert auf ein kontinuierliches Verbessern der Prozessleistung durch inkrementelle und innovative technische Verbesserungen. Verbesserungen, die Ursachen von Prozessschwankungen, Fehlern und anderen Problemen behandeln, werden identifiziert und bewertet. Jene, die messbar die Prozesse einer Organisation verbessern, werden übernommen. Diese Verbesserungen werden aufgrund ihres Beitrages zu den Zielen der Prozessoptimierung ausgewählt, wobei die dadurch entstehenden Kosten für eine Organisation abgewägt werden müssen. Es wird also ständig versucht, die Leistung der betreffenden Organisationsprozesse zu verbessern. Die Effekte der Prozessverbesserungen werden gemessen und begutachtet bzgl. der vorgegebenen Ziele. In einem optimierenden Prozess werden Prozessschwankungen bzgl. Leistung und Qualität damit behandelt, den Prozess so zu verändern, dass die Schwankungen vermindert oder eliminiert werden und die Prozessleistung steigt.

  Eine wichtige Unterscheidung zwischen einem quantitativ gemanagten Prozess und einem optimierenden Prozess ist der, dass ein optimierender Prozess kontinuierlich verbessert wird, indem gängige Ursachen von Prozessschwankungen behandelt werden. Ein quantitativ gemanagter Prozess behandelt spezielle Ursachen der Prozessschwankung und liefert statistische Vorhersagbarkeit der Ergebnisse. Obwohl der Prozess vorhersagbare Ergebnisse produzieren mag, können diese immer noch nicht ausreichend sein, um die Ziele der Prozessoptimierung zu erreichen. Das Erreichen des generischen Ziels auf Stufe 5 eines Prozessgebietes heißt, das die Prozesse im jeweiligen Gebiet von nun an optimierende Prozesse sind.

5 Modelldarstellungen

Um den Entwicklungspfad, den Organisationen bzw. ihre Prozesse durchlaufen, zu beschreiben, gibt es im CMMI Modell zwei Darstellungen: Die *kontinuierliche Darstellung* (Continuous Representation) und die *stufenförmige Darstellung* (Staged Representation). Diese beiden Darstellungen zeigen Organisationen zwei unterschiedliche Wege auf, ihre Produkte und Dienstleistungen durch prozessoptimierende Schritte zu verbessern. In diesem Abschnitt werden diese beiden Darstellungen näher vorgestellt und miteinander verglichen.

5.1 Kontinuierliche Darstellung

Die kontinuierliche Darstellung (Abb. 8) ist auf die Weiterentwicklung eines oder mehrerer Prozessgebiete ausgerichtet, die eine Organisation nach gewissen Kriterien (Hauptschwerpunkt der Organisation, Dringlichkeit, o.a.) selbst ausgewählt hat. Diese Darstellung ermöglicht es Organisationen, inkrementelle Prozessverbesserungen in ausgewählten Prozessgebieten vorzunehmen. Die Entwicklungsstufen dieser Verbesserungen werden in der kontinuierlichen Darstellung *Fähigkeitsgrade* (Capability Levels) genannt. Sie beschreiben Verbesserungen eines Prozesses von einem planlos durchgeführten Prozess bis zu einem Prozess, der quantitative Informationen nutzt, um sich selbst ständig zu verbessern hinsichtlich vorgegebener Ziele. Fähigkeitsgrade bilden das zentrale Thema der kontinuierlichen Darstellung. Ein Fähigkeitsgrad besteht aus den zugehörigen spezifischen und generische Zielen (siehe Abschnitt 3), die die Organisationsprozesse in einem betreffenden Prozessgebiet verbessern sollen. Wenn man die spezifischen und generischen Ziele eines Prozessgebiets erreicht, erlangt man die Vorteile der Prozessverbesserung. Es gibt 6 Fähigkeitsgrade, die mit steigender Verbesserung von 0 bis 5 durchnummeriert sind (siehe Abbildung 7):

Level	Fähigkeitsgrad
0	Unvollständig (Incomplete)
1	Durchgeführt (Performed)
2	Gemanagt (Managed)
3	Definiert (Defined)
4	Quantitativ gemanagt (Quantitativly Managed)
5	Optimierend (Optimizing)

Abbildung 7: Fähigkeitsgrade der kontinuierlichen Darstellung ([1])

Die Tatsache, dass die Fähigkeitsgrade dieselben Bezeichnungen haben wie die jeweiligen generischen Ziele, ist beabsichtigt, weil die Konzepte von Fähigkeitsgraden und generischen Zielen eng miteinander verwandt sind. Ein Prozess der Stufe 0 heißt unvollständig, weil er nicht oder nur teilweise durchgeführt wird. Sobald die spezifischen Ziele eines Prozessgebiets erreicht sind, was gleichbedeutend ist mit der Tatsache ist, dass der Prozess zum ersten mal vollzogen wurde, wird Level 1 erreicht; der Prozess ist nun ein durchgeführter Prozess (siehe Abschnitt 4). Mit dem dem Erreichen der

generischen Ziele auf Level 2 wird dieser Level abgeschlossen; der Prozess ist nun ein gemanagter Prozess usw. Das Erreichen dieser Fähigkeitsgrade ist auch ein wichtiger Indikator nach außen hin, wie gut die jeweilige Organisation ihre internen Prozesse verbessert bzw. verbessern kann.

Abbildung 8: Kontinuierliche Darstellung ([3])

5.2 Stufenförmige Darstellung

Die stufenförmige Darstellung (Abb. 10) zielt auf die Verbesserung von Prozessen einer fest vorgegebenen Gruppe von Prozessgebieten ab. Sie gibt Organisationen eine geprüfte Richtlinie mit an die Hand, wie sie, ohne spezielle Prozessgebiete selbst auswählen zu müssen, Prozessoptimierungen vornehmen können. In der stufenförmigen Darstellung spielen Reifegrade (Maturity Levels) eine zentrale Rolle. Ein Reifegrad besteht aus spezifischen und generischen Praktiken einer vorgegebenen Gruppe von Prozessgebieten. Diese Praktiken verbessern die Gesamtleistung einer Organisation. Ein Reifegrad der Stufe 2 wird erreicht, wenn alle zur Stufe 2 gehörenden Praktiken der jeweiligen Prozessgebiete erfolgreich vollzogen und somit die Ziele erreicht wurden. Ein Reifegrad einer Organisation spiegelt ihre Leistungsfähigkeit in einer bestimmten Disziplin wieder, wie z.B. der des Software Engineering. Mit dem Erreichen eines Reifegrades wird sichergestellt, dass gewisse Prozessgebiete auf einer gemeinsamen Stufe der Entwicklung stehen und dass für das Erreichen des nächsten Reifegradeseine stabile Grundlage vorhanden ist. Es gibt 5 Reifegrade, die jeweils den Weg für den nachfolgenden Reifegrad ebnen (siehe Abbildung 9):

Level	Reifegrad
1	Anfänglich (Initial)
2	Gemanagt (Managed)
3	Definiert (Defined)
4	Quantitativ gemanagt (Quantitaivly Managed)
5	Optimierend (Optimizing)

Abbildung 9: Reifegrade der stufenförmigen Darstellung ([2])

Reifegrade stellen das Entwicklungsstadium einer Organisation hinsichtlich interner Prozessabläufe dar. Sie sind immer noch das Hauptkriterium um einzuschätzen, wie leistungsfähig eine Organisation ist, was Produkte und deren Qualität betrifft.

- In Organisationen mit Reifegrad 1 ist das Abliefern eines Produktes oder einer Dienstleistung von individuellen Einzelleistungen abhängig. Sie bieten kein stabiles Umfeld für gerichtete Prozesse. Organisationen mit Reifegrad 1 haben eine Tendenz, sich überanzustrengen und Prozesse in Krisenzeiten abzubrechen, und sind häufig unfähig, Erfolge zu wiederholen.
- In Organisationen mit Reifegrad 2 laufen gemanagte Prozesse ab, also geplante, vollzogene, bewertete und kontrollierte Prozesse, die auch in Krisenzeiten nicht einbrechen. Zu jeder Zeit ist der aktuelle Status von zu entwickelnden Produkten einsehbar.
- Erreicht eine Organisation den Reifegrad 3, steht fest, dass sie ihre Prozesse angepasst, gut verstanden und in Dokumenten über Standards, Prozeduren, Werkzeugen und Methoden beschrieben hat. Diese Standards tragen zur Vereinheitlichung und Konsistenz in einer Organisation bei. Einzelprojekte schneidern ihre definierten Prozesse aus dem Standardset von Organisationsprozessen zurecht. Es ist also nicht wie bei den Organisationen der Stufe 2, dass für gleiche Projekte, die von unterschiedlichen Gruppen durchgeführt werden, zwei verschiedene Standardprozesse entstehen.
- Eine Organisation mit Reifegrad 4 benutzt quantitative Vorgaben bzgl. Qualitäts- und Prozessleistung, um damit die Prozesse zu managen. Für ausgewählte Subprozesse werden genaue Messungen der Prozessleistung gesammelt und analysiert, um zukünftige Prozessabläufe zu stabilisieren oder zu optimieren. Prozessergebnisse werden somit vorhersehbarer.
- Schafft es eine Organisation Reifegrad 5 zu erreichen, bedeutet dies, das sie kontinuierlich ihre Prozessleistung zu optimieren sucht, indem sie quantitative Methoden der Prozessoptimierung anwendet. Im Gegensatz zur Stufe 4 geschieht diese Optimierung aber auf einem globaleren Level und beinhaltet auch die Modifikation von Standardprozeduren einer Organisation falls nötig.

5.3 Darstellungswahl

Die kontinuierliche Dartstellung bietet eine flexiblere Möglichkeit zur Prozessverbesserung. Organisationen, die ihre Schwachstellen kennen, können mit dieser Darstellung die Leistung einzelner dort angesiedelter Prozesse verbessern. Außerdem haben sie mit dieser Darstellung die Möglichkeit, verschiedene Prozesse verschieden stark zu verbessern. So können sie z.B. in einem Prozessgebiet den Fähigkeitsgrad 4 anstreben, und es in einem anderen bei Fähigkeitsgrad 2 belassen. Man sollte dabei bedenken, dass ein extremes Voranschreiten in einem Prozessgebiet aufgrund gewisser Abhängigkeiten unter den Prozessgebieten nicht durchführbar ist. Die stufenförmige

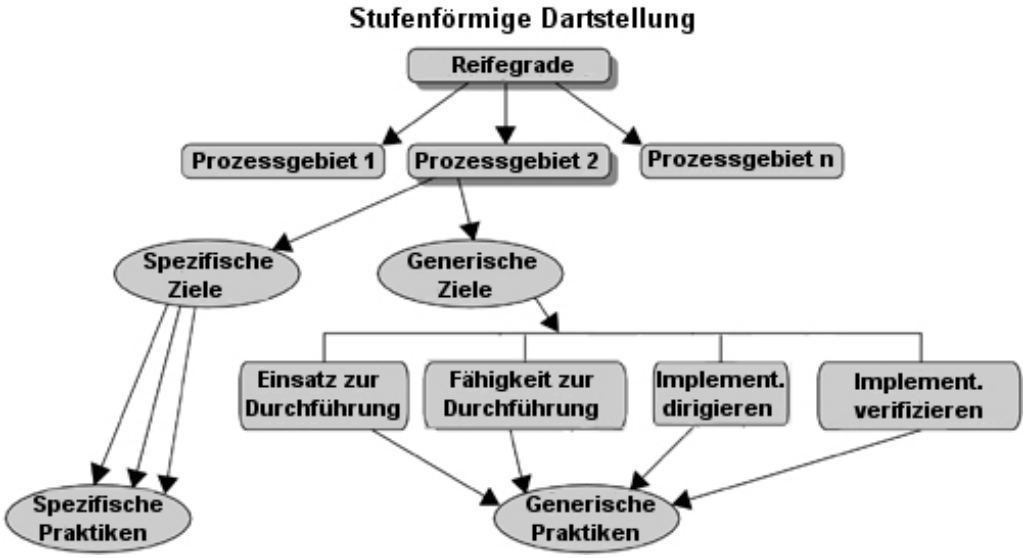

Abbildung 10: Stufenförmige Darstellung (aus [3])

Darstellung bietet eine strukturiertere Möglichkeit der Verbesserung. Mit dem Erreichen jedes Reifegrades wird sichergestellt, dass eine stabile Grundlage für das Erreichen des nächsten gegeben ist. Die stufenförmige Darstellung nimmt Organisationen mit an die Hand, sie müssen sich keine großen Gedanken darüber machen, welche Prozesse besondere Bedeutung erfahren müssen. Sie brauchen nur Verbesserungen in der fest vorgegebenen Gruppe von Prozessgebieten zu machen, um den nächsten Reifegrad zu erlangen. Organisationen, die bisher schon in einem Vorgängermodell mit der einen oder anderen Darstellung gearbeitet haben, sollten auch im CMMI Modell bei dieser Darstellung bleiben. Organisationen, die bisher nicht mit einem CMM Modell gearbeitet haben, aber ein tiefes Verständnis von den intern ablaufenden Prozesses und deren Zusammenhängen haben, sollten die kontinuierliche Darstellung wählen. Diese ist auch die erste Wahl für Organisationen, die genau wissen, welche Prozesse man verbessern muss, um ein bestimmtes Geschäftsziel zu erreichen. Für absolute Neulinge empfiehlt sich jedoch die stufenförmige Darstellung.

6 Das CMMI-Modell in der Praxis

Das CMMI-Modell reduziert die Kosten für die Bemühungen einer organisationsweiten Verbesserung, indem es eine einheitliche Terminologie, einheitliche Komponenten und einheitliche Bewertungs- und Trainingsmethoden benutzt, ohne dabei auf eine Disziplin beschränkt zu sein. Die Schwierigkeit liegt in der Anpassung und Interpretation des CMMI Modells an die jeweils schon vorhandenen Prozesse einer Organisation, die Prozessverbesserung auf bestimmten Gebieten sucht. Solche Organisationen benötigen professionelle Beratung, um CMMI Praktiken zu übernehmen. Um Praktiken zu interpretieren ist es wichtig, den Zusammenhang, in dem sie gebraucht werden,

zu erkennen und zu bestimmen, wie gut sie den Zielen eines Prozessgebiets gerecht werden. Organisationen, die schon CMMs benutzt haben, haben natürlich mit der Anpassung auf CMMI weniger Schwierigkeiten, da sie nur die Unterschiede zwischen den Modellen erkennen müssen. Das CMMI Modell deckt zum Beispiel im Gegensatz zum SW-CMM Modell viel besser die Gebiete von Konstruktions- und Risikomanagement, sowie Mess- und Analyseprozessen ab. Organisationen, die bereits Maturity Level 4 oder 5 in einem Modell erreicht haben, müssen den Zeitpunkt auf das CMMI Modell zu wechseln genau bestimmen, um dessen Vorteile zu erlangen. Organisationen, die neu auf dem Gebiet der Prozessverbesserung sind, sollten sich darüber im Klaren sein, dass der erste Schritt der Prozessverbesserung eine solide finanzielle Basis durch das Seniormanagement oder andere Sponsoren ist. Dann kann eine technisch kompetente Gruppe gebildet werden, die die Prozessverbesserung in den einzelnen Gebieten vorantreibt. Das so genannte *Modelltraining* ist ein Schlüsselelement für die Fähigkeit einer Organisation, dass CMMI Modell zu übernehmen. Das SEI bietet dafür eine Reihe von Kursen an, die einen grundlegenden Überblick über das CMMI Modell vermitteln sollen, aber auch Kurse für Fortgeschrittene, die sich mit CMMI Adoption, Bewertungen und der internen Mitarbeiterschulung befassen.

Eine weitere relevante Komponente neben der Interpretation des CMMI-Modells ist die *Modellbewertung* (Model Appraisal). Modellbewertungen dienen der Identifizierung von Verbesserungsmöglichkeiten und dem Einschätzen der Leistungsfähigkeit von Organisationen, indem Bewertungsteams nach fest vorgegebenen Methoden (Stichwort SCAMPI) Reifegrade zuordnen.

Die *Modellanpassung* (Model Tailoring) stellt einen Prozess dar, indem nur eine Untermenge des CMMI-Modells ausgewählt wird, um den Anforderungen einer speziellen Domäne oder Anwendung gerecht zu werden. Bei dieser Auswahl muss aber mit höchster Vorsicht vorgegangen werden, da sonst die Effektivität des CMMI-Modell leiden könnte. Die Anpassung kann sich dabei auf einzelne Disziplinen, Prozessgebiete, Reifegrade oder Fähigkeitsgrade beziehen, um bestimmten Zielen einer Organisation gerecht zu werden. Die Modellanpassung bezieht sich auch auf die Vergleichbarkeit von Prozessverbesserungen in verschiedenen Projekten. Wenn das eine Projekt gewisse Teile des Modells ausschließt, ein anderes Projekt wiederum andere, wird durch den Zuschnitt wieder eine Vergleichbarkeit zwischen diesen Projekten ermöglicht. Aber nicht nur organisationsinterne Vergleiche, sondern auch Vergleiche unter den einzelnen Organisationen ermöglicht die Modellanpassung. Eine Organisation sucht zum Beispiel eine Verbesserung nur in den Disziplinen Systems-Engineering und Lieferantenfindung, wohingegen sich eine andere Organisation nur mit dem Systems-Engineering befasst, da sie alle Teilprodukte selbst herstellt. Um diese Organisationen miteinander zu vergleichen, müssen die dabei verwendeten Modelle so angepasst werden, dass eine Bewertung in Form von Reifegraden ermöglicht wird.

7 Zusammenfassung

Mit dieser Ausarbeitung sollte das CMMI-Modell näher dargestellt werden. Die grundlegende Struktur des CMMI Modells wurde zuerst dargestellt. Es folgte eine detaillierte Beschreibung der drei Modellkomponenten bzw. deren Teilkomponenten. Es wurde aufgezeigt, wie Prozesse im CMMI Modell beschrieben und eingeordet werden, und welche Darstellungsformen es im CMMI Modell gibt. Im letzten Abschnitt wurde beleuchtet, wie das CMMI Modell in der Praxis angewendet und umgesetzt wird. Es ist abzusehen, dass das CMMI-Modell früher oder später seine Vorgänger ersetzen wird. Wie stark es sich gegen andere Modelle durchsetzen wird, hängt wohl davon ab, wie gut eine Organisation die Theorie hinter diesem Modell verstanden hat und wie schwierig und mit welchem finanziellen Aufwand sich diese Theorie dann in die Praxis umsetzen lässt. Für eine intensive Auseinandersetzung mit dem CMMI Modell, insbesondere mit den verschiedenen Prozessgebieten, auf die hier nur oberflächlich eingegangen werden konnte, verweise ich auf die folgenden Literaturangaben.

Literatur

[1] CMMI Product Team: *CMMI-SE/SW/IPPD/SS, V1.1 Staged Representation*, www.sei.cmu.edu/cmmi/models/, März 2002

[2] CMMI Product Team: *CMMI-SE/SW/IPPD/SS, V1.1 Continuous Representation*, www.sei.cmu.edu/cmmi/models/, März 2002

[3] M.B.Chrissis, M.Konrad, S.Shrum: *CMMI: Guidelines for Process Integration and Product Improvement*, Addison Wesley Professional, 2004

Capability Maturity Model Integration (CMMI) - Appraisals

Boris Bierbaum

Inhaltsverzeichnis

1 Einleitung

Hat eine Organisation, die Software entwickelt, den Wunsch, einen von ihr durchgeführten Prozess, z. B. einen Software-Entwicklungsprozess, weiterzuentwickeln, so kann sie dies mit Hilfe eines Prozessmodells, wie etwa einem CMMI-Modell, tun. Um festzustellen, wo die Stärken und Schwächen des Prozesses liegen und darauf aufbauend dann die Weiterentwicklung des Prozesses zu planen, kann die Organisation eine Bewertung des Prozesses gegen das Modell durchführen. Eine solche Bewertung zur Unterstützung von unternehmensinternen Entscheidungen nennt man *Assessment*.

Es gibt aber auch Gründe für eine Organisation, nicht einen eigenen Prozess, sondern einen Prozess, der in einer anderen Organisation abläuft, zu bewerten oder bewerten zu lassen. Beispiele hierfür sind etwa die Auswahl eines Unternehmens für die Durchführung eines zu vergebenden Auftrags, bei der man die Fähigkeiten der möglichen Auftragnehmer hinsichtlich der Beherrschung ihres Prozesses zugrundelegen möchte, oder die Überprüfung der Fähigkeiten eines Vetragspartners während der Vertragslaufzeit. Die Bewertung eines Prozesses im Auftrag von Leuten, die nicht der Organisation selbst angehören, heisst *Capability Evaluation*.

Beide Arten von Bewertungen, die sich ja möglicherweise in der Praxis lediglich durch die dahinterstehende Motivation, nicht aber durch die Methode ihrer Durchführung unterscheiden, nennt man *Appraisal*. Für die verschiedenen Maturity-Modelle, die CMMI historisch zugrundeliegen, z. B. SW-CMM und SE-CMM, gibt es definierte Appraisal-Methoden, um Bewertungen gegen das jeweilige Modell aufstellen zu können. Entsprechend dem Ansatz von CMMI, die verschiedenen Maturity-Modelle unter einem gemeinsamen, abstrakteren Modell zu vereinen, das dann auf die einzelnen Disziplinen angepasst werden kann, hat man sich am SEI auch über die Vereinheitlichung von Appraisal-Methoden auf der Basis von CMMI Gedanken gemacht. Die angestrebten Vorteile des integrierten Ansatzes von CMMI (bessere Zusammenarbeit der Prozesse verschiedener Disziplinen, Kostenersparnis etc.) sollen dadurch auch auf den Bereich der Appraisals übertragen werden.

Das Resultat dieser Arbeit am SEI sind insbesondere zwei Dokumente: Die Appraisal Requirements for CMMI (ARC), die Thema des Abschnits 2 dieser Ausarbeitung sind, und die Standard CMMI Appraisal Method for Process Improvement (SCAMPI), welche in Abschnitt 3 vorgestellt wird. In Abschnitt 4 schliesslich finden sich ein paar Daten zur Praxis von CMMI-Appraisals.

2 Appraisal Requirements for CMMI (ARC)

2.1 Ziele und Inhalt der ARC

Die Appraisal Requirements for CMMI (ARC) stellen eine Weiterentwicklung des CMM Appraisal Framework (CAF) dar. Dieses war gedacht als gemeinsame Basis

für Appraisal-Methoden basierend auf dem Capability Maturity Model for Software (SW-CMM). Die ARC definieren Anforderungen an Appraisal-Methoden, die für den Einsatz mit CMMI-Modellen entwickelt werden. Ein Beispiel für eine solche Methode wird in Abschnitt 3 vorgestellt. Entwicklern von CMMI-Appraisal-Methoden können die ARC als Definition für den Umfang der von ihnen zu leistenden Arbeit dienen. Die ARC legen ihre Anforderungen entlang einer Einteilung in 3 Klassen (A, B und C) fest, die jeweils einem typischen Anwendungsbereichen von Appraisals entsprechen.

2.2 Die Klasseneinteilung von CMMI-Appraisal-Methoden

Appraisals der Klasse A sind die einzigen, die Quantitative Bewertungen (engl.: *ratings*) produzieren. Solche Appraisals erfüllen alle Anforderungen, die von den ARC gestellt werden (Abschnitt 2.3 geht auf diese Anforderungen näher ein). Klasse B-Methoden erfüllen nur einen Teil der ARC-Anforderungen. In den ARC ist genau definiert, welche Anforderungen für eine Klasse B-Methode nicht verpflichtend sind. Appraisals, die die Anforderungen an eine Klasse B-Methode erfüllen, werden von Organisationen benutzt, die am Beginn der Einführung von CMMI stehen, oder für Zwischenbewertungen zwischen zwei Klasse A-Appraisals. Appraisals, die nach einer Klasse C-Methode durchgeführt werden, schließlich werden von Organisationen benutzt, um einen kostengünstigen, schnellen Einblick in einen Prozess zu erhalten. ([APP], S. 6)

Eigenschaft	**Klasse A**	**Klasse B**	**Klasse C**
Umfang der nötigen Datenquellen	Hoch	Mittel	Gering
Produziert Rating	Ja	Nein	Nein
Umfang der nötigen Ressourcen	Hoch	Mittel	Gering
Teamgröße	Hoch	Mittel	Gering
Anforderungen an Teamleiter	Lead Appraiser	Lead Appraiser oder trainierte und erfahrene Person	Trainierte und erfahrene Person

Abbildung 1: Eigenschaften der CMMI-Appraisal-Klassen

Die Eigenschaften der 3 Appraisal-Klassen sind in Abbildung 1 dargestellt. Sie unterscheiden sich insbesondere im Umfang der Datenquellen (engl.: *objective evidence*), die sie benutzen, in den Zeit- und Geldressourcen, die für ihre Durchführung benötigt werden, und der Frage, ob sie quantitative Bewertungen produzieren ([APP], S. 5). Während Methoden der Klasse A auf insgesamt drei verschiedenen Typen von Datenquellen basieren, nämlich Instrumenten (Fragebögen, Umfragen usw.), Interviews und Entwicklungsdokumenten, beruhen Klasse B-Methoden nur auf zwei dieser Datenquellen, darunter Interviews. Appraisals, die nach einer Klasse C-Methode durchgeführt werden, schließlich benutzen nur eine Datenquelle.

2.3 Anforderungen an CMMI-Appraisal-Methoden

Um den ARC zu genügen, muss eine CMMI-Appraisal-Methode Anforderungen in 7 verschiedenen Bereichen erfüllen ([APP], S. 7 ff):

1. *Responsibilities*: Die Methode muss die Verantwortlichkeiten für den Appraisal-Sponsor (d.h. denjenigen, der das Appraisal in Auftrag gibt) und den Teamleiter definieren. Dazu gehören einige Aktivitäten, beispielsweise die Sicherstellung der Verfügbarkeit der nötigen Ressourcen im Falle des Sponsors oder die Unterrichtung der Teilnehmer am Appraisal über Ziel, Umfang und Ansatz des Appraisals für den Teamleiter.

2. *Appraisal Method Documentation*: Die Appraisal-Methode muss ausreichend dokumentiert sein, z. B. muss sie Angaben darüber enthalten, welche ARC-Anforderungen sie erfüllt. Sie muss weiterhin Anleitungen zu verschiedenen Tätigkeiten enthalten. Darunter fällt etwa die Bestimmung der Größe des Appraisal-Teams, die Festlegung der Rollen der einzelnen Teammitglieder und die Art, wie die Quellen gesammelt und auf das zugrundegelegte CMMI-Modell bezogen werden können.

3. *Planning and Preparing for the Appraisal*: Die Methode muss die Vorbereitung der Teilnehmer auf das Appraisal regeln und definieren, welche Dinge notwendig sind, damit dass Appraisal beginnen kann, z. B. Geldmittel, Zeitpläne etc.

4. *Appraisal Data Collection*: Die Methode muss auf bestimmten Datenquellen basieren, je nach Klassenzugehörigkeit, s. Abschnitt 2.2

5. *Data Consolidation and Validation*: Die Methode muss für bestimmte Entscheidungen den Konsens aller Teammitglieder verlangen. Beispielsweise muss Einigkeit über die Richtigkeit der Qantitativen Bewertungen bestehen. Ausserdem müssen Angaben darüber vorhanden sein, wie bestimmt werden kann, ob zu einem zu überprüfenden Teil des Modells genügend Daten vorliegen.

6. *Rating*: Die Methode muss einen Prozess definieren, wie zu den Bewertungen gelangt werden kann und bestimmte Bedingungen an diesen Prozess müssen erfüllt sein.

7. *Reporting Results*: Die Methode muss Verfahren angeben, wie das Ergebnis des Appraisals präsentiert und weiter verwendet wird.

In den ARC sind detaillierte Angaben darüber zu finden, was in den eizelnen Bereichen jeweils gefordert wird. Bei jeder Teil-Anforderung findet sich zudem eine Angabe, welchen der drei Klassen sie zuzuordnen ist.

3 Standard CMMI Appraisal Method for Process Improvement (SCAMPI)

3.1 Einführung: Ziele und Anwendungsbereiche

Zusätzlich zur Definition der Anforderungen an CMMI-Appraisal-Methoden durch die ARC wurde am SEI auch eine vollständige Appraisal-Methode entwickelt. Sie trägt den Namen SCAMPI (Standard CMMI Appraisal Method for Process Improvement) und erfüllt die ARC-Anforderungen an eine Klasse A-Methode ([SCA], S. I-9). Sie stellt damit eine Referenzimplementation der ARC dar, die Entwicklern von Appraisal-Methoden für CMMI-Modelle als Orientierung dienen kann. Außerdem ist sie mittlerweile auch praktisch im Einsatz. In diesem Abschnitt wird SCAMPI vorgestellt, in Abschnitt 4 befinden sich einige Angaben zu Erfahrungen, die beim praktischen Einsatz von SCAMPI gemacht wurden.

Bei der Entwicklung von SCAMPI wurde eine Reihe von Zielen verfolgt. Diese sind:

- SCAMPI soll quantitative Bewertungen eines Prozesses bezüglich eines CMMI-Modells ermöglichen. Diese Bewertungen sollen so genau sein, dass sie einen Vergleich zwischen Organisationen hinsichtlich deren Stärken und Schwächen zulassen. Sie sollen außerdem so gut definiert sein, dass bei mehrfacher, unabhängiger Durchführung von SCAMPI unter vergleichbaren Bedingungen auch vergleichbare Resultate zustande kommen. Die Bewertungen können durch Capability Level/Maturity Level-Ratings ausgedrückt werden, je nachdem, ob dem Appraisal eine kontinuierliche oder eine stufenweise Repräsentation eines CMMI-Modells zugrundeliegt. ([SCA], S. I-16)
- SCAMPI soll für die verschiedenen Arten von Appraisals geeignet sein: Sowohl für interne Prozessbewertungen durch eine Organisation selbst als auch für Bewertungen durch organisationsexterne Personen zum Zwecke der Vertragspartnerauswahl oder Prozessüberwachung. ([SCA], S. I-15)
- Die Durchführung von SCAMPI soll mit einem vetretbaren Aufwand an Kosten und Zeit praktisch möglich sein. ([SCA], S. I-15)
- SCAMPI soll an die konkreten Umstände des Appraisals, wie etwa das verwendete CMMI-Modell und die vorgesehene Größe des Appraisal-Teams, anpassbar sein. ([SCA], S. I-12)

3.2 Prinzipien und Grundlagen

Um die angestrebten Ziele zu erreichen, werden in SCAMPI anerkannte Techniken aus dem Appraisal-Bereich verwendet. SCAMPI baut auf bereits vorhandenen Appraisal-Methoden für verschiedene Bereiche auf ([SCA], S. I-15). In diesem Abschnitt werden die grundlegenden Prinzipien und Verfahren, auf denen SCAMPI basiert, vorgestellt.

Dabei wird insbesondere daruf eingegangen, wie das Ergebnis des Appraisals erzeugt wird. Abschnitt 3.3 beschäftigt sich dann mit dem gesamten Ablauf von SCAMPI, inklusive Vor- und Nachbereitung, und der Art, wie dieser in [SCA] beschrieben ist.

SCAMPI wird durchgeführt von einem Appraisal-Team aus qualifizierten und erfahrenen Mitgliedern unter der Führung eines Teamleiters. Das Team identifiziert Stärken und Schwächen eines vorhandenen Prozesses und orientiert sich dabei an den im verwendeten CMMI-Modell aufgeführten Zielen (engl.: *goals*). Das Ergebnis des Appraisals basiert auf dem Ausmaß, zu dem diese Ziele nach Meinung des Appraisal-Teams erreicht werden. Um zu jedem Ziel den Erfüllungsgrad feststellen zu können, muss das Team ermitteln, inwieweit die diesem Ziel zugeordneten Praktiken (engl.: *practices*) im Prozess tatsächlich implementiert sind. Die Untersuchung bedient sich dabei 4 verschiedener Arten von Datenquellen ([SCA], S. I-23):

- *Instrumente* (engl.: *instruments*): Instrumente sind schriftliche Unterlagen, die von der bewerteten Organisation erstellt wurden, um das Appraisal-Team über den verwendeten Prozess und dessen Beziehung zum CMMI-Modell zu informieren, z. B. Abbildungen des organisationseigenen Prozessmodells auf das CMMI-Modell.
- *Präsentationen* (engl.: *presentations*): Diese werden von Mitarbeitern der untersuchten Organisation durchgeführt, um Mitglieder des Appraisal-Teams über Aspekte des verwendeten Prozesses oder die Art der Umsetzung von Teilen des CMMI-Modells zu informieren, beispielsweise durch die Vorführung verwendeter Werkzeuge.
- *(Entwicklungs-)Dokumente* (engl.: *documents*): Dokumente sind Unterlagen in schriftlicher oder elektronischer Form, die die Durchführung einer oder mehrerer Praktiken belegen, etwa die Festlegung bestimmter Verfahrensweisen. Im Unterschied zu Instrumenten resultieren sie aus der Arbeit der bewerteten Organisation und sind nicht direkt an das Appraisal-Team als Leserschaft gerichtet.
- *Interviews* (engl.: *interviews*): Die Mitglieder des Appraisal-Teams können in Interviews Personen aus der untersuchten Organisation direkt befragen, sowohl einzeln als auch in Gruppen.

Die Datenquellen werden herangezogen, um zu jeder Praktik aus dem CMMI-Modell zu bestimmen, inwieweit diese durch die bewertete Organisation tatsächlich angewandt wird. Die Quellen werden vor Ort durch das Appraisal-Team einem mehrstufigen Verarbeitungsprozess unterzogen, um zu einer Bewertung zu gelangen. Dieser Prozess ist in Abbildung 2 schematisch dargestellt.

Auf der ersten Stufe wird die Umsetzung einer bestimmten Praktik in den einzelnen Projekten einer Organisation untersucht. Dies kann jeweils durch das gesamte Team oder aber für verschiedene Projekte parallel durch einen zugeordneten Teil des Teams („Mini-Team“) geschehen. Als erstes werden aus der Fülle der Datenquellen diejenigen identifiziert, die die Durchführung der zu verifizierenden Praktik belegen, diese

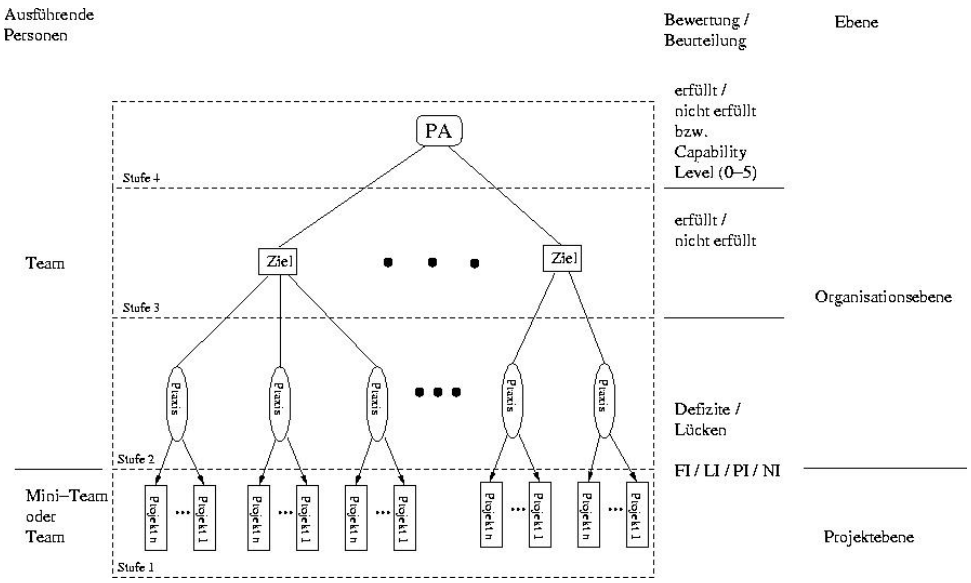

Abbildung 2: Struktur der SCAMPI-Beurteilung

nennt man *Practice Implementation Indicators, PIIs*. Sie werden in die folgenden Kategorien eingeteilt ([SCA], S. I-25):

- *Direktes Artefakt* (engl.: *direct artifact*): angestrebtes Resultat der Durchführung einer bestimmten Praktik und als solches möglicherweise auch im CMMI-Modell erwähnt, z. B. ein auslieferungsbereites Produkt
- *Indirektes Artefakt* (engl.: *indirect artifact*): Belege dafür, dass eine bestimmte Praktik durchgeführt wurde, z. B. Sitzungsprotokolle
- *Bestätigung* (engl.: *affirmation*): zusätzliche Versicherung, dass eine Praktik durchgeführt wurde, z. B. die Aussage eines Mitarbeiters während eines Interviews

Die Idee hinter den PIIs ist, dass die Durchführung einer Praktik in einer Organisation „Spuren“ hinterlässt. Das ist im Falle von direkten Artefakten unmittelbar klar, da sie ja als gewünschtes Ergebnis der Praktik erstellt werden. Aber auch während der Durchführung einer Praktik entstehen Spuren, z B. indem in regelmäßigen Abständen Statusberichte angefertigt werden oder Sitzungen abgehalten werden, deren Protokolle dann vorliegen. Solche PIIs heißen indirekte Artefakte. Die Bestätigungen entstehen als direkte Aussage eines Mitglieds der bewerteten Organisation über die durchgeführte Praktik, etwa während eines Interviews oder bei einer Präsentation.

Die Begutachtung einer Praktik in einem Projekt basiert auf den PIIs, die dem zuständigen Mini-Team zu diesem Thema zur Verfügung stehen. Dabei ist darauf zu achten, dass alle drei PII-Kategorien angemessen berücksichtigt werden müssen, um ein ausgewogenes Bild zu bekommen ([SCA], S. I-26). Insgesamt können während eines SCAMPI-Appraisals erhebliche Mengen an Quellen anfallen. Für eine effiziente Durchführung des Appraisals sind daher geeignete Methoden zur Datensammlung

und Datenverwaltung unverzichtbar. Die Prinzipien, denen diese Methoden genügen müssen, sind in SCAMPI unter dem Terminus „Focused Investigation“ beschrieben ([SCA], S. I-23 f) und sind die folgenden:

- Es soll klar sein, welche Quellen verfügbar sind und auf welche Weise jede einzelne zum Verständnis der Praktiken in der Organisation beiträgt.
- Die Dokumentation der Praktiken durch die Quellen soll beständig verbessert werden, bis sie ausreicht, um ein Urteil zu fällen.
- Die Suche nach weiteren Quellen soll auf die Bereiche konzentriert werden, in denen die Quellenlage noch nicht ausreichend ist.
- Unnötiger oder mehrfacher Aufwand, der nicht wesentlich zum besseren Verständnis beiträgt, soll vermieden werden.

Das Mini-Team untersucht zunächst die Durchführung der Praktik in den einzelnen Projekten unter Zuhilfenahme der PIIs. Die dabei gemachten Beobachtungen werden als Beschreibung der Stärken und Schwächen schriftlich niedergelegt. Dabei wird vor allem auf die Identifikation der Defizite im Vergleich zum CMMI-Modell geachtet. Stärken werden nur dann erwähnt, wenn sie zur Umsetzung der Praktik in besonderer Weise beitragen. Möglicherweise werden auch Beobachtungen festgehalten, die sich auf Praktiken beziehen, die als gleichwertige Alternativen zu den Praktiken aus dem CMMI-Modell gelten können ([SCA], S. I-27). Das Mini-Team setzt seine Tätigkeit fort, bis Konsens darüber besteht, dass zur Beurteilung der Umsetzung der Praktik genügend PIIs gesichtet wurden und die dokumentierten Beobachtungen die identifizierten Stärken und Schwächen hinreichend genau reflektieren. Zum Abschluss seiner Arbeit vergibt das Mini-Team eine Bewertung für den Grad der Umsetzung der untersuchten Praktik in den Projekten, die aus einer von 4 Stufen besteht:

- *Vollständig Implementiert* (engl.: *Fully Implemented, FI*)
- *Überwiegend Implementiert* (engl.: *Largely Implemented, LI*
- *Teilweise Implementiert* (engl.: *Partially Implemented, PI*)
- *Nicht Implementiert* (engl.: *Not Implemented, NI*).

Auf der zweiten Stufe des Verarbeitungsprozesses werden die Ergebnisse der Mini-Teams zusammengeführt, um die Durchführung einer bestimmten Praktik für die gesamte Organisation zu beurteilen. Die Verwirklichung der Praktik innerhalb der Organisation wird mittels Konsens-Beschluss des Appraisal-Teams mit einer der obigen 4 Stufen bewertet. Ebenso werden die Beobachtungen der Mini-Teams zusammengefasst, so dass sie für die Organisation Gültigkeit haben. Wird auf einer der beiden Stufen festgestellt, dass die Beurteilung einer Praktik auf der Basis des bisher gesichteten Quellenmaterials nicht möglich ist, so wird zusätzliches Material gesammelt und der Zeitplan entsprechend angepasst.

Aus den Bewertungen der Implementation der Praktiken innerhalb der gesamten Organisation werden dann auf der dritten Stufe Bewertungen für die zugehörigen Ziele erzeugt. Auf der vierten Stufe können dann Bewertungen für Process Areas und Maturity Level/Capability Level-Ratings erstellt werden.

SCAMPI basiert auf der Erwartung, dass die Datenquellen, soweit möglich, schon gesammelt und vorbereitet werden, bevor das Team mit seinen Untersuchungen vor Ort beginnt. Diese Tätigkeit obliegt demnach der bewerteten Organisation ([SCA], S. I-11). Dabei handelt es sich um das Sammeln von Dokumenten, die Vorbereitung von Präsentationen und die Erstellung von Instrumenten. Insbesondere letztere werden als sehr wichtig angesehen, um dem Appraisal-Team einen möglichst tiefen Einblick in die Umsetzung des Prozessmodells in der Organisation zu geben und die Effizienz des Appraisals insgesamt zu erhöhen. Da das Appraisal-Team seine Arbeit entlang der im CMMI-Modell erwähnten Praktiken strukturiert, sollten sich die Instrumente möglichst auf die Umsetzung der Pratiken in jedem zu untersuchenden Projekt beziehen ([SCA], S. I-30).

Um festzustellen, ob die vorhandenen Quellen als Basis für ein Appraisal ausreichen, werden sie gesichtet, bevor das Appraisal-Team seine Tätigkeit innerhalb der Organsation aufnimmt. Sind die Quellen unvollständig, muss der Zeitplan für das Appraisal entsprechend angepasst werden, um dem Team zusätzliche Zeit, in der es selbst die Datensammlung vervollständigen kann, einzuräumen. ([SCA], S. I-22 f)

3.3 SCAMPI-Ablauf

3.3.1 Überblick

Der Ablauf eines SCAMPI-Appraisals ist in 3 Phasen eingeteilt, von denen jede wiederum aus mehreren Prozessen zusammengesetzt ist. Dies ist in Abbildung 3 dargestellt (s. [SCA], S. I-35 ff).

In der ersten Phase muss zunächst mit dem Sponsor Einigkeit über die Ziele des Appraisals erzielt werden. Es wird festgelegt, welche Projekte in das Appraisal einbezogen werden und welche Teile des CMMI-Modells (Process Areas) als Basis für die Bewertung dienen sollen. Der Appraisal-Plan wird erstellt und Schätzungen bezüglich Zeit und Kosten werden vorgenommen. Währenddessen werden in der zu bewertenden Organisation bereits die Quellen zusammengestellt. Eine Sichtung des Quellenmaterials kann dann als Basis für Anpassungen der Planungen und Schätzungen dienen. ([SCA], S. I-10 f)

Die zweite Phase dient dann der weiteren Sammlung, Sichtung und Verarbeitung der Daten, wie sie in Abschnitt 3.2 bereits beschrieben wurde, bis ein Gesamtresultat erzielt wird. Während der Phase 2 werden die Planungen und Schätzungen aus Phase 1 ständigen Anpassungen unterzogen, um den zunehmenden Erkenntnisstand zu reflektieren. ([SCA], S. I-11 f)

Phase	Process
1 Plan and Prepare for Appraisal	1.1 Analyze Requirements 1.2 Develop Appraisal Plan 1.3 Select and Prepare Team 1.4 Obtain and Analyze Initial Objective Evidence 1.5 Prepare for Collection of Objective Evidence
2 Conduct Appraisal	2.1 Examine Objective Evidence 2.2 Verify and Validate Objective Evidence 2.3 Document Objective Evidence 2.4 Generate Appraisal Results
3 Report Results	3.1 Deliver Appraisal Results 3.2 Package and Archive Appraisal Assets

Abbildung 3: Die Phasen von SCAMPI

Während der dritten Phase werden die Resultate des Appraisals dem Sponsor und der untersuchten Organisation vorgestellt. Der Umfang der Resultate und die Einhaltung eventueller Geheimhaltungsvorschriften werden in Phase 1 festgelegt.

Als Gesamtzeit für ein SCAMPI-Appraisal, inklusive Planung, Vorbereitung und Durchführung, sind etwa 3 Monate anzusetzen.

Die in Abbildung 3 erwähnten Prozesse setzen sich ihrerseits zusammen aus mehreren Aktivitäten. Abbildung 4 zeigt beispielhaft die Aktivitäten, aus denen Prozess 2.4 (*Generate Appraisal Results*) besteht. Alle Prozesse und Aktivitäten sind in [SCA] ausführlich und formal konsistent beschrieben, eine vollständige Darstellung übersteigt aber den Rahmen dieser Ausarbeitung. Zur Illustration der Form der SCAMPI-Prozessbeschreibungen wird deshalb im nächsten Abschnitt auf den Prozess *Generate Appraisal Results* näher eingegangen.

3.3.2 Beispiel: Prozess 2.4 (*Generate Appraisal Results)*

Die Beschreibungen der Prozesse und der Aktivitäten, aus denen sie bestehen, sind in [SCA] der Form nach für alle Prozesse und Aktivitäten gleich. Im Folgenden werden die einzelnen Teile der Beschreibungen der Reihe nach vorgestellt und anhand des Prozesses *Generate Appraisal Results*, durch den das Endresultat des Appraisals erzeugt wird, beschrieben. Eine Prozessbeschreibung besteht aus den folgenden Punkten ([SCA], S. II-111 ff):

- *Purpose*: Zusammenfassung der innerhalb des Prozesses zu erledigenden Aufgaben. Im Falle von *Generate Appraisal Results* ist dies die Aufstellung von Bewertungen für den Erfüllungsgrad der Ziele und darauf aufbauend von Capability Level/Maturity Level-Ratings, basierend auf den PIIs.

Aktivität	**Beschreibung**
2.4.1 Derive Findings and Rate Goals	Festlegung der endgültigen Befunde und der Bewertung bzgl. jedes untersuchten Ziels aus dem CMMI-Modell
2.4.2a Determine Process Area Capability Level	Festlegung des erreichten Capability Levels für jede untersuchte Process Area (kontinuierliches Modell)
2.4.2b Determine Satisfaction of Process Areas	Festlegung der erreichten Process Areas (Stufenmodell)
2.4.3a Determine Capability Profile	Aufstellung eines Capability-Profils, sofern gewünscht (kontinuuierliches Modell)
2.4.3b Determine Maturity Level	Festlegung des erreichten Capability Maturity Levels, sofern gewünscht (Stufenmodell)
2.4.4 Document Appraisal Results	Schriftliche Dokumentation der endgültigen Befunde und der Bewertung

Abbildung 4: Der Prozess 2.4: *Generate Appraisal Results*

- *Entry Criteria*: Bedingungen, die erfüllt sein müssen, damit mit diesem Prozess begonnen werden kann. Hier: Zu den überprüften Praktiken müssen vorläufige Beobachtungen verfügbar sein, insbesondere Angaben über Lücken in ihrer Umsetzung. Das Team muss darin übereinstimmen, dass die gesammelten Daten und Beobachtungen ausreichen, um zu einer Beurteilung zu kommen.

- *Inputs*: Daten, auf deren Basis dieser Prozess arbeitet. Hier: die vorläufigen Beobachtungen, zu jeder Praktik, Angaben, durch welche PIIs sie dokumentiert wird, und Unterlagen, die während der Auswertung der Quellen angefallen sind, z. B. Checklisten

- *Activities*: Liste der Aktivitäten, aus denen dieser Prozess besteht. Hier: s. Abbildung 4

- *Outputs*: Daten, die aus der Durchführung dieses Prozesses resultieren. Hier: endgültige Beobachtungen und quantitative Bewertungen

- *Outcome*: Beschreibung des Ziels der Durchführung des Prozesses, d. h. der Situation die erreicht ist, wenn der Output vorliegt. Hier: Ein Bewertung wurde erstellt für jeden Teil des zugrundegelegten CMMI-Modells, für den dies geschehen sollte und für den ausreichend Daten vorlagen, um dies tun zu können.

- *Exit Criteria*: Bedingungen, die erfüllt sein müssen, damit der Prozess beendet werden kann. Hier: Bewertungen zu allen Teilen des CMMI-Modells, die bewertet werden sollten, liegen vor

- *Key Points*: Hinweise, was besonders wichtig für eine erfolgreiche Durchführung des Prozesses ist und welche Probleme erfahrungsgemäß dabei auftreten können. Hier: Wichtig ist, dass die Bewertung des Ausmaßes, zu dem ein bestimmtes Ziel erreicht wurde, der Beurteilung des Umfangs, zu dem die diesem Ziel zugeordneten Praktiken durchgeführt werden, entpricht. Um dies zu erreichen,

müssen die Mitglieder des Appraisal-Teams zu objektiven Urteilen fähig sein. Erfahrungsgemäß kannn in einer solchen Beurteilungssituation viel Stress auftreten und der Teamleiter muss in der Lage sein, damit umzugehen.

- *Tools and Techniques*: Hinweise auf Werkzeuge, um den Prozess zu unterstützen, und auf Vorgehensweisen, die sich bei der Durchführung des Prozesses bewährt haben. Hier: Aufgrund der anfallenden großen Datenmengen sind in *Generate Appraisal Results* Werkzeuge, die den Entscheidungsfindungsprozess zu strukturieren helfen, sehr nützlich. Im Falle unterschiedlicher Meinungen innerhalb des Teams muss der Teamleiter darauf achten, dass die Diskussion sich auf die wesentlichen Punkte konzentriert. Eine angemessene Führung ist bei der Entscheidungsfindung besonders wichtig.

- *Metrics*: quantitative Größen, die während des Prozesses ermittelt werden. Hier: geplanter und tatsächlicher Zeitaufwand für die Bewertung jeder Modellkomponente sowie die Anzahl der Komponenten, die als erfüllt oder nicht erfüllt bewertet wurden

- *Verification and Validation*: Maßnahmen, die zu treffen sind oder getroffen werden können, um sicherzustellen, dass der Prozess korrekt durchgeführt wird und dass die Ergebnisse des Prozesses den Anforderungen genügen. Hier: Auf der Basis des Appraisal-Plans kann der Teamleiter überprüfen, ob die Bewertungen entsprechend den Vorgaben gewonnen wurden. Die Gründe für die Bewertungen sollten so festgehalten werden, dass sie später nachvollzogen werden können.

- *Records*: Dokumente, die während des Prozesses erstellt werden. Hier: Aufzeichnung der Bewertungsentscheidungen

- *Tailoring*: Hinweise darauf, inwieweit Anpassungen von SCAMPI an das verwendete CMMI-Modell, an spezielle Umstände des Appraisals usw. sich auf diesen Prozess auswirken. Hier: die Erzeugung von Capability Level/Maturity Level-Ratings ist optional, die Bewertung einzelner zusätzlicher Praktiken ist möglich

- *Interfaces with Other Processes*: Schilderung, wie andere Prozesse innerhalb von SCAMPI zu der Arbeit während dieses Prozesses beitragen. *Generate Appraisal Results* hängt von allen vorhergehenden Prozessen direkt ab. Beipielsweise werden durch Prozess 1.1 (*Analyze Requirements*) die Regeln für die Bewertung festgelegt und in Prozess 2.2 (*Verify and Validate Objective Evidence*) wird das Ausmaß ermittelt, zu dem die Praktiken aus dem Modell in der Organisation implementiert sind.

- *Summary of Activities*: Übersicht über den Inhalt der Aktivitäten, die zu diesem Prozess gehören. Hier: Es werden Urteile über den Erfüllungsgrad für jedes Ziel aus dem CMMI-Modell gefällt, das durch das Appraisal zu betrachten ist. Darauf aufbauend können dann, sofern dies gewünscht wird, quantitative Bewertungen ermittelt werden. Dies ist zunächst ein Capability Level Rating für jede zu bewertende Process Area im Falle eines kontinuierlichen Modells bzw. ein Urteil, ob die Process Area erfüllt wird oder nicht für ein Stufenmodell. Im

kontinuierlichen Fall kann aus den Bewertungen für jede Process Area dann ein Profil erstellt werden. Im Falle eines Stufenmodells kann ein Maturity Level ermittelt werden.

Der allgemeinen Beschreibung des Prozesses mit Hilfe der oben angegebenen Punkte folgt dann die Beschreibung der einzelnen Aktivitäten des Prozesses. Anhand der Aktivität 2.4.1 (*Derive Findings and Rate Goals*) wird diese Beschreibung nun beispielhaft dargestellt ([SCA], S. II-114 f):

- *Activity Description*: Beschreibung von Ziel und Inhalt der Aktivität. Hier: Die Entscheidung über den Erfüllungsgrad eines Ziels basiert auf den vorläufigen Befunden, in denen sich insbesondere Angaben über Lücken in der Implementation der zugehörigen Praktik finden. Anhand dieser Befunde muss das Appraisal-Team entscheiden, inwieweit die angestrebten Ziele erreicht wurden.
- *Required Practices*: Tätigkeiten, die während dieser Aktivität durchgeführt werden müssen. Hier sind dies der Reihe nach:
  1. Aufstellung endgültiger Befunde auf der Basis der vorläufigen Befunde und zusätzlicher Erkenntnisse und Daten, die durch die Überprüfung der vorläufigen Befunde gewonnen wurden
  2. Für jedes zu bewertende Ziel Entscheidung darüber, ob es erreicht wurde oder nicht
  3. Über die Beschreibung der Implementation der Praktiken, die endgültigen Befunde und die Bewertungen zu jedem Ziel muss im Team Konsens erreicht werden
- *Parameters and Limits*: Quantifizierungen und Präzisierungen dazu, wie die Aktivität durchzuführen ist. Hier: Ziel der Aktivität ist die Erstellung von Befunden darüber, welche Lücken in der Implementation aller untersuchten Praktiken gefunden wurden, und zwar auf der Ebene der bewerteten Organisation. Auf der Basis der Befunde für alle zu einem Ziel gehörenden Praktiken wird dann für alle betrachteten Ziele entschieden, ob es erreicht wurde oder nicht. Wenn nämlich keine Lücken vorliegen, wurde das Ziel erreicht, ebenso, wenn alle zugehörigen Praktiken entweder als überwiegend (LI) oder als vollständig (FI) implementiert gewertet werden.
- *Optional Practices*: Tätigkeiten, die während der Aktivität zusätzlich durchgeführt werden können. Hier: Falls der Appraisal-Sponsor es wünscht, können die Befunde und Bewertungen auch auf der Basis einzelner Praktiken erstellt werden, nicht nur auf der Basis der Ziele. Dadurch wird natürlich das Ergebnis des Appraisals detaillierter.
- *Implementation Guidance*: Hinweise darauf, was bei der Durchführung dieser Aktivität besonders wichtig ist. Hier: Entscheidend ist hier, das das endgültige Ergebnis für jedes Ziel (erreicht/nicht erreicht) auf dem Urteil über die Implementation der zu diesem Ziel gehörenden Praktiken basiert. Das Ergebnis muss so dokumentiert sein, dass dieser Zusammenhang später nachvollziehbar ist.

4 Praktische Bedeutung

Nach dem Erscheinen von SCAMPI V 1.1 im April 2002 wurden bis zum Juni 2003 100 Appraisals nach dieser Methode durchgeführt, deren Ergebnisse an das SEI berichtet wurden. In [PMP] finden sich Angaben zur Struktur dieser Ergebnisse, auf die in diesem Abschnitt kurz eingegangen werden soll.

Die 100 Appraisals fanden in 93 verschiedenen Organisationen statt, von denen 54% ausserhalb der USA ansässig waren, übrigens keine davon in Deutschland. Insgesamt wurden 357 Projekte betrachtet. Die Hälfte der bewerteten Organisationen hatte mindestens 200 Mitarbeiter. 47,3% der Organisationen waren Unternehmen, die die Appraisals durchgeführt haben, um ihre eigenen Prozesse zu bewerten, 45,1% der Organisationen wurden von einer Behörde zum Zwecke der Vetragspartnerauswahl begutachtet.

Der überwiegende Teil der Appraisals fand in den Bereichen Software Engineering und System Engineering statt: knapp 45% für beide Bereiche gemeinsam, ca. 25% nur für Software Engineering und etwa 10 % nur für System Engineering. In rund 75% der Appraisals wurde ein CMMI-Stufenmodell zugrundegelegt, in den anderen Fällen entsprechend eine kontinuierliche Repräsentation.

Abbildung 5 zeigt das Profil der bewerteten Organisationen, allerdings nur für einen Ausschnitt der Teilnehmer. Auffällig ist dabei sicherlich die hohe Zahl von Organisationen mit einem Maturity Level von 3, vor allem in den USA, verglichen mit relativ wenigen Oranisationen auf Level 4. Im Vergleich zu den ersten Ergebnissen nach der Einführung von SW-CMM, zeigen die CMMI-Profile einen höheren Reifegrad der Prozesse.

Hinweise auf die Existenz anderer CMMI-Appraisal-Methoden als SCAMPI sind dem Autor nicht bekannt, so dass sich die Darstellung hierauf beschränkt.

5 Zusammenfassung

In dieser Ausarbeitung wurde ein kurzer Überblick über Appraisal-Methoden im Rahmen von CMMI gegeben. Dabei wurden in Abschnitt 2 zunächst die Appraisal Requirements for CMMI (ARC) als Definition der Anforderungen an CMMI-Appraisal-Methoden vorgestellt. Sodann wurde in Abschnitt 3 mit der Standard CMMI Appraisal Method for Process Improvement (SCAMPI) eine Referenzimplementation der ARC und damit ein Beispiel für eine praktisch benutzbare CMMI-Appraisal-Methode besprochen. Abschnitt 4 enthält einen Einblick in den Stand der Praxis von Appraisals auf der Basis von CMMI.

Da CMMI vom SEI als Nachfolger der verschiedenen Maturity-Modelle, auf denen es basiert, entworfen wurde, ist damit zu rechnen, dass diese in der Praxis zunehmend durch CMMI ersetzt werden. Dies wird sich entsprechend natürlich auch auf

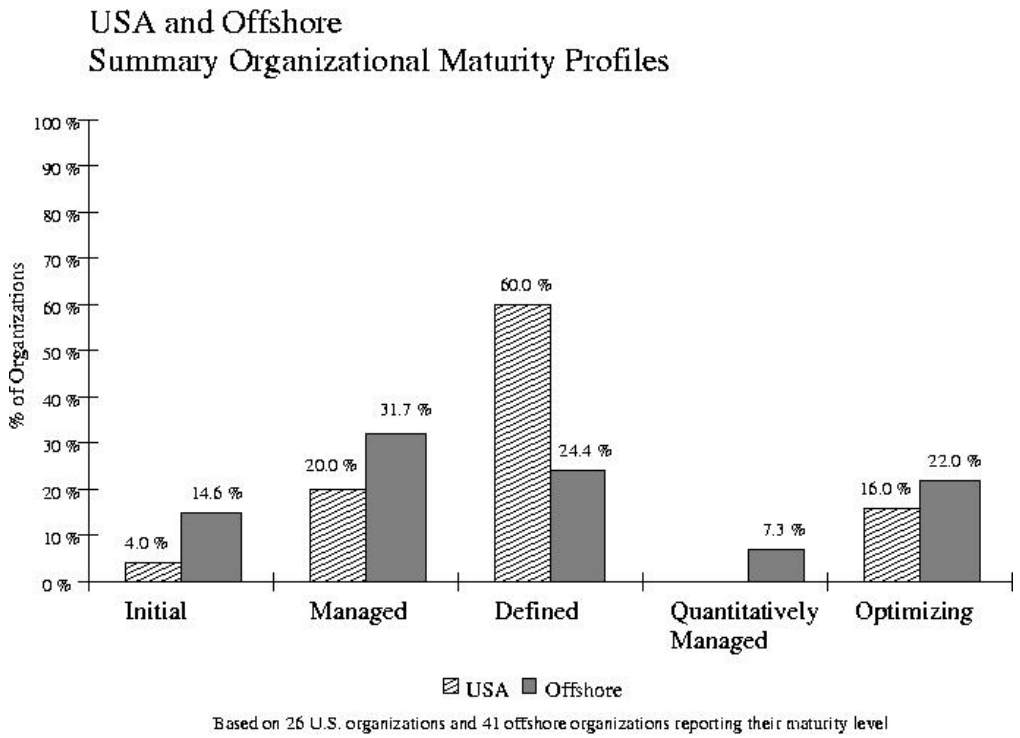

Abbildung 5: Ergebnis der bisherigen SCAMPI-Appraisals

den Bereich der Appraisals auswirken, d. h. dass etwa die Zahl der Appraisals auf der Basis von SW-CMM zurückgehen dürfte, zugunsten einer steigenden Zahl von CMMI-Appraisals. Interessant wird es sein, zu sehen, ob es zur Entwicklung weiterer ARC-konformer Appraisal-Methoden neben SCAMPI kommen wird.

Literatur

[APP] Appraisal requirements for cmmi, version 1.1 (arc, v 1.1). http://www.sei.cmu.edu/publications/documents/01.reports/01tr034.html.

[PMP] Process maturity profile, cmmi v 1.1, scampi v 1.1 appraisal results, first look. http://www.sei.cmu.edu/sema/profile.html.

[SCA] Standard cmmi appraisal method for process improvement (scampi), version 1.1: Method definition document. http://www.sei.cmu.edu/publications/documents/01.reports/01hb001.html.

Qualitätssicherung und Projektmanagement
Rational Unified Process (RUP)/
Unified Process (UP)

Holger Grosse-Plankermann

Inhaltsverzeichnis

1 Einleitung

1.1 Warum Software-Entwicklungsprozesse?

Von vielen Entwicklern und Entwicklungsabteilungen wird die Ansicht vertreten, dass die Entwicklung von Software um einige wenige hochqualifizierte Mitarbeiter herum organisiert werden soll. Es herrscht die Meinung, dass der Entwicklungsprozess genau auf diese zugeschnitten sein soll, da sie genau wissen, was zu tun ist, und daher kaum von der Entwicklungsorganisation angeleitet werden müssen. Dies ist größtenteils ein Irrglaube und wird in den seltensten Fällen zu einem guten Ergebnis führen. Da die Informatik, und insbesondere organisierte Softwareentwicklung sehr jung ist, brauchen (auch hochqualifizierte) Softwareentwickler Unterstützung im Organisationsbereich.
Ein guter Prozess zur Unterstützung von Softwareentwicklern sollte folgende Punkte berücksichtigen:

- Er soll die Aktivitäten eines Entwicklungsteams in geordnete Bahnen bringen.
- Die Aufgaben von einzelnen Mitgliedern, sowie des kompletten Teams sind zu koordinieren.
- Die zu entwickelnden Daten (Artefakte) sollen spezifiziert werden.
- Er soll Kriterien definieren, um den Projektfortschritt messen und bewerten zu können.

Es gibt diverse Ansätze, Softwareentwicklungsprozesse zu organisieren. Ein Beispiel dafür ist das Wasserfallmodell. Ein anderer, iterativ angelegter Prozess, ist der RUP/UP,

der im folgenden beschrieben wird.

1.2 Warum UP?

Der Unified Process setzt nun dort an, wo Defizite im Wasserfallansatz gesehen werden. Er versucht die Risiken zu verringern, die bei diesem Prozess auftreten, in dem ein iterativer Ansatz verwendet wird. Falls bei Benutzung des Wasserfallansatzes, ein Fehler in den Anforderungen erst am Ende der Implementierung entdeckt wird, so hat man unter Umständen viel Zeit und Geld vergeudet. Durch den iterativen Charakter des UP/RUP werden schon in frühen Iterationen kritische Aspekte angegangen und man hat Möglichkeiten angemessen zu reagieren. Der Unified Process ist nicht nur ein Software-Entwicklungsprozess, er ist ein Framework für Entwicklungsprozesse, der an eigene Bedürfnisse angepasst werden kann. Mit diesen Grundgedanken will der UP helfen, besser auf die Nutzer angepasste Software zu erstellen, er will bei der Entwicklung immer komplexer werdender Software unterstützen. Darüber hinaus soll die Entwicklung auch schneller als mit einem gewöhnlichen Entwicklungsprozess geschehen.

1.3 Geschichte

Der Unified Process ist das Produkt aus 3 Jahrzehnten Entwicklung und Anwendung und gilt deswegen laut den Autoren von [JBR99b] als ausgewogen. Er hat dabei größere Phasen durchlaufen:

- Der Ericsson Ansatz
- Objectory Process (1987-1995)
- Rational Objectory Process (1996-1997)
- Rational Unified Process (1997 - heute)

Bei Ericsson wurde sehr früh (um 1967) die komponentenbasierte Entwicklung initiiert (Ivar Jacobson). Software wurde als eine Menge verbundener Blöcke angesehen. Es wurden die ersten Vorläufer der Use Cases, Message Sequence Charts (Nachrichtenfluss-Diagramme) und Zustandsgraphen verwendet. 1987 verließ der Initiator Jacobson Ericsson und fasste mit Objectory AB in Stockholm Fuß. Objectory AB entwickelte das Use-Case Konzept weiter. Der Use-Case Begriff wurde seit dieser Zeit bis heute geprägt. Auch wurde das Schema der sukkzessiven Arbeitsschritte ausgebaut: Es wurde eine Reihe von Modellen eingeführt: Anforderungen, Use-Cases, Analyse, Design, Implementierung und Test. Das besondere daran war der Ansatz, dass ein Systemmerkmal, mit Hilfe der Beziehung zwischen den einzelnen Modellen (vom Anfang bis zum Ende), verfolgbar war. Objectory selbst entwickelte nach diesem Muster. Im Herbst 1995 wurde Objectory von Rational gekauft. Diese verbesserten den Prozess (Rational Objectory Process) in den Punkten Requirements-Management (über Use-Cases

hinaus), Implementation, Test, Projektmanagement, Konfigurationsmanagement, Deployment und der Vorbereitung der Entwicklungsumgebung. Sie fügten den Phasenansatz und die gezielte Iteration hinzu (Inception, Elaboration, Construction, Transition). Der Architekturgedanke wurde explizit mit der Architekturbeschreibung formuliert. UML, das sich zu dieser Zeit in Entwicklung befand, wurde als Modellierungssprache verwendet. Im Laufe dieser Phase des Unified Process aquirierte Rational weitere Firmen, die tiefere Kenntnisse im Requirements-Management, Testen, Konfigurations-Management, Performance-Tests und Data-Engineering brachten. Der Prozess wurde auch um einen Workflow für Geschäftsmodelle erweitert und es wurden Tools hinzugefügt, die es ermöglichten, im Rahmen des RUP, Nutzerschnittstellen aus Use-Cases zu entwickeln. Der Rational Unified Process war nun ein Prozess, der den gesamten Softwareentwicklungs- Lebenszyklus unterstützt. Die Namensänderung beruhte auf der Tatsache, dass eine Vereinheitlichung in verschiedenen Dimensionen stattgefunden hat: Vereinheitlichung und Zusammenführung von Entwicklungsansätzen durch UML und Zusammenführung der Arbeit von vielen Methodologien. Der RUP ist nun eine (kommerziell vertriebene) Implementation des UP. Rational stattet den UP unter anderem mit speziell angepassten Werkzeugen aus. Später dazu etwas mehr, ich werde mich in meiner Ausarbeitung ersteinmal auf den Unified Process beschränken.

2 Prinzipien des RUP/UP

Wie schon gesagt, der Unified Process ist ein Softwareentwicklungsprozess. Der Unified Process ist aufgrund seines eher abstrakten Ansatzes überdies ein Rahmenwerk für generische Softwareentwicklungsprozesse, das die Spezialisierung und Anpassung auf viele Arten von Software-Systemen, Anwendungsgebiete, Organisationen etc. erlaubt (*Tailoring*). Der Unified Process ist ein Prozess, der voraussetzt, dass die zu entwickelnde Software aus Komponenten besteht. Diese Komponenten müssen ihrerseits nun mittels wohl-definierter Schnittstellen gekoppelt sein.
Der Unified Process basiert auf UML. Alle Arten von Planungsdokumenten werden mit Hilfe von UML entwickelt. Dies ermöglicht eine durchgehende Notation und Vereinheitlichung der Zusammenhänge der verschiedenartigen Dokumente. Ich werde im Rahmen dieser Ausarbeitung nicht näher auf UML eingehen. (Für mehr Informationen siehe [JBR99a])

Die wichtigsten Punkte im Unified Process sind

- *Betonung der Anwendungsfälle*,
- *Konzentration auf die Architektur*,
- *der Prozess ist iterativ und inkrementell*.

Ich werde in den folgenden Kapiteln auf die Bedeutung dieser Eigenschaften, die den Kern des Unified Process bilden, näher eingehen.

2.1 Angetrieben von Anwendungfällen (Use-Case-Driven)

Software wird im allgemeinen dafür erstellt, den Bedürfnissen seiner Benutzer zu dienen. Dies wird im Unified Process durch die Einbeziehung und die Betonung der Anwendungsfälle (Use-Cases) erreicht. Unter einem Use-Case soll eine Funktionalität im zu betrachtenden System verstanden werden, die dem Benutzer einen Wert liefert. Alle Use-Cases eines Systems bilden das so genannte Use-Case-Modell (Anwendungsfallmodell). Dieses Modell ersetzt die übliche Funktionsspezifikation. Das Use-Case-Modell spezifiziert somit die Fähigkeiten des Systems bezüglich der Benutzer. Darüber hinaus haben Use-Cases eine bedeutende Rolle im Unified Process: Das System-Design basiert auf den Use-Cases, die Implementierung wird mit den Use-Cases gegengeprüft und die Tests werden mit Hilfe der Use-Cases entwickelt. Man kann also sagen, dass die Use-Cases den Unified Process antreiben. Die Use-Cases entstehen allerdings nicht “auf der grünen Wiese”; diese werden zusammen mit der System-Architektur entwickelt und beeinflussen sich gegenseitig.

2.2 Architektur-zentriert

Der Begriff der Architektur kann im Software-Umfeld ähnlich wie im Bau-Umfeld beschrieben werden: Ähnlich wie beim Gebäude soll die Software aus verschiedenen Blickwinkeln betrachtet werden. Sie umfasst die für einen bestimmten Aspekt wesentlichen Dinge, wobei sie auf der anderen Seite Dinge “unterschlägt”. Es findet Modellbildung für den gerade betrachteten Aspekt statt. Wie man wesentliche Dinge erkennt ist größtenteils Erfahrungssache. Die Architektur soll bestimmten Personen einen klaren Blick auf das Konzept ermöglichen. Sie wird aber auch geprägt durch andere Faktoren, wie z.B. die zu unterstützenden Plattformen, oder schon vorhandene Komponenten. Der Zusammenhang zwischen Architektur und Use-Cases kann man grob mit dem Zusammenhang zwischen Form (Architektur) und Funktion (Use-Cases) umreißen. Der Softwarearchitekt vollführt also folgende Schritte:

1. Einen Grobentwurf der Architektur aufstellen. Dieser sollte einen nicht anwendungsfall-spezifischen Teil umfassen (z.B. die Plattform). An diesem Punkt muss der Architekt aber sehr wohl schon einen ersten Eindruck von den Use-Cases haben.

2. Die Teilmenge der Use-Cases, die die Kernfunktionalität beschreibt, wird herausgearbeitet.

3. Mit der Weiterentwicklung der Use-Cases wird auch mehr und mehr die Architektur klarer. Dies führt im Gegenzug zu einem besseren Verständnis der Use-Cases.

Dieser Vorgang ist nun solange fortzuführen, bis Use-Cases und Architektur als ausgereift und stabil gelten.

2.3 Inkrementell und Iterativ

Entwicklung von kommerzieller Software ist meistens ein sehr langwieriger Prozess, der durchaus über ein Jahr lang dauern kann. Um die Effektivität zu steigern, ist es sinnvoll, diesen Entwicklungsprozess in Mini-Projekte zu untergliedern. Diese Mini-Projekte sind Iterationen, die in Inkrementen münden. In jeder Iteration werden nun einige Use-Cases, die die Funktionalität erhöhen, zusätzlich betrachtet. Die Menge der betrachteten Use-Cases bildet dann ein Inkrement. Zusätzlich setzt diese Iteration den Akzent aufdie wichtigsten Risiken, die im Folgenden auftreten könnten und versucht diese zu minimieren. Selbstverständlich sind die Inkremente am Ende einer Iterationsphase nicht zwingend additiv, da die Entwickler beispielsweise gewisse Teile durch neue ersetzt haben können. In einer Iteration identifizieren die Entwickler die relevanten Use-Cases, erstellen ein Design, das sich an der Architektur orientiert, implementieren das Design mit Hilfe von Komponenten und stellen sicher, dass diese den Use-Cases genügen. Falls das gesteckte Iterationsziel erreicht wurde, wird mit der nächsten Iteration fortgefahren, sonst wird die Iteration wiederholt. Das benutzen von Iterationen im Software-Entwicklungsprozess bietet folglich einige Vorteile:

- *Kostenvorteile*, im Falle eines Fehlers in einer Iteration verliert das Unternehmen nur die Kosten einer einzigen Iteration, nicht die des gesamten Projektes (angenommen die Unternehmung wendet mehr als eine Iteration an).
- *Risikominimierung* durch Risiko- und Fehlerfrüherkennung. Im traditionellen Einzelinkrement werden Fehler erst im Systemtest am Ende der Entwicklung entdeckt und führen häufig zu Verzögerungen.
- *Geschwindigkeitsvorteile* durch kurze, prägnante Arbeitsschritte. Dies führt im allgemeinen zu mehr Effizienz der Mitarbeiter.
- *Flexibilität* durch bessere Anpassungsmöglichkeiten bei sich ändernden Anforderungen.

2.4 Personen im UP

Der Unified Process ist ein Prozessmodell, welches die Personen, die mit ihm arbeiten (hauptsächlich die Entwicklerseite, aber natürlich auch die Kundenseite) in ihrer Arbeitsweise anleitet. Da es bei der Entwicklung immer komplexerer Software unabdingbar ist, in großen Teams zu arbeiten, versucht nun der UP diese Zusammenarbeit zu unterstützen. Der UP kennt in diesem Zusammenhang den Begriff *Worker* und *Ressource*. Unter Ressource versteht man den Entwickler als Person, der als Worker gewisse (sich ändernde) Funktionen übernimmt. So kann ein Entwickler verschiedene *Rollen* während des Projekts einnehmen, beispielsweise zum einen als Use-Case-Spezifizierer und zum anderen als Integrationstester tätig sein. Auch hier findet Tailoring des Prozesses an die Bedürfnisse der Entwicklungsorganisation statt.
Es wird eine UML-Notation verwendet um diese Zusammenhänge zu modellieren. Auch diese Dokumente sind ein Ergebnis des UP.

2.5 Ablauf

Wie bereits erwähnt, besteht der Unified Process aus einer Reihe sich wiederholender *Zyklen*. Ein Softwaresystem durchläuft in seinem “Leben” eine Reihe solcher Entwicklungszyklen. Jeder Zyklus wird durch die Veröffentlichung eines auslieferbaren Produkts beendet. Dieses Produkt besteht im Unified Process nicht nur aus Quelltext, ausführbarem Programm sowie einem Handbuch, es besteht aus den gesammelten Modellen die im Laufe des Prozesses entwickelt werden: Use-Case-Modelle, die Repräsentation der Architektur, Analyse-Modelle, die die Use-Cases verfeinern sollen, das Design-Modell, das Implementierungs-Modell, das Deployment-Modell, das die Verteilung der einzelnen Komponenten auf das Zielsystem darstellt und das Test-Modell. Durch die durchgehende Benutzung von UML können diese unterschiedlichen Modellarten untereinander verknüpft werden. Dies kann man am Beispiel Use-Case-Modell in Abbildung 1 sehen.

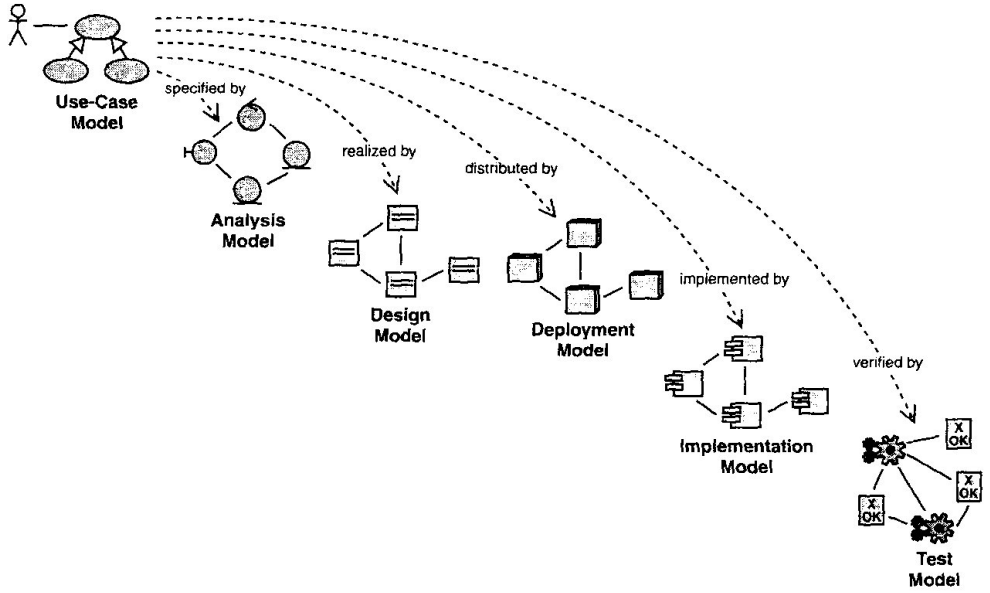

Abbildung 1: aus [JBR99b]

So können beispielsweise die Auswirkungen der Änderungen an den funktionalen Anforderungen (den Use-Cases) im Deployment-Modell erfasst werden.
Jeder dieser Zyklen besteht nun wieder aus vier *Phasen*: Inception, Elaboration, Construction, Transition. Die Phasen werden schließlich in *Iterationen* aufgeteilt. Der Endpunkt einer Phase wird durch einen *Meilenstein* markiert. Die Meilensteine sind hilfreich, um den Stand der Entwicklung festzustellen. Aufgrund der Meilensteine kann die Administration des Projektes auch gut Entscheidungen bezüglich des Fortganges des Projektes treffen.

Was hat man nun unter einem Workflow zu verstehen: Ein *Workflow* ist eine Menge von *Aktivitäten* (z.B. Finden von Use-Cases). Diese Aktivitäten werden von ent-

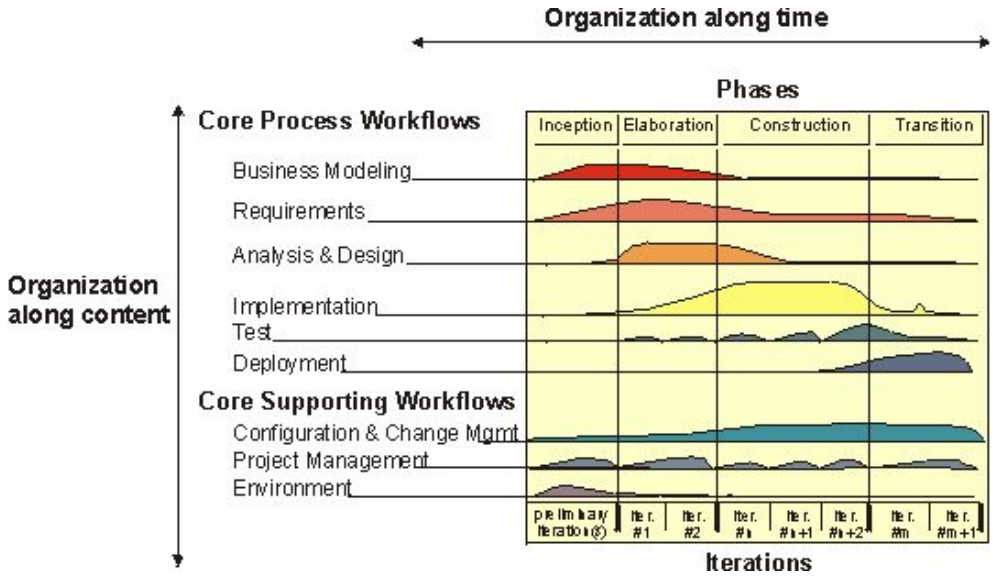

Abbildung 2: Die Abbildung zeigt grob, welche Intensität in welcher Phase welcher Workflow hat. nach [JBR99b]

sprechenden Workern ausgeführt (in diesem Beispiel dem System-Analysierer). Jegliche Beschreibung oder Information, die während einer Aktivität von Workern erstellt, verändert oder benutzt wird, nennt man im UP *Artefakt*: In diesem Fall sind das das Use-Case-Modell, ein (UML-) Actor und das Glossar.
Die Workflows, die im UP verwendet werden, sind Anforderungserfassung, Analyse, Design, Implementierung und Test. Diese sind in jeder Iteration anders ausgeprägt und produzieren (iterativ und inkrementell) oben genannte Dokumente. Siehe auch Abbildung 2. [1] Kurz gesagt wird nach dem Motto: “Ein wenig Design, dann ein wenig implementieren, dann ein wenig testen” vorgegangen. Was passiert nun in den einzelnen Phasen: In der Inception-Phase wird die grobe Vorstellung von dem zu entwickelnden Endprodukt erstellt. In der Elaboration-Phase werden die meisten Use-Cases sowie die System-Architektur entworfen. In der Konstruktion-Phase wird das eigentliche System implementiert und die Transition-Phase ist die Phase in der das System in das Beta-Stadium eingeht. Sie beschäftigt sich somit vornehmlich mit dem Test.

[1] Die Abbildung bezieht sich auf den RUP und geht auf einige spezielle nicht im klassischen UP vorkommenden Workflows ein. Unter *Business-Modelling* wird im RUP das Ausformulieren des Geschäftskontextes in dem das zukünftige System arbeiten wird. *Deployment* bezeichnet den Installation des Systems auf die Rechner des Kunden. Die *Supporting-Workflows* umfassen adminstrative Tätigkeiten: Aufbau und Pflege des Configuration- und Change-Managements, des Projekt-Managements und der Entwicklungsumgebung (inklusive möglicher Simulationsumgebungen).

3 Core-Workflows

3.1 Requirements (Anforderungen)

Die Ausarbeitung der Anforderungen für ein Softwaresystem ist im Allgemeinen eine schwierige Angelegenheit, da der Kunde in der Regel nicht genau weiß, was sein gewünschtes System können soll und was überhaupt technisch möglich ist.
Um dies zu systematisieren werden folgende Arbeitsschritte werden im Requirements-Workflow ausgeführt:

- *Mögliche Anforderungen auflisten*
- *Den Systemkontext verstehen*: Hier wird entschieden, ob ein so genanntes *Business-* oder *Domain-Modell* erstellt wird. Unter einem Business-Modell versteht man die Beschreibung der Geschäftsprozesse eines Unternehmens. Es wird eine UML-Notation verwendet, wobei besonderes Augenmerk auf Business-Entities (Use-Cases, Testfälle etc.), Work-Units (Menge von Business-Entities, z.B. Architekturbeschreibung) und Workern (z.B. Use-Case-Spezifizierer) gelegt wird. Zusätzlich werden die wichtigsten Zusammenhänge notiert.
- *Funktionale Anforderungen festhalten*: Hier wird ein Use Case-Modell des betrachteten Systems erstellt.
- *Nicht-funktionale Anforderungen festhalten*: Diese werden erfasst als zusätzliche Anforderungen oder Use-Cases.

Da sich einige Anforderungen erst im Laufe des Projektes ergeben und einige sich ändern, wird durch die Iteration dieses Workflows erreicht, dass die Anforderungen laufend überarbeitet werden. Die meisten Anstrengungen in dieser Hinsicht werden naturgemäß in den frühen Phasen des Unified Process unternommen (Inception und Elaboration). Wichtig ist hier, dass die Sprache der Kunden gesprochen wird, da diese selten die von den Entwicklern gewünschten Formalismen beherrschen. Natürliche Sprache ist deshalb so wichtig, damit eine optimale Anforderungsausbeute erzielt wird. Dies wird oft in Workshops bestehend aus Spezialisten beider Seiten erreicht. Das Ergebnis ist ein *Use-Case-Modell*, das aus *Use-Case-Systemen* besteht. Diese wiederum bestehen aus *Use-Cases* und den zugehörigen *Aktoren* (siehe auch [JBR99a]). Welche Worker sind nun an diesem Workflow beteiligt: Es gibt hier den *System-Analysierer*, der verantwortlich für die ganze Menge von Anforderungen ist, die als Use-Cases modelliert werden: Er ist verantwortlich für das System als ganzes, dargestellt durch das *Use-Case-Modell*, für das *Glossar*, und für das Auffinden der Aktoren der Use-Cases. Für die einzelnen Use-Cases ist der *Use-Case-Spezifizierer* verantwortlich. Dieser muss in seiner Arbeit eng mit dem Kunden zusammenarbeiten. Der *Architekt* betrachtet gewisse Use-Cases verstärkt, um eine frühe *Architektur-Beschreibung*, als eine Abstraktion des Use-Case Modelles zu erstellen. Auch arbeitet in dieser Phase ein *User-Interface-Designer*, der einen ersten *User-Interface-Prototypen* erstellt.

3.2 Analyse

In der Analyse- Phase werden die erfassten Anforderungen verfeinert und strukturiert. Dadurch soll das gesamte System und dessen Architektur besser verstanden werden. Hier wird die die natürliche Sprache der Anforderungserfassung in eine formale, vom Entwickler wartbare übersetzt:
Das *Analyse-Modell* ist das Resultat dieses Workflows. Hier werden UML-Notationen wie das Interaktion-Diagramm benutzt um die Dynamik des Systems zu modellieren. Das Analyse-Modell kann als eine frühe Version des Design-Modells angesehen werden. Es wäre natürlich denkbar diesen Schritt zu überspringen und die Analyse in das Design oder in die Implementierung einzubauen. Da aber in diesen Phasen noch deutlich mehr zu erledigen ist, ist es sinnvoll, diesen Teil der Arbeit auszulagern und separat zu betrachten. Das Analyse-Modell dient somit als notwendiger Input für die folgenden Phasen. Das Analyse-Modell besteht aus dem *Analyse-System*, das als das "Top-Level"-Paket zu verstehen ist. Man benutzt dann zusätzliche *Analyse-Pakete*, um das Analyse-Modell zu strukturieren und das System-Design auf einer abstrakten Ebene nachzubilden. *Analyse-Klassen* dienen als Repräsentanten von Klassen und Subsystemen und realisieren die Use-Cases. Die Art der Realisation wird durch die *Use-Case-Realisations-Analyse* ausgedrückt. Analyse wird vornehmlich in den frühen Elaborations-Iterationen betrieben. Der Architekt ist in diesem Workflow dafür zuständig, die Architekturbeschreibung weiter zu pflegen und er ist die verantwortlich Person, für die Konsistenz des Analysemodells. Der *Use-Case Techniker* ist für die Integrität von einer oder mehreren Use-Case-Realisationen. Er stellt sicher, dass die textuellen Informationen und die verwendeten Diagramme lesbar sind und den gewünschten Zweck erfüllen. *Komponenten-Techniker* definieren und pflegen die Eigenschaften, Anforderungen und Beziehungen von einigen Analyse-Klassen. Auch sind sie verantwortlich für Integrität einiger Analyse-Pakete. Es ist darüber hinaus sinnvoll diesen Komponenten-Technikern auch die Verantwortung für gewisse Design-Subsysteme zu übertragen, falls eine direkte Beziehung zwischen diesen herrscht.

3.3 Design

Im Design-Workflow wird das System geformt und die Architektur des Systems ausgearbeitet. Es soll ein detailliertes Verständnis bezüglich der nicht-funktionalen Eigenschaften des Systems erlangt werden. Es werden Dinge wie das zu unterstützende Betriebssystem, die zu verwendende Programmiersprache, oder aber die Art der Wiederverwendung von Komponenten besprochen. Die Systementwicklung, insbesondere die Implementierung soll in dieser Phase in besser handhabbare Teile herunter gebrochen werden. Dies soll eine möglichst parallele Bearbeitung ermöglichen. Verstärkt soll dies in Bereichen angewendet werden, in denen ein solches Aufbrechen aufgrund der Anforderungsspezifikation nur schwer möglich ist. Ein weiteres wichtiges Ziel des Design-Workflows ist eine Abstraktion von der Systemimplementierung zu erstellen, die es ermöglicht die Implementierung als geradlinige Verfeinerung des Design-Modells ausuführen. Geradlinig meint hier, dass die grundlegende Struktur in der Implementierung nur noch mit Code ausgefüllt werden muss. Dies ermöglicht u.a. die

automatische Code-Generierung. Auch sollen die wichtigen Schnittstellen zwischen den einzelnen Subsystemen gefunden werden.

Das Ergebnis des Design-Workflows ist das *Design-Modell*. Das Design-Modell besteht aus einem Top-Level *Design-System*, das durch weitere *Design-Subsysteme* in besser handhabbare Teile herunter gebrochen wird. Ein Design-System oder Subsystem besteht seinerseits wiederum aus *Design-Klassen* und *Schnittstellen*.

Die Kollaboration[2] *Use-Case-Realisierung-Design* stellt die Realisierung der Use-Cases durch die Design-Klassen dar. Es beschreibt die Realisierung der Use-Cases als interagierende Design Objekte (Interaction-Diagrams). Die Design-Klasse ist eine direkte Abstraktion der System-Implementation (z.B einer Klasse). Es wird hierfür die in der Implementation benutzte Programmiersprache verwendet. Es können so bereits Schnittstellen erstellt werden, falls dies in der gewünschten Programmiersprache unterstützt wird. Falls ein Problem zu code-spezifisch ist, so wird dieses auf den nachfolgenden Implentierungs-Workflow verlegt. Das Design-(Sub-)System dient dazu, das System in besser handhabbare Subsysteme aufzuteilen. An dieser Stelle muss besonders darauf geachtet werden, dass die Inhalte eines Subsystems stark zusammenhängend sind und dass die Subsysteme untereinander lose gekoppelt sind, ein äußerst wichtiges Design-Merkmal. (Siehe auch [McC93])

Ebenfalls ist an dieser Stelle die Art der Integration wiederverwendbarer Software-Komponenten zu überdenken. Dieses wird auch mit Hilfe von Design-Subsystemen modelliert. Im Design-Workflow ist der Architekt verantwortlich für die Integrität des Design-Modells. Auch sorgt er dafür, dass das Design-Modell die Vorgaben der Architekturbeschreibung erfüllt. Das *Deployment-Modell* zählt auch zum Aufgabenbereich des Architekten. Das Deployment-Modell beschreibt die Verteilung von Funktionen der Systemkomponenenten auf verschiedene Rechner im Zielsystem. Der *Use-Case-Techniker* beschäftigt sich im Design-Workflow mit der Use-Case-Realisierung-Design. Er ist für die Integrität der einzelnen im Design-Workflow betrachteten Use-Cases verantwortlich. Er sorgt dafür, dass die das Analyse-Workflow, im Design-Workflow richtig umgesetzt wird. Die Operationen, Methoden, Attribute und Beziehungen mehrerer Design-Klassen werden vom Komponenten-Techniker betreut. Er stellt auch die Integrität von einem oder mehreren Design-Subsystemen sicher. Oft ist es sinnvoll, dass der Komponenten-Techniker auch für die modellierten Details in den von ihm behandelten Subsystemen verantwortlich ist. Eine “good practice” ist es, den jeweiligen Komponenten-Techniker auch “seine” Artefakte implementieren zu lassen.

Mit dem Design beschäftigt man sich im Unified Process besonders am Ende der Elaboration-Phase und am Anfang der Construction-Phase. Besonders im Round-Trip-Engineering ist es nötig das Design-Modell, während der ganzen Lebensdauer der Software zu pflegen, da dieses hilft die Implementation zu visualisieren und graphische Programmiertechniken unterstützt.

[2]Die Spezifikation eines Elementes (z.B. ein Use-Case), die ausdrückt wie dieses Element durch Bezeichner und Assoziationen, mittels einem bestimmter Rollen realisiert wird. Um die Struktur darzustellen wird die UML-Notation des Interaktions-Diagramms benutzt.

3.4 Implementation

Die Implementation setzt beim Ergebnis des Design-Workflows an und setzt diesen in Komponenten um. Da der größte Teil der Architektur in dem Design-Workflow erdacht wurde, besteht die Hauptaufgabe im Implementations-Workflow darin, die Design-Klassen des Design-Modells in Code umzusetzen. Dazu wird in jeder Iteration die System-Integration durchdacht. Da der RUP/UP inkrementell angelegt ist, führt dies zu einer Implementierung in kleinen, gut handhabbaren Schritten. Implementierung wird besonders in der Construction-Phase betrieben. Ebenso in der Elaboration-Phase, um die Architektur zu festigen und in der Transition-Phase, um auf späte Fehler zu reagieren. Das Ergebnis des Implementations-Workflows ist das *Implementations-Modell*. Dieses besteht (analog zu den vorherigen Phasen) aus einem Top Level *Implementation-System* und verschiedenen *Implementations-Subsystemen*. Diese Systeme bestehen aus *Komponenten* und *Schnittstellen*. Unter Komponente versteht man hier das ausprogrammierte Modell. Da das Design-Modell als eine Abstraktion des zu programmierenden Systems erdacht wurde, existieren in der anderen Richtung auch Beziehungen der Komponenten zum Design-Modell (oder auch anderen erstellten Modellen). Das ermöglicht das Auffinden einer bestimmten Programmsystemsstelle im UP Modell. Eine Komponente implementiert im Allgemeinen nicht nur ein Modell-Element. Analog zum vorherigen Workflow ist der Architekt für die Integrität des Implementations-Modells zuständig und beschäftigt sich mit der Architekturbeschreibung und dem Komponentenmodell. Ein Komponententechniker ist in diesem Workflow mit dem Sourcecode einer (oder mehrerer) Komponente(n) beschäftigt. Außerdem ist er für Verifikation von einigen Subsystemen verantwortlich. Der *System-Integrierer* erstellt einen *Integration-Build-Plan*, da die Integration des Systems eine zu komplexe Aufgabe für den einzelnen Komponenten-Techniker ist. Das Implementations-Modell sollte (wie auch das Design-Modell), das ganze Leben des Softwaresystems gepflegt werden.

3.5 Test

Im Test-Workflow werden die Resultate der Implementierung verifiziert. Zu diesem Zweck werden sowohl Endversionen, als auch Zwischenversionen einem Test unterzogen. Hierfür werden die Tests, die in der jeweiligen Iteration angewendet werden, geplant. Die Tests werden dann designed und implementiert. Es soll ein möglichst automatisierter Test erlangt werden. Schließlich werden die Tests ausgeführt und systematisch ausgewertet und im Fehlerfall wieder in die jeweiligen Core-Workflows gegeben. Getestet werden soll im UP während aller Phasen: In der Inception-Phase werden einige Testszenarien angedacht, in der Elaboration- und Construction-Phase werden Architektur und der Großteil des Systems getestet. In der Transition-Phase verlagert sich der Fokus hin zum Auffinden von Fehlern aus früheren Phasen und zu Regressionstest. Analog zu den obigen Workflows sollte das Testmodell über die gesamte Lebensdauer des System gepflegt werden: Verfeinerung der Tests, Neuerstellung von Tests, Entfernung überflüssiger Tests.
In diesem Workflow ist der *Testtechniker* für die Erstellung des *Test-Modells* zuständig. Das Test-Modell besteht aus *Test-Systemen*, die wiederum aus *Testfällen* und *Testkom-*

ponenten zusammengesetzt sind. Ein Testfall beschreibt die Art und Weise, wie ein Use-Case zu testen ist. Eine Testkomponente automatisiert diesen Prozess. Für die Ausformulierung einer solchen Komponente ist allerdings der *Komponenten-Techniker* zuständig. Er legt im Vorhinein eine Teststrategie fest und überprüft die Ergebnisse des Testvorgangs. Die *Integrationstester* und die *Systemtester* sind für die Behandlung der auftretenden Fehler im Integrationstest bzw. Systemstest verantwortlich. Mehr zum Thema Test in [Het93].

4 Phasen

Im vorigen Kapitel habe ich die Workflows im einzelnen beschrieben. Wie aber bereits gesagt, werden diese nicht nacheinander komplett abgearbeitet, sondern immer im Rahmen von Iterationen, die Teil der in diesem Kapitel beschrieben *Phasen* sind. Die Workflows bilden ein Grundgerüst der Planung. In den verschiedenen Phasen, werden diese Workflows aber unterschiedlich betrachtet (siehe auch Abb. 2): In den frühen Phasen wird an den Dingen gearbeitet, die man für sehr kritisch für den Projektfortgang hält. In späteren Phasen, werden die Ergebnisse dieser Arbeit mehr und mehr zu dem Softwareprodukt geformt, mit dem der Kunde arbeiten möchte. Auch wird mit dem Phasenmodell, die Arbeit der Entwickler in kleinere, besser zu überschauende Teile heruntergebrochen.

4.1 Inception

In dieser Phase wird der Bereich abgesteckt, den das System umfassen soll, eine Vision des zukünftigen Systems soll entstehen. Es soll herausgefunden werden, ob es überhaupt ökonomisch sinnvoll ist, das System zu entwickeln. Eine grobe Architektur wird in dieser Phase entwickelt. Das Ziel hier ist nicht, eine möglichst vollständige Beschreibung zu erhalten, sondern es soll herausgefunden werden, ob überhaupt eine sinnvolle Architektur möglich ist. Besonderes Augenmerk soll hier auf Risiken und auf solche Dinge gelegt werden, die neu für das Projektteam sind. Dieses Team besteht in der Regel in dieser Phase aus dem Projekt-Manager, dem Architekt, ein oder zwei erfahrenen Entwicklern und einem Testtechniker. Ein weiteres Ergebnis dieser Phase kann eine Art “Wegwerf-Prototyp” sein, der dem Kunden, einige Möglichkeiten aufzeigt. Im Normalfall findet lediglich eine Iteration in der Inception-Phase statt, wobei das Hauptaugenmerk hier auf das Herausfiltern der Anforderungen gelegt wird. Analyse und Design wird in geringerem Maße betrieben. Auch wird in dieser Phase der Projektplan erstellt: Es wird festgelegt, wieviel Zeit man für die einzelnen Phasen brauchen wird. Es werden Kriterien für Meilensteine, Systemzustände, die das Phasenende signalisieren, festgeschrieben. Auch wird versucht, die Anzahl der Iterationen zu schätzen. Natürlich kann sich das alles noch ändern, aber ein erfahrener Projekt-Manager kann schon recht genau die benötigten Werte schätzen.

4.2 Elaboration

Das Ziel der Elaborations-Phase ist das Erstellen einer soliden Architektur. Es soll eine Grundsatzarchitektur entworfen werden, die die signifikanten Funktionalitäten umfasst. Zu den Ergebnissen zählen die oben genannten Modell-Artefakte, eine Architekturbeschreibung und eine ausführbare Implementierung. Diese Implementierung ist im Allgemeinen kein “Wegwerf-Prototyp”, wie in der Inception-Phase, sondern ein evolutionärer Protoyp, der weiterentwickelt wird. 80 % der Use-Cases sollen in dieser Phase entwickelt werden. Möglichst der gesamte Kontext des Systems soll in dieser Phase durchdrungen sein (Komplette Erstellung des Business-Modells). Qualitätsbegriffe, wie Zuverlässigkeit oder Reaktionszeiten werden in der Elaborationsphase definiert. Es wird versucht möglicherweise in späteren Phasen auftretende Risiken abzuschätzen und Gegenmaßnahmen zu entwickeln. Schließlich wird auf Basis des Kundenangebots die Projektplanung detailliert. In einem durchschnittlichen Projekt sind zwei Iterationen dieser Phase denkbar: In der ersten wird die Architektur initiiert, in der zweiten reift die Architektur zur Grundsatzarchitektur. In dieser Phase ist das Entwicklungsteam noch relativ klein und somit ergeben sich Möglichkeiten verschiedene Dinge mit Hilfe von Iterationen auszuprobieren. Das Team ist ein wenig größer als in der Inception-Phase: Die frühen Mitglieder dienen als “Gedächnis”, zusätzlich sollten Mitarbeiter hinzugezogen werden, die in der Konstruktionsphase die Leitung übernehmen können, oder beispielsweise jemand der fundiertes Wissen im Bereich der Wiederverwendung hat. Aus Workflow-Sicht werden die meisten Ressourcen für Anforderungserfassung, Analyse und Design verwendet, weniger für die restlichen.

4.3 Construction

Das Ziel der Construction-Phase ist eine ausführbare Version im Beta-Stadium. Zu diesem Zweck sollte auch schon eine aussagekräftige Anleitung vorhanden sein. Diese Phase benötigt deutlich mehr Kapazitäten als die vorhergehenden Phase, es ist von einer Verdopplung der Kapazitäten auszugehen. Deswegen ist es wichtig, in den vorherigen Phasen weitestgehende Klärung erreicht zu haben. In dieser Phase sollten alle restlichen Use-Cases erkannt und realisiert werden. Auch die Analyse wird hier zu Ende gebracht. Bei allen Tätigkeiten, besonders bei den Implementierungstätigkeiten, muss darauf geachtet werden, dass die Integrität der Architektur erhalten bleibt. Die aus den ersten Phasen notierten und noch nicht ausgemerzten Risiken, müssen beobachtet und gegebenenfalls bearbeitet werden. Es sind in der Construction-Phase mindestens zwei Iterationen gefordert, um zu gewährleisten, dass das System ordnungsgemäß arbeitet. Der Fokus wendet sich in der Construction-Phase hin zur Entwicklung, weg von der “Forschung”. In den frühen Phasen wird das Wissen rund um das Projekt gesammelt. Nun wird dieses gesammelte Wissen dazu eingesetzt, um daraus eine ausführbare Software zu bauen. Hauptaufgaben sind nun Implementierung und Test.

4.4 Transition

In die Transition-Phase tritt das Projekt, wenn das System Beta-Status erreicht hat. Der Projekt-Manager entscheidet am Ende der Construction-Phase, dass das System reif genug für den Einsatz beim Kunden ist, obwohl es wahrscheinlich noch nicht perfekt ist. In dieser Phase soll nun das Projekt zur Zufriedenheit des Kunden zu Ende geführt werden, indem sichergestellt wird, dass alle Anforderungen korrekt umgesetzt wurden. Es soll auch dafür gesorgt, dass das System beim Kunden vernünftig arbeitet, dazu zählt auch das angemessene Verarbeiten von Kunden-Feedback. Hier wird auch Fehlern, bisher nicht beachteten Risiken und Problemen auf den Grund gegangen. Das Benutzerhandbuch muss wahrscheinlich noch ein wenig angepasst werden. Es ist nicht das Ziel, das ganze System noch einmal umzukrempeln, sondern kleine Fehler zu finden. Falls es dennoch größere Probleme gibt, so hat man entweder in den vorherigen Phasen nicht vernünftig gearbeitet, oder aber der Kunde hat seine Anforderungen nicht genau genug kommuniziert. In diesem Fall sollte man abhängig von der Wichtigkeit entweder das Gewünschte im nächsten Entwicklungszyklus einbauen oder das Projekt ein paar Phasen zurück verlegen. Die Anforderungen an die benötigten Mitarbeiter ist ähnlich wie in der Construction-Phase, aber weniger aus Entwicklungsgründen, sondern eher, um adäquat auf Kundenwünsche zu reagieren. Die Core-Workflows spielen nicht so eine große Rolle in der Transition-Phase, es ist unter Umständen noch ein wenig Arbeit im Design-, Implementation- und Test-Workflow zu erledigen. Vielmehr ist auf Entwicklerseite, die Installation des Systems eine wichtige Tätigkeit. Es werden abschließend die ausstehenden Artefakte zu Ende gebracht und das Projektende wird festgelegt. Zum Endergebnis zählt nicht nur das Software-System selbst (inklusive der abschließenden Architekturbeschreibung), sondern auch Verträge und Handbücher.

5 Praktische Anwendung

5.1 Unterschied Unified Process/Rational Unified Process

Der Unified Process ist nur ein Rahmenwerk für ein Prozessmodell. Um den Unified Process wirklich anwenden zu können, bedarf es weiterer Informationen. Die Art und Weise, wie der Prozess ausgestaltet wird, hängt stark vom System ab, das zu erstellen ist: Beispielsweise wird in ein sehr großes Projekt, das ein Gebiet bearbeiten soll, welches für die Entwickler neu ist, eine sehr lange Inception- und Elaboration-Phase haben. Auch findet sich in dieser Beschreibung des Unified Process (wie auch im zugrundeliegenden Buch [JBR99b]) keine “Betriebsanleitung”. Um alle Aspekte bearbeiten zu können, benötigt man fundierten Tool-Support. Dieser Tool-Support wird vom Rational Unified Process, einem von Rational vertriebenem kommerzielles Produkt, erreicht. Falls ein Unternehmen den RUP einkauft, so bietet Rational eine umfangreich Datenbasis für jeden Arbeitsschritt und für jedes Teammitglied. Auch Support und Beratung seitens Rational wird angeboten. Ein weiterer Unterschied ist eine detailliertere Aufteilung der Workflows im RUP.

5.2 Anwendung

Es gibt zahlreiche Beispiele, die zeigen, dass der Unified Process in der Praxis wirklich funktioniert. Zugegebenermaßen betreibt Rational hier auch sehr viel Marketing. Ein erfolgreiches Beispiel ist ein Projekt der Bank von Neuseeland: Die Bank benötigte eine Erweiterung ihrer Internetbanking Anwendung. Sie wandten hierfür den Rational Unified Process inklusive seiner verfügbaren Tools an. Als Resultat ergab sich eine nahezu fehlerfreie Internet-Applikation. Anforderungen und Fehlerreporte wurden erfolgreich mit Rational Tools gehandhabt. Durch den iterativen Prozess wurden Risiken innerhalb der Entwicklung vermindert. Auch der eingeführte Entwicklungsvorgang erwies sich als wiederholbar. Insgesamt war dieses Projekt also erfolgreich für die Bank von Neuseeland. Siehe [Rat04]

6 Fazit

In meiner Ausarbeitung habe ich versucht, die Grundgedanken des Unified Process darzulegen. Auf Grund des Umfangs des Themas kann dies jedoch nur sehr oberflächlicher Überblick sein. Für weitergehende Informationen siehe auch [JBR99b]. Ich denke, man hat gesehen, dass der Unified Process versucht, die Komplexität des Softwareentwicklungsprozesses handhabbarer zu machen. Er hat dabei mehrere Ziele vor Augen:
Zunächst sollen die Anforderungen richtig aufgenommen und in Architektur umgesetzt werden.
Desweiteren fordert der Unified Process die Verwendung von Komponenten, was die Wiederverwendung zumindest in einem Unternehmen fördert.
Auch ist die Benutzung von UML als Standard-Modellierungssprache zu begrüßen, es ist zumindest ein guter Schritt in die Richtung, Dokumente verschiedener Entwicklungsstufen in Beziehung zu setzen.
Die Iteration im UP erhöht natürlich (richtig umgesetzt) die Qualität des Produktes in jeder Hinsicht. Obwohl ich unter Umständen das Problem von gewissem Overhead innerhalb des Prozesses sehe (verglichen mit einem straffen Wasserfallansatz). Das gute Risikomanagement ist sicher ein positiver Punkt im UP .
Problematisch bei der Einschätzung des Unified Process/Rational Unified Process ist aber meiner Meinung nach das Fehlen wirklich aussagekräftiger kritischer Stimmen in Fällen in denen der Prozess nicht wie gewollt funktioniert hat. Vielleicht hat Rational einfach nur eine gute Rechtsabteilung, die dies zu verhindern weiß.
Die Ideen des Unified Process, oder seines kommerziellen Ableger Rational Unified Process, sind durchaus sehr gut. Doch es gibt meiner Meinung nach bei einigen Punkte durchaus Kritikbedarf. Meiner Ansicht nach ist der Schritt vom Design-Modell hin zum Code nicht so einfach, wie es das Prozessmodell vorsieht, es ist eine eher komplizierte Aufgabe zu verifizieren, ob das Programmierte nun wirklich dem entworfenen Modell entspricht. Auch sehe ich in den vielen verschiedenen Rollen (die ich in meiner Ausarbeitung ja nur angerissen habe) Problempotential: Wenn ein Mitarbeiter Tätigkeiten für eine bestimmte Rolle ausführt, ist es denkbar, dass er nur seinen

Problembereich betrachtet und mögliche Seiteneffekte nicht bedenkt oder bewusst außen vor lässt. Es wird im UP gefordert, dass Mitarbeiter durchgehend, ihr “Spezialgebiet” von der Anforderung bis zum Code betreuen. Effizient ist das sicherlich, aber es sind auch “Scheuklappeneffekte” denkbar. In kleineren Entwicklungsabteilungen, ist dies aber auch nicht immer umsetzbar. Eine Anpassung des Unified Process an eigene Bedürfnisse (abgesehen vom RUP selber), scheint für eine Einzelorganisation schwer durchführbar, und somit ist man mehr oder weniger auf die Rational Variante angewiesen.
Ob man dies dann wirklich in Erwägung zieht, ist nicht zuletzt eine finanzielle Entscheidung. Der Prozess an sich wird in den Beschreibungen als einfach durchführbar dargestellt; die bunten Diagramme tragen sehr dazu bei. Wenn man ein wenig in die Tiefe des Prozesses schaut, stellt man fest, dass dieser auch schon in seiner Ansetzung relativ komplex und unübersichtlich ist und die Probleme der Softwareentwicklung nicht von alleine löst. Nichtsdestotrotz zeigen die erfolgreich abgeschlossenen Projekte, dass der Unified Process funktionieren kann.
Letzenendes ist die Einführung des Unified Process eine Entscheidung des Managements und der Prozess ist in seiner Ausführung stark abhängig von der Unternehmenskultur. Der UP beeinflusst die gesamte Organisation einer Unternehmung und es kann in einigen Fällen schwierig werden, eine bestehendes System zu ändern, da alle Mitglieder der Entwicklungsorganisation, den Prozess auch unterstützen und umsetzen müssen. Ob sich dieser Prozess weitläufig durchsetzt ist schwer zu sagen, zumal er mit hohen Kosten verbunden ist. Auf der anderen Seite haben die Erfinder des Prozesses durchaus Recht, dass die immer komplexer werdende Softwarewelt einen der Komplexität gewachsenen Prozess braucht.

Literatur

[Het93] Bill Hetzel. *The complete guide to Software testing*. John Wiley + Sons, 1993.

[JBR99a] Ivar Jacobson, Grady Booch, and James Rumbaugh. *The Unified Modelling Language User Guide*. Addison Wesley, 1999.

[JBR99b] Ivar Jacobson, Grady Booch, and James Rumbaugh. *The Unified Software Devolpment Process*. Addison Wesley, 1999.

[McC93] Steve McConnell. *Code Complete*. Microsoft Press, 1993.

[Rat04] IBM Rational. Ibm rational - success stories: Bank of new zealand develops internet banking and a platform for the future with rational unified process, rational tools, and services url=http://www-306.ibm.com/software/success/cssdb.nsf/cs/jens-5mqjlu?opendocument&site=rational [stand 06.12.04], 2004.

Teil 2

Aspekte des Projektmanagements

Agiles Projektmanagement vs. konventionelles Projektmanagement

Alexander Goeres

Inhaltsverzeichnis

1 Projektmanagement - was ist das eigentlich?

Der Begriff Projektmanagement wird heutzutage bei allen möglichen und unmöglichen Gelegenheiten von allen möglichen Personen benutzt. Er ist derart eingängig, das es den Autor dieser Ausarbeitung überraschte, wie wenig er über dieses Thema und die damit verbundenen Schwierigkeiten eigentlich wußte.

1.1 Was ist ein Projekt?

Der Produktionsprozess in Betrieben ist auf die Herstellung bestimmter Produkte ausgerichtet. Dabei ist es unerheblich ob es sich um Waren oder Dienstleistungen handelt. Einmalige Aufgaben die stark von der üblichen Produktpalette abweichen lassen sich schwer in den normalen Fertigungsprozeß integrieren. Die Alternative ist die Bildung eines Teams, das sich außerhalb der normalen Abläufe nur um diese Aufgabe kümmert.

Softwareentwicklung, insbesondere die Erstellung neuer Software wird meist als Projekt realisiert. Damit dies geschehen kann sind bestimmte Rahmenbedingungen erforderlich, die die Aufgabe definieren. Die fünf Kriterien die ein Projekt ausmachen sind:

- Einmaligkeit - Schaffen eines bisher nicht dagewesenen
- konkrete Ziele - das oder die gewünschte/n Produkt/e sind klar definiert (überprüfbar)
- Beschränktheit der Ressourcen - die Kosten des Projektes sind abgegrenzt
- Umfang und Komplexität - Routine-Aufgaben sind keine Projekte
- Interdisziplinarität - Mehrere Experten müssen zusammen arbeiten

Definition: Ein Projekt ist ein zeitlich begrenztes Vorhaben zur Schaffung eines einmaligen Produktes (oder einer Dienstleistung) [PMI00].

Hieran ist zu erkennen, dass zum Beispiel die Einrichtung eines Firmenintranets ein Projekt ist, die daraufhin erforderliche Betreuung dieses Netzes aber nicht, da hier die zeitliche Beschränkung fehlt.

1.2 Was ist überhaupt zu managen?

Die klassischen Aufgaben eines Managers (siehe Abbildung 1) gelten selbstverständlich auch für einen Projektmanager. Die Besonderheiten beim Projektmanagement gehen auf den Ursprung dieses Begriffes zurück. Im zweiten Weltkrieg wurde von den amerikanischen Regierungsbehörden das Projektmanagent für die Rüstungsindustrie entwickelt. Im Vordergrund stand vor allem, innerhalb der beschränkten Ressourcen

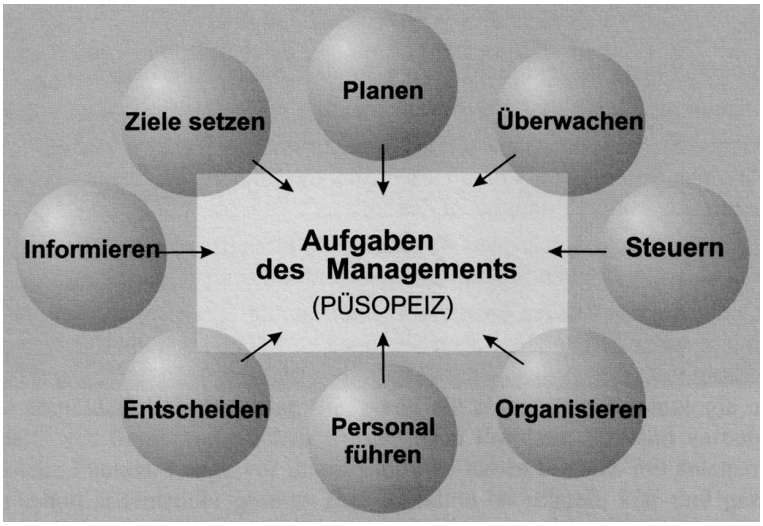

Abbildung 1: Aufgaben des Managements (aus [CG01])

(in diesem Fall - Zeit), ein angemessenes Ergebnis zu erzielen. Management war also nicht mehr die unbefristete Aufgabe, die Existenz des Unternehmens zu sichern, sondern mit Hilfe der klassischen Management-Methoden die gestellte Aufgabe zu lösen. Die Hauptarbeit eines Projektmanager ist es das Projekt innerhalb des magischen Dreiecks (siehe Abbildung 2) aus Qualität, Ressourcen und Zeit zu halten, wenn er das Projekt zum Abschluß bringen will.

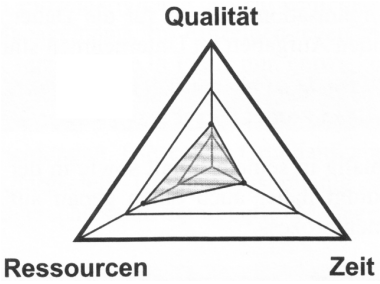

Abbildung 2: Das magische Dreieck (aus [Ger03])

2 Agiles Projektmanagement - Projektphasen

Das Hauptziel von Unternehmen ist der Verkauf von Produkten und Dienstleistungen. Diese müssen entwickelt werden, wobei sich die Durchführung in Form von Projekten als sehr effektiv herausgestellt hat. Dabei darf man aber nicht einem Projekt ein Produkt so zuordnen, das sie quasi synomym werden, sondern sollte aus folgenden Gründen strikt auf eine Trennung achten.

Die Lebensspanne eines Produktes gliedert sich in zwei große Phasen : Produkterstellung und Produkteinsatz. Hierbei ist zu beachten, dass der Produkteinsatz die zentrale Position einnimmt, da hier der Nutzen für Kunde und Hersteller entsteht. Die Arbeit eines Projektes bezieht sich aber meist auf die Produkterstellung. Danach sollte das Projekt beendet sein. Ein Projekt, das das Produkt während seiner ganzen Existenz begleitet ist meist nicht sinnvoll. Die Lebensspanne eines Produktes ist meist in der Regel nicht abzusehen und somit widerspricht ein solches Projekt der Definition eines Projektes. Abgesehen davon ändern sich die Anforderungen an Personalstärke und Qualifikation so stark, das man eh nicht mit einem Team, sondern eher mit zweien arbeiten muss.

Soll ein größeres Projekt realisiert werden, steht der Projektmanager vor der Aufgabe alles zu koordinieren. Insbesondere die Zielsetzungen und deren Prioritäten stellen höchste Ansprüche an Ressourcenverteilung und Organisation. Die Unterteilung eines Projektes in mehrere Phasen soll hier Abhilfe schaffen. Die übliche Unterteilung erfolgt auf vier Hauptbereiche : Vorbereitung, Initialisierung, Durchführung und Abschluß. Dies ist sehr ähnlich zu den in der Softwareentwicklung bekannten Vorgehensmodellen (Wasserfall, RUP, ...), die helfen sollen, den Überblick zu behalten und Struktur in diesen kreativen Prozess zu bringen. Die Änlichkeit geht teilweise soweit, dass sich Projektphasen und Softwareentwicklungsphasen eins zu eins entsprechen, was allerdings keineswegs Vorraussetzung ist, sondern sich aus den Gegebenheiten ergibt.

2.1 Vorbereitung

Die Vorbereitungsphase ist nicht direkt Teil der eigentlichen Projektarbeit, da hier entschieden wird, ob und in welcher Form das Projekt überhaupt realisiert werden kann. Die Aufgabe des Projektmanagers ist es, hier alle beteiligten Stakeholder (Personen die mit dem Projekt in Verbindung stehen) zu identifizieren und zur Zusammenarbeit zu bringen. Ebenso werden erste Entwicklungsarbeiten geleistet. Kern dieser Phase ist es, einen Projektauftrag zu erarbeiten, der folgende Aspekte klärt:

- Ziele : Der Umfang der zu erbringenden Leistung sollten geklärt werden. Insbesondere auch Kriterien zur Klärung, ob die Vorgaben eingehalten wurden oder nicht.
- Grenzen : Dinge die außerhalb des Projektes liegen, sollten explizit vermerkt werden, was späteren Mißverständnissen vorbeugt. Paradebeispiele sind Funktionen einer Software, die erst in späteren Versionen einfließen sollen.
- Ressourcen : Verbindliche Zusage von Ressourcen. Als Ressource zählen neben Geld, Gerätschaften und Arbeitsräumen vor allem auch Personal und Zeit
- Termine : Fertigstellungstermine und Meilensteintermine
- Risiken : Alle jetzt schon bekannten Risiken mit Bewertung

Das Erarbeiten des Projektauftrages ist keineswegs trivial. Das Ermitteln der Stakeholder zum Beispiel beschränkt sich nicht nur auf den Auftraggeber, sondern umfasst auch das eigene Management, Fremdfirmen sowie die Personen, die später mit dem Produkt arbeiten sollen. Eine sinnvolle Gewichtung der Bedeutung, der einzelnen Betroffenen, für das Projekt zum aktuellen Zeitpunkt, ist zu erarbeiten und über die Projektdauer später immer wieder zu überprüfen, da sich die Schwerpunkte verlagern können.

Den schwierigsten Punkt stellt die Definition der Ziele und Grenzen des Projektes dar. Obwohl der Projektmanager hier noch nicht die alleinige Verantwortung hat, wird sie ihm oft zugeschoben. Er muß also versuchen das eigene Management und den Auftraggeber dazu zu bringen verbindliche Aussagen zu machen und muss dann einen Kompromiß ausarbeiten. Während dieser Phase wird das Projekt auch zum ersten Mal geplant. Umsetzungsmöglichkeiten müssen erörtert werden und Zeit-, Kosten- und Personalbedarf abgeschätzt werden. Wichtig ist es bei der Planung auch schon die ersten Risiken abzuschätzen. Diese können von integrierten Fremdlösungen oder neuen Verfahren/Technologien stammen. Am einfachsten dürften noch die benötigten Ressourcen zu benennen sein, da sie sich teilweise direkt aus den Zielen ableiten lassen.

Viel Erfahrung bedarf die Einschätzung von Terminen. Hier gibt es keine Faustregel oder Kochbuchlösungen. Jedes Projekt ist anders und es hängt nicht nur vom Umfang der Aufgabe, sondern auch vom Erfahrungsgrad der zukünftigen Projektmitglieder, dem Einsatz neuer Technologien und den Beschränkungen des Projektes durch Ressourcen, Budget und ähnlichem ab.

Erschwerend kommt hinzu, dass es sich hier nicht um einen iterativen Prozess handelt, sondern Management und Entwicklungsaufgaben parallel laufen müssen (siehe Abbildung 3). Das Ergebnis ist der Projektauftrag, auf dessen Basis ein formeller Auftrag erstellt wird und die Einrichtung des Projektes beginnt. Der Anteil dieser nun abgeschlossenen Phase am Gesamtprojekt beträgt ca. 10 Prozent. Der Aufwand lohnt aber im Vergleich zu möglichen Schwierigkeiten, die durch einfaches 'Drauflosprobieren' entstehen können.

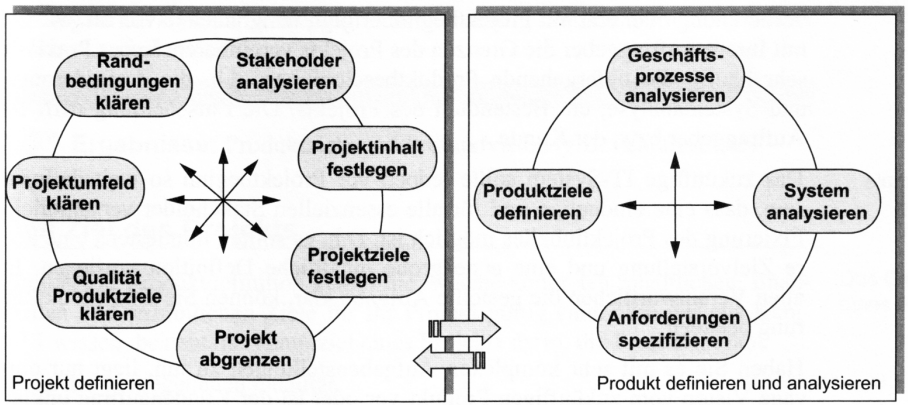

Abbildung 3: Parallelität von Management- und Entwicklungsaufgaben (aus [Ger03])

2.2 Initialisierung

Der Initialisierungsteil bildet das Fundament für das weitere Vorgehen in der Projektumsetzung. Je nach Projektgröße sollte zu diesem Zeitpunkt ein sogenanntes Kernteam die Arbeit aufnehmen und sich um alle organisatorischen Details kümmern. Idealer Weise bleiben diese Leute auch später im Projekt, um sicherzustellen, dass die Entwicklung auch wirklich in die gewünschte Richtung geht.

Eine Anforderungsspezifikation muss ausgearbeitet werden. Sie muss nicht komplett ausgearbeitet sein, allerdings muss auf ihrer Basis eine Abnahme durch den Kunden möglich sein, sowie eine Systemarchitektur entworfen werden können. Bei agilem Management eines IT-Projektes wird meistens eine Implementation in Releases angestrebt. Man startet mit einer kleinen funktionierenden Einheit, die von Release zu Release erweitert wird. Die Architektur muss darauf abgestimmt werden. Wie dies geschieht liegt in der Verantwortung des/der Systemarchitekten. Die Nutzung moderner Programmiertechniken wie Objektorientierung kann dabei sehr hilfreich sein. Dieses Vorgehen wird iterativ-inkrementelles Vorgehen genannt. In Abbildung 4 wird es der klassischen Variante gegenübergestellt.

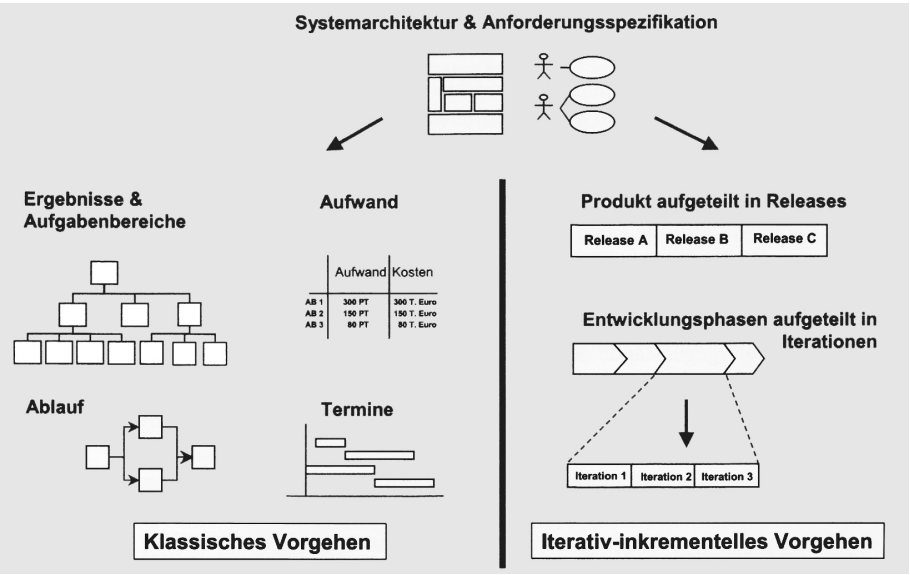

Abbildung 4: Projektdurchführung (aus [Ger03])

Für die weitere Planung ist es sinnvoll, einen Projektstrukturplan aufzustellen. Er besteht aus der ergebnisorientierten Anordnung von Projektelementen, die hierachisch strukturiert sind und den Gesamtinhalt und -umfang des Projektes definieren [PMI00] (siehe auch 5). Ob die Strukturierung verrichtungsorientiert (aus Sicht der Aufgaben), objektorientiert (aus Sicht der zu erzeugenden Ergebnisse) oder als Kombination (wechselndes Strukturprinzip je Element) daraus erfolgt ist nicht vorgeschrieben. Je nach Projekt haben beide Arten Vor- und Nachteile.

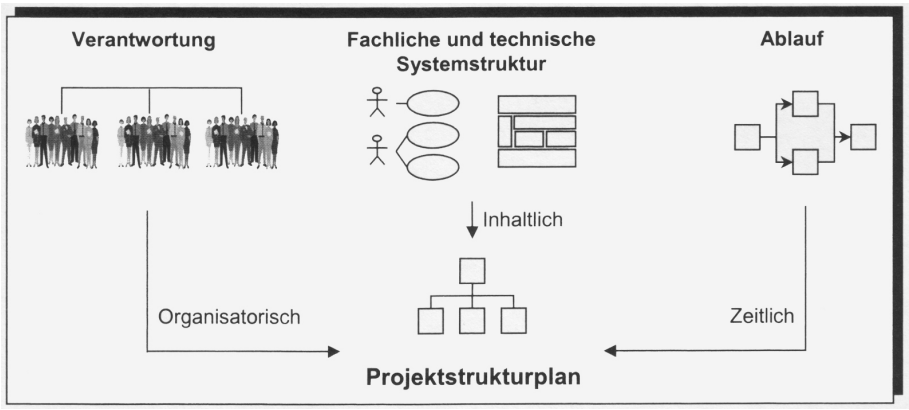

Abbildung 5: Einflußfaktoren auf Projektstrukturplan (aus [Ger03])

Der nächste Schritt sollte die Erstellung eines Arbeitsplanes sein. Er legt fest, wer was mit welchen Mitteln und konkreten Ergebnissen zu entwickeln hat. Hierbei werden unvermeidbar Konfliktsituation durch Überschneidungen auftreten. Diese sind mit Augenmerk auf den Projektstrukturplan und die zeitliche Abfolge soweit wie möglich aufzulösen. Jetzt sollte auch schon für jeden Arbeitsschritt eine Aufwandsabschätzung erfolgen. Sie hilft später bei der Kontrolle des Projektfortschritts und bei Kostenschätzungen.

Ein Ressourcenplan dient zur Aufteilung der in der Vorbereitung ermittelten Ressourcen. Üblicherweise besteht er aus den Teilplänen für Personal, Kosten und Einsatzmittel. Auch hier werden wieder Kollisionen auftreten, die nach Priorität aufgelöst werden müssen.

Alle bisher gewonnen Erkenntnisse über das Projekt und sein Umfeld fließen in der Systemarchitektur zusammen. Sie ist weit genug ausgearbeitet, wenn auf ihrer Grundlage die inhaltliche Planung der Durchführung, die Identifikation von Risiken und die Strukturierung von Teams möglich ist. Folgt man objektorientierten Entwicklungsmethoden kann dies die Aufteilung in relativ unabhängige Teilprodukte und Releases erleichtern. Trotzdem gibt es Projekte wo dies nicht möglich ist.

Sehr hilfreich für die interne und externe Kontrolle des Fortschritts ist auch die Definition von Entwicklungsgegenständen. Im Gegensatz zu den Liefergegenständen (Software, Inbetriebnahme, Benutzerhandbuch) sind diese genauer untergliedert und dienen als Meilensteine der Entwicklung.

Das Hauptpoblem in dieser Phase, genau wie in der Vorbereitung, ist, dass oben genannte Schritte nicht sequentiell sondern parallel bearbeitet werden müssen. Als Ergebnis müssen auf jeden Fall eine überprüfte Systemarchitektur, ein Projetstrukturplan und ein Arbeitsplan vorliegen.

Der zweite Teil der Initialisierungsphase betrifft die Organisation des gesamten Projektes. Es gibt zwei Hauptorganisationsformen:

- autonome Projektorganisation

  Alle Beteiligten werden aus ihrer aktuellen Unternehmensstruktur herausgelöst und im Projekt neu zusammengefasst. Dieses Vorgehen eignet sich besonders für große und komplexe Projekte, oder für sehr kleine abgegrenzte Aufgaben, die in einer bestehenden Organisationseinheit durchgeführt werden können. Besonderer Vorteil: externe Mitarbeiter werden besser integriert.

- Matrix-Projektorganisation

  Alle Beteiligten werden nur fachlich unterstellt, verbleiben aber in ihrer derzeitigen Organisationseinheit. Diese Art der Organisation ermöglicht Projekte mit vielen externen Mitarbeitern, die nicht die ganze Zeit über entbehrlich sind. Der Hauptnachteil ist die unklare Kompetenzenverteilung und menschliche Konflikte: Die Beteiligten sind Diener zweier Herren

Ist die Organisationsform geklärt, müssen noch das Berichtswesen, die Rollenverteilung und die Dokumentationsmethode festgelegt werden. Hier ist wieder alles von der Projektgröße abhängig: Je größer desto formaler, je kleiner desto direkter (Kaffepause). Als Beispiel für eine Projektorganisation dient Abbildung 6.

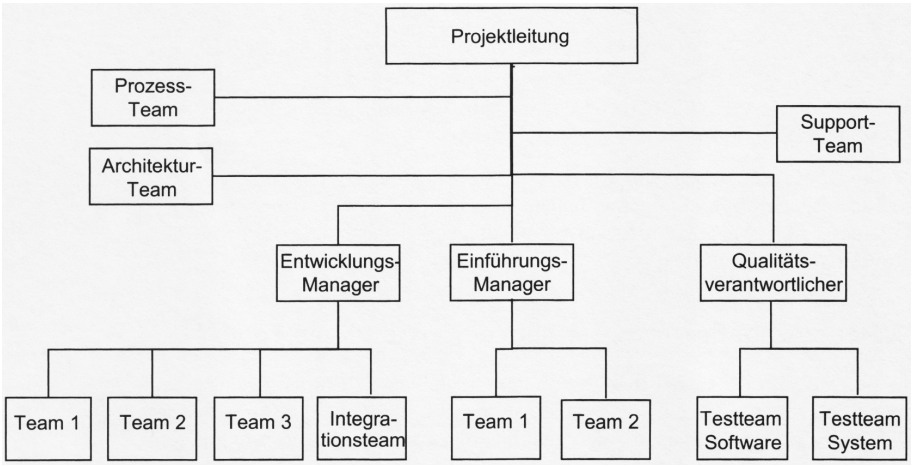

Abbildung 6: Projektorganisation (aus [Ger03])

Die Dokumentation der in der Initialisierungsphase anfalleden Ergebnisse darf nicht vernachlässigt werden, da sonst später dieselben Fragen erneut diskutiert werden müssen. Auch sollte man hier schon das weitere dokumentarische Vorgehen klären. Das Qualitätsmanagement beginnt hier durch Aufstellen eines Projekt-Qualitätsplanes (was wird wann von wem womit geprüft), sowie eines Akzeptanzplanes (welche Eigenschaften muß das zu prüfende Produkt haben um die Tests zu bestehen). Alle Ergebnisse sind dementsprechend zu prüfen.

Der letzte und auch sehr wichtige Teil dieser Phase besteht in der Risikoanalyse. Da die meisten Vorgehensweisen jetzt bekannt sind und auch die Systemarchitektur steht, ist jetzt der richtige Zeitpunkt um alle bekannten Risiken zusammenzutragen, zu analysieren und im sogenannten Risikoportfolio zu dokumentieren. Bei der Analyse sollte das Hauptaugenmerk auf der Klassifizierung von Risiken liegen. Die Merkmale Eintrittswahrscheinlichkeit und Auswirkungen sind entscheidend. Damit man aber auch frühzeitig reagieren kann, sollten soweit möglich auch Warnhinweise vor dem Eintreten des entsprechenden Risikos (Risikoindikatoren) gesucht und benannt werden. Zum Risikomanagement gehört es natürlich auch entsprechende Gegenmaßnahmen zu erarbeiten.

Eine Sorgfältige Durchführung der Initialisierung führt zu Zeit- und Kostenersparnis in der Durchführung. Ihr Projektanteil von 20 Prozent ist daher gerechtfertigt. Zum Abschluß veranschaulicht Abbildung 7 noch einmal alle Aufgaben der Initalisierung auf einen Blick.

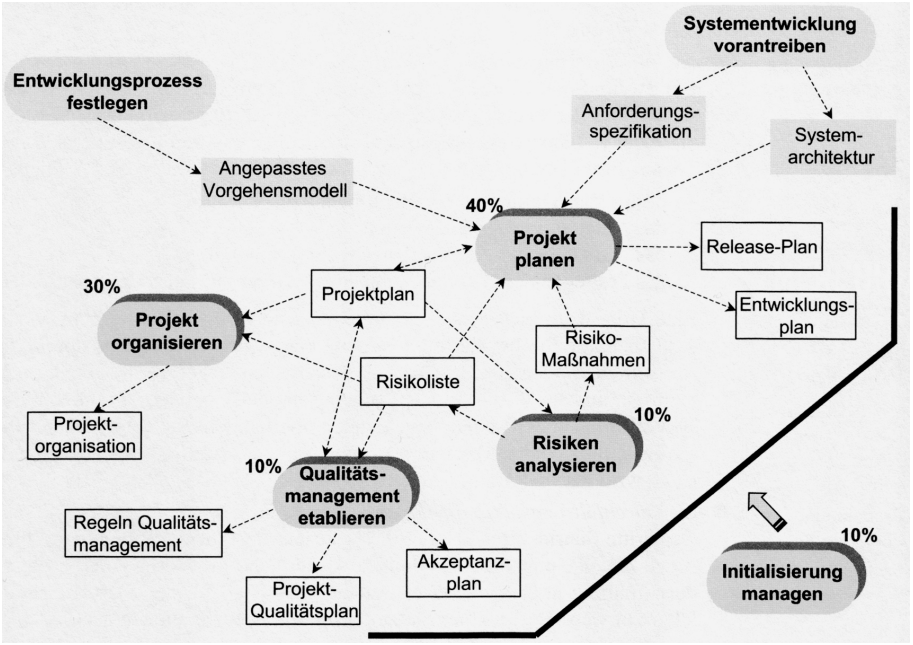

Abbildung 7: Übersicht der Projektinitialisierung (aus [Ger03])

2.3 Durchführung

Nachdem in der Initialisierung alles vorbereitet wurde, kann nun die Durchfürung des Projektes starten. Jetzt erst kommen alle Mitglieder an Bord und sollten eine funktionsfähige Organisation vorfinden. Die Aufgaben des Projektmanagements verlagern sich. Waren es bisher fast ausschließlich Planung und Vorbereitung, sind es jetzt vor allem Kontrolle und Steuerung des Projektes. Diese Phase macht mit 65 Prozent den Hauptanteil des Projektes aus.

Wenn das Projekt iterativ inkrementell vorgeht, muss zu Beginn jeder Iteration natürlich wieder die Planung in den Vordergrund treten. Da dies aber in der Initialisierungsphase bereits ausführlich besprochen wurde, wird hier darüber hinweggegangen.

Die Projektüberwachung/-kontrolle ist jetzt das hauptsächliche Betätigungsfeld des Projektmanagers. Die Art, wie er dies tut, ist abhängig von Projektgröße und Komplexität. Eine einfache und elegante Lösung ist es, sich einmal pro Woche (Monat) von seinen Teamleitern bei Projekt-Meetings den aktuellen Status zu holen. Ein nicht zu unterschätzender Vorteil dabei ist, dass alle Teams vom Status der anderen unterrichtet sind, wodurch schneller reagiert werden kann. Außerdem ist so der generelle Fortschritt auch in den einzelnen Teams ersichtlich, was zur Motivation beiträgt.

Alternativ oder auch begleitend bietet sich ein Berichtswesen an. Insbesondere bei verteilt arbeitenden Teams ist dies von Vorteil, da nicht zuviel Zeit durch An- und Abreise für Meetings verloren geht. Wenn man das Berichtswesen mit ausfüllbaren Standartberichten versieht, kann man es nutzen um von jedem Projektmitglied zu erfahren, wieviel Zeit es auf eine Aufgabe verwendet hat.

Die Kunst ist es, hier die Genauigkeit dieser Berichte so zu wählen, dass man alle nötigen Informationen erhält, ohne in Details zu versinken und den Projektmitgliedern die Zeit für die eigentliche Arbeit zu nehmen.

Die gesammelten Informationen sind permanent auf Konsistenz mit dem im Vorfeld festgelegten Zeitplan abzugleichen. Je eher die Probleme erkannt werden, desto besser funktioniert die zweite Hauptaufgabe: die Projektsteuerung (siehe Abbildung 8)

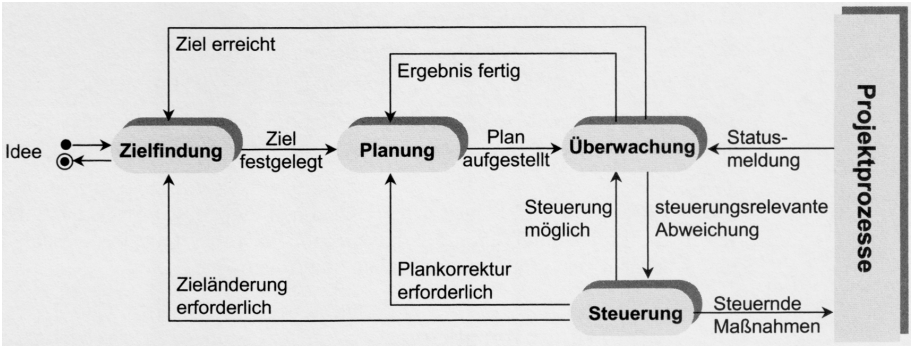

Abbildung 8: Der Regelkreis des Projektcontrollings (aus [Ger03])

Neben den Statusinformationen gibt es noch weitere Ereignisse, die eine Steuerung des Projektes erforderlich machen können, wie zum Beispiel eine Auswertung der Risikoindikatoren, ein Review oder eine Änderung in den Rahmenbedingungen des Projektes. Eine detailierte Ursachenforschung zu Abweichungen jeglicher Art ist Bedingung für erfolgreiche Gegenmaßnahmen.

Für den Fall, dass tatsächlich ein Risikofall eingetreten ist, treten jetzt die vorher ausgearbeiteten Maßnahmen in Kraft. Sind es lediglich geringere Abweichungen, ent-

scheidet die Genauigkeit der Statusinformartionen darüber, ob die richtige Ursache gefunden und behoben wird. In jedem Fall ist eine Neubewertung der Risiken erforderlich und eine Abstimmung mit dem Zeitplan.

Der Projektmanager ist die Schnittstelle des Projektes zum eigenen Management und zum Auftraggeber. Es gehört also auch zu seinen Aufgaben, diese regelmäßig über Fortschritte oder Probleme zu informieren. Ein kontinuierlicher Informationsaustausch an dieser Stelle ist genauso wichtig, wie innerhalb des Projektes, da das Projekt so Hilfe von außerhalb erfahren kann. Frühzeitige Diskussion mit Management und Auftraggeber können so zu Maßnahmen, wie z.B. Terminänderungen oder Auftragsänderungen führen, wodurch ein Scheitern des Projektes verhindert werden kann.

Eine weitere permanente Aktivität während der Durchführung ist die Qualitätssicherung. Sie begleitet alle Aktivitäten des Projektes und hat einen sehr hohen Stellenwert. Kein Meilenstein gilt als erreicht, wenn er die Qualitätskontrolle nicht besteht.

2.4 Abschluss

Der Abschluß ist die kürzeste Phase eines Projektes. Nach erfolgreichem Projektabschluß gibt er die Möglichkeit, liegengebliebene Dokumentation fertig zu stellen und ein abschließendes Review zu machen. Insbesondere der Vergleich aus alten Planwerten und dem späteren tatsächlichen Ressourcenverbrauch ist zu erörtern. Die dabei entstehenden Erkenntnisse werden im Projektabschlußbericht zusammengefasst damit sie bei späteren Projekten beachtet werden. Auch die Identifikation von wiederverwertbarer Software gehört in diese Projektphase. Ein sauberer Abschluß soll das motivationsfördernde Gefühl geben, das Projekt wirklich beendet zu haben und nicht einfach nur zum nächsten überzugehen.

3 Agiles vs. konventionelles Projektmanagement

Das prinzipielle Vorgehen beim agilen Projektmanagement, wie oben vorgestellt, unterscheidet sich kaum von demjenigen des konventionellen Projektmanagament. Es gibt jedoch einige Punkte die herausgestellt werden müssen.

Das Prädikat agil, als Nachfolgebegriff von leicht, soll zum Ausdruck bringen, dass Management und Steuerung dynamisch und flexibel gestaltet werden kann. Während beim konventionellen Projektmanagement nach Vorgehensmodellen auf DIN Basis (DIN 69904 und 69905) gearbeitet wird, bietet das agile Projektmanagement ein Baukastensystem von Methoden und sogenannten ’Best Practices’ an.

Agiles Projektmanagement wurde aus folgenden Gründen geschaffen:

- Die sehr hohe Innovationsgeschwindigkeit im IT-Bereich erzwingt entsprechend schnelle Produktzyklen
- Kunden und Markt sind nicht mehr in der Lage eigene Anforderungen zu definieren, sondern reagieren nur noch auf die Präsentation neuer technischer Möglichkeiten und Produkte
- Die Erstellung von Software ist für sich bereits ein hoch strukturierter Prozess, so dass die Notwendigkeit von systematischem Projektmanagement für Sofware-Entwicklungsprojekten oftmals nicht erkannt wird.

(aus [Mag02])

Durch die Bildung von Releases und Iterationen gelingt es den Kunden, der die Projektabwicklung von Anfang an begleitet, sehr früh mit fertigen Teilprodukten zu konfrontieren, die dann auch noch entsprechend geändert werden können. Durch die Implementation von risikoträchtigen Projektteilen gleich zu Beginn, wird auch das Scheitern eines Projektes in der Endphase vermieden (siehe Abbildung 9).

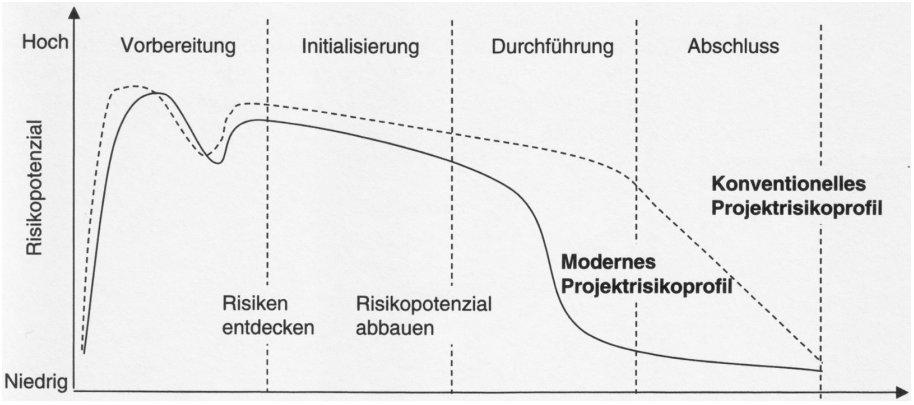

Abbildung 9: Risikoprofile (aus [Ger03])

Die wichtigsten vom agilen Management geforderten Punte sind:

- Offenheit für Änderungen statt Festhalten an starren Vorgaben
- Ergebnisorientierung an Stelle von Prozessorientierung
- Kommunikation statt Dokumentation
- Vertrauen statt Kontrolle
- Untereinander 'Best Practices' austauschen und etablieren statt zentral Anforderungen festschreiben

- Beständig Überprüfung von Risiken durch alle Beteiligten
- Denken im Ganzen - Entscheiden und Handeln im Kleinen
- Kurze Iterationen statt langer Projektphasen
- Schätzungen der Wahrscheinlichkeit der Zielerreichung statt Restaufwände
- Reviews statt Zahlenkolonnen
- schnelles Feedback statt endlose Abnahmetests
- Plankorrektur statt Ad-hoc Reaktionen

(aus [Mag02])

Bei größer werdenden Projekten nutzt aber auch das agile Projektmanagement mehr und mehr Methoden des konventionellen Projektmanagements, so dass die Unterschiede verwischen.

4 Fazit

Das agile Projektmanagement hat einen sehr einfachen und pragmatischen Leitsatz : So viel wie nötig, so wenig wie möglich.

Dieser Satz beschreibt sehr gut worum es geht. Man soll sich bei diesem Vorgehen nämlich am Projekt orientieren und den Aufwand, der für das Management betrieben wird, immer an das Projekt anpassen. Bei großen und verteilten Projekten wird eine sehr formale und dem konventionellen Managment sehr ähnliche Struktur gewählt. Bei kleinen, nicht verteilten Projekten, kann alles etwas einfacher erfolgen. So sind bei kleinen Teams zum Beispiel keine großen Projektverwaltungstools erforderlich, sondern Standard-Office Lösungen funktionieren hier genausogut. Ähnlich ist es beim Berichtswesen: Sprache, Art und Umfang hängen vom Projektumfeld ab.

Agiles Projektmanagement versucht nicht das konventionelle Management zu ersetzen, sondern erweitert es um Flexibilität und Anpassungsfähigkeit. Ein wesentlicher Aspekt liegt auch auf dem Risikomanagement. Durch das iterativ inkrementelle Vorgehen wird versucht, die Hauptrisiken so früh wie möglich zu lösen.

Agiles Management scheint ein sehr vielversprechender Ansatz zu sein, solange man nicht flexibles, angepasstes Vorgehen mit strukturlosem 'wir brauchen kein Management' Chaos, verwechselt. Problematisch ist hingegen, die Anforderung an die Projektmitarbeiter. Je kleiner und informeller, desto höher müssen die Vorkenntnisse und Erfahrungen der Mitarbeiter sein. Aktive Beteiligung und selbstverantwortliches Arbeiten sind ein Muss bei diesem Vorgehensmodell.

Literatur

[CG01] N. Ahrend Christiane Gernert. *IT-Management: System statt Chaos*. Oldenbourg, 2001.

[Ger03] Christiane Gernert. *Agiles Projektmanagement - Risikogesteuerte Softwareentwicklung*. Hanser, 2003.

[Mag02] Projekt Magazin, 2002.

[PMI00] PMI. *Projektmanagement. A Guide to the Project Management Body of Knowledge*. Rhombos, 2000.

Mitarbeiter-/Personalmanagement

Sandra Geisler

Inhaltsverzeichnis

1 Einführung

Betrachtet man den heutigen, immer noch wachsenden Konkurrenzdruck in der IT-Branche, insbesondere im Bereich Softwareentwicklung, wird klar, wie entscheidend und wichtig die richtige Auswahl von Personal für den Verlauf eines Projektes oder sogar die Existenz eines Unternehmens wird. In einer Software-Firma ist der Erfolg eines Produktes oder eines Projektes nicht in erster Linie davon abhängig, ob besonders moderne oder teure Hard- bzw. Software für die Entwicklung eingesetzt wird. Viel mehr ist entscheidend, wie effizient, effektiv und harmonisch die Menschen zusammenarbeiten, die an einem Projekt beteiligt und dafür verantwortlich sind. Mit Ihrer Kreativität, Lern- und Teamfähigkeit, Flexibilität und detailliertem Know-how können Mitarbeiter die Produktivität einer Firma maßgeblich beeinflussen.

In diesem Zusammenhang lässt sich erkennen, welch hohen Stellenwert das Personalmanagement hat oder haben sollte; denn in vielen Unternehmen wird der Faktor „Mensch“ als ebenso austausch- und einsetzbar gesehen, wie Maschinen. Dass dem nicht so ist, möchte ich in dieser Seminararbeit herausstellen. Die Arbeit besitzt zwei Schwerpunkte. Sie behandelt zum Einen den einzelnen Mitarbeiter, d.h. die Personaleinstellung, die Faktoren, die die Arbeit jedes einzelnen Mitarbeiters beeinflussen und seine Produktivität fördern bzw. hemmen können und die Vermeidung von Fluktuation. Zum anderen werde ich die Zusammenarbeit der Mitarbeiter in Projektteams beleuchten und dabei zeigen, wie ein solches Team zusammengestellt und die Arbeit innerhalb des Teams organisiert werden kann.

1.1 Definition Personalmanagement

Unter Personalmanagement versteht man das „Management des betrieblichen Produktionsfaktors „Personal“ unter Berücksichtigung der Zielsetzungen moderner Unternehmen (einschl. nachhaltigen Erfolgs, Sozial- und Umweltverträglichkeit, TQM [1], EFQM [2], Corporate Governance [3]), also nicht ausschließlich im Interesse der Erfüllung kurzfristiger ökonomischer Erfolgsziele.“[Sch00] Diese (sehr technische) Definition beschreibt das Personalmanagement aus der Sicht des Unternehmens. Der Faktor „Personal“ soll so verwaltet werden, dass auch nachhaltig die Ziele aus Sicht unterschiedlicher Interessen der Firma verfolgt werden.
Dabei sollte das Personalmanagement nach folgenden Grundsätzen handeln [Sch00]:

[1] Total Quality Management (TQM) = Umfassendes Qualitätsmanagement Nach DIN EN ISO 8402 „Eine auf die Mitwirkung aller ihrer Mitglieder gestützte Managementmethode, die Qualität in der Mittelpunkt stellt und durch das Zufriedenstellen der Kunden auf langfristigen Geschäftserfolg sowie auf Nutzen für die Mitglieder der Organisation und für die Gesellschaft abzielt“.(vgl. http://www.olev.de/)

[2] European Foundation for Quality Management (Europäische Stiftung für Qualitätsmanagement) (vgl. http://www.olev.de/

[3] Verhaltenskodex (Leitlinien) für die gute Unternehmensführung, verantwortlich und fair sowohl gegenüber den Anteilseigner als auch gegenüber anderen Interessen (-gruppen) und der Allgemeinheit entsprechend dem Governance-Konzept. (vgl. http://www.olev.de/

Erfolgsorientierung	Richte die personalwirtschaftlichen Aktivitäten explizit auf ökonomische Zielgrößen aus!
Flexibilisierung	Erwerbe die Fähigkeit zur kurzfristigen Anpassung an Unvorhergesehenes!
Individualisierung	Gewähre den Mitarbeitern den Freiraum zur Erfüllung ihrer persönlichen Ziele!
Kundenorientierung	Erstelle die Leistungen so, dass die Empfänger der Leistungen subjektiv und objektiv zufrieden sind!
Qualitätsorientierung	Integriere die Personalarbeit in den TQM-Ansatz!
Akzeptanzsicherung	Stelle sicher, dass Mitarbeiter Veränderungen unterstützen und nicht blockieren!
Professionalisierung	Aktualisiere ständig den eigenen Wissensstand und baue spezifische Kompetenzen aus!

Tabelle 1: Grundpostulate des Personalmanagements

1.2 Aufgaben im Personalmanagement

Das Personalmanagement muss viele unterschiedliche Funktionen und Aufgaben erfüllen. Dabei kommen als Funktionsträger die Unternehmensleitung, der Betriebsrat, Vorgesetzte und Betriebsabteilungen der Personalwirtschaft (Sachbearbeiter etc.) in Frage. Die Hauptaufgabengebiete sind im folgenden aufgelistet und beschrieben:

1. **Personalbestandsanalyse:**
   Bei einer **Personalbestandsanalyse** (kurz: PBA) wird der gegenwärtige (und zukünftige) Personalbestand quantitativ und qualitativ erfasst. Sie bildet die Grundlage für Personalentscheidungen, wie z.B. die Einstellung, Fortbildung, Umverteilung oder Entlassung von Mitarbeitern. Im Mittelpunkt der Betrachtungen steht das Leistungspotential der Mitarbeiter, d.h. es wird hinterfragt, ob die geforderten und benötigten Leistungen von bestimmten Mitarbeitern erbracht werden können und wenn nicht, welche Maßnahmen dann zur Erreichung dieser Ziele getroffen werden müssen.

2. **Personalbedarfsbestimmung:**
   Auf gleicher Ebene mit der Personalbestandsanalyse steht die **Personalbedarfsbestimmung**. Mit ihr wird der Soll-Stand des Personals für das gesamte Unternehmen oder auch einzelne Projekte analysiert.

3. **Personalbeschaffung:**
   Die **Personalbeschaffung** ist die Bereitstellung von Arbeitskräften für das Unternehmen in quantitativer, qualitativer, örtlicher und zeitlicher Hinsicht [Uni]. Wird nach der Personalbestands- und Personalbedarfsbestimmung festgestellt, dass der Personalbedarf über dem Personalbestand liegt, muss eine Personalbeschaffung erfolgen. Es können hier verschiedene Formen der Personalbeschaffung unterschieden werden, wie z.B. die Neueinstellung von Personal (s. 3.1), Besetzung von offenen Stellen aus dem internen Personalbestand oder die Beauftragung externer Mitarbeiter.

4. **Personalentwicklung:**
   Mit **Personalentwicklung** wird in erster Linie der Vorgang der Fortbildung von Mitarbeitern beschrieben. Die Personalentwicklung wird benötigt, wenn sich in Bezug auf die Qualität Personalbedarf und Personalbestand unterscheiden. Dabei können als Maßnahmen interne oder externe Fortbildungen angeboten werden.

5. **Personalfreisetzung:** Die Aufgabe der **Personalfreisetzung** beinhaltet die Verminderung von Personalüberdeckung. Sie wird angewendet, wenn der quantitative, qualitative, zeitliche oder örtliche Bedarf an Personal unter dem des Bestandes liegt. Die Personalfreisetzung kann vom Personalmanager z.B. in Form von Entlassungen, Kündigungen von Verträgen (externe Mitarbeiter) oder Umverteilungen von Personal durchgeführt werden.

6. **Personalveränderungsmanagement:** Das Personalveränderungsmanagement umfasst die Bereiche Personalbeschaffung, -entwicklung und -freisetzung und koordiniert diese.

7. **Personaleinsatzmanagement:** Das **Personaleinsatzmanagement**(Disposition) verteilt das Personal auf bestimmte Aufgabengebiete oder Projekte. Dabei werden die Fähigkeiten und Ressourcen der Mitarbeiter in Hinblick auf die gestellten Aufgaben berücksichtigt.

8. **Personalführung:** Bei der Aufgabe der **Personalführung** wird davon ausgegangen, dass die Mitarbeiter schon ihre Aufgaben und Stellen angetreten haben. Die Personalführung beschäftigt sich dann mit der Organisation und den Regelungen zwischen den Vorgesetzten und den ihnen unterstellten Mitarbeitern.

9. **Personalcontrolling:** Das **Personalcontrolling** schliesst die Lücke zwischen dem Personalmanagement und der restlichen Unternehmensplanung. Dabei stehen folgende Schwerpunkte im Mittelpunkt des Controllings: Effektivitätscontrolling, Effizienzcontrolling und Kostencontrolling [Uni], insbesondere aber das Kostencontrolling.

2 Die Organisation

2.1 Organisationsstrukturen

Ein wichtiger Punkt für Entscheidungen im Personalmanagement ist die Unternehmensstruktur selbst. Sie beeinflusst, wie die Mitarbeiter ausgewählt, gefördert und eingesetzt werden. Des Weiteren wird durch sie bestimmt, wie im Unternehmen Teams, im Softwareentwicklungsbereich meist Projektteams, für festgelegte Anforderungen zusammengestellt werden und mit welchem Hintergrund die Projektteilnehmer zum Projekt hinzukommen. Im folgenden werden drei der häufigsten Unternehmensformen vorgestellt, die insbesondere bei projektorientierten Firmen zu finden sind.

2.1.1 Zentralisierte Organisation

In einer **zentralisierten Organisation** werden Abteilungen aufgebaut, die allein für einen gesamten Aufgabenkomplex zuständig sind. Fachabteilungen, wie z.B. Schulungsabteilungen oder EDV-Abteilungen sind fester Bestandteil der Unternehmensstruktur und bleiben dauerhaft bestehen (Fixgruppen-Struktur). Dabei wird jede Abteilung von einem Vorgesetzten auf Gesamtunternehmensebene geleitet (Abteilungsleiter). Die Mitarbeiter werden vom Personalmanagement ihren Fähigkeiten entsprechend den einzelnen Abteilungen und insbesondere den darin vorhandenen Projekten langfristig zugeordnet. Ein wichtiger Vorteil dieser Struktur ist, dass wenn Projekte an eine Abteilung übertragen werden, werden diese durch ein Team, das schon länger zusammenarbeitet, durchgeführt. Zusätzlich werden hierarchische Streitigkeiten vermieden, da allein der Abteilungsleiter über die angenommen Projekte entscheidet. Als primären Nachteil dieser Organisation ist zu beachten, dass wichtige Kenntnisse aus anderen Fachabteilungen bei einem Projekt fehlen können, da ja nur Personen aus der eigenen Fachabteilung am Projekt arbeiten.[Por01]

2.1.2 Funktionalorganisation

In einer **Funktionalorganisation** hat jeder einzelne große Unternehmensbereich (funktionale Gruppe), wie z.B. die Produktion, seine eigenen Fachabteilungen. Das bedeutet, dass die einzelnen Fachabteilungen nur Projekte annehmen, die aus ihrem eigenen Unternehmensbereich kommen. Auch hier sind die Fachabteilungen fester Bestandteil des Unternehmens. Die Vorteile in dieser Struktur liegen darin, dass eine Entlastung der Fachabteilungen stattfindet. Projekte können schneller abgewickelt werden und die Mitarbeiter in diesen Abteilungen sind auf ihre funktionale Gruppe spezialisiert. Nachteile können daraus entstehen, dass die einzelnen Fachabteilungen unterschiedliche Vorgehensweisen in der Durchführung oder Dokumentation von Projekten entwickeln und so keine einheitliche Firmenkultur in diesem Bereich entsteht. Es entstehen auch finanzielle Probleme, da für jede Fachabteilung Material angeschafft wird, das sonst von einer großen Abteilung zusammen verwendet würde. Ohne genügenden Wissenstransfer könnten auch wichtige Erkenntnisse nicht über die Fachabteilung einer funktionalen Einheit hinausgelangen. [Por01]

2.1.3 Matrixorganisation

In einer **Matrixorganisation** arbeiten Mitarbeiter aus ganz verschiedenen Funktionsbereichen und Fachabteilungen an Projekten zusammen. Das bedeutet, dass Personal flexibel, je nach Anforderungen eines Projektes eingesetzt werden kann. Sollte eine Fachkraft mit einem Projekt nicht hundertprozentig ausgelastet sein, kann diese auch noch bei anderen Projekten mit ihrem speziellen Wissen eingesetzt werden. Auf diese Weise kann schnell und flexibel auf kurzfristige Projektanfragen reagiert werden. Die Vorteile bei einer solchen Struktur bestehen darin, dass Fachwissen projekt- und bereichsübergreifend genutzt werden kann. Es findet ein guter Know-how-Transfer statt,

und man kann schnell Teams zusammenstellen. Nachteile ergeben sich dadurch, dass Mitarbeiter gleichzeitig mehreren Vorgesetzten unterstellt sein können. Des Weiteren muss sich das Personal schnell auf die Probleme und Arbeitsmethoden der verschiedenen Projekte umstellen. Die Mitglieder der Projektteams sind meist nicht eingespielt und haben vielleicht noch nie zusammen gearbeitet.[Por01]

3 Die einzelnen Mitarbeiter

In diesem Kapitel werden Aspekte der Personaleinstellung, Gründe für Personalfluktuation und deren Vermeidung und schliesslich auch die Mitarbeitermotivation aus verschiedenen Blickwinkeln beleuchtet.

3.1 Personal richtig auswählen

Die Aufgaben des Personalmanagements können kurz in einem Satz zusammengefasst werden: Die richtigen Personen zur richten Zeit am richtigen Ort für die richtige Aufgabe auswählen! Das bedeutet, dass nur, wenn ein Manager einen Pool von „richtigen“ Mitarbeitern zur Verfügung hat, diese entsprechend eingesetzt werden können. Wie ich in der Einleitung schon kurz umrissen habe, ist in den meisten Unternehmensformen (also z.B. Softwareentwicklungs-Unternehmen) das Personal der entscheidende Faktor, der für den Erfolg von Projekten oder des Gesamtunternehmens ausschlaggebend ist. Daraus resultiert, dass schon bei der Einstellung des Personals sehr wichtige Entscheidungen getroffen werden.
In den USA wurde 2001 eine Studie über Callcenter-Agents in einer großen Kreditgesellschaft durchgeführt. Es wurde festgehalten, wieviel Geld in die einzelnen Mitarbeiter während der Aquisition, der Einarbeitung, der produktiven Phase bis hin zum Austritt des Mitarbeiters aus der Firma investiert werden musste. Die Studie zeigt, dass gerade Mehrinvestitionen bei der Einstellung des Personals und der späteren Investition in diese Mitarbeiter, längerfristig eine höhere Produktivität ergeben. Das Personal bleibt in der Regel dann auch länger bei dem Unternehmen. [Ube01]
In den nächsten beiden Unterkapiteln werden die Kriterien, die zur Auswahl eines Mitarbeiters betrachtet werden sollten und verschiedene Methoden zur Auswahl eines Mitarbeiters beschrieben.

3.1.1 Kriterien zur Auswahl eines Mitarbeiters

Damit das Unternehmen auch weit nach der Einstellung des Mitarbeiters dessen Leistungen schätzt und der eingestellte Mitarbeiter selbst eine hohe Arbeitszufriedenheit hat, sollte vor der Einstellung zunächst verglichen werden, ob Mitarbeiter und Tätigkeitsfeld zusammenpassen (Eignungsdiagnostik). Der Mitarbeiter muss für die Anforderungen der Anstellung die entsprechenden Fähigkeiten mitbringen. Dieser Vergleich

hilft eine Über- bzw. Unterforderung zu vermeiden. Des Weiteren sollte das Befriedigungspotential der Tätigkeit für den Mitarbeiter auch in späterer Zukunft ausreichend sein und mit dessen Interessen und Bedürfnissen nicht in Konflikt stehen. Dies unterstützt, dass der Mitarbeiter mit genügender Motivation bei der Arbeit ist und seine Aufgaben nicht einfach nur abarbeitet. Ausserdem muss der Mitarbeiter über ein entprechendes Entwicklungspotential verfügen, so dass die während der Tätigkeit auftretenden Veränderungen gut gemeistert werden können. Dies spiegelt die Flexibilität und Lernfähigkeit des Mitarbeiters bei neuen Aufgaben wieder. Allgemein lässt sich also sagen, dass die Eignung eines Mitarbeiters das Maß der Übereinstimmung zwischen Anforderungen der Organisation und Leistungsvoraussetzungen des Bewerbers ist.

Diese Vergleiche können nur gezogen werden, wenn sich der zuständige Manager Gedanken über die zu besetzende Stelle gemacht hat. (Situationsdiagnostik mit Arbeits-/Anforderungsanalayse). Des Weiteren muss der Personalmanager geeignete Methoden vorbereiten (Persondiagnostik), um einen Bewerber zu finden, so dass bei den Vergleichen möglichst viele Zielkriterien getroffen werden. Bei der Beschreibung des Tätigkeitsfeldes sollte nicht nur auf die aktuell anstehenden Aufgaben geachtet werden, sondern auch darauf, ob der zukünftige Mitarbeiter später für andere Aufgaben, z.B. als Projektmanager in Frage kommen soll. Die Fähigkeiten, die der Mitarbeiter für eine solche Stelle benötigt, müssen dann berücksichtigt werden. Die Einschätzung, in wie weit sich der Mitarbeiter während seiner Arbeitszeit im Unternehmen weiterentwickeln kann, ist sehr schwer zu treffen. Es gibt aber einige Eigenschaften, die sich für auch sehr unterschiedliche Berufszweige als erfolgsrelevant herausgestellt haben.
Dazu gehören Intelligenz und Lernfähigkeit, allgemeine Leistungsmotivation, soziale Kompetenz, psychische Stabilität und Veränderungsbereitschaft.Hierbei sollte gerade im Bereich der sozialen Kompetenz ein Schwerpunkt gesetzt werden. Insbesondere in der Projektarbeit ist es sehr wichtig, dass sich neue Mitarbeiter schnell einarbeiten können und sich gut in das schon vorhandene Team einfügen. Gerade die soziologischen Fähigkeiten werden als Eigenschaft oft unterschätzt bzw. gar nicht überprüft. Dies kann zu Spannungen und schlechtem Klima im Arbeitsumfeld führen.
Es gibt sehr viele Methoden, Eigenschaften, die Bewerber für eine offene Position benötigen, zu überprüfen. Diese können in zwei verschiedene Untergruppen eingeteilt werden.

1. **Eigenschaftsorientierte Verfahren:**
   Dazu gehören psychologische Tests (u.a. Intelligenztests, Persönlichkeitstests, Leistungstests), Einstellungsinterviews oder computergestützte Tests.

2. **Simulations-/Verhaltensorientierte Verfahren:**
   Dazu zählen das Sichten von Bewerbungsunterlagen, Arbeitsproben, eine Probezeit, computergestütze Simulationen, das multimodale Einstellungsinterview oder Assessment Center [4]

Für die meisten dieser Methoden kann eine prognostische Validität angegeben werden,

[4]multiple Verfahrentechnik, bei der mehrere eignungsdiagnostische Instrumente oder leistungsrelevante Aufgaben zusammengefasst werden[Ros95] - engl.: Beurteilungszentrum

d.h. ein Maß dafür, wie gut diese Maßnahmen die Eignung des Bewerbers „vorhersagen“ können. Nach diesem Maß haben folgende Methoden eine gute Validität: Probezeit, Arbeitsproben, biographischer Fragebogen, Assessment Center, multimodales Interview und kognitive Fähigkeitstests. Eine schlechtere Validität wurden mit konventionellen Einstellungsgesprächen, Persönlichkeitstests und Bewerbungsunterlagen erreicht. [Ros95]

3.1.2 Das Einstellungsgespräch

Das Einstellungsgespräch, auch Interview genannt, ist in Deutschland eine der beliebtesten Personalauswahlformen [Pau97] und wird neben der Sichtung der Bewerbungsunterlagen am häufigsten angewandt[Ros95]. Allgemein sollte vor dem Einstellungsgespräch eine Vorbereitung durch den Interviewer stattfinden. Dazu muss er z.B. die Bewerbungsunterlagen auswerten und alle benötigten Unterlagen bereitlegen.
Es gibt verschiedene Formen des Bewerbungsgesprächs. Darunter fallen:

- **Unstrukturiertes Interview:** Das Interview hat eher die Form eines zwanglosen Gesprächs. Der Interviewer ist evtl. unvorbereitet und stellt ungeordnete Fragen.
- **Halbstrukturiertes Interview:** Der Interviewer weiss aus welchen Themenbereichen er Fragen stellen möchte und aus welchen nicht. Er hat evtl. Fragen vorbereitet, hat aber keinen festen Gesprächsablauf
- **Strukturiertes Interview:** Die Fragen sind sämtlich vorbereitet und werden in einer festen Reihenfolge gestellt. Dazu kann der Interviewer einen Fragebogen (Interviewleitfaden) verwenden und die Ergebnisse in einen Bewertungsbogen eintragen. Durch diese Methodik steigt die objektive Vergleichbarkeit der Bewerber.
- **Multimodales Interview:** Ähnlich wie das strukturierte Interview. Der Interviewleitfaden wird aber aus einer Tätigkeitsanalyse heraus erstellt. Es werden verschiedene Fragetechniken angewandt, wie z.B. situative Fragen oder Fragen, bei denen der Bewerber sich selbst bewerten soll.

Obwohl das Bewerbergespräch eine sehr beliebte Form der Auswahl darstellt, besitzt sie doch eine geringe Validität. Die hauptsächlichen Gründe dafür liegen in der überwiegend subjektiven Einschätzung und dem mangelnden Anforderungsbezug der Interviewer[Ros95]. Subjektive Fehleinschätzungen können z.B. aufgrund der äußeren Erscheinung oder der Artikulationsweise getroffen werden. Andere wichtige Eigenschaften, wie z.B. die soziale Kompetenz werden oft nicht überprüft. Die Eignung des Kandidaten im Gespräch zu testen, kann durch verschiedene Fragetechniken erreicht werden. Bei der situativen Fragetechnik wird ihm beispielsweise eine Aufgabe oder eine fiktive Situation vorgegeben. Auf diese Weise können die Interviewer das Verhalten und die Entscheidungsfindung des Bewerbers einschätzen.

Eine Alternative zum konventionellen Vorstellungsgespräch ist die Anhörung. Der Bewerber hält vor seinen zukünftigen Kollegen einen berufsbezogenen Vortrag. Die Kollegen geben nach dem Vortrag dann ein Urteil über den Bewerber ab. Diese Vorgehensweise verspricht viel Erfolg, da die Mitarbeiter besser einschätzen können, ob der neue Kollege in ihr Team passt und ob er die fachliche Kompetenz für diese Aufgaben mitbringt.
Aber auch die Validität des konventionellen Bewerbergesprächs kann durch verschiedene Methoden verbessert werden. Neben den strikteren Formen (s. strukturiertes bzw. multimodales Interview) kann eine Kombination mit anderen Auswahlverfahren angewendet werden. Gerade in der IT-Branche sind Kombinationen mit Methoden, wie z.B. Arbeitsproben sehr sinnvoll. Die Präsentation von Arbeitsbeispielen gewährt dem Interviewer einen Einblick in die Arbeitsweise und -qualität des Bewerbers. Leider wird eine solche Vorgehensweise nicht häufig angewendet [TD99].

3.2 Personalfluktuation vermeiden

Unter Personalfluktuation versteht man das aus Sicht des Unternehmens ungeplante Ausscheiden von Mitarbeitern[Frö96]. Die Bedeutung des Verlusts eines Mitarbeiters und die daraus resultierenden Folgen werden von Unternehmen häufig unterschätzt. In vielen Firmen wird jeder Mitarbeiter als ersetzbar eingestuft. Wenn dies nicht so wäre, würde der Erfolg des Unternehmens ja von einzelnen Personen abhängen und dies einzusehen, fällt Managern schwer. Aber gerade in der Softwareentwicklung kann in keinster Weise von der einfachen Ersetzbarkeit eines Mitarbeiters gesprochen werden. Sehr häufig hängen Softwareprojekte von dem Know-how einzelner Personen ab und der Verlust eines solchen Mitarbeiters würde den Verlust dieses Know-hows bedeuten.

Kosten der Fluktuation:
Die Kosten für den Ersatz eines wertvollen Mitarbeiters sind immens. Zunächst muss ein Ersatz für den Mitarbeiter durch z.B. die Personalabteilung gefunden werden. Nach der Einstellung eines neuen Kollegen muss dieser zunächst eingearbeitet werden und sich in das neue Team einfügen. Dabei kann nicht nur seine Zeit nicht in vollem Maße genutzt werden, sondern andere Mitarbeiter werden durch Fragen und Hilfestellungen auch von ihrem gewohnten Arbeitsrhythmus abgehalten.

Kündigungsgründe
Die Kündigungsgründe sind in der Regel auf zwei entscheidende Faktoren zurückzuführen: mangelnde Arbeitszufriedenheit und mangelnde, organisationale Verbundenheit [NS96]. Die mangelnde Arbeitszufriedenheit resultiert häufig daraus, dass der Mitarbeiter mit den Arbeitsinhalten nicht mehr zufrieden ist, sich unter- oder überfordert fühlt oder sich mit seiner Arbeit nicht identifizieren kann. Die fehlende feste Bindung an das Unternehmen kann durch unzureichende Investitionen in Fortbildung und schlechte Aufstiegschancen, generelle große Fluktuationszahlen im Unternehmen oder einem vom Management suggerierten Ersetzbarkeitsgefühl verursacht werden.

3.2.1 Einfluss von Zeitdruck

Die Ausübung von Zeitdruck auf die Mitarbeiter wird häufig verwendet, um eine Produktivitätssteigerung zu erreichen. Dabei berufen sich Manager häufig auf das Parkinsonśche Gesetz. Dieses Gesetz aus dem Jahre 1954 besagt, dass der Arbeitsaufwand immer so weit anwächst wie die Zeit, die für die Arbeit zur Verfügung steht.[TD99] Dies würde den Schluss erlauben, dass je weniger Zeit man dem Personal gibt, eine Aufgabe zu erledigen, desto schneller würden diese fertig. Das dem nicht so ist, wurde in Studien belegt. Die Ausübung von Druck auf die Mitarbeiter hat so gut wie keinen Einfluss auf die Produktivität der Mitarbeiter.[DeM97] Meiner Meinung nach hat die Ausübung von übermäßigem Zeitdruck sogar einen negativen Effekt auf die Produktivität, da aussichtslose Abgabetermine nur zu Frustration und sogar Aufgabe führen.

3.2.2 Einfluss von Überstunden

Das Thema „Überstunden“ ist in der Softwareentwicklung ständig präsent und aktuell. Der immer größer werdende Konkurrenzdruck und schlecht geplante Zieltermine führen zu einer steigenden Anzahl von Überstunden. Überstunden in Extremform sind bei Arbeitssüchtigen zu finden, die sich völlig aufopfern und ihr gesamtes Privatleben vernachlässigen. Dies führt schliesslich zum Burnout-Syndrom. Unter dem Burnout-Syndrom versteht man einen chronischen Erschöpfungszustand mit Krankheitsgefühl, der mindestens sechs Monate andauern kann. Einige Unternehmen möchten mit Überstunden ihre Produktivität erhöhen und die Einhaltung von gesetzten Terminen garantieren. Die DV-Angestellten lassen sich nur allzu leicht mit z.B. Termindruck und schlechtem Gewissen („Ich bin zu langsam!“) sogar zu unbezahlten Überstunden regelrecht „erpressen“. Dabei ist schon längst erwiesen worden, dass Überstunden die Produktivität nicht fördern, eher im Gegenteil. Sicherlich haben Überstunden eine kurzfristige Produktivitätssteigerung zur Folge. Aber je mehr Überstunden über einen längeren Zeitraum gemacht werden, desto mehr nimmt auch die Produktivität (bzw. die der erzielte Umsatz pro Arbeitsstunde) ab (vgl. [Gme95]).

3.2.3 Personalmotivation

Damit ein Unternehmen gute Mitarbeiter möglichst lange erhalten und für sich produktiv einsetzen kann, muss es gute Arbeitsbedingungen für sein Personal schaffen und eine möglichst hohe Arbeitszufriedenheit ermöglichen. Die Motivation von Mitarbeitern ist eine wichtige Aufgabe des Personalmanagements. Nur wenn die Mitarbeiter mit ihrer Arbeit zufrieden sind, erbringen sie die gewünschten Leistungen und entwickeln Loyalität für das Unternehmen. Folgende Punkte sind wichtige Motivationsfaktoren:

- **Lob und Anerkennung:**
  Kann in Gesprächen durch das Management, bei Meetings durch die Kollegen oder bei internenWettbewerben in der Firma erfolgen. Sollte richtig dosiert und ernst gemeint sein.

- **Zugehörigkeit:**
  Kann durch richtige Zusammenstellung von Teams, viel Teamarbeit, einem klargesetzten Firmenziel (mit dem sich die Mitarbeiter identifizieren können), entsprechenden Räumlichkeiten (Aufenthaltsräume etc.) und geplanten Freizeitaktivitäten erreicht werden.

- **Selbstverwirklichung und Veränderung:**
  Kann durch abwechslungsreiche Aufgaben, Fortbildungen oder Veränderungen in der Büroumgebung erreicht werden. Abwechslung im Berufsalltag steigert die Produktivität (auch **Hawthorne Effekt** genannt.

4 (Projekt-)Teams in Organisationen

In das große Aufgabenfeld „Personaleinsatz“ des Personalmanagements fällt nicht nur die Zuordnung der Mitarbeiter zu Abteilungen, sondern, insbesondere in projektorientierten Unternehmen, die Bildung von Teams. Auch hier ist es entscheidend für den Erfolg eines Projektes, wie die Teams zusammengesetzt werden und wie ihre Arbeit organisiert wird.

Definition Team:
„Von einem Team spricht man ganz allgemein, wenn mehrere Personen in gegenseitiger Abhängigkeit bemüht sind, etwas zu vollbringen (d.h. wenn die Handlungen eines Mitglieds Einfluss auf den Erfolg aller anderen besitzen.) Ein Team ist weiters eine aufgabenorientierte Arbeitsgruppe mit starkem persönlichen Kontakt und direkter Kommunikation. Ein gemeinsames Ziel ist demnach Voraussetzung.“[Pat86]

4.1 Verschiedene Teamformen

Dem Begriff „Teamform“ kann man sich aus verschiedenen Richtungen nähern. Zum Einen kann man Teams nach ihrer Größe in **Kleingruppen** (3-7 Mitglieder, es besteht Kontakt zwischen allen Mitgliedern) und **Großgruppen** (Mehr als 7 Mitglieder, evtl. Aufsplittung in kleinere Teilgruppen, es müssen feste Regelungen zur Zusammenarbeit getroffen werden) unterscheiden. Des Weiteren kann man Teams nach örtlichen Bedingungen klassifizieren, z.B. **Face-to-face-Gruppen** (Alle Teammitglieder arbeiten physisch am selben Ort zusammen) oder **Verteilte Gruppen** (Örtliche Trennung der Gruppen, z.B. verschiedene Bürokomplexe. Kommunikation gestaltet sich schwieriger.)Durch die Einteilung in Aufgabenbereiche kann man auch Teams unterteilen in Kreativitäts-, Entscheidungs-, Umsetzungs- oder Problemslösungs-Teams.[Pat86]

Speziell auf das Softwareengineering bezogen, gibt es eine Klassifizierung der Teams aus hierarchischer Sicht:

- **Democratic Team Approach:**
  Das Team trifft Entscheidungen zusammen. Die Form unterstützt das „Egoless Programming“[5]. Funktioniert gut bei schwierigen Fragestellungen und schafft eine positive Einstellung zum Auffinden von Fehlern.[Lan01]

- **Chief Programmer Approach:** Kommunikation findet hauptsächlich über den Chefprogrammierer statt, der als Manager und technischer Leiter fungiert. Die anderen Teammitglieder sind Spezialisten, wie z.B. Datenbankspezialisten. Vorteil dieser Form ist, dass die Kommunkation hauptsächlich über eine Person läuft, also kanalisiert ist (jeder bekommt die benötigten Informationen).[Lan01]

- **Technical Managerial Approach:** Hier wird das Management auf zwei verschiedene Personen übertragen. Zum Einen gibt es den Projektmanager, der für Finanz- und Leistungsaspekte zuständig ist. Zum Anderen gibt es den Teamleiter, der die Verantwortung für technische Entscheidungen trägt. Die Entscheidungsfindung wird dezentralisiert, um die Vorteile demokratischer Teams nutzen zu können. Informationsaustausch findet zwischen den Teams über die Teamleiter und den Projektleiter statt. [Lan01]

4.2 Vor- und Nachteile der Teamarbeit

Im folgenden möchte ich kurz beschreiben, aus welchen Gründen Teamarbeit heute in vielen Firmen, insbesondere im IT-Bereich, so häufig zu finden ist.

Im Wesentlichen liegen die Vorteile der Teamarbeit darin, dass durch den Austausch mit anderen Personen, die Ideenausschöpfung viel größer ist. Die Mitarbeiter können sich mit ihren Ideen gegenseitig „anstecken“ und daraus neue produzieren. Ein weiterer Vorteil ist, dass durch das Gespräch mit den anderen z.B. prinzipielle Denkfehler frühzeitig erkannt werden können. Des Weiteren neigen Menschen immer dazu, sich gegenseitig zu vergleichen und übertreffen zu wollen. Dies kann in gesundem Maße der Produktivität zuträglich sein. Die Kommunikation, insbesondere wenn es sich um Mitarbeiter verschiedener Abteilungen handelt, fördert zudem den Know how Transfer. Ein wichtiger Vorteil wird auch durch den Spruch „Gemeinsam ist man stark!“ ausgedrückt. Eine gemeinsame Zielsetzung (es reicht die Zielsetzung, das Ziel muss nicht unbedingt erreicht werden) fördert den Zusammenhalt und Probleme können in der Gruppe besser besprochen werden.

Diesen Vorteilen stehen gewisse Nachteile gegenüber. Je größer das Team wird, desto schwieriger wird es, sich auf etwas zu einigen. Zudem werden die Kommuni-

[5] Begriff geprägt 1971 von Gerald M. Weinberg in seinem Buch „The Psychology of Computer Programming“. Der Begriff beschreibt eine Debugging Philosophie, die häufig technische Peer-Reviews einsetzt und die Einstellung der Programmierer wiederspiegelt (vgl. `http://builder.com.com/5100-6374-1054456.html`)

kationswege immer länger und komplizierter. Nicht jedes Teammitglied kann dann alle Informationen bekommen (es sei denn es wird ein zentraler Informationskanal, wie z.B. eine Verteilerliste, ein kleines Intranet o.ä. aufgebaut.). Besprechungen dauern länger, da man ja auch alle Beiträge der Teammitgleider ernst nehmen möchte. Des Weiteren wird der Organisationsaufwand auch wesentlich größer. Einen Termin zu finden, an dem alle Mitglieder eines 10-köpfigen Teams teilnehmen können, wird schwierig, ebenso wie die Aufgabenverteilung. Ist das Team aus sehr unterschiedlichen Personen mit sehr unterschiedlichen Arbeitsweisen, Vorstellungen und Einstellungen zusammengesetzt, können schnell Reibungen zwischen diesen entstehen.

4.3 Teambildung

Die Teambildung gehört auch zu einer sehr wichtigen Aufgabe des Personalmanagements (Bereich „Personaleinsatz“). Je besser ein Team zusammengesetzt wird, desto produktiver, kreativer und schneller arbeiten ihre Mitglieder. Aus diesem Grund muss genau überlegt werden, welche Personen zunächst natürlich von ihren Fähigkeiten für das gewünschte Team in Frage kommen. Aber das Personalmanagement muss ebenso bedenken, wie gut die potentiellen Mitglieder aus sozialer Sicht zusammen arbeiten können.

4.3.1 Mitglieder und Rollen in einem (Projekt-)Team

Die Beteiligten eines Projektteams unterscheiden sich, je nach Organisation eines Projektes. Typische Mitglieder eines Projektes in einer Matrixorganisation sind:

- Projektmanager: Ist für die erfolgreiche Durchführung des Projektes verantwortlich. Legt Ziele, Zeitplan und Ressourcenbudget fest
- Projekt-Teammitglieder: Sind dafür verantwortlich, die einzelnen Projekttätigkeiten durchzuführen
- Funktionalmanager: sind die direkten Vorgesetzten der Teammitglieder, organisieren die Arbeit seines Teams
- Unternehmens-Management: ist kein wirkliches Mitglied des Projektes, legt aber den äußeren Rahmen (Bezahlung, Arbeitsstandards etc.) für das Projekt fest

Die einzelnen Mitglieder nehmen automatisch und unbewusst in einem Team unterschiedliche Rollen ein, wobei es für die Rollenverteilung in einem Team viele verschiedene Ansätze gibt . Ich möchte den Ansatz des Teammanagementkreises von C. Margerison und D. McCann (s. Abbildung 5) vorstellen, weil ich der Auffassung bin, dass er sehr gut für Projektteams im DV-Bereich angewendet werden kann.
Der Ansatz beschreibt, dass es in einem Team 8 verschiedene Rollen gibt, die Teammitglieder unbewusst einnehmen. Die Rollen müssen nicht notwendig von 8 Personen

ausgefüllt werden, sondern mehrere Rollen können auch von einer Person gleichzeitig eingenommen werden. Dabei liegen die Rollen, die eine Person gleichzeitig einnimmt, meist im Managementkreis möglichst nah beieinander. Dabei gibt es folgende Rollen

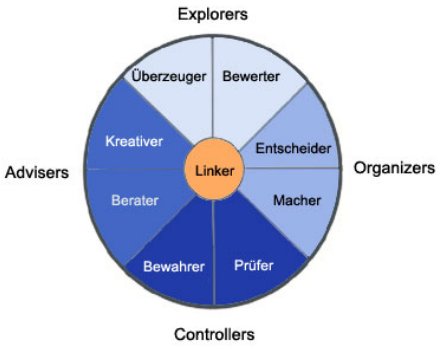

Abbildung 1: Der Teammanagementkreis

mit ihren Bedeutungen:

- **Berater (Reporter-Advisor):** Stellt die benötigten Informationen für das Team bereit. Er liefert vor allen Dingen inhaltliche Beiträge und leistet Aufbauarbeit.
- **Kreativer (Creator-Innovator):** Diese Person liefert in erster Linie Ideen und zeigt sich innovativ. Sie stellt vieles in Frage, besonders bewährte Verfahren, und kann sich schlecht Hierarchien unterordnen.
- **Überzeuger (Explorer-Promoter):** Der Überzeuger schafft benötigte Kontakte und hat guten Überblick. Er kann gut nach aussen Ideen repräsentieren, aber auch Ideen entwickeln. Er ist sehr kontaktfreudig.
- **Bewerter (Assesor- Developer):** Personen in dieser Rollen können gut abschätzen, ob Ideen realisierbar sind und haben ein gutes Einschätzungsvermögen für Ergebnisse. Sie machen nicht gerne Routinearbeiten.
- **Entscheider (Thruster- Organizer):** Initiiert die Umsetzung von Ideen und Plänen und plant Termine und Ziele. Diese Person kommt mit hierarchischen Strukuren gut zurecht und kann Krisen gut managen.
- **Macher(Concluder- Producer):** Der Macher erledigt gerne Routinearbeiten und hat ein hohes Durchhaltevermögen. Er ist sehr zuverlässig und sorgt sich darum, dass Pläne und Vorgaben eingehalten werden.
- **Prüfer(Controller- Inspector):** Ist hauptsächlich für die Qualitätssicherung verantwortlich und arbeitet eher im Hintergrund.
- **Bewahrer (Upholder- Maintainer):** Der Maintainer hat zwar geringe Führungsqualitäten, ist aber emotionale Anlaufstation für die übrigen Mitglieder, kann gut vermitteln und achtet auf Teamnormen und Werte.

- **Linker:** Diese Person koordiniert alle Informationen und repräsentiert das Team nach außen.

4.3.2 Auswahl der Projektmitglieder

Die Auswahl der Projektmitglieder passend zu einem geplanten Projekt ist entscheidend für den Erfolg des Projektes. Um die Projektmitglieder angemessen auswählen zu können, muss man sich zunächst darüber Gedanken machen, welche Fähigkeiten überhaupt notwendig sind. Je nach hierarchischer Struktur, die für das Projekt angestrebt wird, müssen verschiedene wichtige Posten besetzt und natürlich alle nötigen technischen Fähigkeiten durch die Teammitglieder abgedeckt werden.
Allgemeine Eigenschaften, die aber zunächst alle Mitglieder erfüllen sollten, sind [Pat86]:

- **Fachkompetenz:** Die Teammitglieder sollten Fachwissen über den Projektgegenstand besitzen und, wenn möglich, an einem ähnlichen Projekt schon teilgenommen haben. Außerdem sollte es Mitglieder mit Spezialwissen geben, die für das Projekt wichtig sind, wie z.B. Datenbankexperten. Generell sollten alle Mitglieder den für das Unternehmen oder die Abteilung typischen Projektablauf kennen und mit grundsätzlichen Arbeitsmethoden vertraut sein (Entwurfsmethoden etc.), damit alle eine gemeinsame Arbeitsbasis haben.

- **Sozialkompetenz:** In erster Linie sollten die Teammitglieder natürlich die nötige Teamfähigkeit mitbringen. Arbeitet jemand nicht gerne im Team, kann er eine Gefahr für das Projekt sein und den gesamten Ablauf stören. Die Beteiligten müssen zusätzlich Führungsfähigkeiten, aber auch die Fähigkeit zur Unterordnung mitbringen.Teamfähigkeit zeichnet sich auch dadurch aus, dass andere Meinungen akzeptiert und ernst genommen werden, Konfliktfähigkeiten vorhanden sind und genereller Respekt den anderen Kollegen und dem Projektleiter entgegengebracht wird.

Damit man sich ein besseres Bild über die Qualifikationen der potentiellen Mitglieder machen kann, kann ein Qualifikationsplan aufgestellt werden. Ein Qualifikationsplan ist eine Tabelle, deren Reihen die gesuchten Fähigkeiten und die Spalten die potentiellen Teammitglieder bilden. In die einzelnen Felder wird dann die Güte der Kenntnisse der Personen eingetragen. Auf diese Weise kann man feststellen, ob alle Fähigkeiten abgedeckt sind und welche Personen in welchen Bereichen Führungsqualitäten besitzen. [Por01]

4.3.3 Phasen der Teamentwicklung

Nachdem die Teambildung abgeschlossen ist, beginnt die Teamentwicklung. Die Teamentwicklung besteht aus mehreren Phasen [Pat86]:

1. **Forming (Test- oder Formierungsphase):** Die Formierungsphase beginnt sofort nach der Zusammenstellung des Teams. Jedes Teammitglied ist noch verunischert, hat Bedenken, Angst und bestimmte Erwartungen. Die Mitglieder haben noch unterschiedliche Ziele und Interessen und versuchen, die anderen Beteiligten kennenzulernen.

2. **Storming (Konflikt-, Frustrations- oder Nahkampfphase):** In dieser Phase beginnen die Mitglieder mit der richtigen Arbeit. Dabei werden bei Planungen und Besprechungen Interessenkonflikte und Meinungsverschiedenheiten offenbar. Es wird Widerstand gegen die Führungsperson und gegen andere Meinungen geben. In dieser Phase wird um die Rangordnung und Aufgaben gestritten. Es kommt zur Cliquenbildung.

3. **Norming (Normierungs-, Akzeptanz- oder Organisationsphase):** Erstmals wird das gemeinsame Ziel als solches erkannt, die eigenen Interessen diesem Ziel untergeordnet und die Mitglieder können sich damit identifizieren. Es entsteht echter Teamgeist. Die Mitglieder gehen auch offener miteinander um und teilen sich Verantwortungen.

4. **Performing (Arbeits-, Routine- oder Verschmelzungsphase):** Die meisten Probleme zwischen den Mitgliedern sind gelöst. Die Energie wird nun hauptsächlich in die Arbeit gesteckt. Teammitglieder können auch andere Rollen übernehmen, wenn Hilfe gebraucht wird. Kennzeichen dieser Phase sind hauptsächlich, dass die Motivation, das Wir-Gefühl und das Selbstvertrauen sehr hoch sind. Die Teammitglieder arbeiten sehr effektiv zusammen, helfen sich gegenseitig und erbringen so hohe Leistungen.

5. **Adjourning(Teamauflösung):** Das Projekt ist beendet und (im Idealfall) das angestrebte Ziel wurde erreicht. Alle Mitglieder wenden sich ihren alten oder ganz neuen Aufgaben zu. Dabei entsteht Enttäuschung und Frustration über den Verlust der sicheren Position in dem gut eingespielten Team.

4.4 Charakteristika eines erfolgreichen Teams

Nachdem Teams schon längere Zeit zusammengearbeitet haben, lässt sich feststellen, dass manche Teams erfolgreicher sind als andere. Woran liegt das? Folgende wesentliche Gemeinsamkeiten können bei erfolgreichen Teams beobachtet werden [TD99]:

- **Identitätsbewußtsein des Teams:** Alle Teammitglieder sehen sich als eine feste Einheit. Jeder wird für den Erfolg gebraucht. Es entwickelt sich eine Teamkultur, d.h das Team hat vielleicht einen eigenen Namen oder es gibt eine eigene „Teamsprache“. Die beteiligten Personen identifizieren sich stark mit dem Team und dessen Zielen.

- **Persönliche Beziehungen zwischen den Teammitgliedern:** Die Teammitglieder verstehen sich untereinander sehr gut. Sie arbeiten gerne und reibungslos zusammen und vertrauen einander. Es finden auch gemeinsame Freizeitaktivitäten

statt. Es herrscht eine lockere Arbeitsatmosphäre. Die Diskussionen sind tolerant, alle Personen und deren Beiträge werden ernst genommen. Zudem werden alle Konflikte offen angesprochen und gelöst.

- **Selbstbewußtsein:** Das Team besitzt ein unerschütterliches Selbstvertrauen. Die Mitglieder fühlen sich als Teil etwas Großem und Besonderem. Alle beteiligen sich aktiv am Erreichen des gemeinsamen Ziels und sind stolz dem Team anzugehören.
- **Geringe Fluktuationsrate:** Das Team verliert keine oder nur wenige Mitglieder während der Durchführung des Projektes. Die beteiligten Personen fühlen sich dem Projekt und den anderen Mitgliedern verpflichtet und geben nicht so leicht auf.
- **Freude an der Arbeit:** Alle Teammitglieder zeigen eine große Begeisterung an ihrer Arbeit. Ihr Ideenreichtum ist kaum zu bremsen.

Es ist schwer ein Team so zusammenzustellen und zu führen, dass viele oder vielleicht alle dieser guten Eigenschaften auf das Team zutreffen. Trotzdem gibt es Verhaltensweisen als Projekt- oder Teammanager, die dazu beitragen, dass das Team gut zusammenwächst und Dinge, die dies verhindern. In der Tabelle 2 finden sie eine Aufstellung der Do's und Dont's im Teammanagement.

Fördernde Massnahmen	**Hemmende Massnahmen**
Qualität als eins der wichtigsten Ziele setzen	Qualität als nicht so wichtig einstufen
Anerkennung und Bestätigung deutlich machen	Unnötige, bürokratische Regeln
Das Selbstvertrauen des Teams fördern, Elitebewußtsein erhalten	Örtliche Trennung der Teammitglieder
Erfolgreiche Teams nicht auseinanderreissen, sondern fördern und erhalten	Personen in verschiedenen Teams gleichzeitig einsetzen
In den entscheidenden Momente Führungsqualitäten beweisen	Mißtrauen der Teammitglieder gegenüber

Tabelle 2: Do's und Dont's im Teammanagement

4.5 Organisation der Teamarbeit

Ein wichtiger Erfolgsfaktor für das Erreichen eines Ziels in Teamarbeit ist die Art, wie man sie organisiert. Dazu gehören Methoden, wie Informationen an alle Beteiligten weitergegeben werden, wie man zusammen Probleme löst, Entscheidungen trifft und schliesslich auch Konflikte im Team löst.

1. **Kommunikationsformen:** Besondere Aufmerksamkeit bei der Organisation der Teamarbeit verdienen die zu schaffenden Kommunikationswege zwischen den

Teammitgliedern. Dabei ist zu beachten, dass alle Mitglieder des Teams die von ihnen benötigten Informationen in einem geeigneten Umfang erhalten. Zu wenige oder zu viele Informationen können kontraproduktiv wirken. Im folgenden stelle ich verschiedene Kommunikationsformen vor.

Schriftlich Informationen weitergeben:
Zu den schriftlichen Kommunikationswegen für Informationen gehören z.B. Statusberichte, Fehlerberichte, Artikel über neu gewonnenes Wissen oder Gesprächsprotokolle.

Meetings:
Meetings können aus verschiedenen Gründen einberufen werden, wie z.B. Entscheidungsfindung, Problemlösung, Feedback geben und schliesslich auch zum Berichten, also zur Informationsweitergabe. Sie können als klassische Konferenz, über Video, Telefon oder das Internet stattfinden.

2. **Strategien zur Lösungsentwicklung:** Im Laufe einer Teamarbeit bzw. eines Projektes kommt es immer wieder zu Situationen, bei denen das Team gemeinsam nach Lösungen suchen muss. Vor allen Dingen am Anfang eines Projektes, wenn es um die Planung geht, müssen zusammen Ergebnisse erarbeitet werden, auf denen dann die einzelnen Mitglieder oder Teilgruppen ihre Arbeit aufbauen können. Um gemeinsam zu Ergebnissen zu kommen, gibt es verschiedene Techniken, wie z.B. [Pat86]:

   - Brainstorming
   - Mindmapping
   - Nominal-group-technique (NGT)

3. **Konfliktbewältigung:**
   Ein Konflikt kann entstehen, wenn eine Person (oder Personengruppe) Pläne oder Absichten hat, deren Verwirklichung jemand anderen beeinträchtigen (verletzen, behindern, blockieren oder bedrohen) würde oder diese Absichten schon durchgeführt hat. Kleinere Konflikte kommen bei der Teamarbeit immer mal wieder vor und werden zumeist von den Konfliktparteien selbst geklärt. Handelt es sich aber um einen Konflikt, der ohne Hilfe von außen nicht gelöst werden kann, kann dies das gesamte Team behindern. Man kann die Konfliktbewältigung in drei Stufen einteilen [Pat86].

   (a) Damit Konflikte nicht so häufig und ausufernd stattfinden, kann **Konfliktprophylaxe** durch den Teammanager und das Team selbst betrieben werden. Vorschläge dazu sind z.B., dass die Auswahl der Mitglieder nach Kriterien, die ein gesundes Maß an übereinstimmenden Wertevorstellungen gewährleisten, geschieht. Das gemeinsame Ziel soll immer wieder vor Augen geführt und die Identifikation mit dem Team gefördert werden. Es sollte zusätzlich ein ausreichender Informationsaustausch garantiert und eine Teamstimmung erreicht werden, in der Probleme offen und schnell ausgesprochen werden. Ausserdem sollte man Aufgabenbereiche und Verantwortlichkeiten klar verteilen.

(b) Als nächste Stufe in der Konfliktbewältigung steht das **Aushandeln des Konfliktes zwischen den Konfliktparteien** selbst. Das bedeutet, dass die Beteiligten den Konflikt kooperativ untereinander in einem vernünftigen Gespräch lösen. Bei diesem Gespräch sollte dann eine Verhandlungsabfolge festgelegt werden, damit alle Punkte, die Gegenstand des Konfliktes sind, geklärt werden. Wichtig ist, dass die Parteien sich auf zunächst grundlegende Punkte einigen und erst später Details besprechen. Es sollten auch zunächst erst alle Punkte angesprochen werden und dann über Lösungen nachgedacht werden, da manche Punkte zusammenhängen können.

(c) Die letzte Stufe der Konfliktbewältigung ist eine **Aushandlung mit einer dritten Partei**. Diese Methode wird verwendet, wenn die Konfliktparteien sich untereinander nicht mehr einigen können. Dabei schlägt die dritte Partei keine Patentlösung vor, sondern schafft die Rahmenbedingungen, so dass die Parteien selbst doch noch zu einer Einigung gelangen können. Die dritte Partei plant das Gespräch zwischen den beiden Parteien und gibt einen Ablauf vor. Im Gespräch selbst kann sie die Konfliktparteien dazu anhalten, sich in die Lage des anderen zu versetzen und Emotionen so weit wie möglich zu unterdrücken. Einigungen werden von der dritten Partei als verbindlich erklärt und z.B. schriftlich festgehalten. Außerdem sollten Massnahmen festgelegt werden, die bei Nicht-Einhaltung der Verbindlichkeiten durchgeführt werden.

5 Zusammenfassung

Als Fazit lässt sich feststellen, dass gutes Personalmanagement eines der wichtigsten Faktoren ist, die den Erfolg eines Unternehmens mitbestimmen. Es stellt die Ressource in den Mittelpunkt ihrer Arbeit, die entscheidend für die Erfüllung des Unternehmenszwecks und -ziels ist: den Mitarbeiter. Es hat sich bestätigt, dass die Leistungsfähigkeit und Produktivität einer Firma im Wesentlichen von der Güte ihres Humankapitals, also ihres Personalmanagements abhängt.
Die wichtige Aufgabe des Personalmanagements besteht also darin dieses Kapital zu vermehren, zu fördern und zu erhalten. Weitere Feststellungen sind:
Im Kontext der umgebenden Organisation werden mit Hilfe des Personalmanagements Entscheidungen über die Personaleinstellung, das Personaleinsatzmanagement bis hin zur Personalführung getroffen. Dazu habe ich im zweiten Kapitel zunächst die verschiedenen Organisationsformen vorgestellt.
Die Aufgaben des Personalmanagements können in dem Satz zusammengefasst werden: Die richtigen Personen für die richtige Aufgabe zur richtigen Zeit und am richtigen Ort einsetzen! Daraus ergibt sich zunächst die Aufgabe der Personaleinstellung der „richtigen Mitarbeiter“. In diesem Zusammenhang habe ich die zu beachtenden Kriterien bei der Auswahl der Mitarbeiter und gängige Methoden vorgestellt.
Doch das Personalmanagement trägt nicht nur die Verantwortung für die Personaleinstellung. Es muss auch die geeignete Umgebung und Verhältnisse schaffen, so dass das Personal dem Unternehmen lange erhalten bleibt, da eine hohe Mitarbeiterfluktuation

sehr kostspielig ist (wobei kostspielig nicht nur die Finanzen, sondern auch Wissen etc. meint). In diesem Zusammenhang habe ich Faktoren erläutert, die eine Fluktuation begünstigen und Maßnahmen, die ihr entgegenwirken.
Des Weiteren ist das Personalmanagement verantwortlich für den richtigen Einsatz der Mitarbeiter. Gerade in projektorientierten Unternehmen, wie es in der IT-Branche häufig der Fall ist, ist die Zusammenstellung von Teams und die Organisation innerhalb der Teams entscheidend für den Erfolg der Projekte und damit auch des Unternehmens. Zu diesem Themenschwerpunkt habe ich aufgezeigt, aus welchen Mitgliedern und Rollen ein Team bestehen sollte, welche Teamformen es gibt und was ein erfolgreiches Team ausmacht. Ausserdem habe ich Formen der Kommunikation, Problemlösungsstrategien und Methoden zur Konfliktbewältigung vorgestellt.

Literatur

[Büh94] Rolf Bühner. *Personalmanagement*. verlag moderne industrie, 1994.

[DeM97] Tom DeMarco. *Der Termin*. Hanser-Verlag, erste edition, 1997.

[Frö96] Werner Fröhlich. *Führung und Personalmanagement*. ADIA, 1996.

[Gme95] Volker Gmelin. *Effizientes Personalmanagement durch Personalcontrolling*. expert verlag, 1995.

[Lan01] Landay. *The Software Process*. Applied Software Management UC Berkeley, Januar 2001. `http://bmrc.berkeley.edu`.

[NS96] Jürg Baillod Norbert Semmer. Fluktuation bei computerfachleuten: Eine follow-up studie. *Zeitschrift für Arbeits- und Organisationspsychologie*, 40, 1996. Fluktuation bei Computerfachleuten: Eine follow-up Studie.

[Pat86] Rattay Patzak. *Projekt Management*. Linde Verlag, dritte edition, 1986.

[Pau97] Jochen Paulus. Personalauswahl - weniger als 10 minuten. *Wirtschaftswoche*, 047, November 1997.

[Por01] Stanley E. Portney. *Projektmanagement für Dummies*. mitp-Verlag, erste edition, 2001.

[Ros95] Domsch Rosenstiel, Regnet. *Führung von Mitarbeitern*. Schäffer Poeschel, dritte edition, 1995.

[Sch00] Christian Scholz. *Personalmanagement*. Verlag Vahlen, fünfte edition, 2000.

[TD99] Timothy Lister Tom DeMarco. *Wien wartet auf Dich!* Hanser-Verlag, zweite edition, 1999.

[Ube01] Marc C. Ubelhart. Measuring the unmeasurable. *Shareholder Value*, Mai 2001.

[Uni] Unister GmbH. *Begriffsdefinitionen*. `http://www.unister.de`.

Risk Management

Moaffak Assassa

Inhaltsverzeichnis

1 Einleitung

Diese Ausarbeitung beschäftigt sich mit dem Risikomanagement im Rahmen des Projektmangements von Softwareprojekten. Sie stellt vor, wie das Risikomanagement funktioniert, was es für Vorteile bringt und wie es angewendet wird. Doch bevor ich in das Thema einsteige, möchte ich kurz erläutern was man unter einem Risiko versteht, um den Begriff ein wenig abzugrenzen und eine Grundlage zu schaffen, die im Verlauf der Ausarbeitung verwendet wird. Im Folgenden sind einige Definitionen aufgelistet, die den Begriff erklären:

- *risk - is the possibility of suffering loss*. [web]
- *Unter einem Risiko ist das Ausmaß zu verstehen, mit dem die Erreichung geschäftlicher und technischer Ziele und die Umsetzung geschäftlicher Strategien durch negative Ereignisse oder Handlungen bzw. Unterlassungen innerhalb und außerhalb des Projektumfeldes gefährdet sind.*
  Die Risikodefinition umfasst nicht nur reines Verlustrisiko, sondern auch die Nichtausnutzung des Chancenpotenzials. [Bur00]
- *Ein Risiko beschreibt die Möglichkeit, dass eine Aktivität einen körperlichen oder materiellen Verlust oder Schaden zur Folge hat.*
  Ein Risiko ist ein potenzielles Problem. [Bal98]

Die Entstehung von Risiken kann mehrere Gründe haben. Die Hauptgründe dafür sind die drei folgenden:

- **Informationsmangel**
  Wegen *Informationsmangel* können einige Aspekte nicht richtig einkalkuliert werden. Wenn man nicht alles über einen Sachverhalt weiß, kann man keine genauen Aussagen darüber treffen. Der Ausgang ist ungewiss. Es muss also geschätzt werden, oder es müssen Annahmen getroffen werden, die nicht mit absoluter Sicherheit korrekt sind. Es ist beispielsweise nicht möglich, einen Gerätetreiber für ein Gerät zu schreiben, wenn man nicht alles über dieses Gerät weiß.
- **Mangel an Kontrolle**
  Fehlt einem die *Kontrolle* über bestimmte Teile des Projektes so resultiert das ebenfalls in Risiken. Man ist abhängig von Dritten, die ebenfalls Fehler machen und somit die Fertigstellung des eigenen Projektes in Mitleidenschaft ziehen können. Wird die genaue Dokumentation eines Gerätes nicht zum genannten

Termin fertig, ist es für mich schwierig den Termin für die Fertigstellung der Treiber einzuhalten, obwohl es nicht mir zuzuschreiben ist. Das ist ein Risiko mit dem man sich beschäftigen muss.

- **Zeitmangel**
  Zeitmangel ist im Grunde die Mutter aller Risikoursachen. Entwickler machen unter Zeitdruck Fehler. Je weniger Zeit also verfügbar ist, desto wahrscheinlicher ist es, dass Fehler auftreten.

1.1 Warum geht man Risiken ein?

Es stellt sich die Frage, warum man überhaupt Risiken in Kauf nimmt oder gar bewusst eingeht. Sicherlich sind manche Risiken in vielen Fällen unvermeidbar, denn sie ergeben sich aus dem Umfeld. Es können beispielsweise in einem Projekt immer Entwickler durch Krankheit für einen kurzen Zeitraum ausfallen. Das ist ein Risiko, was man zwar bewusst eingeht, aber von Anfang an klar ist, dass ohne dieses Risiko kein Projekt möglich ist.

Man könnte aber von einer anderen Art von Risiken ausgehen. Und zwar solche, die man eingeht, um sich einen eventuellen Vorteil der Konkurrenz gegenüber zu verschaffen, beispielsweise durch den Einsatz neuer Technologien, deren Auswirkungen noch nicht ganz erforscht sind. Es besteht also ein Zusammenhang zwischen dem Risiko das man eingeht und der Ausnutzung des Chancenpotenzials. Das Rikomanagement hilft dem Projektmanager dabei abzuwägen, ob es sinnvoll ist, ein bestimmtes Risiko bewusst einzugehen oder nicht.

2 Ziele des Risikomanagements

Ziel des Risiko Managements ist es, negative Effekte der Risiken gegenüber dem potenziellen Gewinn an Möglichkeiten abzuwägen. Dazu müssen Risiken zunächst systematisch identifiziert und bewertet werden um Maßnahmen zu ergreifen, die die erkannten Risiken mindern, bevor sie zu einer Gefahr für das Projekt werden. Da während der Entwicklungszeit neue Risiken auftreten oder alte beseitigt sein können, beschäftigt man sich nicht nur am Anfang des Projektes mit den Risiken. Es ist vielmehr ein fortlaufender Prozess, weswegen man auch vom fortlaufenden Risikomanagement (*Continuous Risk Management*) spricht. Das stellt sicher, dass bestimmte Veränderungen im Projekt mit ihren Auswirkungen auf die Planung und daraus resultierende Effekte rechtzeitig identifiziert werden können, um erneut einen Handlungsspielraum zu schaffen. Ziel des Software-Risikomanagements ist es, die Wechselbeziehungen zwischen Risiken und Erfolg zu formalisieren und in anwendbare Prinzipien und Praktiken umzusetzen.

3 Schritte des Risikomanagements

Das Risikomanagement ist in drei Schritte unterteilt. Am Anfang steht die Risikoanalyse in der die Risiken identifiziert und bewertet werden. Wenn man die Risiken kennt, kann man im zweiten Schritt Maßnahmen entwickeln um diese Risiken zu handhaben, welche wiederum bewertet werden. Dieser Schritt ist die Risikoabsicherung. An dritter Stelle steht die Risikoüberwachung. Der Ablauf ist in Abbildung 1 beschrieben.

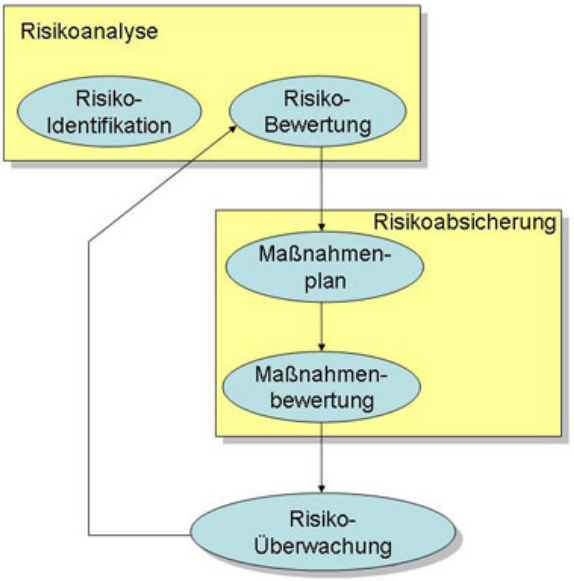

Abbildung 1: Schritte des Risikomanagements

3.1 Risikoanalyse

3.1.1 Risikoidentifikation

Der erste Schritt im Risikomanagement ist die Identifikation möglicher Risiken. Hierbei sollte allerdings noch keine Bewertung stattfinden, wobei in einigen Fällen, aufgrund der Erfahrungen des Managers, manche als sehr unwahrscheinlich eingestufte Risiken von vornherein keine Beachtung finden. Die erkannten Risiken werden üblicherweise in bestimmte Arten unterteil, sodass man sie besser einordnen kann. Hierbei unterscheidet man zwischen technologischen, personen-bezogenen und unternehmensbezogenen Risiken, sowie Risken durch Werkzeuge, Anforderungs- und Schätzrisiken.

- *Technologische Risiken* ergeben sich aus den Hard- und Softwaretechnologien,

die bei der Entwicklung des Projektes zum Einsatz kommen. Hierbei kann es sich um selbst Entwickelte Rahmenwerke handeln, die die Funktionalität des eigentlich Projektes einschränken, weil sie weniger leisten als eigentlich erwartet.

- *Personalrisiken* hängen mit den Personen zusammen, die direkt am Projekt beteiligt sind. Das Personal ist einer der wichtigsten Faktoren in der Softwareentwicklung und Softwarewartung und birgt zugleich die meisten Risiken. Das Beste was einem passieren kann ist es, Leute zu haben, die genau wissen was sie tun, die Disziplin besitzen, es zu tun und die Courage haben, die Wahrheit bezüglich möglicher Probleme auch dann zu sagen, wenn sie nicht gerne von Projektmangern und -leitern gehört wird. Besondere Beachtung verdient der letzte Punkt, da er es den Projektmanagern erlaubt, die Situation immer realistisch einzuschätzen und Handlungsspielräume schafft, bevor es zu spät ist und man sich in einer Katastrophe wieder findet. Beispiele für mögliche Personalrisiken sind:

  - man hat nicht die Leute mit dem fürs Projekt nötige Know-how
  - man hat zwar die richtigen Leute aber sie erbringen nicht die erwartete Leistung. Das kann unter Umständen daher rühren, dass bestimmte Entwickler wegen Krankheit oder sonstigen Faktoren nicht für das Projekt verfügbar sind.

- Risiken, die aus finanziellen oder Problemen im Management resultieren werden als *unternehmensbezogene Risiken* eingestuft. Dies ist der Fall, wenn beispielsweise Umstrukturierungen im Unternehmen stattfinden, die mit dem Projekt in Verbindung stehen.

- Benutzt man Werkzeuge zur Unterstützung der Projektentwicklung, so können dort ebenfalls Risiken auftreten. Man spricht von *Risiken durch Werkzeuge*.

- *Anforderungsrisiken* stehen auch im Zusammenhang mit dem Personal. Das Personal könnte mit dafür verantwortlich sein, dass das fertige Produkt die Anforderungen nicht erfüllt. Allerdings liegt hier der Schwerpunkt auf dem Produkt und dem Kunden. Ändert er die Anforderung, so können große Veränderungen im ganzen Entwurf anfallen.

- Da man selten unendlich viel Zeit für ein Projekt hat wird bei der Planung eine Zeitplan festgelegt. Dieser ist allerdings nur eine Schätzung. Diese Schätzung kann überschritten werden, also spricht man von *Schätzrisiken*. Wie gut diese Schätzungen sind liegt in der Verantwortung des Projektmanagements. Der Projektmanager sollte unter keinen Umständen einen Zeitplan und das Budget von einer höheren Instanz im Unternehmen diktiert bekommen. Fehlt die Planung völlig oder bekommt man einen willkürlichen Zeitplan auferlegt, so resultiert das mit großer Wahrscheinlichkeit in einem Problem.

3.1.2 Risikobewertung

Hat der Manager nun alle Risiken erkannt und eingeordnet, muss er diese Risiken bewerten. Dabei gilt es abzuschätzen, wie hoch die Eintrittswahrscheinlichkeit eines Risikos und die Abwägung seiner Auswirkung ist. Es stellt sich jedoch als recht schwierig heraus, genaue Schätzungen zu bestimmen, da es keinen allgemeinen Weg hierfür gibt. Alles beruht auf den Erfahrungen des Managers und seiner Fähigkeit, mit den verfügbaren Informationen über die möglichen Risken umzugehen. Generell gilt jedoch, je mehr Informationen über das Projekt, die Entwicklung, das Entwicklungsteam und das Unternehmen verfügbar sind, desto einfacher ist es, die Eintrittswahrscheinlichkeit und die Auswirkung eines Risikos abzuschätzen.

Im fortlaufenden Risikomanagement kommt noch hinzu, dass sich Informationen während der Projektplanungs- und Entwicklungszeit nicht nur neue Risiken hinzukommen, sondern auch weitere Informationen bezüglich der bereits identifizierten und analysierten Risiken verfügbar sein können. Dies sollte dann wiederum in die Risikobewertung einfließen um zu jedem Zeitpunkt eine möglichst realistische Bewertung zu gewährleisten.

Typischerweise werden die nun erkannten und bewerteten Risiken in Tabellen zusammengefasst. Dabei werden sie nach der Bedeutung des Risikos sortiert, so dass die potenziell für das Projekt gefährlichsten Risiken am Anfang der Tabelle wieder zu finden sind.

Wie viele Risiken letzten Endes betrachtet werden hängt stark mit dem Projekt und seinen Anforderungen zusammen. In kritischen Projekten, die besondere Aufmerksamkeit verlangen, werden mit Sicherheit mehr Risiken berücksichtig, so unbedeutend sie zu sein scheinen, als es bei einem kleineren nicht ganz so kritischen Projekt der Fall wäre. Es gibt also keine für jedes Projekt gültige absolute Zahl. In der Praxis werden in der Regel die Top 10 Risiken verfolgt.

3.2 Risikoplanung

3.2.1 Maßnahmenplan

Nachdem in der Risikoanalyse alle Risiken erfolgreich analysiert wurden, und man nun weiß, womit man es zu tun hat, kann man zum nächsten Schritt übergehen: zur Risikoplanung. Hierbei geht es in erster Linie um den Umgang mit den Risiken. Dabei gibt es wieder keinen allgemeinen Weg, den man verfolgen kann, denn auch hier ist das Urteilsvermögen und die Erfahrung des Projektmanagers gefragt. Allerdings gibt es einige Strategien, die man verfolgen kann:

- *Vermeidungsstrategie:* Die Vermeindungstrategie zielt auf die Eintrittswahrscheinlichkeit des Risikos. Es wird also versucht Vorkehrungen zu treffen, die das Eintreten des Risikos unwahrscheinlicher machen. Stellt zum Beispiel ein ver-

wendetes Tool ein Risiko dar, da es mögliche Fehler beinhaltet, so wäre es eine Reduktion der Eintrittswahrscheinlichkeit, ein anderes Tool für den gleichen Zweck aber mit bekannter Zuverlässigkeit zu benutzen.

- *Minimierungsstrategie:* Die Minimierungsstrategie verfolgt das Ziel, die Auswirkungen des Risikos auf ein Minimum zu reduzieren. Beispielsweise wäre bei dem Risiko *Personal-Krankheit* eine Reduktion der Auswirkung, wenn das Team so organisiert ist, dass es mehr Überschneidungen bei den Entwicklern gibt, und diese die Arbeit der anderen Entwickler dadurch besser verstehen. Wäre also ein Entwickler für einen kurzen Zeitraum nicht verfügbar, könnten die anderen seine Arbeit übernehmen.

- *Notfallplanung:* Wie der Name schon vermuten lässt ist die Notfallplanung für den Fall gedacht, dass der worst-case, also ein Risiko mit seinen schlimmsten Auswirkungen eintritt. Ein Manager muss auf diesen Fall vorbereitet sein, so dass eine direkte Reaktion erfolgen kann, um dem entgegenzuwirken.

3.2.2 Maßnahmenbewertung

Da der Maßnahmenplan selten ohne zusätzliche Kosten von statten geht, muss genau festgehalten und bewertet werden, welche monetären Auswirkungen jede einzelne Maßnahme mit sich bringt.

3.3 Risikoüberwachung

Die dauerhafte Überprüfung der Risiken und der Änderungen ihrer Eintrittswahrscheinlichkeiten und ihrer Auswirkungen ist Gegenstand der Risikoüberwachung. Dieser Schritt ist das Kernstück des fortlaufenden Prozesses des Riskomanagements und ist von großer Wichtigkeit. Allerdings gibt es Schwierigkeiten, die Änderungen unmittelbar zu überwachen, denn man ist, wie bei der Analyse auch, auf Informationen aus dem Umfeld angewiesen, die Hinweise auf Risikoveränderungen geben.

4 Software Risk Evaluation - SRE

Für das Risikomanagement in der Softwareentwicklung werden einige mehr oder weniger formalisierte Methoden eingesetzt. Das Software Engeneering Institute (SEI) beschäftigt sich seit 1992 mit Risikomanagement und hat einige Formalismen entwickelt, die im Rahmen der Softwareentwicklung dienlich sind. Ein Ergebniss ist die Software Risk Evaluation (SRE). SRE ist sowohl eine stand-alone Diagnose-Lösung, die einem Unternehmen den Erfolg seiner Projekte sichern kann, als auch ein solides Fundament für das Riskomanagement. Eine zentraler Rolle in der SRE haben Interviews, mit deren Hilfe Informationen bezüglich der Probleme erfasst werden können. SRE basiert auf den folgenden Prinzipien:

- *Globale Perspektive*: Softwareentwicklung sollte im Kontext der Definitionen, des Design und der Entwicklung der übergeordneten Systemebenen betrachtet werden. Dies ermöglicht es, einen Blick für den potenziellen Nutzen der Möglichkeiten und ihrer negative Effekte zu bekommen.

- *Vorausschauende Betrachtungsweise*: sie macht es möglich, Ungewissheiten frühzeitig zu erkennen und deren Ausmaße einzuschätzen.

- *Offene Kommunikation*: Der Kommunikationsaustausch zwischen allen Projektebenen sollte ermutigt werden. Jeder, der am Projet beteiligt ist, sollte Beachtung finden damit mehr Informationen für die Prozesse des Risikomanagements verfügbar sind.

- *Integriertes Management*: Man sollte dem Risikomanagement eine tragende Rolle geben und die Projektinfrastruktur so anpassen, dass seine Methoden und Tools aufgenommen werden können.

- *Fortlaufender Prozess*: Risikomanagement muss fortlaufend stattfinden. Zu jeder Zeit sollte es möglich sein, Risiken zu identifizieren und zu analysieren.

- *Gemeinsame Produktvorstellung*: Sie sorgt für eine durchgängige Einigkeit und ein gemeinsames Ziel, dass alle am Projekt beteiligten vor Augen haben.

- *Teamwork*: Die gemeinsamen Vorstellungen können durch Teamarbeit und Aufteilung der einzelnen Fähigkeiten der Projektmitarbeiter erreicht werden

Unterteilt wird der Prozess in fünf Phasen, wie in Abbildung 2 dargestellt. Sie werden im Folgenden eingehender betrachtet.

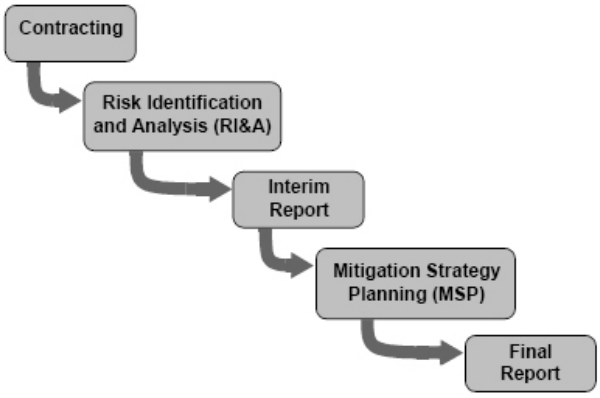

Abbildung 2: Die fünf SRE Phasen [Wil99]

4.1 Contracting

Die erste Phase, das *Contracting*, ist auch zugleich eine der wichtigsten. Werden hier die Rollen der Beteiligten richtig gewählt und Ziele und Erwartungen an das SRE vernünftig gesetzt so ist der Weg für ein erfolgreiches SRE geebnet. Bevor mit den Maßnahmen begonnen werden kann, muss SRE formal in Auftrag gegeben werden. Der Auftraggeber ist im Allgemeinen auch der Projektmanager. Er trägt die Verantwortung für die Organisation des SRE, muss sämtliche Ressourcen zur Verfügung stellen und aktiv den Prozess unterstützen. Es ist nicht damit getan, dem Projektpersonal mitzuteilen, dass jetzt "Risikomanagement" gemacht werden soll. Eine dieser Verantwortungen ist es, das SRE-Team um sich aufzubauen, welches in der Regel aus den folgenden Rollen besteht:

Site-Visit Coordinator: Der Site-Visit Coordinator hat die Aufgabe, alle nötigen Vorkehrungen für SRE zu treffen. Diese Position ist idealerweise mit einem Verwaltungsassistenten oder einem erfahrenen Mitarbeiter belegt, der mit der Koordination von Personal- und Zeitplänen vertraut ist. In der Regel ist es eine schlechte Idee technisches Personal oder Ingenieure für diesen Job auszuwählen.

"Risk-Management Champion": Der Auftraggeber sollte eine Person bestimmen, die das "Gewissen" der Risikomanagement-Aktivitäten verkörpert. Damit das restliche Projektpersonal die Wichtigkeit des Risikomanagements erkennt, sollte die Position des "Risk-Management Champion" mit einer bereits respektierten Person belegt werden. Diese Person wird dafür sorgen, dass Risiken bei Sitzungen ein Thema sind und dass Risiko-Aktivitäten aufrechterhalten und jeder Zeit sichtbar sind. Außerdem ist der "Champion" dafür verantwortlich, dass Risiko-Informationen sowohl in Richtung Auftraggeber als auch in Richtung Projekt-Personal weitergeleitet werden.

SRE-Team-Mitglieder: Des Weiteren wird ein SRE-Team benötigt, welches aus erfahrenen und fähigen Leuten aus dem Unternehmen bestehen sollte. Die Hauptaufgabe des Teams ist es, ein klares und verständliches Bild über die Risiken, die das Projekt betreffen, zur Verfügung zu stellen.

SRE-Teilnehmer: Die Teilnehmer sind meist Personen, die am Projekt beteiligt sind. Sie geben das Feedback vom Standpunkt der Entwickler und sind offensichtlich recht wichtig, da sie eine der Hauptinformationsquellen sind. Daher liegt es in der Verantwortung des Projektmanagers diese Leute sorgfältig auszusuchen. Die Qualität der SRE schwankt stark mit den Auswahl der Teilnehmer.

Das Resultat der ganzen Aktivitäten und Aufwendung ist einzig Eigentum des Projektmanagers. Dieser bestimmt alleine darüber, ob und an wen das Resultat weiter gegeben wird. Selbst der Auftraggeber, sofern er nicht zugleich der Projektmanager ist, hat kein Anrecht auf die Resultate.

Hat man alle Positionen belegt, folgt die Zeitplanung für die SRE Aktivitäten. Dabei gibt es ziemlich genaue Richtlinien zur Zeitplanung der Risikoidentifikations- und Anaylsephase (RI&A). Beispielsweise wird eine mindestens vierstündige Sitzung vor-

geschlagen, um dem SRE-Team die technischen Herausforderungen, die Organisation und die Zeitplanung des Projektes zu veranschaulichen. Ebenfalls vierstündig fällt jedes Interview mit einem SRE Teilnehmer und die Team-Analyse aus. Außerdem gibt es noch weitere zeitliche Richtlinien.

Um Risiken einordnen zu können, werden Metriken benötigt. Diese werden ebenfalls im Contracting beschrieben. Die eingesetzten Metriken sind:

- *Risikoausmaß*: Bewertung auf einer Skala von 1 bis 4. 4 ist Katastrophal, 3 kritisch, 2 marginal und 1 vernachlässigbar.
- *Eintrittswahrscheinlichkeit*: Hierbei ist 3 sehr wahrscheinlich, 2 wahrscheinlich und 1 unwahrscheinlich.
- *Risikogefährlichkeit*: ist eine Funktion, die sich aus den oberen beiden Werten ergibt auf einer Skala von 1 bis 6, wobei die Werte 6 bis 5 für *Hoch*, 4 bis 3 für *Mittel* und 2 bis 1 für *Gering* stehen.

Der Projektmanager kann diese Risikogefährlichkeits-Einstufungen dann in einer Matrix (Abbildung 3) zusammenfassen, die zusätzliche Informationen über den Risikobereich beinhaltet.

Component→ Category ↓	Performance	Support	Cost	Schedule
Catastrophic	nonachievement of technical performance	unsupportable software	major budget overrun (>50%)	unachievable IOC
Critical	significant degradation of technical performance	major delays in software modifications	serious budget overrun (~30%)	serious delay in IOC (>30% late)
Marginal	some reduction in technical performance	minor delays in software modifications	budget overrun (~10%)	delay in IOC (>10% late)
Negligible	minimal to small reduction in technical performance, at detail level	irritating and awkward maintenance	consumption of some budget cushion	consumption of some slack—not on critical path

Abbildung 3: Risikogefährlichkeits-Matrix [Wil99]

4.2 Risk Identification and Analysis

4.2.1 Vorbereitung

Stehen nun alle Formalismen fest, kann man mit der Identifikations- und Analysephase beginnen. Sie besteht hauptsächlich aus Befragungen der Projekt-Teilnehmer. Begonnen wird mit einem vom Projektmanager herbei geführtem Briefing, um dem SRE-

Team einen Überblick über das Projekt zu geben und ihnen sowohl sie Ziele als auch den Zweck des Projektes näher zu bringen. Es steht dem Projekt-Manager frei, zusätzlich zum SRE-Team auch andere Leute zu diesem Briefing einzuladen. Das Ergebnis dieses Briefings ist, dass alle SRE-Teammitglieder verstehen, welches Produkt wann fertig sein muss, wie die organisatorische Struktur des Projektes mit all seinen Rollen funktioniert, welche technischen Herausforderung es zu bewältigen gibt, und nicht zuletzt, welche Kosten und Einschränken man hat.

Sobald das SRE-Team mit dem Projekt vertraut ist, beginnt die erste für das Projekt sichtbare Handlung. Das Eröffnungs-Briefing für alle Projektteilnehmer wird abgehalten. Inhalt des Briefings ist in der Regel die Vorstellung des SRE-Teams, und eine Einführung in die Ziele und Methoden des SRE. Es wird erklärt, welche Informationen von den Projekt-Teilnehmern benötigt werden, wie sie formuliert sein müssen und was mit den Informationen danach passiert. Außerdem wird der Interview-Zeitplan der Projektteilnehmer vorgestellt.

Bevor es aber mit den Interviews losgeht, wird das SRE-Team noch einmal vorbereitet. Danach sollte dann jeder wissen, welche Rolle er hat und was für Vorgaben er in der Identifikations- und Analysephase zu befolgen hat.

4.2.2 Interviews

Die Risikoidentifikations- und Analysephase kann beliebig viele Interview Sitzungen haben. Jede Sitzung ist drei Stunden lang und wird in zwei Teile aufgespaltet. Die ersten 2,5 Stunden dienen der Risikoidentifikation die restliche halbe Stunde wird dazu benutzt, die gewonnenen Ergebnisse von den Interviewten bewerten zu lassen. Die Interviews sind die Hauptinformationsquelle für das SRE und daher von großer Wichtigkeit für den Erfolg der SRE.
Das Ergebnis eines Interviews sind 15-40 *Risk-Statements*, eine inhaltliche Zusammenfassung der Sitzung und von den Interviewten ausgefüllte Bewertungsformulare. Risk-Statements haben die Form “Ursache ; Wirkung”. Doch wie sieht die Rollenverteilung in so einer Sitzung aus? Vom SRE-Team werden 4 Leute benötigt:

- *Interviewer*: stellt die Fragen während der Sitzung und hält sich an ein bestimmtes Schema, welches ich später erläutern werde.
- *Risikoprotokollführer*: hilft dem Interviewten dabei, seine Risk Statements zu formulieren und notiert sie für alle Sichtbar auf einer Tafel.
- *Sitzungsprotokollführer*: protokolliert Diskussionen und nichtverbale Kommunikation um die knapp formulierten Risk-Statements später richtig verstehen zu können. Manchmal sind die Risk-Statements zwar klar formuliert, aber wenn man den Zusammenhang nicht mehr vor Augen hat könnte es Missverständnisse bei der Auswertung geben. Der Sitzungsprotokollführer sollte jedoch darauf achten, die Risk-Statements und Fragen nicht zu protokollieren. Es ist viel wichtiger die Diskussion so gut wie möglich festzuhalten, denn alles andere wird anderweitig protokolliert oder wurde bereits vor der Sitzung formuliert.

- *Datenübersetzer*: schreibt die Risk-Statements in eine Tabelle und erstellt die für den nächsten Schritt benötigten Bewertungsformulare.

Die Befragung läuft immer nach einem fixen Schema ab und besteht aus 4 Schritten:

- Schritt 1: Der Interviewer liest seine Fragen exakt so vor, wie er sie auf seinem Hilfszettel notiert hat. Gibt die Antwort Grund zur Sorge, dann wird mit Schritt 3 weitergemacht.
- Schritt 2: Stößt man in Schritt 1 auf keine Probleme bezüglich der Frage so kann der Interviewer an dieser Stelle weitere Fragen diesbezüglich stellen. Ergeben sich immer noch keine Probleme kehrt man zu Schritt 1 zurück.
- Schritt 3: Ist der Interviewer auf ein Problem gestoßen, versucht er der Sache näher auf den Grund zu gehen und fordert den Interviewten auf, dieses Problem als Risk-Statement zu formulieren.
- Schritt 4: Der Risikoprotokollführer geht an die Tafel, schreibt das Risk-Statement auf und überprüft mit dem Interviewten, ob das genau seine Aussage ist oder ob er etwas anderes meint. Ist das Risk-Statement von allen verstanden und vom Interviewten bestätigt, kann der Interviewer mit der nächsten Frage fortfahren (Schritt 1).

Unmittelbar nach der Risikoidentifikation werden die Interviewten aufgefordert, die gerade verfassten Risk-Statements zu bewerten. Dabei werden für jedes Statement die Eintrittswahrscheinlichkeit und das Ausmaß angegeben, nach der vom Projektmanager in der Contracting Phase bestimmten Matrix (Sihe Abbildung 3). Außerdem ist es noch nötig, die Risiken nach Wichtigkeit zu ordnen. Nur die Top-Risiken finden in der SRE Beachtung. Nach der Sitzung bewertet das SRE-Team die erarbeiteten Statements. Die Ergebnisse werden abschließend zusammengefasst und in einem Diagramm in die gesamten Risiken, die Top-Risiken des SRE Teams und die, der Projekt Teilnehmer dargestellt, aufgeschlüsselt.

4.3 Interim Report

Während dieser Phase werden die Resultate aus der vorhergegangenen Phase erneut ausgewertet. Diesmal aus einer anderen Sichtweise: von Interesse ist der Zusammenhang der Risiken und deren Auswirkung auf einander. Alle Ergebnisse aus der Risikoidentifikation und -analyse fließen mit in den Interims-Report ein. Dieser stellt auch die Basis für den nächsten Schritt, die MSP-Phase (Mitigation Strategy Planning), in der Strategien zur Risikominderung und -umgehung geplant und überprüft werden. Der Report enthält sowohl einen Zeitplan für die nächste Phase als auch Vorschläge, für welche Risiken als erstes Gegenmaßnahmen und Umgehungsstrategien entworfen werden sollen.

4.4 Mitigation Strategy Planing

In dieser Phase wird ein konkreter Plan entwickelt, der Aktionen und Strategien enthält um die in der zweiten Phase erkannten und analysierten Risiken abzuschwächen oder zu umgehen. Dazu bedient man sich in insgesamt vier Schritten einiger Methoden und Metriken, um die Risiken handhabbar zu machen und die erarbeiteten Pläne auf ihre Wirksamkeit zu überprüfen. Im ersten Schritt findet ein Team-Briefing statt um die Rollenverteilung für die weiteren Schritte jedem Team-Mitglied verständlich zu machen. Danach geht man über in die MSP-Sitzungen in denen effektiv die Risikobereiche angesprochen werden und durch strukturiertes Brainstorming und anderen Methoden Gegenmaßnahmen entwickelt werden. Stellt sich heraus, dass einige Maßnahmen sich überschneiden, sich gegenseitig ausschließen oder in einer Wechselwirkung zu einander stehen, so muss eine Bereichsübergreifende-Strategie-Sitzung abgehalten werden in der Abhängigkeiten aufgelöst und Konflikte und Redundanzen beseitigt werden. Diese Sitzung ist jedoch optional. Abschließend wird die MSP-Resultats-Sitzung abgehalten, in der die erarbeiteten Lösungen präsentiert und logistische und organisatorische Fragen geklärt werden.

An den MSP-Sitzungen sind das SRE-Team, die Risikobesitzer aus Phase 2 und jemand, der für das Sitzungsprotokoll aus Phase 2 verantwortlich ist, beteiligt. Das Ganze wird von einem Moderator geleitet. Aufgabe des Moderators ist es, durch gezielte Frage eine Diskussion mit den Beteiligten zu entfachen, deren Resultat adäquate Lösungen darstellt. Dabei sollten seine Methoden möglichst flexibel sein damit er auf die Probleme, die sich ergeben, eingehen kann. Der Zeitplan für die Sitzung ist nur eine recht grobe Vorgabe, denn es ist teilweise nötig, ihn während der Sitzung anzupassen, damit alle Risikogebiete ausgiebig erörtert und Lösungen erarbeitet werden können. Die oben angesprochenen Metriken stellen allerdings ein Problem dar, da sie schwierig zu bestimmen sind. Der Moderator muss dafür sorgen, dass man sich nicht zu lange daran aufhält. Findet man in der gegebenen Zeit keine passende Metrik, ist es sinnvoller, mit dem nächsten Risikogebiet weiterzumachen. Ähnlich verhält es sich mit Budget-Schätzungen, die aufgeschoben werden sollten. Üblicherweise bewegt sich der Zeitrahmen für diese Sitzung zwischen einem halben und einem ganzen Tag.

Die *bereichsübergreifende Stratetegie-Sitzung* benötigt die gleichen Rollen wie die MSP-Sitzungen. Die Sitzung ist nur dann nötig, wenn alle Projekt-Teilnehmer und SRE-Teammitglieder nicht an allen MSP-Sitzungen anwesend waren oder die Risikobereiche nicht deutlich von einander abgegrenzt sind. Das heißt also, dass im Vorhinein vom SRE-Team überprüft werden sollte, ob sich irgendwelche Konflikte, Abhängigkeiten oder Redundanzen in den Gegenmaßnahmen befinden. Ist das der Fall, so wird die Sitzung einberufen und läuft ähnlich ab wie die MSP-Sitzung. Auch hier muss der Moderator dafür sorgen, dass effektive Lösungen gefunden werden. Spätestens hiernach sollte man vernünftige Gegenmaßnahmen gefunden haben.

Die *MSP-Resultats-Sitzung* ist der letzte Schritt dieser Phase. Abgesehen von der Präsentation der Lösung werden hier Fragen diskutiert, die die Umsetzung der Gegenmaßnahmen betreffen. Beispielsweise wird herausgefunden, welche Personal-Ressourcen, Autorisierungen, monetäre Aufwendungen und weitere organisatorische Dinge nötig

sind.

4.5 Final Report

Abgesehen vom abschließenden Report beinhaltet der letzte Schritt der SRE die Zusammenstellung der Daten für die Projektdatenbank, die Kundenpräsentation des Reports und ein abschließendes Meeting. Der Abschlussreport wird vom Interimsreport abgeleitet und beinhaltet die Zusammenfassung aller Resultate aus den vorangegangenen Phasen. Dieser kann dann dem Kunden vorgestellt werden. Wie bereits erwähnt gehören alle Resultate der SRE dem Projekt-Manager und es steht ihm zu, diese für sich zu behalten. Jedoch empfiehlt das SEI ausdrücklich dem Kunden die Resultate zu präsentieren. Das abschließende Meeting würde diese Möglichkeit bieten. Abgesehen davon ist die Sitzung im Sinne der SRE-Verbesserung, da dort finale Fragen zum SRE geklärt und diskutiert werden können.
Da die SRE ein Teil des fortlaufenden Risikomanagements ist, ist es notwendig die Resultate in die Projekt-Datenbank zu geben, damit diese Daten für dem weiteren verlauf des Risikomanagements eingesehen werden können.

5 Zusammenfassung

Ich denke man hat gesehen, dass Risikomanagement gerade in großen Projekten von großer Wichtigkeit ist. Es bietet viele Möglichkeiten, Software zu verbessern, da zum einen Probleme von Anfang an umgangen werden und zum anderen das Chancenpotenzial maximiert werden kann. Allerdings gibt es kaum Stand-Alone-Tools, die diesen Prozess unterstützen. Meistens ist es nur ein Teil der Projektmanagement-Tools, aber es würde durchaus Sinn machen, eigene Risikomanagement-Tools einzusetzen. Diese könnten besser an die Bedürfnisse des fortlaufenden Risikomanagements angepasst sein, und dadurch effektiver arbeiten.

Literatur

[Bal98] Balzert. *Lehrbuch der Softwaretechnik, Band 2*. 1998.

[Bur00] Burghardt. *Projektmanagement*. 2000.

[web] *Webster's Dictionary*.

[Wil99] Behrens Williams, Pandelios. *Software Risk Evaluation (SRE) Method Description (Version 2.0)*. 1999.

Change- und Configuration Management

Andreas Dornbusch

Inhaltsverzeichnis

1 Configuration Management

1.1 Ziele des Configuration Management

Bei einer Software-Entwicklung in großem Umfang tauchen immer wieder Fehler und Probleme auf, welche insbesondere auf der Tatsache beruhen, dass mehrere Entwickler zur gleichen Zeit am selben Projekt arbeiten. Einen Problembereich stellt die Fehlerbehebung und die häufige Änderung der Software dar. Oft ist nicht nachvollziehbar, ob und von wem ein Fehler bereits korrigiert wurde oder welche Änderungen an der Software vorgenommen wurden. Software Configuration Management soll diese Probleme verhindern, indem der gesamte Änderungsprozess einschließlich der Änderungswünsche und der vorgenommenen Änderungen dokumentiert wird. Ebenso problematisch ist es nachzuvollziehen, welche Änderungen fehlerhaft waren und daher aus der Konfiguration (siehe 1.2) entfernt werden müssen oder welche Eigenschaften bestimmte Software-Elemente zu einem bestimmten Zeitpunkt haben. Das Configuration Management soll helfen, ein Produkt mit bestimmten Eigenschaften zu konfigurieren und dabei sicherzustellen, dass diese konsistent zueinander sind. Ein weiterer Grund für den Einsatz des Configuration Management ist das Übersetzen und Binden der Bestandteile, um keinen Arbeitsschritt bei der Bearbeitung der Software auszulassen.

1.2 Begriffe

Ein fertiges Produkt besteht aus mehreren Software-Elementen. Dabei ist ein *Software-Element* ein eindeutig bezeichneter und identifizierbarer Bestandteil der Software (z.B. Quellen, Handbücher, Testfälle, etc.). Eine *Konfiguration* ist eine konsistent zusammenpassende Menge an Software-Elementen (z.b. Test-Konfiguration, Kunden- Konfiguration). Um zu dokumentieren, welche Elemente zu einer Konfiguration gehören, wird ein Konfigurations-Identifikationsdokument erstellt. Dabei werden nicht nur Bestandteile, welche später beispielsweise an den Kunden ausgeliefert werden, aufgeführt, sondern auch jene Elemente (Werkzeuge, Hilfsmittel), die zur Erstellung benötigt werden [Bal01]. Um eine Ausprägung eines Software-Elements eindeutig zu identifizieren, werden Versionsnummern benutzt. *Versionsnummern* bestehen dabei meist aus zwei Teilen: der Release-Nummer und der Level-Nummer (in der Regel durch einen Punkt getrennt). Bei jeder gravierenden Änderung an dem Software-Element wird die Release-Nummer erhöht und die Level-Nummer auf 0 gesetzt, bei kleineren Änderungen die Level-Nummer erhöht. Wenn eine neue *Version* eines Elementes entsteht, wird zwischen zwei Arten von Versionen unterschieden: Zwischen der *Variante*, welche alternativ zum Ausgangsdokument verwendet werden kann, und zwischen der *Revision*, die statt des Ausgangsdokumentes verwendet wird[Bal01]. Ein elementarer Begriff bei CM-Systemen (vor allem bei den nachher vorgestellten) ist das *Repository*. Dieses erfüllt die Funktion einer zentralen Dateibibliothek. Im Repository befinden sich alle Dateien, die unter der Kontrolle des CM-Systems stehen. Ändert ein Entwickler eine Datei, so wird eine neue Version angelegt (anstatt die Ursprungsversi-

on zu ersetzen). Die ursprüngliche Version bleibt erhalten. In einem Repository werden somit alle Versionen der Software-Elemente inlusive Informationen wie ”Wer hat wann was geändert?” gespeichert. Am häufigsten wird das *Checkin/Checkout-Modell* zur Verwaltung von Versionen benutzt. Dabei wird zwischen Checkout-(Ausbuche-) und Checkin-(Einbuche-) Operationen unterschieden. Eine Checkout-Operation reserviert das Element für den Ausbucher und übergibt ihm eine Kopie, welche er ändern kann. Wenn der Ausbucher fertig ist, löscht die Checkin-Operation die Reservierung und dokumentiert die Änderungen (einschließlich Autor und Einbuchungszeitpunkt). Möchte während einer Reservierung ein anderer Entwickler eine Ausbuchung vornehmen, wird entweder eine Fehlermeldung angezeigt oder eine neue Verzweigung hinzugefügt (mehr dazu bei den einzelnen Systemen, Abschnitt 3). Dadurch werden Änderungen nicht überschrieben[Bal01].

1.3 Change Management

Das Change Management befasst sich mit dem Erfassen und Analysieren von Änderungsanträgen bezüglich einzelner Software-Elemente eines Produktes. Dazu werden die Änderungsanträge (Change-Requests) erfasst, archiviert und an entsprechende Stellen weitergeleitet. Im Allgemeinen können Änderungsanträge von allen beteiligten Personen (Entwickler, Projektmanager, Kunden) kommen. Das sogenannte Change Control Board (Projektleiter, Manager, Entwickler) entscheidet, ob Änderungsanträge angenommen werden. Dabei müssen die eingereichten Anträge einer gegebenen Form folgen und zwingend bestimmte Informationen enthalten. Die folgenden Fragen liegen diesen Informationen zugrunde: Wo trat das Problem auf? Wann trat das Problem auf? Was wurde beobachtet? Wie wurde das Problem ausgelöst? Wie schwerwiegend ist das Problem? Wie hoch waren die verursachten Kosten? Auch sollte die Art des Änderungsantrags angegeben werden. Hier wird zwischen einem Fehler (”fault”, ”defect”) und einem Verbesserungs- oder Änderungswunsch unterschieden.

2 Change und Configuration Management bei verteilten Projekten

2.1 Besonderheiten und Probleme

Bei verteilten Projekten (gemeint sind in diesem Fall tatsächlich geographisch verteilte) ergeben sich weitere Probleme, die Konfigurationen und Änderungen sowie Änderungsanträge vernünftig zu verwalten. Insbesondere stellen sich an die C&CM-Werkzeuge eines solchen verteilten Projektes zusätzliche Anforderungen. Grundsätzlich strebt man natürlich auch mit einem im Rahmen eines verteilten Projektes eingesetzten CM-System die in Abschnitt 1.1 genannten Ziele an. Vor allem sollte das System nicht nur aus einer Versions- /Revisions- Verwaltung bestehen, sondern auch Release-Management genauso wie ein effektives Änderungsmanagement beinhalten.

Aufgrund der Verteilung der Benutzer kommen noch weitere Anforderungen hinzu [vdH00]:

- Mehrere Repositories sollten unterstützt werden. Dadurch wird eine zu große Abhängigkeit von einem einzigen Server verhindert. Diese Repositories sollten in bestimmten Zeitabständen synchronisiert werden.
- Eine Verbindung zu den Repositories sollte nur für Checkin-/Checkout-Operationen benötigt werden (nicht permanent), um den Server zu entlasten und erhöhte Verbindungskosten auf der Clientseite zu vermeiden.
- Vor allem bei großen Projekten wäre eine private Versionsverwaltung für einzelne Entwickler sinnvoll. Das heißt ein Entwickler kann seine momentanen Arbeiten über das CM-System verwalten, ohne dass eine Veränderung an den öffentlich sichtbaren Entwicklungszweigen sichtbar ist, bis der Entwickler seine Arbeit freigegeben hat. Insbesondere bei früheren Systemen musste der Entwickler seine nur ihn betreffenden Entwicklungsschritte an seinem eigenen Arbeitsplatz verwalten. Diese waren also bis dahin kein Bestandteil der Entwicklung.
- Selbstverständlich sollte das parallele Arbeiten mehrerer Entwickler am gleichen Software-Element unterstützt werden. Es sollte also von einem einfachen Checkout-System, welches ein Element gleichzeitig nur einem Clienten zur Verfügung stellt, abgesehen werden.
- Das System sollte zumindest auf Clientseite weitgehend Betriebssystem und Hardware unabhängig sein. So läßt sich das System besser in bestehende Entwicklungsumgebungen integrieren.

3 Werkzeuge des C&CM bei verteilten Projekten

3.1 Vorstellung von Werkzeugen für Configuration Management

3.1.1 CVS

CVS (Concurrent Versions System) ist ein CM-Werkzeug, das auf RCS (Revision Control System, bietet lediglich einfache Version-/Revisionsverwaltung) basiert. Bedient wird CVS über die Kommandozeile. Zusätzlich zu der Benutzerführung durch die Kommandozeile existieren auch grafische Frontends für CVS (z.B. WinCVS). CVS lässt sich zudem in viele Entwicklungs-IDEs integrieren (z.B. Eclipse). Das System bietet grundlegende Funktionen eines CM-Systems. CVS unterstützt die Markierung und Identifikation von Dateien. Ebenso lassen sich Revisionen und Varianten verwalten. Hauptsächlich werden Text-Dateien als Software-Elemente unterstützt. Für Binär-Dateien stehen nicht alle Funktionen in vollem Umfang zur Verfügung. Die verwalteten Dateien werden in einem zentralen Repository abgelegt. Verwaltete Dateien im Repository enthalten Informationen für alle Versionen (Nummern, Kommentare). Der

Inhalt der jeweiligen Version wird in Form von Delta-Speicherungen (nur die Unterschiede zwischen Versionen werden gespeichert) gesichert. Nur die aktuelle Version enthält den vollständigen Inhalt. CVS bietet ein Checkin-/Checkout-Verfahren, das paralleles Arbeiten ermöglicht. Bei einem Checkout werden Software-Elemente nicht gesperrt, sondern es wird Merging beim Checkin verwendet (siehe Abb.1). Merging bedeutet bei CVS: Wenn zwei Entwickler das gleiche Software-Element zur gleichen Zeit bearbeiten, muss der Entwickler (im Beispiel b), der als letzter eincheckt, erst sein Arbeitsverzeichnis aktualisieren. Dann kann er unter der Voraussetzung, dass keine Konflikte bestehen, einchecken. Das CVS verbindet die Entwicklungen zu einer neuen Version.

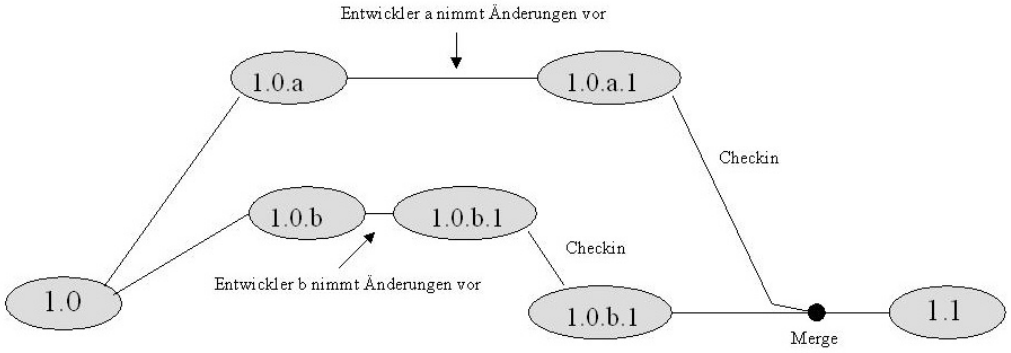

Abbildung 1: Merging bei CVS

Die Verteilung wird über ein klassisches Client/Server Konzept erreicht. Das Repository liegt auf dem Server und das jeweilige Arbeitsverzeichnis auf dem Client. Eine Replication eines CVS-Repository an verschiedenen Standorten wird leider nicht unterstützt. Ebenso fehlt eine Versionsverwaltung am Arbeitsplatz. Das heißt der entsprechende Entwickler muß entweder ein eigenes System zur Versionskontrolle (z.B. einen eigenen CVS-Server) installieren oder auf Versionskontrolle an seinem Arbeitsplatz verzichten. CVS ist ein Open-Source Projekt und für fast alle Betriebssysteme (Microsoft Windows, Linux und die meisten Unix-Derivate) erhältlich[CVS03].

3.1.2 ClearCase MultiSite

MultiSite ist eine optionale Produkterweiterung für das CM-System ClearCase von Rational. Anstatt nur ein einzelnes Repositorium wie beim CVS zu haben, wird das Repositorium an verschiedenen Standorten repliziert. Die Replicas sind dabei so angelegt, dass die Entwicklung an den einzelnen Standorten jeweils an einem anderen Zweig eines globalen Versionsbaumes stattfindet. Um die Entwicklung an den anderen Standorten darzustellen, gibt es an jedem Standort einen Teil im Repository, der die Versionen der Software-Elemente der anderen Standorte enthält. In bestimmten Zeitabständen werden Änderungen an Software-Elementen an die entsprechenden Zweige der anderen Standorte propagiert. Das bedeutet, dass es an einem Standort

verschiedene Versionen eines Software-Elementes geben kann, aber nur eine an diesem Standort veränderbar ist. Innerhalb der einzelnen Standorte ist eine parallele Entwicklung möglich. Von der technischen Seite her bietet Rational Unterstützung für die gängigsten Betriebssysteme (Windows, Linux, Unix). Auch lässt sich das System nahtlos in bestehende Entwicklungs-IDEs integrieren (z.b. Microsoft.NET oder Eclipse). Ein umfangreiches Backup-System sorgt dafür, dass das Repository im Falle eines Systemabsturzes leicht und vollständig wiederherstellbar ist. ClearCase lässt sich problemlos an die Größe der Entwicklerteams anpassen. MultiSite ist also im Gegensatz zum CVS nicht an einen Server-Standort gebunden. Ein Nachteil ist bei diesem System, dass an jedem Standort zwingend die Zweige im Versionsbaum erstellt werden müssen, welche die anderen Standorte repräsentieren. Diese Verzweigungen müssen dann unter Umständen von den Benutzern bzw. Entwicklern selber zu einer einzigen Grundversion zusammengefügt werden. Besonders falls die Abhängigkeiten zwischen einzelnen Software-Elementen verändert wurden, stellt dies ein ernstes Problem dar [RCC03].

3.1.3 Visual SourceSafe

Visual Sourcesafe von Microsoft ist ein erweitertes CM- und Versionskontrollsystem für die Entwicklung von Software, welches neben der Erstellung von Applikationen auch speziell die Erstellung von Webseiten unterstützt. VSS ist so konzipiert, dass es sich vor allem einfach in die Microsoft Visual.NET Entwicklungstools integrieren lässt. Aber auch mit anderen Entwicklungstools lässt sich VSS kombinieren, was allerdings meistens nicht unproblematisch ist. VSS verwaltet beliebige Dateitypen. Der Benutzer kann auf Datei- oder Projektebene arbeiten. Alle Dateien und ihre Historie werden in der zentralen SourceSafe Database gespeichert. VSS sperrt ein ausgebuchtes Element für andere Benutzer. Es lassen sich auch Parallele Checkout-Operationen erlauben. Dies hat sich in der Praxis aber als problematisch erwiesen. Die Benutzeroberfläche wurde dem Windows-Explorer nachempfunden. Das System läuft ausschließlich auf Windows NT/2000/XP [VSS03].

3.1.4 WWCM

WWCM (World Wide Configuration Management) ist eines der Systeme, welche die Verteilung durch das Internet erzielen. Die Clients greifen mittels eines Web-Browsers (Applet) auf das System zu, während ein Server das Repository mit den Software-Elementen verwaltet. Entstanden ist WWCM aus einem System namens WWRC (World Wide Revision Control). Die Funktionalität von WWRC beschränkte sich dabei auf Revision Management. Wirkliches Configuration Management war nicht möglich (siehe 1.1). Die Probleme lagen darin, dass WWRC vollständig auf HTML aufbaute. Dadurch war eine dynamische Interaktion mit den Benutzern schwierig. Für jede Anfrage musste eine neue HTML Seite generiert und an den Client geschickt werden. Für Verbindungen mit langen Round-Trip-Zeiten war dieses System somit völlig unbrauchbar. Außerdem ergaben sich Probleme bei der Aktualität der Seiten: Wenn eine Operation

von einem anderen Client durchgeführt wurde, welche die angezeigten Informationen beeinflusst, wird immer noch nicht die angezeigte HTML-Seite im Browser aktualisiert. Diese Probleme führten dazu, dass WWCM in JAVA entwickelt wurde. Der Zugriff auf das Repositorium erfolgt nun über ein JAVA-Applet (lediglich eine Login-Seite ist zusätzlich vor die Applet-Seite geschaltet). Aufgrund der nun vorhandenen Kommunikationsmöglichkeiten lässt sich ein konstanter Datenstrom zwischen Client und Server öffnen. Die Aktualisierungsproblematik wurde somit gelöst. Ebenso muss nicht mehr bei jeder Interaktion eine neue POST/GET-Anfrage gesendet werden. Das Benutzer-Interface beschränkt sich bei WWCM auf die nötigsten Aktionsmöglichkeiten: Der Benutzer wählt nach dem Login das zu bearbeitende Projekt und kann mit den zugehörigen Software-Elementen die typischen Checkin- und Checkout-Operationen durchführen (siehe Abb.2).

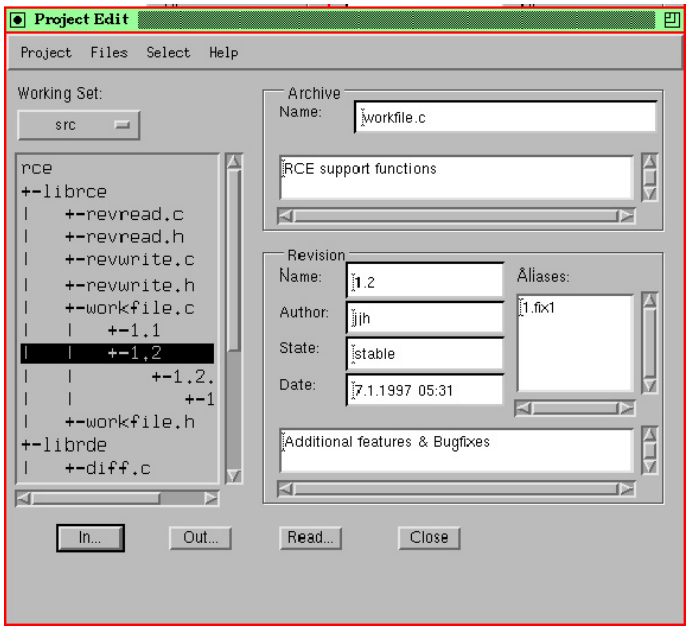

Abbildung 2: Projekt-Bearbeitung bei WWCM

In WWCM wird für diese Operationen ein Mechanismus namens TRAD (Template Regulated Alternative Development) eingesetzt. Dieser Mechanismus unterscheidet zwischen öffentlichen und privaten Revisionen. Öffentliche Revisionen sollen allen am Projekt beteiligten Entwicklern zugänglich sein, während private Revisionen nur für einen einzelnen Entwickler zugänglich bleiben, bis dieser sie öffentlich macht. Nach einer Checkout-Operation (vorgenommen an einem Element des Hauptzweiges) wird eine neue Verzweigung gebildet, auf welcher der Entwickler arbeitet bis er seine Arbeit veröffentlichen will. Neben dieser Verzweigung wird ein Platzhalter für die nächste öffentliche Revision erstellt (siehe Abb.3).

Bei der entsprechenden Checkin-Operation überprüft das System, ob der Platzhalter für die Revision ein direkter Nachfolger des ursprünglichen Vorgängers ist, und setzt in diesem Fall die neue Revision an dessen Stelle. Falls ein anderer Entwickler parallel

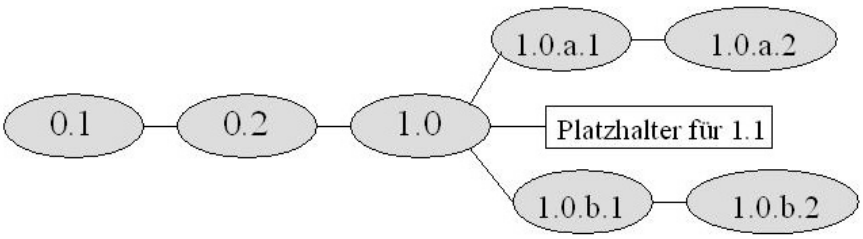

Abbildung 3: Parallele Entwicklung bei WWCM

an gleicher Stelle gearbeitet hat, entscheidet der Zeitpunkt, zu dem der erste Entwickler seine neue Revision fertig hat. Die erste neue Revision erhält die Position an dem Platzhalter. Für den zweiten Entwicklungszweig wird nun nach der neuen Revision ein Platzhalter angelegt. Ist auch der zweite Entwickler mit seiner neuen Revision fertig, erkennt das System, dass der Platzhalter nicht der direkte Nachfolger der Ursprungsrevision ist. Um zu verhindern, dass eine nicht funktionierende Revision auf dem Hauptzweig erscheint, verbindet das System die aktuelle Revision mit der gerade entwickelten und setzt diese an den privaten Zweig des Entwicklers bis diese Revision überprüft und für in Ordnung befunden wurde (siehe Abb.4).

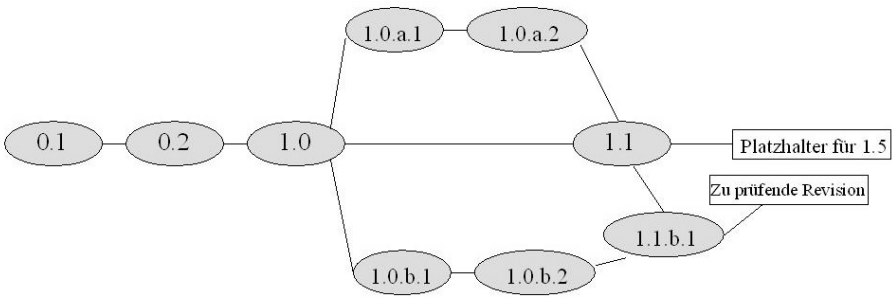

Abbildung 4: Verbinden der Entwicklungszweige bei WWCM

Beim WWCM ist sicherlich die Plattformunabhängigkeit sowie eine leichte Bedienbarkeit von Vorteil. Ein Problem ist hier die Abhängigkeit von nur einem Server [HLRT97].

3.2 Vorstellung von Werkzeugen für Change Management und Bug-Tracking

3.2.1 Bugzilla

Bugzilla stellt heute praktisch den Standard bei der Fehlerverfolgung in Open-Source-Projekten dar. Auch für kommerzielle Software-Entwicklungen wird Bugzilla immer häufiger eingesetzt. Die meisten Optionen, die kommerzielle Produkte bieten, finden sich auch hier wieder. Da Bugzilla auf der Clientseite vollständig über einen Web-Browser zu bedienen ist, kann das Programm plattformunabhängig benutzt werden. Lediglich ein temporärer Internetzugang muss vorhandensein. Auf Serverseite ist die Installation eines Webservers (vorzugsweise wird hier der Apache-Webserver unterstützt), einer MySQL-Datenbank als RDBMS und einer Perl-Distribution nötig. Am problemlosesten ist der Einsatz auf Unix/Linux Systemen möglich.

Bugzilla arbeitet bei der Fehlerverfolgung abhängig vom Produkt. Die einzelnen Software - Elemente des Produktes können in der Datenbank abgebildet werden. Zur Eintragung eines Fehlers besteht dann für jeden Benutzer des Systems die Möglichkleit, produkt-spezifische Zugriffsrechte zu setzen. So erhält ein Benutzer (Kunde) beispielsweise nur Rechte, einen Fehler- oder Verbesserungsantrag einzutragen, während ein anderer Benutzer (Entwickler) auch oder sogar nur Rechte erhält, den Status eines Fehler- oder Verbesserungsvorschlags zu ändern (z.B. von ”neu eingegangen” zu ”wurde zur Bearbeitung übergeben”). Über das Request-Formular können bzw. müssen die für einen Change-Request typischen Daten (Produkt, Software-Element, Test-Plattform, Beschreibung, Reproduzierbarkeit, um die wichtigsten zu nennen) eingegeben werden. Das Formular ist vom Administrator - abhängig vom Benutzer - einstellbar. So sehen zum Beispiel die Tester eines bestimmten Bestandteils der Software nur die zu diesem gehörenden Elemente. Auf der anderen Seite bietet Bugzilla bei der Abfrage von Fehlern und Verbesserungsvorschlägen vielfältige Möglichkeiten. So lässt sich nach sämtlichen Komponenten der Datensätze suchen und sortieren (siehe Abb.3). Auch lassen sich Statistiken aller Art anzeigen (z.B. Welche Fehler sind neu hinzugekommen? Wieviele wurden schon gelöst?). Zusätzlich zu der Option, alle Funktionen Bugzillas über einen Web-Browser zu benutzen, besteht auch die Möglichkeit über ASCII-basierte Formate Daten auszutauschen, unter anderem via XML oder komma-separierte Listen (CSV) [Bug03].

3.2.2 ClearQuest MultiSite

Ebenso wie bei Rational ClearCase gibt es auch für das weit verbreitete Change Management - Tool ClearQuest von Rational eine Erweiterung namens ClearQuest MultiSite. ClearQuest verwaltet die Fehler- und Verbesserungsvorschläge ähnlich wie Bugzilla in einer Datenbank, ist hier aber flexibler. Unterstützt werden unter anderem Oracle, IBM DB2, Microsoft Datenbanken und Sybase SQL Anywhere. Als einziger Webserver wird leider nur der Microsoft IIS unterstützt.

Die Requests können über ein Web-Interface abgegeben werden, welches nahezu ei-

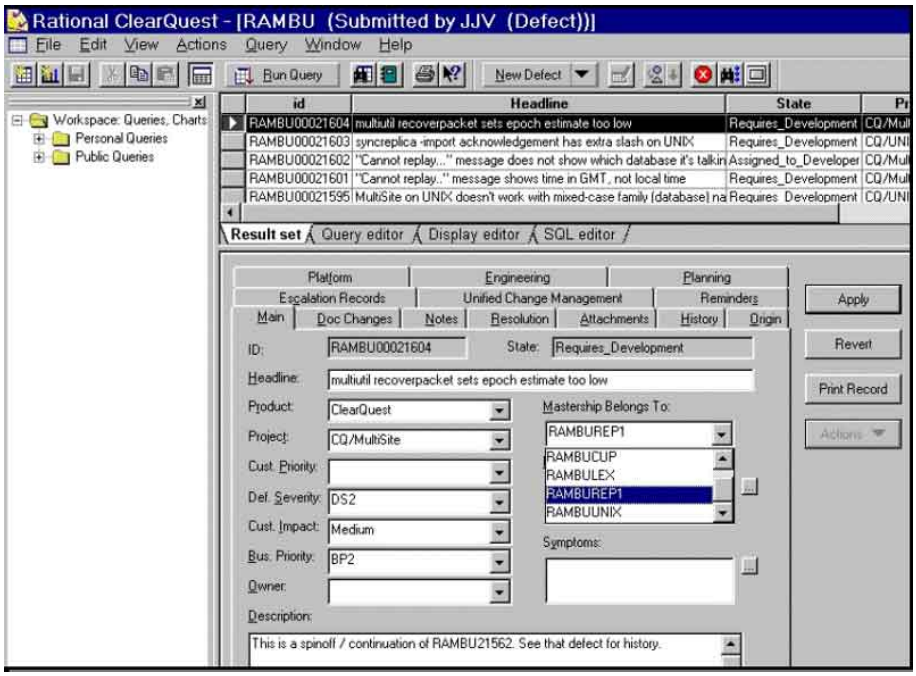

Abbildung 5: Screenshot ClearQuest MultiSite

ne Windowsoberfläche simuliert. Zusätzlich bietet ClearQuest auch ein eigenständiges Desktop-Interface. Mit Hilfe der Interfaces kann der Benutzer alle Funktionalitäten bedienen. Der Umfang reicht vom Abgeben der Requests bis zum Erstellen von Abfragen und komplexen Berichten. Die Benutzer-Interfaces können ebenso wie die zugrunde liegende Datenbank in allen Bereichen angepasst werden. ClearQuest läuft auf Microsoft Windows, Sun Solaris und HP-UX Systemen [RCC03].

4 Fazit

Es ergeben sich tatsächlich einige zusätzliche Anforderungen an ein Change und Configuration Management System, wenn die Entwicklung an geografisch verteilten Orten stattfindet. Alle hier vorgestellten Tools ermöglichen verteilte Software-Entwicklung. Von den genannten Configuration Management Systemen wird im Open-Source Bereich vornehmlich das CVS benutzt. Das CVS genügt den oben genannten Anforderungen allerdings nicht in allen Punkten. So wird nur ein Repository verwendet und es gibt keine private Versionsverwaltung (siehe 2.1). Wer ein in dieser Hinsicht komplettes System sucht, wird auf ClearCase zurückkommen müssen. ClearCase bietet sehr umfangreiche Features und Möglichkeiten, das System den individuellen Anforderungen anzupassen, ist aber auch entsprechend kostspielig. Microsoft Visual SourceSafe ist stark auf die Verwendung in Kombination mit anderen Microsoft-Produkten ausgelegt. Sollte daher eher von Entwicklern eingesetzt werden, die mit Visual.NET arbeiten. Die beiden hier vorgestellten Change Management Tools gehören in ihrem je-

weiligen Bereich sicher zu den Besten. Bugzilla hat sich mittlerweile im Open-Source Sektor etabliert und stellt dort das State-Of-The-Art Change Management und Bugtracking System dar, während ClearQuest sich unter den kostenpflichtigen Systemen hervorhebt. Bugzilla bietet alle grundlegenden Funktionen, die für verteiltes Change Management und Bugtracking notwendig sind, lässt sich aber im Gegensatz zu ClearQuest nur bedingt anpassen (vorausgesetzt man möchte nicht den Quellcode von Bugzilla bearbeiten). Für individuelle Lösungen eignet sich ClearQuest wesentlich besser. Ob das Verhältnis zwischen Kosten und Nutzen stimmt und für welches System man sich entscheidet, ist dementsprechend vom jeweiligen Entwicklerteam und Projekt abhängig.

5 Literaturverzeichnis

Literatur

[Bal01] H. Balzert. *Lehrbuch der Software-Technik*. Spektrum Akademischer Verlag, 2001.

[Bug03] Bugzilla Documentation. http://www.bugzilla.org, 2003.

[CVS03] CVS Documentation. http://www.cvshome.org, 2003.

[HLRT97] Hunt, Lamers, Reuters, and Tichy. Distributed Configuration Management via Java and the World Wide Web. Technical report, University Karlsruhe, 1997.

[RCC03] Rational ClearCase and ClearQuest Website. http://www.rational.com, 2003.

[vdH00] A. van der Hoek. Configuration Management and Open Source Projects. Technical report, Institute for Software Research Irvine CA USA, 2000.

[VSS03] Microsoft Visual SourceSafe. http://www.msdn.microsoft.com, 2003.

Vertragsgestaltung zwischen Auftragnehmer und Auftraggeber

Nils Waterkotte

Inhaltsverzeichnis

1 Einleitung

In diesem Papier geht es um die Vertragsgestaltung und -durchführung zwischen Arbeitgeber und Arbeitnehmer.

Zuallererst mag man bei diesem Thema an juristische Ausformulierungen und Paragraphen denken, die einem die Zusammenarbeit und anschließend, im Falle des Scheiterns, den Stand vor Gericht erleichtern sollen. Deswegen sehen vielleicht viele den Vertrag als ein mit unverständlichen Klauseln übersätes Dokument, welches erst im Falle eines Rechtsstreits aus der Schublade geholt wird. Dass dieses nicht ganz richtig ist, wird im weiteren Verlauf ersichtlich werden, denn hier geht es vielmehr um den Vertrag als Hilfe für die Durchführung des Auftrages mit einem erfolgreichen Abschluss für beide Seiten. Um dieses Ziel zu erreichen, bedarf es eines Überblicks über auftretende Probleme und Krisen, deren Ursachen und Ansätzen, diese zu verhindern bzw. zu bewältigen. Und genau diesen Überblick soll dieses Papier geben.

2 Probleme bei IT-Verträgen

Bei den meisten Projekten der IT-Branche treten Probleme auf. Oftmals haben sie gleiche Ursachen und treten zu gleichen Zeitpunkten im Projektverlauf auf. Welche Ursachen es sind, und vor allem, wann sie auftreten, wird in diesem Kapitel behandelt.

Nun, oftmals werden folgende Ursachen für einen Misserfolg gegeben

- Größe des Projekts
- Komplexität der Lösung
- Interdisziplinärer Charakter von Projekten

- Neuartigkeit des Projekts
- Neuartigkeit der eingesetzten Technik

Wie aus der Abbildung 1 , die Ursachen für Projektmisserfolge darstellt, ersichtlich ist, sind diese Ursachen allerdings nicht für die großen Probleme, oder gar das Scheitern von Projekten verantwortlich.

Überblick über die Ursachen AN = Auftragnehmer		in %	
1. Bei Vertragsabschluss gesetzt			**39**
1.1 Ziele/Aufgabenstellung		**18,6**	
1.1.1 Ziele unklar	1,2		
1.1.2 Aufgabenstellung (Anforderungen des Kunden)	13,3		
1.1.3 Leistungsumfang	4,1		
1.2 Aufwand		**4,8**	
1.3 Termine		**5,2**	
1.4 Sonstige Einflüsse aus Auftragnehmerseite darunter		**7,6**	
Mangelndes IT-technisches Wissen	1,3		
Mangelndes Fachwissen	1,9		
1.5 Sonstige Einflüsse aus Kundenseite		**0,8**	
1.6 Projekt als solches (Tücken der IT-Technik)		**0,2**	
1.7 Sonstige Ursachen		**1,8**	
2. Bei Aufsetzen des Projekts gesetzt			**5**
3. Bei Durchführung aufgetreten			**56**
3.1 Projektarbeit gemeinsam		**7,2**	
3.1.1 Projektleitung gemeinsam	2,5		
3.1.2 Menschliche Faktoren (insb. »Chemie stimmt nicht«)	4,0		
3.1.3 Streit über Anforderungen	0,7		
3.2 Einflüsse aus Kundenseite		**24,7**	
3.2.1 Ziele (Mangelnde Bejahung des Projekts/der Zielsetzung)	5,3		
3.2.2 Aufgabenstellung	3,7		
3.2.3 Mitarbeit des Kunden darunter	12,8		
Mangelnde Kompetenz der Fachseite	2,0		
Mangelhafte/verzögerte Mitwirkung	5,9		
3.2.4 Andere Gründe auf Kundenseite	2,9		
3.3 Einflüsse aus Auftragnehmerseite darunter		**19,7**	
Mangelhafte Projektleitung (einseitig)	10,8		
Erforderliche Mitarbeiter fehlen	3,2		
Fehlerhafte Leistung	2,8		
3.4 Projektbezogene Einflüsse: Komplexität		**0,2**	
3.5 Externe Einflüsse darunter		**4,2**	
Vorlieferant des AN fällt aus/liefert fehlerhaft	2,5		
Vorlieferant des Kunden fällt aus/liefert fehlerhaft	1,1		
Lösung technisch nicht machbar	0,5		
Summe Menge = 850		**100%**	**100%**

Abbildung 1: Überblick über Ursachen, die für das Scheitern von Projekten verantwortlich sind

Vielmehr sind es die so genannten ’weichen Faktoren’, verbunden mit dem menschlichen Verhalten, die das Leben auf beiden Seiten erschweren. Da gibt es zum einen Ursachen, die aus der Nichtbefolgung der später beschriebenen Anforderungen an die Vertragsgestaltung resultieren. Diese legen eine Grundlage für spätere Konflikte, welche sich dann kaum noch beseitigen lassen. Hat man z.B. zeitlich zu knapp kalkuliert,

so lässt sich dieses höchstens durch Verringerung der Qualität oder des realisierten Umfangs ausgleichen, welches allerdings kein erstrebenswertes Ziel ist.

Zum anderen gibt es Ursachen, die aus der Projektdurchführung entstehen. Ist das erwartete Engagement des Kunden ausgeblieben, so dass sich die gesamte Weiterarbeit verzögert hat, führt das schnell zu größeren Konflikten. Der Kunde hat einen Abgabetermin gegeben, und davon wird er sich so schnell nicht abbringen lassen, geschweige denn seine Schuld eingestehen.

Das ganze kann man unter dem Begriff „3-K-Problem“(Kommunikation, Kooperation, Koordination) zusammenfassen. Was man darunter genau versteht, wird in den folgenden Kapiteln beschrieben.

3 Auftragnehmer

In diesem Abschnitt wird auf die Auftragnehmerseite eingegangen. Diese ist zugleich auch der Hauptadressat dieses Papiers, da man sich als Informatiker primär auf dieser Seite wiederfinden wird.

Um einen Projekterfolg zu erreichen, müssen während der gesamten Projektentwicklung einige Punkte beachtet werden. Das fängt damit an, sich über seine eigenen Ziele bewusst zu werden („Was möchte ich erreichen?“). Weiter sind bei der Akquisition von Aufträgen und der Angebotsstellung einige Grundregeln zu beachten, um die Aufmerksamkeit und das Interesse auf sich zu ziehen. Schließlich gibt es für den Vertrag noch einige Gestaltungsregeln und Schreibstile, die den Weg zu einem erfolgreichen Vertragsabschluss ebnen können.

3.1 Ziele

Was möchte man mit einem Auftrag erreichen? Zuallererst steht sicherlich der Projekterfolg. Doch was beinhaltet dieser? Zum einen eine praxistaugliche Lösung und zum anderen einen zufriedenen Kunden. Dazu kommen noch Ziele wie:

- satte Gewinne erzielen
- technologisch/am Markt vorankommen
- Zufriedenheit über erfolgreichen Abschluss
- Verschmelzung einer Gruppe von einzelnen Mitarbeiten zu einem Team

Doch diese Ziele seien hier vernachlässigt. Wie erreicht man nun eine praxistaugliche Lösung und einen zufriedenen Kunden?

3.2 Grundregeln

Sicherlich ist dies keine „goldene Liste“ mit Grundregel, die, wenn sie befolgt werden, zum garantierten Erfolg führen; aber folgende Grundregeln und Verhaltensweisen helfen sicherlich, dem ein oder anderen Konflikt vorzubeugen, so dass erst gar keine größeren Probleme entstehen.

Bevor man überhaupt zu arbeiten anfängt, sollte das Vertragsdokument aus mindestens zwei Dokumenten bestehen: eine für beide Parteien verständliche und umsetzbare Anforderungsdefinition und einen Vertrag (für die rechtliche Absicherung). Da diese Dokumente bei jeder kleinen bis mittleren Meinungsverschiedenheit hervorgeholt werden (bei größeren Konflikten ist beiderseits klar, was das Problem ist, z.B. Liefertermin, etc.), um den genauen Wortlaut zu interpretieren, ist dieser oft entscheidend. Daher versteht sich von selbst, dass diese Dokumente in schriftlicher Form vorliegen müssen. Wie sind diese nun zu verfassen und was gehört dort hinein?

3.2.1 Inhalt der Dokumente

Ein Vertragsdokument sollte präzise, möglichst vollständig und klar abgefasst werden. Darin wird alles festgehalten, angefangen bei den geforderten und den ausgegrenzten Leistungen, über die Regelungen der Aufgabenverteilung zwischen Auftraggeber und -nehmer, bis hin zu den Eskalationspartnern der ersten Stufe auf beiden Seiten, welche bei kleineren Konflikten als Ansprechpartner dienen und auch befugt sind, diesbezüglich Entscheidungen zu treffen.

Beim Verfassen der Anforderungsdefinition ist darauf zu achten, vollständig und verständlich zu schreiben, damit die das Projekt durchführenden Personen, dieses auch verstehen („Gedacht ist noch nicht geschrieben“). Dabei sollte beachtet werden, dass man, soweit dies möglich ist, allgemeinverständlich oder „management-verständlich“ schreibt (soll allerdings nicht heißen, dass es ein Laie verstehen muss). Denn bei Meinungsverschiedenheiten entscheiden oftmals Personen, die nicht immer vom Fach sind, über diese Formulierungen. Um die Vollständigkeit und Verständlichkeit zu erhöhen, sollte ruhig ein Satz zu viel als zu wenig geschrieben werden, allerdings ohne sich dabei zu widersprechen. Zu beachten ist, dass hierbei nichts versprochen wird, was später nicht umgesetzt werden kann. Dazu gehört auch, die Ziele des Kunden - also die Anforderungsdefinition - klar und verständlich für beide Seiten zu verfassen.

Die globalen Ziele des Kunden sollten in Einzelziele und sogenannte „Meilensteine“ gespalten werden, um später zum einen eine Abrechnungsgrundlage nach erreichten Meilensteinen zu erhalten, und um dem Kunden die Möglichkeit zu nehmen, aus diesen Globalzielen zusätzliche Anforderungen an die geschuldeten Leistungen abzuleiten.

Für den Fall, dass man die Ziele klar gegliedert vorliegen hat und auf eine Abrechnung nach erreichten Meilensteine verzichtet, kann man sich auch zu einem Festpreis überreden lassen, wobei man sich allerdings des damit verbundenen Risikos bewusst

sein sollte, und dementsprechend nicht an Risikozuschlägen sparen darf. Auch sind feste Termine mit Vorsicht zu betrachten, da selbst bei einem Verschulden des Kunden dieser weniger dazu bereit sein wird, auf seinen gesetzten Termin zu verzichten.

3.2.2 Schreibstil

Nachdem nun das „Was“ behandelt wurde, wird nun näher darauf eingegangen, wie die Ziele Klarheit, Verständlichkeit und Vollständigkeit im Dokument umgesetzt bzw. erreicht werden können.

Zuallererst sollte man es tunlichst vermeiden, der Sprache der Juristen nachzueifern, die von vielen als umständlich, unklar, gespreizt und schwerfällig gesehen wird. Bei dem Versuch, juristisch zu schreiben, kann man viel schneller Gefahr laufen, als es einem recht ist. Bei Auseinandersetzungen kommt es ganz entscheidend auf jeden Wortlaut an. So kann es sein, dass der verfasste Text der anderen (und auch der eigenen) Seite nicht verständlich ist, oder man unwissentlich Zusicherungen und Garantien gibt und damit eine weitgehende Haftung eingeht. Ebenso kann es sein, dass man juristische Begriffe verwendet, dessen Bedeutung zwar eindeutig , einem selbst aber unbekannt ist. Im Falle eines Gerichtstermins wird ein Richter den Text auch nicht wie den eines Kaufmanns, sondern wie den eines Juristen verstehen und auslegen. All diese Gefahrenquellen können genau das Gegenteil des gewollten bewirken, dass man zum einen nicht das angestrebte absichert und man zum anderen lächerlich wirkt und nicht mehr ernst genommen wird. Ähnliches ist auch beim Gebrauch von englischen Wörtern der Fall. Diese sollte man möglichst nicht verwenden, es sei denn, die Bedeutung ist zu 100% eindeutig. Da nun bekannt ist, was man nicht machen sollte, hier nun ein paar Punkte, die sich positiv auf das Dokument auswirken.

Zum einen ist es von Vorteil, wenn das Dokument gegliedert ist und ein Inhaltsverzeichnis besitzt, damit schnell gefunden werden kann, was gerade benötigt wird. Umfangreiche Erläuterungen und Definitionen von Begriffen sollten sich in einem Glossar wiederfinden, wobei auch sichergestellt sein sollte, dass sowohl Kunde als auch Auftragnehmer das gleiche unter diesen Begriffen verstehen. Diese sollten im gesamten Dokument (bestehend aus Vertrag, AGB, Handbuch, etc.) einheitlich verwendet werden.

Um nicht die Übersicht zu verlieren, gerade wenn viele Informationen auf dichtem Raum stehen, ist es von Vorteil, kurze Sätze zu bilden. Des weiteren ist eine eindeutige und konkrete Formulierung angebracht, d.h. nicht im Passiv schreiben (Wer ist gemeint?), vorsichtig sein bei „wir“ (Das eigene Unternehmen oder die beiden Vertragspartner?) und vor allem keinen Spielraum für Interpretationen lassen („Der Kunde ist verpflichtet,...“ statt „Der Kunde sollte...“). In der Abbildung 2 sind noch einmal die wichtigsten Ziele der Vertragsgestaltung zusammengefasst.

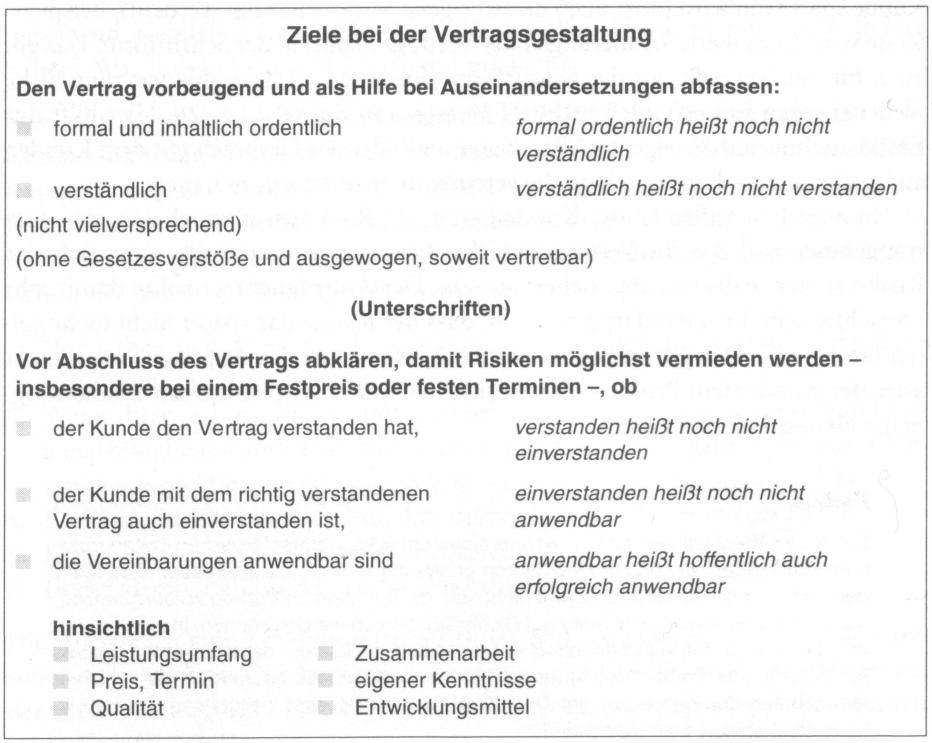

Abbildung 2: Ziele bei der Vertragsgestaltung

3.3 Vom Angebot zum Vertragsabschluss

3.3.1 Angebot stellen

Um einen Auftrag erhalten zu können, muss man den Kunden erst einmal für sich gewinnen. Dieses erreicht man durch ein attraktives Angebot, das neben den Anforderungen an Vollständigkeit, Genauigkeit und Verständlichkeit nach 3.2 auch noch folgende Kriterien erfüllt:

- Professioneller Eindruck (fördert Vertrauen in das Angebot und anschließendes Projekt)
- Handhabbarkeit des Dokumentes (klare Gliederung, Inhaltsverzeichnis, technische Beschreibungen und Erläuterungen im Anhang)
- Kopierfähigkeit (Angebot sollte im kopierten schwarz-weiß noch die gleiche Wirkung haben wie das Original)
- Beachtung der wichtigsten Adressaten, wie Manager, Vorgesetzte, Juristen, etc. welche möglicherweise nicht das gleiche Fachwissen besitzen
- enthält Begriffswelt des Kunden (Vereinfachung für den Kunden, demonstriert Interesse am Kunden und baut Vertrauen auf)

Ein weiteres Thema sind die AGB. Es gibt zwar nicht viele Kunden, die sie sich vor einem möglichen Konflikt anschauen, aber die, die es tun, sollten durch diese nicht abgeschreckt werden. Viele AGB sind mit zahlreichen Parapraphen gespickt, die die eigene Seite absichern sollen. Oftmals sind es Klauseln, die irgendwo abgeschrieben wurden, ohne dass sie eine wirkliche Bedeutung haben, da vieles in den Gesetzestexten geregelt ist. Viele Kunden macht man auch erst durch explizite Leistungsausgrenzungen auf diese aufmerksam, sodass wieder neue Verhandlungspunkte entstehen („Eine Pflege wird für maximal 3 Jahre übernommen“,...). Meist reicht es auch aus, den Wortlaut freundlich zu fassen, damit sich der Kunde nicht direkt auf der Anklagebank sieht. AGB sollen weder kundenfreundlich, noch kundenunfreundlich sein. Hier ist der gesunde Mittelweg gefragt.

3.3.2 Aushandeln von Vertragspunkten

Beim Aushandeln geht es für beide Parteien darum, die für sie besten Konditionen aus den Verhandlungen mitzunehmen. Das heißt für den Kunden, möglichst viel Leistung für wenig Geld zu bekommen, und für den Auftraggnehmer, möglichst wenig Extraleistungen für das Geld zu geben. Dabei sollte darauf geachtet werden, dass man das Nachgeben auf die Schlussverhandlung verschiebt, da dort der Kunde noch ein letztes mal mehr Leistungen für weniger Geld verlangen wird. Dann ist es von Vorteil, noch einige Reserven zu besitzen.

Im Gegenteil dazu sollte die eigene Haftung rechtzeitig angesprochen und abgeschlossen werden, auch wenn es für die meisten ein rotes Tuch ist und gerne nach hinten verschoben wird. Falls dieses nicht geschieht, wird man zum Ende hin leicht erpressbar. Der Kunde kann einerseits verlangen, das Schadensrisiko auf die andere Seite abzuwälzen, da er sich denken kann, dass man dieses Projekt nach so langer Vorbereitungszeit und investierter Arbeit ungern verliert, und man daher eher nachgeben werde. Zum anderen kann es noch offene Punkte geben, wobei der Kunde nachgeben würde, falls die andere Seite über die offenen Punkte der Haftung hinweg sehe.

Deswegen ist ein frühzeitiges Aushandeln und vor allem Abschließen der eigenen Haftung nicht zu unterschätzen (daher unbedingt auf eine Erklärung bestehen, dass es eine endgültige Teilvereinbarung sei).

Bei Verhandlungen besitzt jede Seite einen gewissen Vorrat an Verhandlungsmacht. Diese reduziert sich bei jedem Entgegenkommen der anderen Seite. Deswegen sei gut überlegt, in welchen Punkten man seine Macht einsetzen will. Bei Eventualitäten und unwahrscheinlichen Ereignissen, sowie für den Kunden wichtige Punkte, die einen selbst aber nur wenig belasten, sollte man diese nicht vergeuden.

Bei der Schlussverhandlung sollten auf beiden Seiten Entscheidungsträger anwesend sein. Denn was nützt eine Schlussverhandlung, wenn die ausgehandelten Punkte später noch abgezeichnet werden müssen. Für den Fall, dass die andere Seite keinen solchen Entscheidungsträger teilnehmen lässt, sollte man den eigenen Entscheidungsträger auch nicht teilnehmen lassen, um auf die endgültige Entscheidung noch ange-

messen reagieren zu können.

Im Laufe der Verhandlungen kann es sein, dass der Kunde überzogene Forderungen stellt, sei es aus Verhandlungstaktik, Unsicherheit oder schlechter Erfahrung. Diese sollte man weder ins Lächerliche ziehen, noch sollte man Kritik an dessen Unvernunft üben. Das bringt nur Misstrauen und schadet einer gesunden Vertrauensbasis. Stattdessen sollte man das Vertrauen des Kunden erhalten, seine Forderungen ernst nehmen und diese mit sachlichen Einwänden erwidern. Wenn der Kunde vor Augen geführt bekommt, welcher materielle und personelle Mehraufwand mit seiner Forderung verbunden ist, wird er meist sicherlich von selbst darauf verzichten.

3.3.3 Umgang mit Beratern und Rechtsanwälten

Für den Fall, dass der Kunde einen Berater oder einen Rechtsanwalt zu den Verhandlungen mitbringt, sollte man ihnen mit Vorsicht begegnen. Viele der Berater sind technikverliebt oder wollen sich in ihrer Position profilieren. Sie stellen häufig überzogene leistungsbezogene oder rechtliche Forderungen. Letztere können auch einfach wiederholte Forderungen aus anderen Verhandlungen sein, die zwar nicht verstanden, aber für gut befunden worden sind; und zudem oft auch noch juristisch falsch.

Falls man über juristisches Wissen verfügt, kann man den Berater durch eine gute Wahl von Argumenten in den Hintergrund rücken, indem man seine mangelnde Kompetenz und Unvernunft verdeutlicht. Bei Leistungsbezogenen Forderungen sei wieder auf die finanziellen Mehrkosten hingewiesen, welche bei Umsetzung entstehen würden.

Im Falle eines übereifrigen Beraters, der seinen Kunden vertritt, ohne dass dieser persönlich anwesend ist, hilft nur ein Schreiben an diesen, ob er auch wirklich hinter diesen überzogenen Forderungen stehen würde.

Wenn man nun einen Rechtsberater seinen Gegenüber nennen kann, wird man mit ihm hauptsächlich in rechtlichen Fragen konfrontiert, zumeist mit den AGB. Kritikpunkte an den eigenen AGB sind nämlich oftmals die Einseitigkeit in rechtlichen Belangen, so dass von der anderen Seite befürchtet wird, dass die IT-spezifischen ebenfalls einseitig angelegt seien, welche sie allerdings durch mangelndes Fachwissen nicht überprüfen können.

Kunden haben daher oft Standards, die sie auch durchsetzen wollen. So ist zu prüfen, welche Risiken man beim Verzicht auf die eigenen AGB eingeht. Allerdings kann man versuchen, die wichtigen Punkte der eigenen AGB in den Vertrag zu übernehmen, sodass auf die eigenen AGB ohne zusätzliches Risiko verzichtet werden kann.

3.3.4 Vertragsabschluss

Wenn nun der Punkt der Vertragsunterzeichnung näher rückt, könnte man denken, dass bei Unterschrift des Kunden dieser auch mit dem Vertragsinhalt einverstanden ist. Die-

ses ist rein rechtlich gesehen korrekt. Doch heißt einverstanden sein nicht, dass er den Text auch wirklich verstanden hat! Falls nicht, sind Konflikte und vielleicht auch der Projektabbruch vorprogrammiert. Und dann hat man zwar eine Gerichtsverhandlung gewonnen, aber die Referenz und der Kunde sind verloren. Daher ist es besser, den Vertrag so verständlich wie möglich zu halten und sämtliche Risiken mit dem Kunden abzusprechen (Aufklärungspflichten des Auftragnehmers gegenüber dem Kunden laut IT-Vertragsrecht).

4 Auftraggeber

Das Ziel des Auftraggebers ist ganz primär ein erfolgreiches und praxistaugliches Ergebnis zu bekommen. Um dieses Ziel zu erreichen, ohne übermäßig viel darin zu investieren, gibt es auch für den Auftraggeber einiges zu bedenken. Viele Regeln von dem Auftragnehmer können für den Auftraggeber 1:1 übernommen werden. Allerdings gibt es noch einige besondere Punkte, die der Auftraggeber beachten muss, da seine größte Beachtung dem Auffinden des für ihn besten Auftragnehmers gilt. Somit wird hier noch einmal gezielt auf dessen Ziele und Verhaltensregeln eingegangen.

4.1 Grundregeln

Der wichtigste Schritt um oben genanntes Ziel zu fördern ist, den passenden Arbeitnehmer sorgfältig auszuwählen. Zu hinterfragen sind seine Referenzen, seine Kompetenzen, seine Finanzstärke, seine Kapazitäten an Mitarbeitern, sowie sein Fachwissen in dem zukünftigen Projektgebiet neben seinem IT-Wissen.

Falls es z.B. an einem kompetenten Projektmanagemant fehlt, hat das Projekt keinen guten Start. Wie aus der Abbildung 1 von Seite 121 zu entnehmen ist, sind 11% aller gescheiterten Projekte auf ein schlechtes Projektmanagement zurückzuführen.

Hilfreich hierbei ist das Erkundigen bei früheren Kunden des Arbeitnehmers nach erfolgreichen Projekten. Falls die Recherche positiv ausfällt, und man einen fähigen Projektleiter gefunden hat, welche im IT-Markt rar sind, dann sollte man vertraglich festhalten, dass dieser während des Projekts nicht abgezogen und ersetzt werden darf, um einen ordentlichen Projektabschluss zu sichern.

Weiterhin gilt die Finanzstärke des Arbeitnehmers als große Sicherheit für ein Projekt. So muss man nicht befürchten, dass das Projekt vorzeitig beendet wird, falls die Kosten des Arbeitnehmers die Vorgaben übersteigen, oder im Falle des Scheiterns, etwaige Vorauszahlungen nicht mehr einzufordern sind.

Auch wenn im Laufe des Projekts meist mehr Kosten auf den Auftraggeber zukommen als erwartet, sei es durch Änderungen, Verbesserungen oder zusätzliche Anforderungen, ist es ratsam, eine komplette Aufstellung der erwarteten Kosten (auch die eigenen) erstellen zu lassen, um später unnötige Überraschungen zu vermeiden.

Feste Preise sowie feste Termine sind eigentlich nur bei detaillierter Klärung der Aufgabenstellung zu empfehlen, doch häufig fehlt diese, in dem Glauben, alle Abweichungen auf den Arbeitnehmer schieben zu können. Dabei wird nicht beachtet, dass es sich um Probleme handelt, die später auf einen selbst zurückkommen. So wird zwar oftmals der Festpreis, jedoch der feste Termin selten und die erforderliche Qualität fast nie gehalten.

Darum ist es ratsam, bei Bestehen auf diese, Termine großzügig zu planen und den Programm-Abgabetermin weit vor dem eigenen Einführungstermin zu legen und bei der Kalkulation zusätzliche Mittel im Budget einzuplanen. Dabei sind Termine noch vorsichtiger zu planen als Preise, da das Risiko nicht so einfach auf den Arbeitnehmer abgewälzt werden kann, sondern einen selber trifft.

Als Auftraggeber sollte man sich vor allem vor Augen führen, dass gute Leistung ihren Preis und ihre Zeit braucht.

4.2 Von der Projektausschreibung bis zum Vertragsabschluss

4.2.1 Einholen von Angeboten

Um ein Angebot für ein Projekt zu erhalten, muss zuallererst die Aufgabenstellung hinsichtlich der Funktionalität, der Benutzerqualifikation und des Leistungsverhaltens schriftlich abgefasst werden. Dabei ist es empfehlenswert, auch die Anforderungen schon im voraus zu erstellen, um hinterher keine unausgesprochenen Erwartungen an das Produkt zu haben. Der IT-Fachmann kann den Text in Ruhe durchdenken und sich mit ihm auseinandersetzen, wodurch oftmals auftretende und unterschätzte Verständigungsschwierigkeiten vermieden werden. Später können dann Einwände und Anregungen seitens des IT-Fachmanns gemeinsam integriert werden. Für einen zügigeren und einfacheren Ablauf sorgt ein Angebotsdokument, welches ähnlich dem endgültigen Vertrag aufgebaut ist. Dieses läßt sich schnell und unkompliziert in einen Vertrag wandeln, so dass dieser schnellstmöglich ohne größere Überarbeitung unterzeichnet werden kann, um den Arbeitsbeginn nicht unnötig hinauszuzögern. Dabei ist natürlich unerlässlich, dass das Angebotsdokument bei Verhandlungen ständig fortgeführt und aktualisiert wird.

Einen Zeit-Arbeitsplan zu verlangen, der sämtliche Aktivitäten enthält, auch die vom Arbeitgeber zu erbringenden, hat den Vorteil, dass man die eigenen Mitarbeiter besser einplanen kann, und später nicht unerwartet einige abtreten muss.

Um aus der Menge der Angebote entscheiden zu können, bedarf es einem Kriterienkatalog. Dieser kann K.O.-Kriterien enthalten, sollte aber auf jeden Fall die gewichteten Kriterien enthalten, die man erwartet. Auf dessen Grundlage werden die Angebote aussortiert, so dass nur noch wenige (2-3) zur Auswahl stehen, die man nun genauer analysieren kann. Die Abgewiesenen sollten umgehend informiert werden, damit diese mit besseren neu aufgelegten Angeboten aufwarten können.

4.2.2 Aushandeln von Vertragspunkten

Hier gilt eigentlich, genau das zu machen, was Arbeitnehmer laut Kapitel 3.3.4 nicht tun sollen.

- Von Anfang an Entgegenkommen fordern
- Wert auf Schadensersatzansprüche legen

Was die Zahlungsbedingungen betrifft, seien aber auch hier die Zahlungen an „Meilensteine“, also das Erreichen von Zwischenzielen, geknüpft. Denn bereits gezahltes Geld lässt die Aktivitäten des Arbeitnehmers erlahmen, und nur über Voraussetzungen für einzelne Zahlungen wird das Interesse geweckt, Probleme zu beseitigen.
In der Regel wird in den Schlussverhandlungen versucht, die Preise weitest möglich zu drücken. Doch sei hier gewarnt, nicht zu überziehen. Der Auftragnehmer wird sich den aus seiner Sicht vom fairen Preis fehlenden Betrag spätestens nach der Abnahme wieder hereinholen. Wenn man es trotzdem versucht, kann es sein, dass sämtliche Änderungs-/Zusatzwünsche nur gegen Aufpreis gewährt werden.

4.2.3 Vertragsabschluss

Bei Vertragsabschluss sollte man sich absolut sicher sein, dass es der richtige Vertragspartner ist. Doch wenn die oben genannten Punkte berücksichtigt werden, dürfte man seiner Entscheidung vertrauen. Da eine gute Entscheidung seine Zeit braucht, sollte man sich, außer aus projekt-technischen Gründen (Zeitdruck,etc.), nicht vorschnell zur Unterschrift überreden lassen. Skepsis ist vor allem bei Sonderangeboten angebracht, die nur wenige Tage gültig sind.

5 Durchführung von Verträgen

Nachdem der Vertrag unterzeichnet wurde, beginnt nun das eigentliche Projekt. Und hier treten die meisten Probleme auf, wobei diese weniger technischer Natur sind, als dass sie auf menschliche Verhaltensweisen zurückzuführen sind. Es kann durchaus vorkommen, dass man von vornherein das falsche tut, weil man die Anforderungsdefinition nicht verstanden hat, man sich im Voraus nicht genügend gegen Risiken abgesichert hat, oder man sich bei Schwierigkeiten nicht korrekt verhält, indem man diese vor sich herschiebt.

5.1 Negative Verhaltensweisen

Für die Seite der Arbeitnehmer ist an die Technikverliebtheit der Mitarbeiter zu denken („Simple is beautiful“), oder die Eigenschaft, dass man das löst, was man kann, aber nicht das, was man soll. Weiter ist man versucht, verstärkt an dem eigenen erfolglosen Ansätzen zu arbeiten, als sich einen neuen zu suchen. Die Einstellung des Arbeitnehmers bzw. seiner Mitarbeiter tut meist sein übriges, indem man seine eigenen Ressourcen überschätzt, sich komplett auf den Vertrag als Erfolgsgarantie verlässt, oder von der Gutgläubigkeit des Kunden überzeugt ist, dass dieser einem auch mal entgegenkommt, wenn man dieses tut.

5.2 Schnittstelle zwischen Auftraggeber und Auftragnehmer

Im täglichen Projektmanagement geht es meist um die Schnittstelle zwischen Arbeitgeber und -nehmer. Auf Arbeitnehmerseite liegt das Hauptproblem beim Projektleiter, der oftmals inkompetent und überfordert ist. Doch lässt sich dieses Problem während eines Projekts selten ändern.

Auf Auftraggeberseite liegt das Problem eher in der mangelnden zeit- und sachgerechten Mitarbeit, sowie in der Entscheidungsfreudigkeit. Abhilfe können dabei je ein entscheidungsbefugter Projektmanager auf beiden Seiten sein, die eng zusammen arbeiten und und so das Projekt zum Erfolg führen.

Um die erforderliche Mitarbeit zu organisieren, hat sich das Kick-off-Meeting bewährt. Dabei werden die einzelnen Aufgaben besprochen, sowie die Anforderungsdefinition mit beiden Teams überarbeitet, indem die ursprünglichen Punkte als Grundlage überarbeitet und detailliert werden. Für den Fall, dass sich der Auftraggeber nicht an seine Pflichten zur Mitarbeit hält, sollte dies mindestens in Protokollen festgehalten werden, um im Falle einer Auseinandersetzung, beispielsweise bei Zeitverzug, Beweismittel vorweisen zu können.

Im weiteren Verlauf, insbesondere wenn Änderungen vereinbart wurden, ist es von großem Vorteil, die Dokumente auf dem neuesten Stand zu halten. Spätestens wenn größere Ergänzungswünsche auftreten, sind Change Requests unerlässlich. Dabei sollte der Kunde seine Wünsche selbst verfassen, um direkt mit dem Umfang und deren Auswirkungen konfrontiert zu werden. Nach deren Besprechung, inklusive der mit sich bringenden Terminverschiebungen, muss das Dokument abgezeichnet und der eigentliche Vertrag fortgeschrieben werden.

5.3 Konflikte

Der Auftragnehmer hat grundsätzlich nicht das gewünschte oder das benötigte, sondern das bestellte System abzuliefern. Damit der Kunde auch vom erfolgreichen Werdegang seines Projekts überzeugt ist, sollten Verunsicherungen vermieden werden.

Erhält er regelmäßig positive Meldungen über erreichte Zwischenschritte, wird er sein Vertrauen in den Arbeitnehmer nicht verlieren. Dabei sollte jedoch wahrheitsgemäß berichtet und nicht übertrieben werden, denn sonst ist das Projekt zu lange „fast fertig“ und der Kunde wird verständlicherweise skeptisch.

Neben den positiven Meldungen sollte es einem auch möglich sein, eigene Fehler zuzugeben und diese nicht herunterzuspielen; das weckt Misstrauen und schadet der Geschäftsbeziehung.

Bevor ein System abgegeben wird, sollte es gründlich getestet werden, auch wenn dadurch mit einer Verzögerung des Abgabetermins zu rechnen ist: Lieber spät, und dafür stabil! Bezüglich der Stabilität und Einfachheit der Bedienung sollte man auch schon beim Erstellen darauf achten und nicht zu sehr der Perfektion verfallen. Kurzfristige Änderungen sollten in Hinblick auf die Stabilität ebenfalls vermieden werden, Gleiches gilt für die Verwendung des Wortes „Fehler“ im Gespräch oder im Schriftverkehr mit dem Kunden: Solange nicht wirklich ein Fehler vorliegt (sondern mangelnde Feinabstimmung, etc.), sollte er auch nicht so genannt werden, da dem Kunden diese „Fehler“ im Gedächtnis bleiben, und er später bei der Abnahme ein wenig skeptischer bezüglich der Fehlerfreiheit eingestellt sein kann.

6 Zusammenfassung und Bewertung

Zusammenfassend kann festgehalten werden, dass es bei einem Projekt nicht nur auf die Durchführung selbst ankommt, sondern eine ordentliche Vorbereitung den Grundstein für einen erfolgreichen Projektablauf und -abschluss legt.

Dazu gehört vor allem, dass alles projektzugehörige, wie Anforderungen, Änderungen etc., schriftlich abgefasst wird. Denn nur so kann später ein für beide Seiten vertretbarer Soll-/Ist-Vergleich angetreten werden. Unvollständige Dokumente können unerwartet große Kreise ziehen und den erfolgreichen Abschluss stark behindern.

Eine vernünftige Absicherung jeder Seite ist natürlich nicht zu unterschätzen, allerdings auch nicht zu übertreiben. Schließlich soll in dem Projekt zusammengearbeitet werden, mit dem Ziel, ein ordentliches System erstellt zu haben, welches beide Seiten zufrieden stellt.

Trotzdem ist dem Vertragspartner nicht zu viel Vertrauen entgegenzubringen, denn sobald es an die eigenen Fehler oder Pflichten geht, werden diese schnell verleugnet. Eine vernünftige Dokumentation des Fehlverhaltens der anderen Seite ist spätestens dann von Vorteil, wenn dieses Rückwirkungen auf die eigenen Pflichten und Leistungen nimmt und man diese rechtfertigen muss (z.B. Zeitverzug).

Alles in allem mag dieses Papier Anstoß für eine ordentlichen Projektarbeit geben, aber eine Garantie dafür kann es auf keinen Fall sein („jeder ist seines Glückes Schmied“).

Wenn man die genannten Ursachen und Probleme betrachtet und sich vor Augen führt,

dass beide Seiten nur am erfolgreichen Projekt interessiert sind, sollte man in der Lage sein, ein ordentliches Produkt zu fairen Konditionen abzuliefern bzw. zu erwerben.

Literatur

[Zah02] Christoph Zahrnt. *Projektmanagemant von IT-Verträgen : Ein Ratgeber für Auftragnehmer und Auftraggeber*. dpunkt Verlag, 2002.

Dokumenten-Management in Softwareprojekten

Alexej Penner

Inhaltsverzeichnis

1 Einführung

Neben den Programmen sind Dokumente die wichtigsten Ergebnissen eines Softwareprojekts [2]. Dokumentenerstellung und -verwaltung sind allerdings sehr zeit- und kostenintensiv. Dataquest/IDC hat in diesem Bereich umfassende Untersuchungen durchgeführt und folgende Feststellungen gemacht [1]:

- Unternehmen geben ca. 10 bis 15% der Einnahmen für Erstellung, Verwaltung und Verteilung von Dokumenten aus.
- Arbeit mit Dokumenten nimmt 60% Arbeitszeit in Anspruch.
- Ein Dokument wird durchschnittlich fünfmal kopiert.
- 50 bis 80% der Arbeitszeit wird zum Suchen nach benötigten Informationen verbraucht.

Diese Zahlen beziehen sich auf ein statistisch durchschnittliches Unternehmen, dessen Tätigkeitsschwerpunkt nicht in der Dokumentenverarbeitung liegt. Da bei einem Softwareprojekt fast ausschließlich mit Dokumenten (Quellcode inklusive) in einer oder anderen Form gearbeitet wird, ist davon auszugehen, dass diese Zahlen wesentlich höher ausfallen dürfen. Deswegen ist es sehr wichtig, Maßnahmen zu treffen, die zu einer effektiven Kontrolle über verfügbare Informationen führen. Dadurch können große Produktivitätssteigerungen erzielt werden.

2 Dokumente in einem Softwareprojekt

Um sich einen Überblick über Probleme und Besonderheiten bei Dokumenten-Management in einem Softwareprojekt zu verschaffen, sollen zunächst Dokumente selbst näher untersucht werden. Das kann am besten anhand von unterschiedlichen Dokumentenklassifizierungen gemacht werden. Hier werden *Dokumentenformate*, *Dokumenteneigenschaften* und *Dokumententypen* betrachtet.

2.1 Dokumentenformate

In einem Softwareprojekt werden nicht nur papierbasierte Dokumente (Schriftstücke) eingesetzt, sondern vielmehr andere Ausprägungen, die in elektronischer Form vorliegen. Kampffmeyer [1] definiert ein *elektronisches Dokument* als eine geschlossene Informationseinheit, die in einem DV-System als Datei vorliegt. Dateiformate können grob in drei Gruppen unterteilt werden:

- *Textformate*: Darunter reine Textformate (ASCII oder Unicode) oder strukturierte Textformate (HTML oder XML).

- *Bildformate*: Eingescannte Faksimiles. TIFF-Format hat sich hier als Standard durchgesetzt. Diese Dokumente werden oft mit OCR-Verfahren durch ein Textdokument ergänzt.

- *Proprietäre*: Geschlossene applikationsspezifische Formate. Sehr verbreitet sind: Portable Document Format (PDF), Microsoft Word, Microsoft Excel und andere.

Je komplexer ein Dokumentenformat ist, umso schwieriger erweist sich der Datenaustausch - bedingt durch Plattform- oder Applikationsabhängigkeit. In einem Softwareprojekt sollen daher nach Möglichkeit nur reine oder strukturierte Textformate verwendet werden.

2.2 Dokumenteneigenschaften

Kampffmeyer [1] klassifiziert Dokumente nach folgenden Eigenschaften:

- physische Eigenschaften (Papier, Datei)
- formale Eigenschaften (Aufbau, Gestaltung)
- Inhalt (inhaltlicher Bezug)
- Charakter (dynamische oder statische Dokumente)
- Zeit (Erzeugungsdatum, Verfallsdatum, letzte Veränderung)
- Erzeuger (Ersteller, Absender)
- Nutzer (berechtigter Bearbeiter, Leser)

Aus diesen Merkmalen folgt, welche Maßnahmen zum Verwalten, Zugriff und Bearbeiten eines Dokuments in einem Softwareprojekt getroffen werden müssen. Physische Eigenschaften eines Dokuments legen beispielsweise fest, ob ein Dokument in das Projekt einfach übernommen werden kann oder zunächst mit Imaging-Verfahren in eine geeignete Form überführt werden soll. Merkmale wie Zeit und Charakter werden dazu verwendet, ein geeignetes Dokumentenmanagementsystem für die Dokumentverwaltung auszusuchen.

2.3 Dokumententypen

Alle Dokumente eines Entwicklungsprojekts können nach Frühauf [2] in drei Dokumentenklassen unterteilt werden:

- *Projektdokumente*: Das sind Dokumente, die ausschließlich zur Projektplanung erstellt werden: Projektauftrag, Projektplan, Berichte und Sitzungsprotokolle. Projektdokumente sind sowohl statisch als auch dynamisch und liegen meistens in der Papierform vor.

- *Produktdokumente*: Unter anderem sind das die Anforderungsspezifikation, Entwurfsdokumente, Testspezifikationen, Programmlistings. Dazu gehören auch Benutzerhandbücher. Produktdokumente haben dynamischen Charakter und werden solange nachgeführt, wie das Produkt weiterentwickelt wird. Die meisten Produktdokumente liegen in der elektronischen Form vor.

- *Prüfungsnachweise*: Diese Dokumente werden am Ende von Tests oder Reviews erstellt und dokumentieren vorliegende Prüfungsergebnisse. Sie werden nur einmal erstellt und nicht mehr geändert. Diese Dokumente liegen in elektronischer oder Papierform vor.

3 Dokumentenmanagementsysteme (DMS)

Alle erwähnten Dokumententypen werden durch einen Dokumenten-Management-System (DMS) verwaltet. Unter einem DMS versteht man ein Computersystem mit dem jegliche Art von Information aufgenommen und verwaltet werden kann. [1]

Vorhandene DMS, die in einem Softwareprojekt eingesetzt werden können, werden grob in fünf Gruppen unterteilt:

1. *Klassisches DMS*: Die Verwaltung ist dokumentenorientiert und erfolgt auf Basis von Dokumenteneigenschaften. Organisatorische Aspekte wie gemeinsames Arbeiten mit Dokumenten bzw. Einbinden in Prozesse spielen hier keine Rolle. Hauptaufgaben sind: Dokumentengruppierungen, Versionsmanagement, selbstbeschreibende Dokumentenobjekte.

2. *Archivsystem*: Dient zur Ablage statischen Dokumenten. Gewünschte Dokumente werden normalerweise über einen eindeutigen Suchbegriff bzw. eine eindeutige ID identifiziert.

3. *Recherchesystem*: Meistens eine statische oder wachsende Datenbasis. Im Unterschied zu einem Archivsystem liegt der Schwerpunkt auf Abfrage von Informationen. Datenaktualisierungen finden nur zu bestimmten Zeitpunkten statt, wenn keine Abfragen vorgenommen werden.

4. *Vorgangssystem (Workflow-System)*: Hier wird ein prozessorientierter Ansatz verfolgt. Vorhandene Dokumente werden in die strukturierten Arbeitsabläufe eingebunden. Oft werden vorhandene Archivsysteme oder klassische DMS verwendet, die in das Vorgangssystem integriert werden.

5. *Groupware-System*: Im Vordergrund steht die gemeinsame Verwendung und Organisation von Informationen in einer Mehrbenutzerumgebung. Zusammensetzung von Werkzeugen wird aufgabenorientiert für die Kooperation, Kommunikation und Koordination von Arbeitsgruppen benutzt.

Zwei davon, klassisches DMS und Workflowsysteme werden im folgenden detaillierter betrachtet.

3.1 Klassisches DMS

Es ist davon auszugehen, dass zum aktuellen Zeitpunkt dies der am meisten eingesetzte DMS-Typ bei Softwareprojekten ist. Mit diesen Systemen werden vor allem Source-Code und auch andere Dokumenttypen eines Softwareprojekts verwaltet.

Bei fast allen DMS werden Dokumente an einem zentralen Platz, im so genannten $Repository$ (deutsch: Dokumentenbestand) abgelegt. Repositories werden als eine Datenbank, Verzeichnisstruktur oder Kombination aus beiden implementiert. Zugriff auf Repositories wird meistens über das Netzwerk bereitgestellt.

Änderungen an einem Dokument finden dagegen in einem lokalen Arbeitsbereich statt. Bevor ein Dokument bearbeitet werden kann, wird es lokal kopiert bzw. ausgecheckt. Der $Checkout$-Vorgang stellt sicher, dass der Benutzer die aktuellste Kopie erhält.

Änderungen werden in einem $CheckIn$- oder $Committing$-Vorgang in das Repository geschrieben. Im Repository wird nicht nur eine aktuelle Kopie des Dokuments gespeichert, sondern jede Version, die jemals eingecheckt wurde. Diese Versionen, die normalerweise eine eindeutige Versionsnummer bekommen, werden $Revisionen$ genannt. Mit Hilfe von Revisionen sind folgende Vorgänge durchführbar:

- Wiederauffinden von bestimmten Versionen eines Dokuments.
- Zurücksetzen des persönlichen Arbeitsbereichs auf ein bestimmtes Datum.
- Feststellen der Änderungen an einem Dokument, die zwischen zwei Revisionen stattgefunden haben.

Änderungen, die von dem Benutzer durchgeführt wurden, verändern nicht automatisch den Arbeitsbereich der anderen Teammitglieder. Dokumente des jeweiligen Arbeitsbereichs werden mit einem $Update$-Kommando auf den aktuellen Stand gebracht.

Versuchen mehrere Benutzer ein Dokument gleichzeitig zu modifizieren, so kommt es zu einem $Konflikt$. Ein DMS versucht in diesem Fall den Konflikt aufzulösen. Finden die Änderungen im Dokument an verschiedenen Stellen statt, so werden sie normalerweise ohne Benutzerinteraktion zusammengefügt. (Voraussetzung ist dafür,

dass das Dokument im Textformat vorliegt). Änderungen, die an einer Stelle stattfinden, müssen manuell korrigiert werden. Der letzte Fall - ein Indiz für eine ineffiziente Teamarbeit - tritt normalerweise nicht sehr oft auf.

3.1.1 Branching

Das so genannte $Branching$ ist ein weiteres wichtiges Feature, das viele DMS anbieten. Es erlaubt Abzweigungen in der Entwicklungslinie eines Softwareprojekts zu erstellen. Wir betrachten ein Szenario, in dem ein Softwareprojekt zum Abschluss gebracht wird. Da die Fortführung des Projekts geplant ist, wird ein Release-Zweig von der Hauptentwicklungslinie abgespalten. Das Release-Team arbeitet somit ungestört am Projektabschluss, führt Systemtests durch und macht letzte Fehlerbehebungen. Die weitere Entwicklung, die die Produktqualität zu dem Zeitpunkt beeinträchtigen können, findet in der Hauptentwicklungslinie statt. Siehe Abbildung 1.

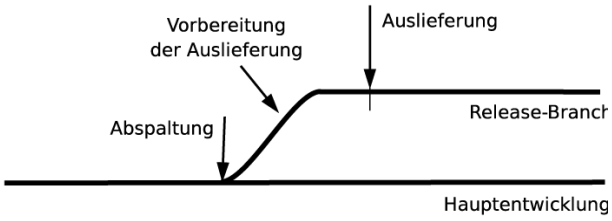

Abbildung 1: Branching

3.1.2 Merging

Nach Projektabschluss wird der Release-Zweig normalerweise weitergeführt. Fehler, die das Release betreffen, werden auch in diesem Release-Zweig behoben. Beispielsweise werden alle Fehlerkorrekturen oder Dokumentationnachbesserungen in diesen Zweig eingecheckt. Damit auch die Weiterentwicklung davon profitieren kann, können diese Änderungen wie in Abbildung 2 gezeigt, mit der Hauptentwicklungslinie zusammengefügt werden. Dieser Vorgang wird $Merging$ genannt.

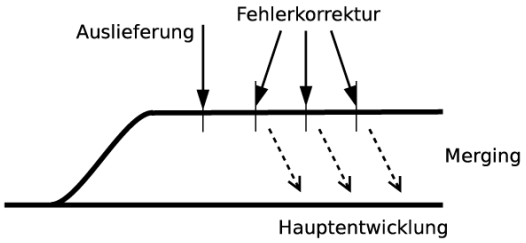

Abbildung 2: Merging

3.2 Workflow-Systeme

Ein klassisches oder dokumentorientiertes DMS stellt zwar viele Funktionalitäten zur Verfügung, muss aber auch vom Anwender aktiv und mit großer Sorgfalt verwaltet werden. Besonders bei großen Projekten können dabei unnötige Risiken bzw. Effizienzeinbüßen entstehen. Daher ist es wünschenswert, dass ein Projekt durch ein computergestütztes System aktiv unterstützt bzw. gemanagt wird.

Workflow-Systeme stellen Technologie zur Umsetzung von Anwendungsprozessen bereit. [3] Da die Menge aller Vorgänge eines Softwareprojekts in den meisten Fällen durch ein festes Modell vorgegeben werden kann, passen Workflow-Systeme sehr gut zum managen eine Softwareprojekts.

Laut Jablonki [3] sollen im Idealfall alle Aspekte eines Projekts berücksichtigt werden: Daten, Funktionen, Organisationseinheiten, benötigte Anwendungen, Kontroll- und Datenflüsse, Sicherheits-, Konsistenz- und Integritätsregeln.

3.2.1 Funktionsweise

Ein Softwareprojekt wird normalerweise nach einem vorgegebenen Modell geführt, das aus einzelnen Phasen besteht. Eine Phase wird meistens durch einen Workflow repräsentiert (Abbildung 3).

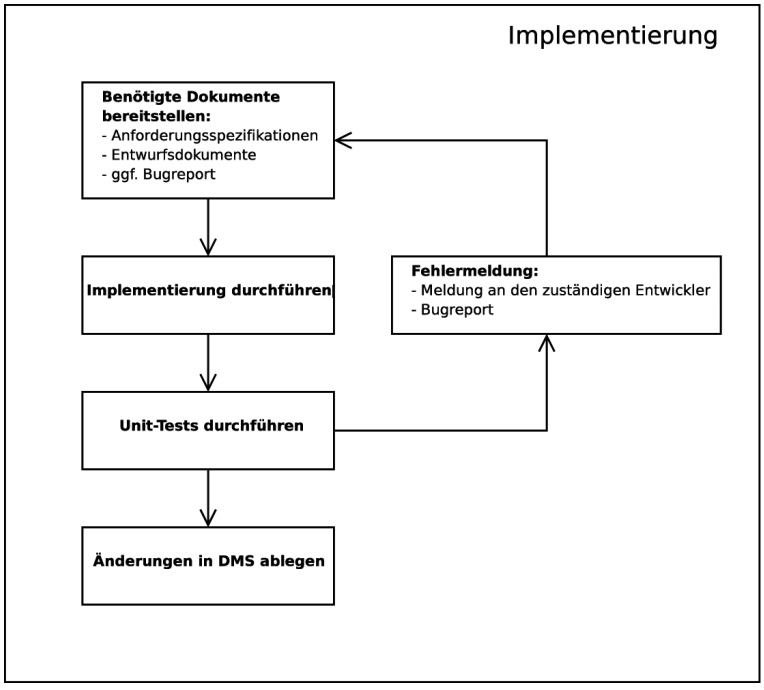

Abbildung 3: Beispiel eines Workflows: Implementierungsphase

Der Workflow besteht seinerseits aus mehreren Arbeitsschritten. Einzelne Arbeitsschritte sollen ausgehend von funktionalen und organisatorischen Anforderungen durch das Workflow-Management-System in eine sinnvolle Reihenfolge gebracht werden. Jablonski [3] unterscheidet zwischen zwei Typen von Arbeitsschritten:

- *Vollautomatisierte Arbeitsschritte, die komplett von Workflowsystem durchführt werden.* Als Beispiel kann hier ein automatisierter Unit-Test dienen, der vom Workflow-System initiiert und ausgewertet werden kann.

- *Arbeitsschritte, die nur der Koordination anderer Vorgänge dienen.* Bei einem Kundengespräch, das weiteren Systemanforderungenspezifizierungen dienen soll, können dem zuständigen Mitarbeiter beispielsweise die Kundenadresse, ein Terminvorschlag und die benötigten Dokumenten vom Workflow-System zugeteilt werden. Die eigentliche Handlung findet ausserhalb des EDV-Systems statt. Ergebnisse dieser Handlung (ein Bericht) werden anschließend wieder in das Workflow-System eingebunden.

Arbeitsschritte, die nicht automatisch durchführt werden können, erzeugen dabei *Aufgaben*, die einzelnen Mitarbeitern zugeordnet werden. Diese Aufgaben können selbst als Dokumente aufgefasst werden und enthalten Eigenschaften wie Beschreibung, Frist und Status. Die Gesamtheit aller Aufgaben, die einer Person zugewiesen sind, bilden eine so genannte *Arbeitsliste* (Abbildung 4). Ist eine Aufgabe von dem Mitarbeiter abgeschlossen, so wird sie vom Workflow-System aus der Liste entfernt. Anschließend werden weitere Arbeitsschritte bestimmt, was ggf. zur Generierung weiteren Aufgaben führt.

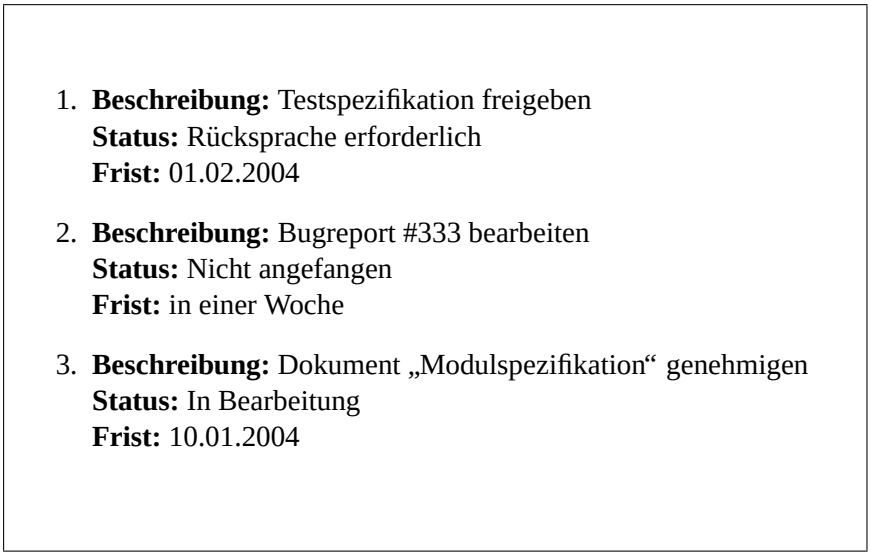

Abbildung 4: Arbeitsliste

3.2.2 Architektur eines Workflow-Systems

Workflow-Systeme können sehr unterschiedlich aufgebaut werden. Trotzdem können dabei einige gemeinsame Komponenten hervorgehoben werden (Abbildung 5):

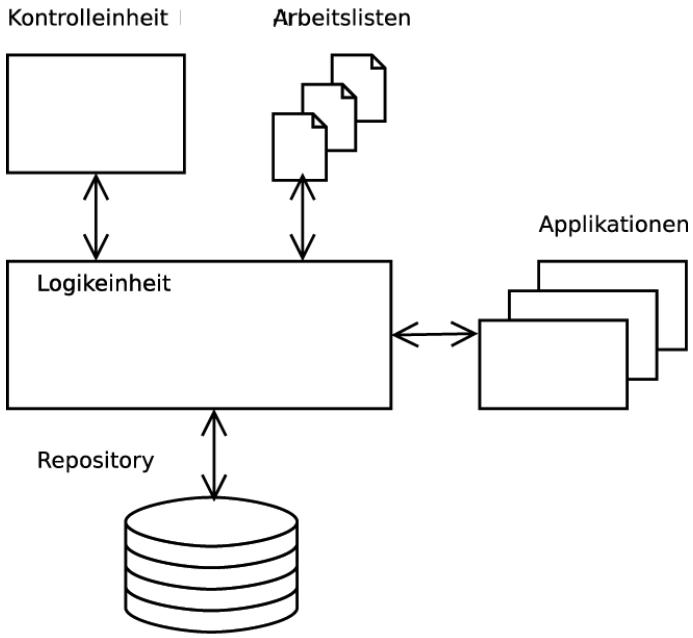

Abbildung 5: Architektur eines Workflow-Systems

- *Logikeinheit* oder *Engine*: Sie implementiert angeforderte Aspekte eines Geschäftsprozesses oder anders ausgedrückt beschreibt den Workflow. Workflowlogik wird meisten mit einer *Workflowsprache* implementiert.

- *Kontrolleinheit*: Dient meistens zum Definieren des Workflows. Der Workflow kann damit aber auch überwacht oder analysiert werden.

- Workflowinstanzdaten werden entweder in der Logikeinheit oder in einem speziell dafür angelegten *Repository* abgelegt.

- *Workflowapplikationen*: Programme, die zum Ausführen von Workflowprozessen benötigt werden: Dokumentenrepository, Entwicklungsumgebung, Bugreportsystem und andere.

- *Arbeitslisten*: Schnittstelle zwischen dem Workflow-System und darin beteiligten Anwendern.

4 Zusammenfassung

Erstellen und Verwalten von Dokumenten verursacht nach wie vor einen enormen organisatorischen Aufwand in einem Softwareprojekt. Dokumenten-Management-Systeme sollen diesen Aufwand durch eine einheitliche Dokumentennutzung reduzieren. Aspekte wie Dokumentenbestand, Einsatzzweck oder Projektgröße stehen maßgebend für die Wahl des richtigen DMS. Es ist anzunehmen, dass für kleinere Softwareprojekte meistens ein klassisches DMS ausreicht. Bei den größeren Projekten kann zusätzlich ein Workflow-System eingesetzt werden, das strukturierte Abläufe gezielt unterstützt. Ein Workflow-System ist dabei nicht als eine in sich geschlossene Technologie anzusehen. Vielmehr soll sie als Middleware-Komponente der Integration von vorhandenen EDV-Komponenten dienen [3].

Literatur

[1] Ulrich Kampffmeyer. Dokumenten-Management: Grundlagen und Zukunft. Hamburg: PROJECT CONSULT GmbH, 1999

[2] Karol Frühauf. Software-Projektmanagement und -Qualitätsssicherung. vdf, Hochsch.-Verl. an der ETH, 1999

[3] Stefan Jablonski. Grundlagen des Workflowmanagements. CSCW-Kompendium: Lehr- und Handbuch zum computerunterstützten kooperativen Arbeiten. Hrsg.: Gerhard Schwabe . . . - Berlin; Heidelberg; New York; Barcelona; Honkong; London; Mailand; Paris; Singapur; Tokio: Springer, 2001

Projektmanagement bei Open Source Projekten

Immo Köster

Inhaltsverzeichnis

1 Einleitung

Mit der Konzeption eines Software-Entwicklungsprojekts als Open Source ändert sich weitaus mehr als man auf den ersten Blick erwarten mag. In den seltensten Fällen beschränkt sich die Interaktion mit der Benutzer- und Entwicklergemeinde darauf, dass man ab und zu auf einen Fehler aufmerksam gemacht wird und vielleicht sogar einen Lösungsweg zugeschickt bekommt. Die organisatorischen, rechtlichen, ökonomischen und technischen Faktoren einer Open Source Entwicklung sollen in dieser Ausarbeitung kurz vorgestellt werden.

Die Konzepte *Open Source* und *Free Software* werden häufig verwechselt. Die wohl größte Gemeinsamkeit beider ist, dass der Quelltext des Programms frei und offen für alle zugänglich sein muss. Zunächst gibt es aber drei andere geläufige Begriffe zur Klassifizierung von Software:

Freeware bezeichnet Software, die kostenlos von jedem genutzt und weiterverbreitet, nicht jedoch verändert werden darf. Da der Quelltext üblicherweise **nicht** veröffentlicht wird, ist Freeware weder Open Source noch Free Software.

Shareware darf ebenfalls beliebig verbreitet werden, wer das Programm jedoch dauerhaft nutzt, muss eine Lizenzgebühr an den Autor zahlen. Oft sind die frei verfügbaren Versionen funktional eingeschränkt und/oder laufen nach einer bestimmten Zeit oder einer festen Anzahl von Aufrufen ab. Shareware ist ebenso wie Freeware urheberrechtlich geschützt, liegt selten im Quellcode vor und darf nicht modifiziert werden.

Public Domain ist ein Konzept, das nur in den USA existiert: dort ist es möglich, das Eigentumsrecht an die Allgemeinheit abzugeben. Nach internationalem Recht gibt es keine Werke ohne Eigentümer/Verantwortlichen. Public Domain ist freie Software, sofern sie im Quelltext vorliegt, allerdings ist es jedem ungenommen, daraus eine modifizierte, proprietäre Fassung zu erstellen.

1.1 Was versteht man unter Free Software?

Der Ausdruck *freie Software* geht auf die Free Software Foundation (FSF) zurück, die von Richard M. Stallman gegründet wurde, eine politische Bewegung, die es sich zum Ziel gesetzt hat, die Benutzer vom Einfluss der Software-Firmen zu befreien.

Das wohl bekannteste Ergebnis dieser Bemühungen ist das GNU Software-Paket. Die FSF will das *free* als frei im Sinne von Freiheit, nicht im Sinne von kostenlos verstanden wissen. Für sie ist die Weitergabe und Veränderung von Software ein demokratisches Grundrecht, da anderen zu helfen die Grundlage der Gesellschaft darstellt.

Die **Free Software Definition** (http://www.gnu.org/philosophy/free-sw.html) fordert von freier Software, dass jeder

- das Programm für beliebige Zwecke ausführen darf
- das Programm an die eigenen Bedürfnisse anpassen darf (dies impliziert eine Offenlegung des Quellcodes)
- das Programm kostenlos oder gegen Gebühr verbreiten darf (z. B. finanziert die FSF so die Entwicklung freier Software teilweise über den Verkauf von CDs)
- modifizierte Versionen verteilen darf, damit alle von Verbesserungen profitieren

Das Gegenstück zum Copyright ist das sogenannte *Copyleft* der FSF, das die oben genannten Freiheiten gewährt, es aber verbietet, Einschränkungen zur Lizenz hinzuzufügen. So wird sichergestellt, dass die Software auch frei bleibt.

1.2 Was ist Open Source?

Im Gegensatz zur FSF ist die Open Source Initiative (OSI) weniger radikal und politisch. Vielmehr geht es der von Eric S. Raymond ins Leben gerufenen Organisation darum, robuste und qualitativ hochwertige Software durch einen offenen Entwicklungsprozess zu fördern. Hier geht es eher um pragmatische als um ideologische Argumente; so wird auch eine Mischung zwischen proprietärer und Open Source Software nicht so stark beschränkt wie bei freier Software.

1.3 Vorteile von Open Source

Stabilität

Ein guter Grund für Open Source Projekte ist die im Allgemeinen hohe Zuverlässigkeit der Software, die ähnlich wie in der Wissenschaft durch *peer reviews* erreicht wird: In beiden Bereichen werden die Ergebnisse der Arbeit von anderen Experten kritisch

geprüft. Da der Quellcode von jedem frei eingesehen werden kann und somit auch bisher unbeteiligte Entwickler Beiträge leisten können, erreicht man leicht einen höheren Wirkungsgrad der Reviews als in der traditionellen Software-Entwicklung, wo oft nur wenige und aus dem direkten Umfeld (der gleichen Unternehmenskultur) stammende Peers beteiligt sind.

Nur wer Zugriff auf den Quellcode hat, kann einen Fehler beheben, was bei Open Source dazu führt, dass viele Fehler auf der Stelle behoben werden können. Zudem ist der mit der Behebung eines bekannten, wichtigen Bugs verbundene Statuszuwachs ein großer Ansporn für viele Entwickler.

Motivation

Es gibt eine weitere Parallele zur Wissenschaft: in der Open Source Welt geht es nicht darum, ein Produkt zu vermarkten. Statt dessen findet man eine sogenannte *gift economy* vor, in der das persönliche Ansehen innerhalb der Gemeinschaft die Motivation ist. In [Ray00] wird der Vergleich gezogen zu „einem freien Markt oder einer Ökologie, einer Ansammlung von selbstsüchtigen Agenten, die versuchen, den Nutzen zu maximieren und im Prozess eine selbstkorrigierende spontane Ordnung erzeugen, die elaborierter und effizienter ist als jeder Aufwand an zentraler Planung es hätte erreichen können“.

Effektivität

Frederick Brooks stellte in [Bro86] fest, dass die Leistung von Entwicklerteams nicht skaliert: Während der Aufwand für die Kommunikation zwischen den Beteiligten quadratisch mit der Anzahl wächst, steigt die erbrachte Leistung nur linear an. Daher stellt sich die Frage, warum Open Source Projekte wie Linux überhaupt Erfolg haben, ja sogar effektiver als die traditionellen Entwicklungsmethoden sein können.

Es gibt drei Effekte, die Brooks’ Law entgegensteuern: Zum einen steigt durch die globale Vernetzung die Qualität der erreichbaren Entwicklergemeinde deutlich an, zum anderen ist die Motivation der an Open Source Projekt Beteiligten höher, da es sich hier um Freiwillige handelt, die ein persönliches Interesse an ihrer Arbeit haben. Und zu guter Letzt trägt das massive *peer review*, das implizit in jeden Open Source Entwicklungsprozess eingebettet ist, zu einer rapiden Verbesserung des Produkts bei.

Standards

Ein wichtiges Argument für die Veröffentlichung eines Programms als Open Source ist die Möglichkeit, dadurch dieses frei zugängliche System als Standard zu etablieren. Ein bekanntes Beispiel dafür ist das X Window System, dass vom MIT entwickelt und unter einer toleranten Lizenz verbreitet wurde, die auch proprietäre Modifikationen

zuliess. Dadurch adaptierten viele Firmen das Programm als Teil ihrer kommerziellen UNIX-Systeme und sorgten so für eine schnelle und weite Verbreitung.

UNIX selbst ist ein Gegenbeispiel für versuchte Standardisierung, da Novells Versuch, innerhalb des X/Open Konsortiums einheitliche Richtlinien vorzugeben, an den Vorbehalten anderer Mitglieder scheiterte [San01, S. 61]. Als einzig glaubwürdiger Ansatz, um Standards zu setzen, gilt heutzutage daher „echte“ Offenheit, also die Möglichkeit aller (im Gegensatz zur Einschränkung auf Mitglieder), standardkonforme Produkte zu erstellen, wie dies auch durch Open Source bzw. freie Software gewährleistet ist.

2 Erste Schritte

2.1 Analyse & Entwurf

Eine wichtige und interessante Beobachtung ist, dass Projekte selten schon im Entwurf oder gar der Analyse Open Source sind. Schließlich weckt eine Projektankündigung immer auch gewisse Erwartungen. Finden Interessenten dann jedoch noch nicht einmal eine Grundfunktionalität (*plausible promise* nach [Ray00]) vor, so werden sich viele wieder abwenden und dem Projekt in Zukunft keine Beachtung mehr schenken.

Ziel ist es also, einen robusten Rahmen für die spätere Entwicklung zu erschaffen, der Anreize schafft, sich an dem Projekt zu beteiligen. Dazu sollte man zunächst das zu lösende Problem so detailliert wie möglich beschreiben. Denn bereits zu Beginn eines OS Projekts spielt die schriftliche Darstellung eine sehr wichtige Rolle. Zum einen fehlt im Gegensatz zum traditionellen Entwicklungsprozess der direkte, (fern)mündliche Kontakt zu den anderen Beteiligten, zum anderen ist es oft die einzige Kommunikationsmöglichkeit, um eine breite Masse ohne Redundanz zu erreichen.

2.2 Evaluation

Zunächst sollte die Frage geklärt werden, ob das Projekt wirklich in einem offenen Entwicklungsprozess ablaufen soll. Wichtig ist beispielsweise, ob das Problem überhaupt für andere von Interesse ist: Eine Lösung für das firmeninterne Netzwerk wird sicherlich nicht die Aufmerksamkeit der internationalen Entwicklergemeinde wecken.

Vorteile eines offenen Entwicklungsprozesses (nach [Kra00]):

- bessere Verantwortlichkeit: Der Prozess kann von außen eingesehen werden, was einen stärkeren Druck auf die Entwickler ausübt
- früheres Testen
- größere Anziehungskraft auf zukünftige Entwickler (breitere Entwicklerbasis)

Ist das Problem von größerem Interesse, so wäre es natürlich denkbar, dass sich bereits andere damit auseinandersetzen. Internet-Portale wie Freshmeat (http://freshmeat.net), Slashdot (http://slashdot.org), SourceForge (http://sourceforge.net) oder die Google Linux-Suche (http://www.google.com/linux) bieten allesamt Möglichkeiten, nach existierenden Open Source Projekten zu suchen. Eine Anfrage an themenspezifischen Newsgroups oder Mailinglisten kann nicht nur die Suche ergänzen, sondern eventuell bereits Interessenten aufdecken.

Da es recht wahrscheinlich ist, dass man dabei zumindest auf Projekte mit ähnlichen, wenn nicht gleichen Zielen stößt, stellt sich schnell die Frage, ob man wirklich ein neues Projekt ins Leben rufen soll. In vielen Fällen mag es sinnvoller (und zeitsparender) sein, sich einer anderen Gruppe anzuschließen, um dieses zu erweitern oder zu verbessern, als mit ihr um Entwickler zu konkurrieren oder das Rad neu zu erfinden. Eventuell trifft man auch auf ein aufgegebenes Projekt, dessen Leitung man übernehmen kann. So hat man eine Ausgangsbasis und profitiert von Erfahrung und Infrastruktur.

2.3 Lizensierung

Wie bereits in der Einleitung beschrieben, definiert sich Open Source und freie Software primär durch die Wahl der Rechte, die vergeben werden. Es können folgende Rechte unterschieden werden (nach [Ray02]):

- Kopieren und Weiterverbreiten
- Benutzen
- Modifizieren für den persönlichen Gebrauch
- Weiterverbreiten modifizierter Kopien

Nach der Open Source Definition sind uneingeschränkte Rechte zur Kopie, Nutzung und Modifikation für den persönlichen Gebrauch Voraussetzung für ein Programm, um sich als Open Source zu qualifizieren. Weiterhin darf die Weitergabe modifizierter Binaries nicht eingeschränkt werden.

Lizenzen, die lediglich einen nicht-kommerziellen Gebrauch des Produkts erlauben, gelten nicht als Open Source Lizenz, da sie bestimmte Berufe, Personen und Gruppen ausgrenzen.

Die **Open Source Definition** (http://www.opensource.org/docs/definition_plain.html) ist selbst keine Lizenz, sondern beschreibt die Rechte, die eine Open Source Lizenz gewähren muss, um als solche anerkannt zu werden. Sie ist aus den **Debian Free Software Guidelines** (http://www.debian.org/social_contract) hervorgegangen.

Für Distributoren ist es wichtig, dass Software unter einer OSD-konformen Lizenz steht, da die Weitergabe des Programmes sonst nicht oder nur sehr schwer möglich ist.

Kein Programm sollte ohne Lizenz veröffentlicht werden, auch von dem Selbsterstellen einer Lizenz sollte man absehen. Falls es sich dennoch nicht vermeiden lässt, ist es möglich, diese neue Lizenz von der OSI zertifizieren und somit als Open Source Lizenz anerkennen zu lassen.

Beispiele für OSD-konforme Lizenzen (http://www.opensource.org/licenses/) sind:

GNU General Public License (GPL) (http://www.gnu.org/copyleft/gpl.html) ist die wohl bekannteste OS-Lizenz, erstellt von der Free Software Foundation und dem GNU Projekt. Wichtigstes Merkmal ist die Anforderung, dass alle von dem lizensierten Produkt abgeleiteten Werke ebenfalls unter der GPL veröffentlicht werden müssen.

GNU Lesser GPL (LGPL) (http://www.gnu.org/copyleft/lesser.html) stellt eine Abwandlung dar, die meist für Bibliotheken eingesetzt wird und das Verlinken mit nicht-freien Programmen erlaubt.

BSD License (http://www.debian.org/misc/bsd.license) Die durch die BSD Unix Versionen bekannte Lizenz erlaubt im Gegensatz zur GPL auch eine kommerzielle Weiterentwicklung der Software, solange der Urheber immer in allen Dokumenten genannt wird. Somit ist zwar nicht mehr garantiert, dass die Änderungen wieder in die Open Source Gemeinschaft zurückfließen, dafür werden solche Projekte aber für Firmen interessanter, die eigene (proprietäre) Fassungen der Software vermarkten können.

Artistic License (http://language.perl.com/misc/Artistic.html) stellt eine sehr liberale Lizenz dar: sie lässt die Wahl, eine modifizierte Fassung frei verfügbar zu machen, die modifizierten ausführbaren Dateien umzubenennen und die Unterschiede zu dokumentieren oder eine beliebige anderen Übereinkunft mit dem Rechteinhaber zu treffen.

2.4 Bezeichnungen & Konventionen

Vor der ersten Bekanntmachung des Projekts gilt es, einen passenden Namen zu finden. Im Idealfall gibt dieser eine Idee davon, was das Produkt später leisten soll und ist überdies einprägsam, so dass jemand, der von dem Projekt gehört hat, dieses auch später wiederfinden kann. Auch hier ist es wichtig, nach gleichen oder sehr ähnlichen Namen zu suchen, um Verwechslungen auszuschließen oder gar mit geschützte Namen in Konflikt zu geraten.

Der nächste Schritt ist, sich ein Schema für die Versionskennzeichnung zu überlegen. Da Open Source für gewöhnlich mehr Revisionen in kürzeren Abständen durchläuft als traditionell entwickelte Software und es keine „internen“ Fassungen gibt, ist es wichtig, darauf zu achten, dass jede Änderung am Programm auch mit einer Erhöhung der Versionsnummer einhergeht.

Major-Minor-Patch (major.minor.patch)

ist das wohl bekannteste Schema: drei durch einen Punkt getrennte ganze Zahlen, wie z. B. 3.1.024 geben den *major level* (wichtige Änderungen und Rewrites), den *minor level* (zusätzliche oder modifizierte Funktionalität bei größtenteils unveränderter Struktur) sowie den *patch level* (Fehlerbeseitigung) an.

Für den Linux Kernel wird eine leicht modifizierte Variante dieses Schemas eingesetzt, hier gibt eine ungerade minor number an, dass es sich um einen Entwicklerkernel handelt, während die stabilen Kernel eine gerade minor number tragen. Die Versionsnummern können also nur innerhalb dieser beiden Klassen direkt verglichen werden. Verwendet man diese Variante nicht, kann es sinnvoll sein, nur gerade minor numbers zu nutzen, um Missverständnisse zu vermeiden.

Year-Month-Day (YYYYMMDD)

Es mag für einige Projekte sinnvoll (oder schlichtweg einfacher) sein, einfach das Datum als Versionskennzeichnung zu nutzen, wie dies beispielsweise beim Windows-Emulator WINE üblich ist. Dabei sollte man darauf achten, die Reihenfolge Jahr, Monat, Tag einzuhalten und mit Nullen aufzufüllen, um sicherzustellen, dass bei einer Interpretation der Version als Ganzzahl keine Probleme auftreten.

In direktem Zusammenhang mit der Versionskennzeichnung steht die Klassifizierung der Stabilität einzelner Versionen. Diese dient den Benutzern als Orientierung bei der Frage, ob die Aktualisierung für ihn wünschenswert ist: In einem Produktionssystem werden gänzlich andere Anforderungen gestellt als für den privaten Gebrauch. Nach [Man00] sollte man sich an folgende Konventionen halten:

pre-alpha feature-(oder API-)unvollständig, sollte grundsätzlich nicht ausserhalb des Entwicklerteams gelangen

alpha feature-vollständig, aber (eventuell) nur teilweise funktionsfähig

beta feature-vollständig und funktionsfähig, aber im Testzyklus

release-candidate mindestens einen Beta-Testzyklus durchlaufen, wird als für den produktiven Einsatz bereit angesehen

2.5 Dokumentation

Die Dokumentation spielt bei Open Source eine noch größere Rolle als in der traditionellen Software-Entwicklung; sie ist *lebenswichtig* für ein Projekt.

Einige Gründe dafür sind:

- Entwickler haben keinen Überblick über vorherige Zusammenarbeit
- Konflikte und Engpässe im Designprozess sind nicht vorhersehbar
- ohne funktionale Spezifikation fehlt dem Entwicklungsprozess die Ziellinie
- später hinzukommende Entwickler benötigen eine Roadmap als Grundlage
- überwiegend schriftliche Kommunikation
- die große Anzahl und der stetige Wechsel der Beteiligten
- die räumliche Verteilung der Beteiligten
- Probleme von Echtzeit-Kommunikationsmedien (Telefon, Videokonferenz)

Es können folgende Zielgruppen ausgemacht werden:

- Benutzer (Readme, Installations- und Bedienungsanleitung, FAQ)
- Tester (aktuelle Fehler, zu testende Bereiche, Vorgehensweise)
- Entwickler (gewünschte Funktionalität, Ziel & Stil des Projekts)
- Projektleiter (Beurteilung des Fortschritts)

Laut [Man00] sollten min. drei essentielle Dokumente vorhanden sein:

- Benutzerhandbuch (Bedienungsanleitung)
- Entwicklerreferenz (Erklärung der API, Namenskonventionen & Werkzeuge)
- Anforderungs-/Entwurfsspezifikation

Gründe für eine Internet-basierte Dokumentation (nach [San01]):

- direkter Zugriff durch alle Beteiligten
- Dezentralität (wahrt Eigentum und Kontrolle der Mitarbeiter über ihre Beiträge)
- Suchmöglichkeiten (insbesondere zukünftig)
- Beschleunigung des Prozesses der Konsensfindung

Zur Webseite eines Projektes sollte neben einer Online-Version der Dokumentation auch ein Bereich für Neuigkeiten rund um das Projekt, Informationen, wie man sich an der Entwicklung oder dem Testen beteiligen kann, direkte Verweise auf Download-Möglichkeiten der Software sowie Verweise auf eventuelle Mailinglisten enthalten.

Zudem sollte es eine zentrale Adresse geben, an der stets die letzte stabile bzw. Entwicklungsversion zu finden ist. Wird keine Bug-tracking Software verwendet, so ist es wichtig, eine konsistente E-Mail Adresse für Fehlermeldungen einzurichten, die (z. B. bei Delegation der Wartungsaufgaben) auch umgeleitet werden kann.

Da gerade Open Source Projekte viele Änderungen in kurzer Zeit durchlaufen, ist es ratsam, den Autor einer Modifikation grundsätzlich auch für die entsprechende Anpassung der Dokumentation verantwortlich zu machen.

Durch eine umfassende und gut strukturierte Dokumentation wird interessierten Entwicklern der Einstieg erleichtert. Im besten Fall können diese so neue Beiträge leisten, ohne dass zuvor die Zeit anderer Beteiligter in Anspruch genommen wurde.

3 Interaktion mit Entwicklern

Auch wenn die Strukturierung dieses Dokuments anders aussehen lässt, so gibt es doch keine klare Trennung zwischen Benutzern, Testern und Entwicklern bei Open Source Software. Die Tatsache, dass die Konsumenten auch an der Produktion teilnehmen, führte zu der Bezeichnung *prosumers*. Mit der Veröffentlichung des Projekts als Open Source legt man die weitere Entwicklung, die Ziele und den Weg dorthin in die Hand der Benutzer- und insbesondere Entwicklergemeinschaft.

Als Leiter eines solchen Projekts muss man in einer Weise Entwickler finden und binden, wie sie in traditionellen Projekten nicht erforderlich ist. Wie [Hil01] es umschreibt: „making responsible decisions and [...] responsibly choosing not to make decisions“.

3.1 Netzwerk-Organisationen

Das Open Source Modell charakterisiert [San01] als *dynamische Netzwerk-Organisation*. Diese besteht aus temporären Allianzen von unabhängigen Partnern mit Schlüsselfertigkeiten, die üblicherweise um ein Projekt herum organisiert sind. Eine Netzwerk-Organisation basiert auf Kontakten und Kommunikation zwischen den Beteiligten und ist weitaus weniger hierarchisch strukturiert als klassische Organisationsformen. Die Autorität geht weniger vom Rang als von der Expertise einer Person aus, und da alle Beteiligten freiwillig an dem Projekt arbeiten, ist es wichtig, sich Vertrauen, Respekt und Einsatzfreudigkeit zu verdienen.

3.2 Delegation von Aufgaben

Sobald das Projekt eine gewisse Bekanntheit erreicht hat, ist es sehr wahrscheinlich, dass sich die ersten Beiträge anderer Entwickler einfinden. An diesem Punkt sollte man

sich langsam damit auseinandersetzen, die Verantwortung für bestimmte Aufgaben zu delegieren. Je größer und komplexer ein Projekt wird, desto wichtiger ist es, eine Gruppe von Entwicklern zu bilden, die den Entwicklungsprozess trägt und wichtige Aufgaben abnimmt.

Dabei sollte man sich an diejenigen wenden, die sich stark für das Projekt engagieren, also viele Ideen, Patches und Diskussionsbeiträge beisteuern. Zudem sollten Aufgaben erst dann abgegeben werden, wenn es notwendig ist. Dafür darf aber auch nicht davor zurückgeschreckt werden, wichtige Rollen wie die des Entwicklungsleiters zu delegieren, wenn sich herausstellt, dass in der Entwicklergemeinschaft jemand mit besseren Fähigkeiten in diesem Feld und kontinuierlicher Beteiligung vorhanden ist.

Ein sinnvoller und praxiserprobter Ansatz ist die Gründung eines Komitees, das zukünftig die Schlüsselentscheidungen trifft, wie dies z. B. bei den Apache und Debian Projekten organisiert ist. Im Falle von Apache spricht man von einer *Meritocracy*, da diejenigen, die sich um das Projekt am meisten verdient gemacht haben, auch die grössten Entscheidungsbefugnisse genießen. Dies ist in vielen Open Source Gemeinschaften (implizit) ähnlich: Hier hat das Wort desjenigen mit der besten Reputation das meiste Gewicht.

3.3 Releases, Branching & Freezing

Zwei Rollen, die sich sehr gut delegieren lassen, sind die des Release Managers und die des Branch Controllers (Kontrolle über einen kompletten Entwicklungszweig).

Die mit einer Release verbundenen Aufgaben sind (nach [Hil01]):

- Koordination des Testens
- Ansetzen eines *code freeze* (s.u.)
- Verantwortung für die Stabilität und Qualitätssicherung
- Zusammenstellung des Installationspakets und Bereitstellung zum Download

Falls das Projekt in verschiedene Zweige unterteilt ist (üblicherweise *stable* und *development*), ist es überlegenswert, die Kontrolle über einen gesamten Zweig an einen geeigneten Entwickler zu übergeben. Branching sollte allerdings mit Bedacht eingesetzt werden, denn auch wenn es einige Probleme im Konflikt zwischen neuer Funktionalität und Stabilität löst, so bringt es doch auch einiges an Verwaltungsaufwand und Verwirrung bei Entwicklern und Benutzern mit sich.

Aus den gleichen Gründen sollte man die Anzahl der Zweige minimal halten, die einzelnen Zweige ausführlich beschreiben und klar voneinander abgrenzen sowie alle jederzeit in konsistenten Stellen (z. B. durch symbolische Verweise) verfügbar halten.

Denkbar ist auch, lediglich im Vorfeld einer neuen Release zwei Zweige zu nutzen, anstatt diese immer parallel laufen zu lassen, wie dies bei der Linux Kernel-Entwicklung geschieht.

Vor einer neuen *stable* Release durchläuft der *development* Branch einen *feature freeze*. Währenddessen werden keine größeren Änderungen oder zusätzliche Merkmale mehr zugelassen, damit man sich auf die Verbesserung der bestehenden Funktionalität konzentrieren kann und Fehler behoben werden können. Dadurch wird eine wiederholte Verzögerung der Veröffentlichung aufgrund ständiger kleiner Verbesserungen und Änderungen vermieden.

Das *code freeze* soll hingegen sämtliche Änderungen am Quelltext verhindern, die nicht der Behebung von Fehlern dienen. Dies findet normalerweise im Anschluss an ein *feature freeze* und unmittelbar vor der Release statt. Freezing ist oft eine gute Möglichkeit, das Ablehnen oder Aufschieben von Patches zu begründen.

3.4 Akzeptieren und Ablehnen von Patches

Das Verwalten von Patches ist eine wichtige Aufgabe, auch und gerade weil man als Projektleiter selten selbst welche schreibt. Zunächst sollte man Vorgaben bezüglich des Formats von Patches machen, um nicht unnötig Zeit auf die Bearbeitung dieser zu verwenden. Da nur bewährte Entwickler Schreibrechte für das Repository haben sollten, sind Patches die sinnvollste Möglichkeit, auch anderen eine Beteiligung an der Entwicklung zu ermöglichen.

Man sollte darauf achten, dass das ursprüngliche Ziel des Projekts stets gewahrt wird und die Evolution lenken, so dass auch Funktionalitäten eingegliedert werden können, die ursprünglich nicht absehbar waren. Dabei ist allerdings darauf zu achten, dass der Fokus nicht zu weit ausgedehnt wird und das Projekt frühzeitig vor lauter „Baustellen“ zusammenbricht. Die Frage sollte stets sein, ob die Modifikation einer signifikaten Anzahl Benutzer zugute kommt und in die Domäne (oder eine natürliche Erweiterung dieser) des Projektes passt.

Daher kann es desöfteren notwendig erscheinen, einen Patch zu verwerfen. Da dieser immer eine freiwillig und unaufgefordert geleistete Arbeit darstellt, ist es sehr wichtig, niemals die Verantwortung für das Projekt und den Respekt vor dem Autor zu vergessen, denn die Ablehnung kann schnell den Verlust des Urhebers zur Folge haben. Open Source Entwicklung bedeutet Teamarbeit, und auch als Begründer und Leiter eines Projekts hat man keine Besitzansprüche mehr an diesem!

Wichtige und/oder schwierige Entscheidungen kann man daher gut an die Gemeinschaft zurückgeben, anstatt sie alleine zu entscheiden, indem man sie z. B. in einem Komitee, einer Mailingliste oder einem Forum für Entwickler zur Diskussion stellt. Dies vermeidet nicht nur den Eindruck eines rigiden und machtbasierten Führungsstils, sondern stärkt auch die Bindung zur Gemeinschaft und kann neue Aspekte zum Vorschein bringen.

Da es (gerade zu Projektbeginn) um den Aufbau einer motivierten Entwicklergemeinde geht, lohnt es sich oft, auch unvollständige oder fehlerbehaftete Patches einzuarbeiten oder zumindest nicht gleich zu verwerfen, sondern aufzuschieben. Dem Urheber sollte die Entscheidung stets begründet werden und eine Ablehnung mit der Aufforderung verbunden sein, auch in Zukunft mitzuwirken. Die Leistung sollte anerkannt und auch kleinere Beiträge öffentlich gewürdigt werden, um die Motivation aufrecht zu erhalten.

4 Interaktion mit Benutzern

Aufgrund der Tatsache, dass es in der Open Source Gemeinschaft nach Möglichkeit vermieden wird (und aufgrund des offenen Quellcodes auch werden kann), dass zwei Projekte in direkter Konkurrenz arbeiten oder anderweitig Arbeit doppelt verrichtet wird, ergibt sich eine ungewöhnliche Situation: Der Mangel an Konkurrenz führt dazu, dass die Benutzer oft nicht die Wahl haben und auf ein Projekt angewiesen sind.

Zudem erlaubt es dieser Mangel sogar, die Benutzer zu ignorieren, da man ohne Konkurrenz auch keine Vorlage hat. Daher muss der Entwicklungsprozess intern und/oder durch die Verpflichtung den Benutzern gegenüber gesteuert werden. Der Interaktion mit und dem Eingehen auf die Bedürfnisse der Benutzer kommt somit eine sehr hohe Bedeutung zu.

4.1 Testen

Ebenso wie die Grenze zwischen Benutzern und Entwicklern fließend ist, so ist auch die Trennung zwischen Benutzern und Testern nicht klar. Viele Benutzer sind bereit, ein Programm zu testen, allerdings muss man ihnen die Wahl freistellen, also instabile Versionen auch explizit als solche kennzeichnen (sofern man nicht ohnehin verschiedene Entwicklungszweige im Projekt angelegt hat). Viele Benutzer erwarten stabile Programme und nicht unvollständige, fehlerbehaftete Versionen.

Insbesondere Programme mit grafischer Oberfläche lassen sich nur schwer testen. Hier ist man auf die Mithilfe möglichst vieler Tester angewiesen, die man zunächst einmal finden und dann auch halten muss. Dazu sollte die (De)Installation, das Bug-Reporting und die Aufgabe der Tester möglichst gut strukturiert sein. Auch hier gilt wieder die Maxime, schnell zu antworten und persönlich sowie öffentlich für ihre Arbeit zu danken, um die Motivation zu erhalten.

Weiterhin bietet es sich gerade für die schnell evolutionierende Open Source Software an, anhand von automatischen Testverfahren (*Unit-Tests* & *Sanity-Checks*) auf häufige Fehler zu prüfen und die Grundfunktionalität sicherzustellen. Um das *peer review* muss man sich meist nicht explizit kümmern, solange genügend Entwickler an allen Bereichen des Quelltextes arbeiten.

4.2 Support-Infrastruktur

Gerade bei größeren Projekten mit vielen Benutzern sollte einige Zeit in eine Infrastruktur zu deren Unterstützung investiert werden, so dass die anderen Entwickler und die Nutzer einander helfen oder ihre Fragen schon in einem Teil der Dokumentation geklärt werden. Dies ist neben der Delegation von Aufgaben die wohl effektivste Möglichkeit, Arbeit und Verantwortung abzugeben.

Zur Kommunikation innerhalb der Gemeinschaft der Entwickler, Tester und Benutzer bieten sich webbasierte Foren, Newsgroups und Mailinglisten an, wobei es oft erstrebenswert ist, zwischen verschiedenen Aspekten (wie Entwickler oder Benutzer) zu trennen. Bei Mailinglisten ist es wichtig, auf einfache Bedienung und die Möglichkeit eines Web-Archives zu achten. Zu guter Letzt kann man nicht genügend Möglichkeiten angeben, wie man selbst zu erreichen ist (z. B. E-Mail, IRC, Instant Messenger).

4.3 Veröffentlichung

Wie bereits erwähnt, ist es schwer, den Zeitpunkt der ersten Veröffentlichung abzuwägen. Es gilt einen Kompromiss zu finden, um einerseits genügend Funktionalität als Anreiz für potentielle Benutzer und Entwickler zu bieten, andererseits sollte das Projekt noch nicht zu weit fortgeschritten sein, um den zukünftigen Mitentwicklern noch genügend Freiraum für eigene Ideen zu lassen.

Auch bei weiteren Releases sollte man sich vorher stets die Frage stellen, ob der Zuwachs an Funktionalität, die Anzahl der behobenen Fehler, der zeitliche Abstand zur letzten sowie die Qualität der neuen Release den Aufwand für die Veröffentlichung, aber auch den auf Seiten der Benutzer rechtfertigt.

Nach dem erfolgten Upload des Installationspaketes sollte man auf der Webseite und den Mailinglisten des Projekts sowie relevanten Newsgroups die neue Version ankündigen. Weiterhin bieten sich die Newsgroup comp.os.linux.announce sowie das Portal freshmeat.net für die Bekanntgabe an.

5 Werkzeuge

Anmerkung: Es muß sichergestellt sein, dass interessierte Entwickler Zugang zu den essentiellen Werkzeugen haben, diese also im Idealfall frei und für alle Plattformen verfügbar sind!

5.1 Versionskontrolle & Konfigurationsmanagement

Ein Versionskontrollsystem ist von essentieller Bedeutung, je mehr Entwickler an einem Projekt arbeiten und je weniger starr die Aufgaben und Zuständigkeiten verteilt sind. Dieses erlaubt es, viele Aufgaben wie z.B. das Erstellen von Installationspaketen für verschiedene Plattformen zu automatisieren und deckt Konflikte bei simultanen Änderungen am Quelltext auf. Es wird nicht nur für Quelltexte, sondern auch für Webseiten, Dokumentation und Entwurfsdokumente genutzt.

Um Konfigurationsdateien vor ungewünschte Änderungen zu schützen, die Verteilung und Aktualisierung von Software zu automatisieren und so für eine homogene Systemkonfiguration beispielsweise innerhalb eines Netzwerks zu sorgen, werden sogenannte Konfigurationsmanagement- und 1Softwareverteilungswerkzeuge genutzt. Zudem können Werkzeuge eingesetzt werden, um Software-Metriken zu ermitteln und so kritische Teile eines Programms zu identifizieren.

Das bekannteste System ist wohl das *Concurrent Version System* (CVS), das neben UNIX auch für Windows und Mac OS verfügbar ist. Es integriert ein Client-Server Modell und ist so auch für internet-basierte Projekte geeignet. Im Gegensatz zu vielen anderen erlaubt es mehrere Check-Outs gleichzeitig (*unreserved check-out model*) und umgeht so einige Konflikte, die besonders bei einer großen Entwicklerzahl auftreten, wenn das Recht, Dokumente auszuchecken und modifiziert zurückzuschreiben, exklusiv an eine Person vergeben wird.

Ähnlich wie bei einem Backup-System werden hier nur die Änderungen an einer Datei gesichert, nicht die einzelnen Versionen. So lassen sich speicherplatzschonend jederzeit alle Versionen eines Dokuments abrufen. Logische Zusammenhänge oder Konflikte können jedoch von CVS nicht aufgelöst werden. Aufgrund der großen Bedeutung für und des kontinuierlichen Einsatzes in Open Source Projekten sollte man sich in Versionskontroll- und Konfigurationsmanagementsysteme wie CVS im Vorhinein gut einarbeiten.

5.2 Bug-tracking Software

Auch die Koordination von Fehlerberichten kann schnell so aufwändig werden, dass sich der Einsatz eines Bug-tracking Systems lohnt. Dieses stellt zumeist strukturierte Schnittstellen bereit, um die Berichte der Benutzer und Tester zu sammeln, weiterzuleiten und ihren Status zu verfolgen. Das Beheben eines Fehlers kann so an den zuständigen Entwickler delegiert, die Wichtigkeit eines Fehlers eingeordnet und gleiche Meldungen zusammengefasst werden.

Ein Großteil der Fehler hat (teils umfassende) Änderungen im Quelltext oder anderen Dokumenten zur Folge, so dass eine Verbindung zum Versionskontrollsystem sinnvoll ist, um die Zusammenhänge nachverfolgen zu können. Bekannte Bug-tracking Systeme sind beispielsweise Bugzilla, GNU GNATS, JitterBug (entwickelt vom Samba-Team, webbasiert) und das Debian Bug Tracking System (mailbasiert).

Bugzilla ist im Rahmen des Mozilla-Projekts entwickelt worden und verwaltet To-Do Listen für Entwickler sowie die Priorisierung, Zeitplanung und Verfolgung von Abhängigkeiten. Basierend auf Perl und MySQL, besitzt das Open Source Programm auch ein Web-Interface.

5.3 Groupware

Die Aufgabe von Groupware ist es, Echtzeit- und asynchone Zusammenarbeit sowie Wissensmanagement und Koordination zu unterstützen. In Tabelle 1 sind die vier verschiedenen Gruppen nach [San01] dargestellt.

	Synchronität	
Verteilung	synchron	asynchron
lokal	Abstimmung, Präsentationsunterstützung	gemeinsam genutzte Computer
entfernt	Videokonferenzen, Chat, Instant Messaging	E-Mail, Workflow, Versionskontrolle

Tabelle 1: Unterteilung von Groupware in Kategorien

Der Einsatz von Groupware ist oft problematisch, da die Wahrung der Privatsphäre oft im Gegensatz zu den Interessen der Projektleitung oder anderer Teilnehmer steht: Nicht jeder ist gewillt, an einer Videokonferenz teilzunehmen, Kontaktdaten (z. B. die eines Instant Messengers) oder Termine freizugeben, wie dies für die Konfliktauflösung bei gemeinsamen Kalendern notwendig ist.

E-mail ist die wohl bekannteste und weitverbreitetste asynchrone Groupware-Technik und stellt mit Merkmalen wie automatische Filterung, Weiterleitung, das Anlegen von Gruppen und Dateianhänge sowie Lese- und Empfangsbestätigungen eine recht umfangreiche Funktionalität bereit. Auch Newsgroups, Mailinglisten und Funktionen einer Textverarbeitung wie Anmerkungen und das Nachverfolgen von Änderungen (samt deren Urhebern) stellen einfache async. Groupware-Funktionalitäten dar.

Einige Programme unterstützen mittlerweile auch die synchrone Mitverfolgung von Änderungen, so dass mehrere Personen gleichzeitig von verschiedenen Orten aus an einem Dokument arbeiten können. Oft wird dazu noch ein zusätzlicher Kommunikationskanal wie Chat oder Videokonferenz angeboten. Textuelle Chats bieten den Vorzug, direkt eine Mitschrift zu erzeugen, die man archivieren oder zum Rückverfolgen (beispielsweise für später dazugestoßene Teilnehmer) nutzen kann.

6 Fazit

Der Entwicklungsprozess eines Open Source Projektes unterscheidet sich in vielen Aspekten von traditionellen Vorgehensweisen. Neben der großen Anzahl an nicht klar

zu trennenden Benutzern, Testern und Entwicklern ist auch der freiwillige Charakter der Mitwirkung zu berücksichtigen. Der klassische Führungsstil, basierend auf Macht und Rang, führt hier nicht zum Erfolg. Vielmehr sind Respekt, Reputation, Vertrauen und Motivation die entscheidenden Faktoren, im Umgang mit der Gemeinschaft.

Auch die Delegation von Aufgaben und eine weitreichende Werkzeugunterstützung gehören zum Open Source Entwicklungsprozess. Wichtiger als die technischen Aspekte sind die organisatorischen: eine gute Infrastruktur und Dokumentation kann hier vieles erleichtern.

Man sollte nicht vergessen, dass man mit der Lizensierung eines Programms als Open Source das Projekt aus den Händen gibt und der Gemeinschaft überträgt, auch wenn man die Leitung innehat. Als Gegenleistung zeichnen sich die Produkte eines offenen Entwicklungsprozesses oft durch Stabilität, rasche Evolution und Zuverlässigkeit aus.

Literatur

[Bro86] Frederick Brooks. *The Mythical Man-Month*. Addison-Wesley, 1986.

[Hil01] Benjamin Hill. Free Software Project Management HOWTO. `http://mako.yukidoke.org/projects/howto/`, 2001.

[Kra00] Robert Krawitz. Free Source Project Management. `http://www.advogato.org/article/196.html`, 2000.

[Man00] Montey Manley. Managing Projects the Open Source Way. `http://news.linuxprogramming.com/news_story.php3?ltsn=2000-10-31-001-05%-CD`, 2000.

[Ray00] Eric S. Raymond. The Cathedral and the Bazaar. `http://www.catb.org/~esr/writings/cathedral-bazaar/cathedral-bazaar/`, 2000.

[Ray02] Eric S. Raymond. Software Release Practice HOWTO. `http://www.tldp.org/HOWTO/Software-Release-Practice-HOWTO/`, 2002.

[San01] Jan Sandred. *Managing Open Source Projects*. John Wiley & Sons, Inc., 2001.

Systemdenken und lernende Organisationen

Mathias Funk

4. Februar 2004

Inhaltsverzeichnis

1 Einleitung

Bisher war es immer so, dass Organisationen, seien es Firmen, Nichtregierungsorganisationen (NGOs) oder andere soziale Strukturen, streng hierarchisch ausgerichtet waren, mit wenigen Personen an der Spitze, die entsprechend ihren Visionen und Vorstellungen das Ganze formten und führten. Man kennt und erwartet es speziell bei großen Organisationen nur so: Managen, Organisieren, Kontrollieren. Je komplexer und größer die Organisationen aber werden, je schneller und undurchschaubarer die Prozesse um und in ihnen ablaufen, desto weniger reicht es aus, auf den obersten Hierarchiestufen zu *lernen* und sich den Gegebenheiten der Umwelt anzupassen - Lernen muss möglichst auf *allen Ebenen* stattfinden, um den zukünftigen Erfolg zu sichern. Ein anderer Aspekt ist, dass das heutige Beschäftigungsverhältnis generell andere Bedürfnisse des Beschäftigten erfüllt und erfüllen muss. Früher diente der Job der Überlebenssicherung, heute ist dies jedoch nicht mehr vorrangig der Fall. Vielfach ist „Selbstverwirklichung“ der Anlass, zu arbeiten bzw. eine bestimmte Beschäftigung für sich auszuwählen. Dementsprechend hoch ist die Identifikation mit dem Beruf, die Selbstachtung hängt stark von der im Job erfahrenen Bestätigung ab.

Jeder hat mit Sicherheit schon einmal die Erfahrung gemacht, Mitglied eines „tollen Teams“ gewesen zu sein, einer klar abgegrenzten Gruppe von Menschen mit einem Ziel und dem gemeinsamen Wunsch, dieses Ziel zu erreichen. Sei es beispielweise in einer Fußballmannschaft, einer Band/Theatergruppe oder im Job. Meist war es so, dass sich individuelle Eigenschaften ergänzten und etwaige persönliche Differenzen keine Rolle spielten. Sehr hoher Motivation und Spaß an der Sache folgten wie selbstverständlich beeindruckende Resultate.

Das Gegenteil sind im schlechtesten Fall zerstrittene Teams, demotivierende Vorgesetzte und Hierarchiestrukturen, die persönliche Einflußnahmen unmöglich machen, klar, dass am Ende nur suboptimale Ergebnisse stehen und leistungsfähige Mitarbeiter - mit einer hohen Motivation und starkem Identifikations- und Integrationswillen - abwandern oder abgeschreckt werden.

Hier setzt das Konzept der *lernenden Organisation* an, die die höheren Anforderungen eines Mitarbeiters an seine Tätigkeit respektieren und fördern kann. Ein effizient und hochmotiviert arbeitendes Team kann jedoch nicht direkt zusammengestellt wer-

den oder urplötzlich entstehen, es muss sich entwickeln. Dazu gehört eine Strategie aufbauend auf 5 Disziplinen, jede für sich notwendig:

1. Systemdenken, das Denken in sich beeinflussenden Prozessen
2. Selbstmanagement (engl. Personal Mastery)
3. Mentale Modelle
4. Gemeinsame Vision
5. Teamlernen

Sämtliche Ausführungen zu den lernenden Organisationen basieren auf dem Buch „The Fifth Discipline“ von Peter M. Senge [Sen90]; der Schwerpunkt soll hier auf dem Systemdenken liegen, das die vier übrigen Disziplinen essentiell verbindet und somit der Schlüssel zu den lernenden Organisationen ist.

2 Lernende Organisationen

2.1 Systemdenken

Das Denken in Systemen ist ein Prozess, der sich nicht nur an kurzfristigen Ereignissen und Details orientiert und auf sie reagiert, sondern das große Ganze sieht und auch in langfristiger Hinsicht abwägt. Systemgrößen werden nicht als starr ansehen, sondern als sich ständig verändernde Größen, denn auch die Stabilität einer Größe wird nur durch irgendeine Aktivität erreicht. Alle Systemteile sind folglich selbst Systeme, die einen Teil ihrer Selbst, ihrer Aktivität, ihrer Struktur und ihrer Energie in das betrachtete System einbringen. Systemdenken ist also ein Handeln *mit* den Ereignissen und das aus einem umfassenden Verständnis der ablaufenden Prozesse heraus.

2.1.1 Probleme

Eine neue Art zu denken einzuführen macht nur Sinn, wenn bekannt ist, wo die Probleme liegen und warum Änderungen durchgesetzt werden sollen. Organisatorische Mängel gibt es in fast allen Institutionen und sozialen Strukturen, dementsprechend breit gefächert ist ist das Spektrum möglicher Ausprägungen. Vieles lässt sich jedoch übergreifend beobachten und damit Problemklassen zuordnen, die hier im Ansatz vorgestellt werden sollen.

Auf die Frage nach ihrem Beruf antworten viele Beschäftigte mit einer Aufzählung aller über den Arbeitstag ausgeführten Tätigkeiten. Welchen genauen Zweck diese Arbeitsschritte im Endeffekt haben, ist demjenigen oftmals nicht präsent. Je mehr er

sich mit der eigenen Aufgabe oder der Position innerhalb einer Organisation identifiziert, um so eher fühlt sich der Einzelne als *Rädchen im Getriebe*, das über die eigene Aufgabe hinaus nichts weiter bewirken und verändern kann und soll. Schnell entsteht die Auffassung, dass die eigene Verantwortlichkeit an der Grenze des Aufgabenbereichs endet. Dabei geht der Blick auf das Zusammenwirken der verschiedenen Aufgaben verloren, was zu mehr Fehlern, höheren Kosten etc. führt, wobei die Fehlersuche wiederum dadurch erschwert wird, dass nicht nach Problemen in der Gesamtstruktur gesucht wird, sondern diese natürlich in den Teilbereichen vermutet werden. Dieser „Tunnelblick“ führt zur beliebten Pauschalisierung:

„Irgendjemand hat’s verbockt!“

Es scheint jedem Menschen eigen zu sein, die Schuld zuerst bei anderen, dann evtl. bei sich selbst zu suchen, dabei ist dieses Denken nur eine logische Konsequenz des oben beschriebenen „Tunnelblick“–Phänomens: Wenn man nicht imstande ist, die Folgen seines Handelns zu überblicken, können sie leicht zum Bumerang werden - nur logisch, dann nicht nach eigenen Unzulänglichkeiten, sondern nach externen Fehlerursachen zu suchen. Kommen noch Vorurteile und Ressentiments ins Spiel, gerät die Suche nach den Ursachen eines Problems leicht außer Kontrolle.

„Angriff ist die beste Verteidigung!“

Es ist sicherlich ein guter Ansatz, die Ursachen von Krisen und Problemen so schnell wie möglich zu beseitigen und nicht zu warten, bis deren weitere Entwicklung zu Tatsachen führt, die nur noch begrenzten Handlungs- bzw. Reaktionsspielraum lassen oder überhaupt nicht mehr beherrschbar geworden sind. Jedoch muss man unterscheiden zwischen einem emotional gesteuerten „Erstschlag“, der vielleicht nur die Symptome eines Problems tangiert oder gar im Grunde nur Show ist, und proaktivem, lösungsorientiertem Handeln, also dem Vermögen zu sehen wie man zur Lösung eines Problems beitragen kann. Hierbei spielt die Sichtweise auf Ereignisse in ihrem zeitlichen Verlauf eine große Rolle:

> *„Ein Frosch wird, in kochendes Wasser geworfen, direkt wieder herausspringen, setzt man ihn jedoch in einen Topf mit kaltem Wasser, das dann langsam erwärmt wird, bleibt er sitzen bis das Wasser so heiß ist, dass er wirr im Kopf wird und erst recht nicht mehr heraus kann.“* (Parabel)

Viele Veränderungen in den uns umgebenden Systemen laufen so langsam und unauffällig ab, dass sie sich unserer Aufmerksamkeit komplett entziehen. Oftmals helfen Messinstrumente, langsam ablaufende Prozesse beobachten zu können, so können beispielweise schleichende Verschlechterungen im Wasser, den Nahrungsmitteln, der Luft nur mit genauen Messgeräten und langfristig ausgelegten Messmethoden zuverlässig

erkannt und eingeschätzt werden. Ohne sie würde man vielleicht durch Unwohlsein, Krankheiten oder Todesfälle auf Probleme aufmerksam, ohne jedoch zu wissen, wo die Ursachen liegen und ob Abhilfe überhaupt noch möglich ist.

Die Fähigkeit, auf kurzfristige Ereignisse zu reagieren, hat in der Steinzeit ihre Wurzeln und liegt damit in unserer Natur. Zum Überleben war es nicht unbedingt notwendig, den Zeitpunkt der Sonnenwende durch dauerhafte Naturbeobachtung sicher vorhersagen zu können, sondern den Säbelzahntiger rechtzeitig zu bemerken und darauf zu reagieren. Heutzutage hat sich die Lebenssituation geändert, langfristige, graduelle Veränderungen bestimmen direkt unser Leben und das unserer Organisationen.

> *„Aus Fehlern lernt man."*

Lernen ist im Grunde Konditionierung: Unser Tun provoziert Folgen, sind diese positiv, werden wir die auslösende Handlung als positiv konditionieren, sind sie negativ, werden wir negativ konditioniert. Wir können uns darauf verlassen, dass aus schlechten Erfahrungen gelernt wird und derartige Fehler in der Regel nicht noch einmal begangen werden.

Das Dilemma liegt nun aber darin, dass wir die Konsequenzen langfristiger Prozesse, unserer wichtigsten Entscheidungen nie bewusst *erfahren* können. Unser Lernhorizont ist begrenzt, je weiter Ursache und Wirkung zeitlich und auch örtlich auseinander liegen, umso schwächer ist der Lerneffekt. Selbst periodische Ereignisse, sollten sie länger als ein- bis zweijährlich wiederkehren, sind schwer zu erkennen. Wird in den Medien zum Bespiel über einen Mangel an Arbeitsplätzen innerhalb einer bestimmten Berufssparte berichtet, werden mehr Menschen eine entsprechende Ausbildung beginnen, ohne jedoch zu registrieren, dass der Endpunkt der Ausbildung in eine ganz andere Periode fällt und sie so eventuell später in der Masse der Arbeitssuchenden, die damals zur gleichen Entscheidung bewegt wurden, untergehen werden.

Diese Muster und Beispiele irrationalen Fehlverhaltens zeigen deutlich, dass die umgebende Struktur das Verhalten des Einzelnen beeinflusst. Strukturen sind nicht nur äußere Gegebenheiten, in denen sich der Einzelne als *atomarer* Teil des Ganzen bewegt, sondern können auch auf den menschlichen Organismus bezogen sein, also auf die Mechanismen, die uns Wahrnehmungen registrieren und verarbeiten und später verinnerlichte Vorstellungen, Ziele und Normen in unseren Handlungen widerspiegeln lassen. Oftmals ist für einen Lösungsansatz eine neue Art zu denken notwendig.

2.1.2 Prinzipien

Wurde im vorherigen Abschnitt über Probleme verschiedenster Art berichtet, sollen jetzt Prinzipien des Systemdenkens erläutert werden, die der Problemlösung eigen sind. Viele dieser Prinzipien sind trivial, „gesunder Menschenverstand", man kennt meist ein Sprichwort, das sie treffend beschreibt. Dies macht sie uns vertraut. Dass jedoch ein Prinzip nicht nur in Kindheitheitserinnerungen oder im privaten Bereich zu

treffen kann, sondern aufgrund seiner Generalität auch auf nicht so naheliegende Dinge zutrifft, und dass damit vermeintlich komplizierte Sachverhalte enorm vereinfacht werden können, leuchtet oft nicht ein.

Viele Probleme entstehen zum Beispiel nur, weil in der Vergangenheit falsche Entschlüsse getroffen wurden, deren Auswirkungen auf zukünftige, also jetztzeitige Ereignisse nicht genug bedacht wurden. Ein deutliches Beispiel ist das Jahr-2000 Problem in der Informationstechnik: Wenn Programme Daten untereinander austauschen, verstehen sie sich wesentlich besser, wenn sie alle das Jahr mit gleichvielen Stellen verarbeiten. Und wenn man schon 10 Programme hat, die mit zwei Stellen funktionieren, baut man eben das 11. auch wieder so. Häufig wurden die zweistelligen Jahreszahlen auch formal in Standards und Spezifikationen festgeschrieben, die zu missachten sich kein Programmierer leisten konnte, ganz abgesehen davon, dass sonst sein Programm mit Tausenden von Programmen nicht zusammengearbeitet hätte, die nach diesen Spezifikationen bereits erstellt worden waren.

So haben Entscheidungen, die im Laufe eines Tages, eines Jahres, einer Beschäftigung getroffen werden müssen, nicht nur direkte Auswirkungen auf sichtbare und sofort verständliche Dinge, vieles zieht im Hintergrund - wenn die Aufmerksamkeit schon zum nächsten Problem übergeschwenkt ist - weitere Kreise. Die vermeintliche Wirkungskette hört bereit beim erwünschten Zielzustand auf. Mitte der 90er Jahre galt diejenige Firma als ideal strukturiert, die, möglichst schlank, zahlreiche Aufgaben an Fremdfirmen delegierte und ganze Abteilungen aufgab, Managerjargon: „Outsourcing“. Erst einmal wurden massiv gesenkte Personalkosten an der Börse begeistert aufgenommen, der Trend manifestierte sich. Dann jedoch stellte man fest, dass die Kosten nicht verschwunden waren, sondern an anderer Stelle in der Bilanz wiederauftauchten. Ausgelagerte Tätigkeiten kosteten sogar mehr als vergleichbare interne Leistungen. Hinzu kam, dass die Unternehmen nach und nach die Kontrolle über ihre länger werdenden Lieferketten verloren und die Kernkompetenz in wichtigen Bereichen wie Rechnungswesen, Sicherheit und Informationstechnik abhanden kam. Neuerdings wird wieder „eingesourcet“, jedes viertes Unternehmen holt Aufgaben zurück, die zuvor an Fremdfirmen vergeben wurden. [Vie03]

Ähnlich ist es, wenn Ursachen bewusst oder unbewusst nur zum Teil beseitigt wurden. Kurzfristige Problemlösung wird eben dann gefährlich, wenn die grundlegende Ursache des Missstandes nicht angetastet, sondern oberflächliche Symptome behandelt werden. Die Probleme werden mit großer Wahrscheinlichkeit wieder auftreten und müssen erneut berücksichtigt werden. Das kostet im Endeffekt mehr Zeit und Resourcen als bei einer angemessenen Ursachenbekämpfung.

Ein anderes Problem ist, die optimale Leistung einzuschätzen, also wieviel *dauerhaft* geleistet oder wie schnell ein Wachstumsprozeß werden kann, ohne dass Probleme oder Mangelerscheinungen auftreten. In beruflichen Stresssituationen, also beispielsweise bei zeitlichen Engpässen, versucht man häufig, den Zeitverzug durch Mehrarbeit zu kompensieren: „Dann muss ich eben noch härter arbeiten...“ Unbestritten ist, dass man durchaus eine kurze Zeit mehr als 100 Prozent abliefern kann. Danach braucht jedoch der Körper eine mehr und minder lange Erholungsphase, so dass sich das Arbeits-

ergebnis auf lange Sicht nivellieren wird - und zwar deutlich unterhalb des Idealwerts. Andere „Problemzone“, gleiches Prinzip: Einer Crash-Diät folgt fast unweigerlich der berüchtigte Jojo-Effekt. Dies trifft auch für wirtschaftliche Wachstumsprozesse zu, seien es Erträge, Kunden oder der DAX. Wächst etwas zu schnell, wächst ebenso die Gefahr, dass es zum Kollaps kommt und es danach erst einer Erholungsphase bedarf, bis das Wachstum erneut ansteigen kann.

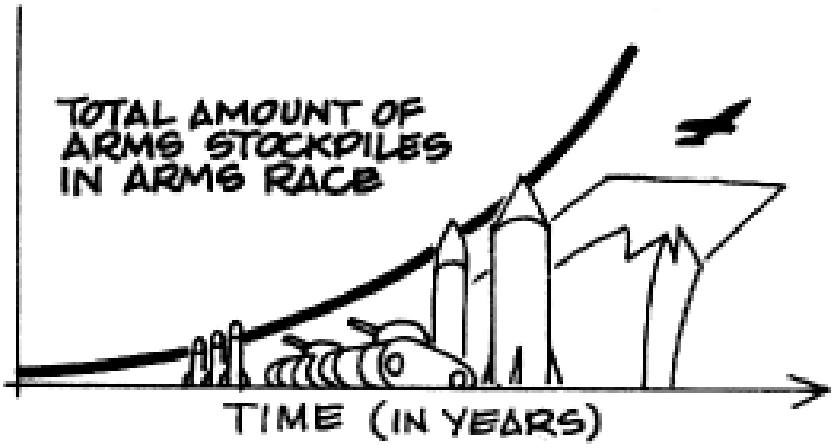

Abbildung 1: Wettrüsten während des Kalten Kriegs

Dies hängt auch mit dem Effekt des „kompensierenden Feedbacks“ zusammen, dem sprichwörtlichen „Teufelskreis“. Jemand versucht einer bestimmten Situation zu entfliehen und die Anstrengungen, die er unternimmt, ketten ihn nur noch fester an diese Situation. Erst wenn der Kreis verstanden wurde, kann er auch durchbrochen werden. Sei es im Kleinen oder im Großen, es ist dasselbe Prinzip. Ein Beispiel ist der Kalte Krieg. Schematisch sieht es so aus: Sowjetische Waffen waren eine Bedrohung für die Amerikaner, die sich wiederum dazu gezwungen sahen, selber aufzurüsten, erneut Bedrohung für die Sowjets usw. Beide linearlogischen Verhaltensweisen (Bedrohung folgt Aufrüstung) beeinflussen sich gegenseitig und schaukeln sich immer weiter auf, mit grausamen Folgen. Der Kreis wurde schließlich von Gorbatschow durchbrochen, ein Wett*ab*rüsten nahm seinen Lauf. All dies hat einen simplen Grund, die Struktur beeinflusst das Handeln (siehe unten). Es gibt keine wirklich trennenden Systemgrenzen, vielmehr ist es oft die geglaubte Separation, die die Existenz gewisser Seiteneffekte verschleiert und so zu falschen Annahmen verleitet, der berühmte „Flügelschlag des Schmetterlings“ der Chaostheorie, der anderenorts Katastrophen auslöst. In diesem Sinne mögen es noch so kleine Ereignisse, Schwachstellen oder (Bau-)Teile sein in der Struktur eines Systems, werden sie entfernt oder funktionieren sie nicht richtig, bricht das System zusammen. Eine kleine Sicherheitslücke im Computersystem kann die Eintrittkarte Böswilliger sein, die nun dem System enormen Schaden zufügen können - an aktuellen Beispielen mangelt es nicht.

2.1.3 Theorie

Systemdenken und damit auch das Konzept der lernenden Organisationen haben ihre Wurzeln in östlicher Religion, verschiedenen Philosophien und weltanschaulicher Strömungen, so gibt es zum Beispiel gewisse Überschneidungen mit dem Taoismus: „[...] Tao ist, wo du dich selbst zu finden hast, wo du dein Inneres zu verstehen hast. Wenn du dich verstehst, verstehst du auch die Anderen. Eine Haltung, die den Anderen so in dich einbezieht, als wäre der Andere du; und auf diese Weise kannst du den Anderen verstehen.“ [vF02]

Hinter dem Systemdenken steckt jedoch mehr als eine (religiöse) Weltanschauung, es baut auf einer konkreten Theorie auf. Das Denken, Planen und Steuern findet gemeinhin auf drei Ebenen statt:

1. Ereignisorientiertes Denken
2. Verhaltensmusterorientiertes Denken
3. Strukturorientiertes, systemisches Denken

Im Abschnitt über die Probleme in Organisationen [Sen90] wird das *ereignisorientierte Denken* als häufige Ursache sich im Nachhinein weiter aufschaukelnder Probleme „gebrandmarkt“. Mehr oder minder kurzfristige Ereignisse werden registriert und als Anlass zum Handeln genommen. Wie oben schon erwähnt, greift dieses *Reagieren* zu kurz, denn längerfristige Abläufe werden selten wahrgenommen, der Impuls zum Verstehen und Handeln findet nicht statt. Betrachtet man nun obige Beispiele mit ein wenig Abstand, fallen die *Verhaltensmuster* direkt ins Auge, mit etwas Übung wird man diese auch selbst erkennen und so nachhaltigere Entscheidungen treffen können, die nicht nur Reaktion, sondern schon *Antwort* sind. Allerdings greift dies immernoch zu kurz, denn ein fester Regelsatz befähigt noch nicht zum eigenen, kreativen Problemlösen. Dazu muss man die Struktur anschauen, denn die umliegende Struktur beeinflusst stark das Handeln: Werden verschiedene Menschen in dieselbe Struktur gesetzt, werden sie sich ähnelnde Ergebnisse produzieren, viele Probleme sind strukturbedingt und eben nicht personell.

Dem Systemdenken liegt zugrunde, dass man Prozesse innerhalb verschiedenster Systeme als aus drei Komponenten zusammengesetzt versteht: Sich verstärkendes Feedback, Ausbalancierung und Verzögerung. Diese grundlegenden Mechanismen laufen parallel ab, interagieren miteinander und beeinflussen so direkt das Gesamtwirken eines Systems:

Sich *verstärkendes Feedback* ist ein zyklischer Wachstumsprozess, der immer dann stattfindet, wenn mehr und mehr Information, Material und Gewinn angehäuft wird oder umgekehrt Arbeitskosten, Ausgaben und die Anzahl von Produktmängeln gesenkt werden. Ein Beispiel ist der Absatz einer Firma, gute Produkte stellen die Verbraucher zufrieden, sie geben ihre Meinung weiter und mehr Leute kaufen die Produkte, die

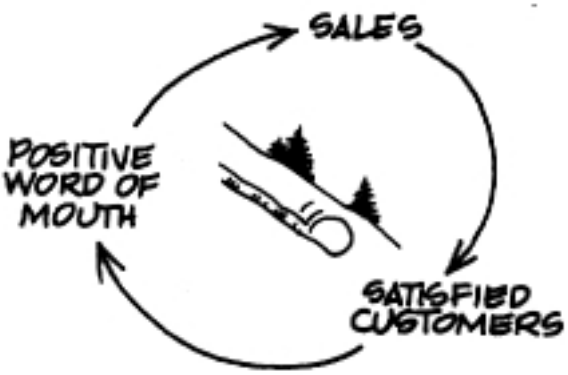

Abbildung 2: Sich verstärkendes Feedback am Beispiel der Kundenbindung

Firma stellt mehr her usw. Wie oben schon beschrieben, gibt es jedoch kein unbegrenztes Wachstum, denn jede sich verstärkende Feedbackschleife wird begleitet von einem *balancierenden* Feedback, die jedes zielorientierte Streben (hier: mehr Umsatz) abbremst, im Beispiel könnte es sein, dass mit steigender Produktion die Firma irgendwann ihr Limit erreicht, nicht genügend Maschinen hat, um die gesteigerte Nachfrage letztendlich befriedigen zu können. Das Wachstum wird zu einem Punkt kommen, an dem es sich stabilisiert und nicht mehr ansteigt.

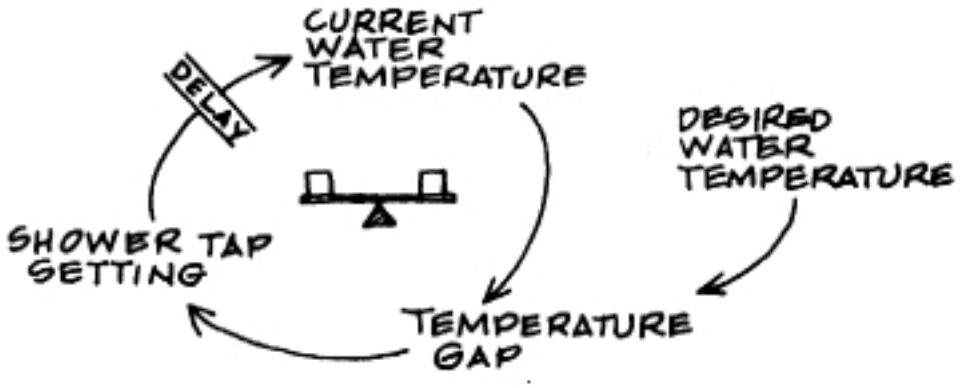

Abbildung 3: Verzögerte Balancierung beim Regulieren der Duschtemperatur

Nun wäre es naheliegend, eine neue Halle zu bauen, frische Maschinen einzukaufen und weiterzuproduzieren. Es gibt aber noch eine dritte Komponente, die *Verzögerungen* innerhalb der Feedbackschleifen. Dies kann im Beispiel so aussehen, dass die aktuellen Verkaufszahlen erst zwei Monate nach der Produktion eintreffen, ansteigende Nachfrage kommt erst 8 Wochen später in den Zahlen zum Ausdruck, vieles kann inzwischen passiert sein, Händler konnten nicht ausliefern, weil die Produktion bis dato noch nicht gesteigert wurde und die Reserven nur knapp waren und aufge-

braucht mittlerweile sind. Die Nachfrage lässt daraufhin nach, weil die Produkte nicht verfügbar sind und das Werk steigert die Produktion, immernoch im Glauben an steigende Kauflust. Das Resultat: Unzufriedene Käufer und Überproduktion. Eine simple Verzögerung kann die bisher „brummende“ Firma des Beispiels komplett aus der Bahn werfen. Mit einem strukturorientierten Ansatz ist die verzögerte Aufstellung der Verkaufszahlen schnell als Behinderung des Wachstums ausgemacht, je aktueller nun die Berichte werden, umso verlässlicher werden sie.

In der Realität sind Systeme allerdings wesentlich komplizierter gestrickt, klar, warum ereignisorientiertes Denken in linearen Wirkungsketten nicht an die Anforderungen der (rückgekoppelten) Realität heranreicht. In der Regel lassen sich systemische Prozesse mit Hilfe dieser drei Mechanismen so weit modellieren, dass Hindernisse und strukturelle Fehler erkennbar werden. Systemdenken ist der ganzheitliche Baustein der lernenden Organisationen.

2.2 Selbstmanagement

Lernende Organisationen können nur entstehen wenn Teilgruppen lernen, was wiederum durch die Lernfähigkeit der einzelnen Mitarbeiter bedingt ist. Man kann jedoch nur erfolgreich lernen, wenn man eine Richtung, ein Ziel vor Augen hat. Grundsätzlich geht es darum, das für einen selbst Wesentliche zu erkennen und eine Vision zu entwickeln.

> *Die Menschen werden geboren, die Menschen sterben, und die Zeit dazwischen verbringen sie mit dem Tragen der Digitaluhren... (Douglas Adams)*

Dieser Vision zu folgen, basiert auf lebenslangem Lernen, einem nicht endenden Prozess. Wichtig ist, dass die Vision intrinsisch ist, d.h. sich an eigenen und nicht an äußeren Maßstäben orientiert, also relativ ist. Relativ angelegte Visionen können kurzzeitig hilfreich und sinnvoll sein, um zum Beispiel eine andere Mannschaft zu schlagen, ohne eine hintergründige, eigene Vision jedoch macht dies keinen Sinn! Eine Vision ist etwas, was man um des inneren Wertes wegen schätzt. Kommt es zur Erfüllung einer Vision, wird eine neue gesetzt, ein kreativer und immer wiederkehrender Prozess des Fokussierens und Refokussierens auf das, was man wirklich will.

Oft ist es schwer, die eigenen Visionen zu formulieren, meist aus dem Grund, dass zwischen ihr und der Realität eine große Lücke klafft, die die Vision unrealistisch aussehen lässt. Diese Lücke jedoch ist der Ausdruck der Kraft von Visionen, denn der Grund, sich zu bewegen, ist gerade der Weg und nicht das Ziel. Wäre der Übergang zwischen Realität und Vision nahtlos, welchen Sinn machten dann überhaupt Visionen? Um also nicht direkt anfangs zu scheitern, ist ein klarer Blick auf die Realität notwendig, so dass Beziehungen zur Vision hergestellt werden können.

2.3 Mentale Modelle

Die Speicherung und Verarbeitung von Aspekten der Realität erfolgt immer über mentale Modelle. Modelle sind Abbilder der Realität die durch Selektion nur spezifische Teilinformationen übernehmen. Dieses angeborene Vorgehen spart Platz und hilft, sich auf die wesentlichen Dinge zu konzentrieren. Gleichzeitig beeinflussen die mentalen Modelle auch das Handeln, da eben nur die gespeicherten Teilinformationen zum Überdenken, Abwägen und Schätzen zur Verfügung stehen. Sie können aus sehr pauschalen Angaben bestehen oder aber auch sehr komplexe Strukturen wie die innerhalb einer Familie genau und korrekt abbilden. Mentale Modelle bilden den Unterschied zwischen „subjektiv“ und „objektiv“, so können zwei Menschen, konfrontiert mit derselben Situation, sich hinterher an verschiedene Dinge besser erinnern, ihre Modelle ließen sie selektiv eben mit verschiedenen Schwerpunkten beobachten.

Probleme treten auf, wenn mentale Modelle unterbewußt sind, sie also von einer Position aus Wahrnehmungen filtern und die Handlungen des Individuums steuern, die nur schwer zugänglich und sehr resistent gegenüber Veränderungen ist. Man nimmt eigene *Annahmen* als *Fakten* war, was mit großer Wahrscheinlichkeit eines Tages zur falschen Entscheidung führen wird. Obwohl das nötige Wissen vorhanden ist, verhindert ein falsches mentales Modell den richtigen Entschluss. Indem die eigenen mentalen Modelle transparent in Frage und vielleicht sogar offen zur Diskussion gestellt werden, kann ein Team einige Hindernisse auf dem Weg zur Vision wegräumen.

2.4 Gemeinsame Vision

Eine gemeinsame Vision ist mehr als eine Idee, sie verbindet eine Gruppe von Menschen, die alle eine starke emotionale Bindung zur ihr aufgebaut haben, nicht zuletzt, weil die gemeinsame Vision auch die persönliche ist. Solche Visionen sind notwendig für lernende Organisationen, sie liefern Fokussierung und Energie, schweißen zusammen zu Menschen „auf einer Mission“. Beispiele gibt es viele von den Kathedralenbauern des Mittelalters bis zur (amerikanischen) Friedensbewegung.

Wie auch persönliche lassen sich auch gemeinsame Visionen positiv oder negativ formulieren. Also entweder mittels „Was wollen wir erreichen?“ oder durch „Was wollen wir vermeiden?“. Man wird letztere negative Visionen häufiger vorfinden als positive. Dies liegt daran, dass es einfach ist, Menschen zu erschrecken, bei ihnen Furcht vor der Arbeitslosigkeit, der Konkurrenz, auf der politischen Ebene vor Kriminalität, Umweltkatastrophen oder Krieg zu wecken. Das *funktioniert*! Gleichwohl sind negative Visionen dreifach beschränkt. Sie binden Energie, die für Neues genutzt werden könnte, um etwas abzuwehren. Zweitens impliziert „ich sollte mich fürchten“ Schwäche, die anfangs nicht vorhanden ist, nun aber fühlbar wird und ein hemmendes mentales Modell der vermeintlichen Gefahr erzeugt. Drittens sind negative Visionen immer kurzfristig angelegt, sie motivieren nur solange die Bedrohung existiert, danach ist die wertvolle Fokussierung verloren: Business as usual.

2.5 Teamlernen

Eine gemeinsame Vision, Fokussierung auf ein Ziel, ist unabdingbar für Teamlernen. Im Idealfall fungiert eine Gruppe als Ganzes, die Ausrichtung auf die Vision bildet das Rahmenwerk, in dem sich alles Gruppendynamische bewegt, sicherlich zentral, die Kommunikation zwischen Teammitgliedern. Hier gilt es Dialog und Diskussion zu unterscheiden. Eine Diskussion hat Gewinner auf der einen Seite und auf der anderen Verlierer, die sich den besten, schlagkräftigsten, manchmal auch nur schillerndsten Argumenten anschließen, denn - nur eine Sicht der Dinge wird am Ende akzeptiert.

Während die Diskussion mehr oder weniger hilft, zwischen mehreren feststehenden Vorschlägen abzuwägen, ist der Dialog ein kreativer Prozess, in dem ein komplexes Thema von vielen Sichtpunkten aus erforscht wird. Jedes Mitglied zieht seine persönliche Schlüsse, die dann kommuniziert werden. Das Ziel ist, gemeinsam weiter voranzukommen, als jeder Einzelne für sich. Dass hier auch ein offenes Arbeiten mit den mentalen Modellen hineinspielt, versteht sich von selbst.

In der Realität ist der Dialog oft gefährdet und droht in eine Diskussion abzugleiten, sobald nur ein Teilnehmer zu beharrlich auf seiner Sicht der Dinge besteht, ein klassischer Teufelskreis: Position A wird behauptet, was eine Bedrohung für Position B darstellt, daraufhin wird Position B bekräftigt, was wiederum eine Bedrohung für A!

Um die Möglichkeiten beider Kommunikationskonzepte im Bewusstsein der jeweiligen Vor- und Nachteile effizient zu nutzen, muss sich ein Team flexibel zwischen beidem bewegen, je nach Bedarf. Grundsätzlich sind auch „funktionierende“ Teams nicht konfliktfrei, deshalb Kommunikationsregeln und Übung! Die Umsetzung kann nur auf freiwilliger Basis erfolgen und ist nicht planbar, das Rahmenwerk ist zwar gegeben, es hängt jedoch von den Teams ab, was sie daraus machen.

2.6 Systemdenken in der Praxis

So universell und geeignet für alle Lebenslagen das Konzept des Systemdenkens auch erscheinen mag, die erfolgreiche Anwendung erfordert Erfahrung und Disziplin. Alle Entscheidungen, die bisher halbautomatisch und aus dem Bauch heraus getroffen wurden, müssen nun einzeln überdacht werden. Da Menschen am besten aus den Konsequenzen des eigenen Handelns lernen, diese aber bei systemischen Entscheidungsprozessen oft außerhalb des Lernhorizonts liegen (siehe oben), wurden Mikrowelten entworfen, in denen die Vorgänge zeitlich und räumlich so komprimiert ablaufen, dass man sie überblicken und mit ihnen experimentieren kann. Mikrowelten begleiten uns von Kindheit an in Form von Puppenhäusern, Legosteinen, Computer- und Rollenspielen und es werden auf diese Weise Regeln und Gesetzmäßigkeiten spielerisch verinnerlicht, die auch in der Realität gelten.

Die Möglichkeiten, Managementprozesse zu simulieren, sind bisher beschränkt, allerdings erlauben es computergestützte Systeme mehr und mehr, komplexe, dynamische Verläufe ablaufen zu lassen. Bislang wird erforscht, wie Mikrowelten Lernprozesse

in Organisationen beschleunigen können. Wichtige Aspekte sind die Ähnlichkeit zwischen Spiel und Realität, die Möglichkeit, Simulationen zu beschleunigen, sie zu verlangsamen oder anzuhalten, bestimmte Parameter zu isolieren und für sich zu betrachten und die einmal gewonnenen Erfahrungen zu archivieren. Vieles ist hier von dem Stand der Technologie abhängig, jedoch hält Peter Senge Simulationen für das Managementwerkzeug der Zukunft [Sen90]. Weitere informationen gibt er zusammen mit anderen Autoren im Buch „The Fifth Diszipline - Fieldbook“ [SKS+94], das konkrete Anleitungen zur Entwicklung lernfähiger Organisationen und Hilfestellungen für konfliktträchtige Situationen enthält - eben die Antworten auf die Frage: „Was müssen wir tun?“

3 Zusammenfassung

Das Konzept der lernenden Organisationen wird vielfach schon seit Jahren erfolgreich angewandt, jedoch ist eine weitere Verbreitung nur langsam möglich. Dies liegt einerseits an den nötigen Umstellungen - nicht jeder hat die erforderliche Beziehung zu seiner Arbeit - empirische Untersuchungen haben ergeben, dass nur ein Viertel der Beschäftigten einer Firma sich aktiv für sie einsetzen, dem größten Teil der Mitarbeiter ist das Schicksal des Arbeitgebers egal, solange der Job gesichert erscheint. Andererseits erscheint das Konzept insgesamt „soft“, etwas esoterisch angehaucht, sich jeder objektiven Messung widersetzend, das fördert die Skepsis. Wenn man sich jedoch anschaut, was der Markt der Managementwerkzeuge und der Manager- und Teamleiterfortbildung sonst so bietet, ist das Konzept der *lernenden Organisationen* im Gegensatz dazu durch eine konsistente Theorie und anerkannte Forschung gestützt und trotzdem aufgrund der anschaulichen Anwendungsbeispiele und dem ausgeprägten Realitätsbezug gut vermittelbar.

Literatur

[Sen90] Peter M. Senge. *The Fifth Discipline*. Currency Doubleday, 1990.

[SKS+94] Peter M. Senge, Art Kleiner, Bryan Smith, Charlotte Roberts, and Richard Ross. *The Fifth Discipline - Fieldbook*. Currency Doubleday, 1994.

[vF02] Heinz von Foerster. *Teil der Welt*. Carl-Auer-Systeme Verlag, 2002.

[Vie03] Ulrich Viehoever. Selbst ist die belegschaft. *Die Zeit*, 2003.

Teil 3

Aspekte der Qualitätssicherung

1 Einleitung

Mit immer größeren und komplexeren Softwaresystemen, die den wachsenden Anforderungen der Anwender gerecht werden müssen, wird es für die Entwickler, die meist unter hohem Zeitdruck arbeiten, immer schwieriger, die Qualität ihrer Produkte zu überwachen. Aus diesem Grund wurde in den letzten Jahrzehnten der Versuch unternommen, neben der Entwicklung eine parallele Qualitätssicherung zu etablieren. Zur Bewertung verschiedener Qualitätsaspekte werden hierzu unter Anderem auch sogenannte *Metriken* erhoben, die ein quantitatives Maß für die Erfüllung einer gewissen Eigenschaft darstellen.

Viele der Ansätze, bestimmte Qualitätsmerkmale zu bewerten, ebenso wie viele der dazu verwendeten Metriken beziehen sich allerdings auf Details der Implementierung eines Systems. So gibt zum Beispiel die *Zyklomatische Zahl* die Komplexität einer Methode auf einer Skala von 1 bis 15 an. Häufig können aber Probleme der Software auf Ungenauigkeiten und Fehler zurückgeführt werden, die bereits im Entwurf der Architektur aufgetreten sind, zu einem Zeitpunkt, an dem die Implementierung noch unbekannt war. Da diese Fehler in späteren Phasen der Entwicklung nur schwer behoben werden können, besteht ein Interesse daran, sie schon im Entwurf zu entdecken. Es werden also Methoden und Formalismen benötigt, die die Qualität eines Entwurfs bewerten können.

In den folgenden Kapiteln sollen 2 Qualitätsmodelle vorgestellt werden, die diese Aufgabe übernehmen können. Als Grundlage dienen hierfür die Eigenschaften objektorientierter Entwürfe, wie zum Beispiel die Anzahl der Klassen und ihre Beziehungen untereinander. Diese können durch die UML modelliert werden und setzen keine Kenntnisse über die Implementierung voraus. Allerdings lassen diese Modelle keine rein objektive Bewertung zu, sondern präsentieren ihre Ergebnisse als Bewertungsvorschläge, die teilweise von Sachverständigen aufgrund gegebener Metriken festgelegt werden müssen. Entwickelt wurden beide Modelle hauptsächlich von Ralf Reissing ([Rei01], [Rei02]), einem ehemaligen Mitarbeiter des Instituts für Softwaretechnologie an der Universität Stuttgart. Bevor jedoch die Qualitätsmodelle vorgestellt werden, beschreibt das nächste Kapitel zunächst kurz das UML Meta-Modell, das als Grundlage dienen soll. Dieses Meta-Modell definiert, wie Entwürfe in UML aufgebaut werden und welche Elemente zu diesem Zweck genutzt werden können.

2 Das UML Meta-Modell

Ziel der UML (**U**nified **M**odelling **L**anguage) ist es, eine eindeutige Sprache und Notation für den Entwurf und die Analyse von objektorientierten Systemen bereit zu stellen.

Die Elemente und ihre möglichen Beziehungen, die in der UML zur Darstellung und Definition von Architekturen genutzt werden können, werden im UML-Standard festgelegt. Ein auf der Basis des UML Meta-Modells entworfenes Modell gilt dann als

eine Instanziierung dieses Meta-Modells. Hier sollen lediglich die Teile des UML Meta-Modells kurz vorgestellt werden, die zur Entwicklung der Qualitätsmodelle in den späteren Kapiteln benötigt werden. Die verwendeten Definitionen basieren auf der Version 1.3 der UML-Spezifikation[OMG99], höhere Versionen verwenden hierfür die Notation MOF (**M**eta **O**bject **F**acility).

2.1 Modellelemente - Entitäten

Abbildung 1 zeigt einen Ausschnitt aus der UML-Spezifikation, der hier betrachtet werden soll, als sogenanntes Klassendiagram.

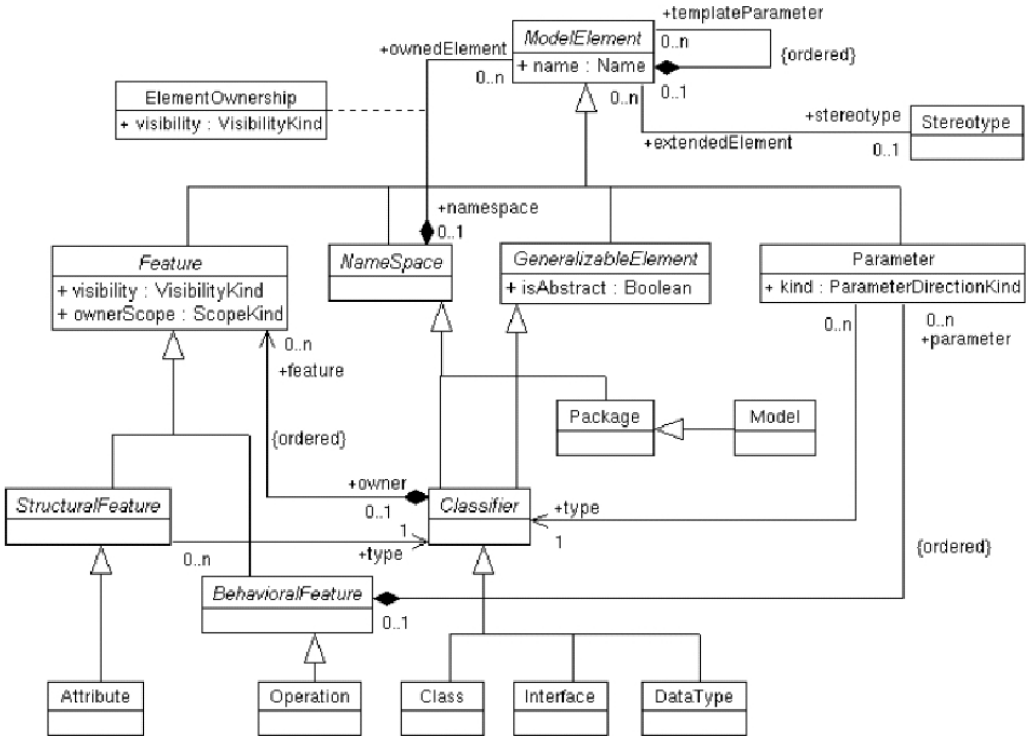

Abbildung 1: Ausschnitt des UML Meta-Modells für Klassendiagramme

Das allgemeine Modellelement *ModelElement* stellt die Klasse dar, von der alle weiteren Elemente erben. Sie besitzt lediglich einen eindeutigen Namen zur Identifizierung und kann sich in einem *Namespace* befinden. Letzterer ist selbst eine Unterklasse von *ModelElement* und kann eine beliebige Anzahl von Elementen enthalten, womit es neben *GeneralizableElement* eine Oberklasse von *Package* darstellt, von der sich wiederum das eigentliche *Model* ableitet. Weitere direkte Unterklassen des allgemeinen Modellelements sind *Feature* und *Parameter*. Ebenfalls von *Namespace* und *GeneralizableElement* abgeleitet ist die Klasse *Classifier*, die mehrere *Feature* und *Parameter* enthalten kann und Oberklasse von *Class*, *Interface* und *DataType* ist.

2.2 Modellelemente - Beziehungen

Auch die Beziehungen in UML-Modellen leiten sich von dem allgemeinen *ModelElement* ab, wie in Abbildung 2 deutlich wird.

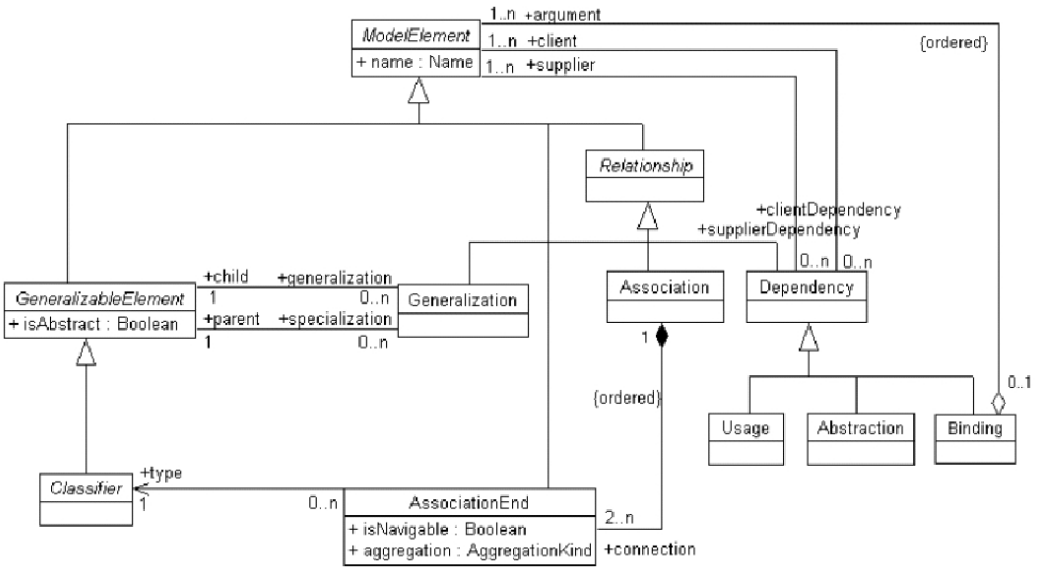

Abbildung 2: Beziehungen im UML Meta-Modell

Relationship ist dabei Oberklasse der Beziehungen *Generalization*, *Association* und *Dependency*, die in Modellen zwischen Klassen bestehen können. Die *Association* hat dabei beliebig viele *AssociationEnds*, denn eine *Association* kann mehrere *Classifiers*, also *Classes*, *Interfaces* und *DataTypes* verbinden, die sich von *GeneralizableElement* ableiten, während *Generalization* und *Dependency* jeweils genau 2 *Classifiers* verbinden. *Usage* sowie *Abstraction* stehen für Abhängigkeiten und erben somit von *Dependency*.

3 ODEM - das Referenzmodell

Auf der Grundlage des UML Meta-Modells kann nun ein Referenzmodell erstellt werden, dass die Eigenschaften eines Entwurfs definiert. Diese können dann für die Definition von Metriken genutzt werden, die die Eigenschaften quantifizieren und einen Entwurf unabhängig von der Implementierung bewerten. Als Basis sollte dazu der Teil des UML Meta-Modells identifiziert werden, der bei objektorientierten Entwürfen heutzutage am häufigsten zum Einsatz kommt. Hierdurch bleibt das Referenzmodell übersichtlich und kann auf einen Großteil der Entwürfe angewandt werden. Aus diesem Grund liegt hier der Focus auf dem Teil der UML, der den Aufbau von Klassendiagrammen beschreibt, wie er schon im vorigen Kapitel vorgestellt wurde. Das darauf

basierende Referenzmodell, das hier vorgestellt werden soll, heisst ODEM (**O**bject-oriented **De**sign **M**odell) und beschränkt sich nicht nur auf die so getroffene Auswahl an Aspekten der UML, sondern enthält zusätzliche Attribute. Letztlich sollen Metriken abgeleitet werden, die später zur Bewertung von Entwürfen herangezogen werden können. Dabei bezieht sich ODEM auf reine Entwürfe, deren Implementierung noch unbekannt sind.

3.1 Focus im UML Meta-Modell

Das Fehlen jeglicher Kenntnis der Implementierung führt dazu, dass bestimmte Aspekte der UML gar nicht zum Einsatz kommen können. Die Definition von Datentypen zum Beispiel werden ausgelassen, während die Datentypen selber als Typ von Parametern und Attributen Verwendung finden. Ebenso wird nicht festgehalten, ob ein Attribut eine Konstante ist oder nicht.

Beziehungen zwischen Operationen, die üblichweise in Sequenzdiagrammen dargestellt werden, finden keine Berücksichtigung. Ob die definierten Operationen abstrakt sind, also ob sie durch konkrete Methoden umgesetzt werden müssen, wird nicht beachtet, da ODEM Methoden als Details der Implementierung auffasst und sie aus diesem Grund übergeht. Ausserdem sind Konstruktoren nicht von Bedeutung, da sie zwar eine Sonderrolle als erzeugende Operation in der Klasse spielen, aber ihre Komplexität in der Regel nicht beeinflussen. Vererbt werden sie nicht, da jede Klasse ihren eigenen Konstruktor braucht, der über den Namen an sie gebunden ist.

Darüber hinaus werden Template-Klassen sowie die Möglichkeit, Klassen oder Interfaces innerhalb bereits bestehender Klassen zu definieren, weggelassen. Dies soll verhindern, das Modelle zu komplex und unübersichtlich werden. Assoziationsklassen und Subsysteme entfallen auch, da sie Mischungen aus Klassen und anderen Modellelementen darstellen und nur sehr selten benötigt werden.

3.2 Zusätze zum UML Meta-Modell

Mit den so gegebenen Mitteln lassen sich Metriken allerdings kaum beschreiben. Aus diesem Grund führt ODEM neue Elemente und Eigenschaften ein, die auf den bereits vorhandenen Aspekten aufbauen.

Der gesamte Entwurf wird nun als System S aufgefasst. Die Elemente des Systems sind darin wie folgt in Mengen aufgeteilt:

- P enthält alle Pakete des Systems, auch S selbst.
- I enthält alle Interfaces des Systems.
- C enthält alle Klassen des Systems.
- O enthält alle Operationen des Systems.

- *A* enthält alle Attribute des Systems.
- *M* enthält alle Parameter des Systeme.

Desweiteren benötigt ODEM einige Relationen, die die Beziehungen von Modellelementen formal festhalten:

- *Associates* beschreibt Assoziationen zwischen Klassen, sowie Klassen und Interfaces.
- *Realizes* beschreibt, welches Interface eine Klasse realisiert.
- *Extends* beschreibt, welche Klasse oder welches Interface eine Klasse erweitert.
- *Depends_on* ist die Vereinigung aus *Associates*, *Realizes* und *Extends*.
- *Contains* beschreibt, welche Packete, Interfaces und Klassen ein Packet enthält.
- *Has* beschreibt, welche Operationen und Attribute ein Interface oder eine Klasse enthält, oder welche Parameter eine Operation bei einem Aufruf erhält.
- *Uses* beschreibt, welche Klassen oder Interfaces eine Klasse oder ein Interface benutzt.

Um auch geerbte Beziehungen erfassen zu können, sind in ODEM auch die transitiven Abschlüsse der genannten Relationen enthalten. Bezogen auf eine Klasse k enthält zum Beispiel *Extends** zusätzlich die Klassen und Interfaces, die von den Klassen/Interfaces erweitert werden, die die ursprüngliche Klasse k erweitern, die von diesen erweiterten Klassen und Interfaces, usw. Sollten mehrere Relationen gleichen Typs, zum Beispiel mehrere Assoziationen, zwischen 2 Elementen bestehen, so kann dies über eine entsprechende Gewichtung in einer *Associates*-Beziehung zusammengefasst werden. Abbildung 3 zeigt die wichtigsten Elemente und die möglichen Relationen von ODEM im Überblick.

3.3 Entwurfs-Metriken

So ist es möglich, verschiedenste Metriken zu definieren, die aufgrund der Herleitung von Elementen und Relationen in ODEM unabhängig von Implementierungsdetails funktionieren. Zusätzlich soll versucht werden, bekannte Metriken, die sich in der Praxis bereits als nützlich erwiesen haben, in ODEM zu beschreiben, um dessen Praxistauglichkeit zu zeigen. In beiden Fällen werden nur einige Beispiele vorgestellt, um die Handhabung zu verdeutlichen.

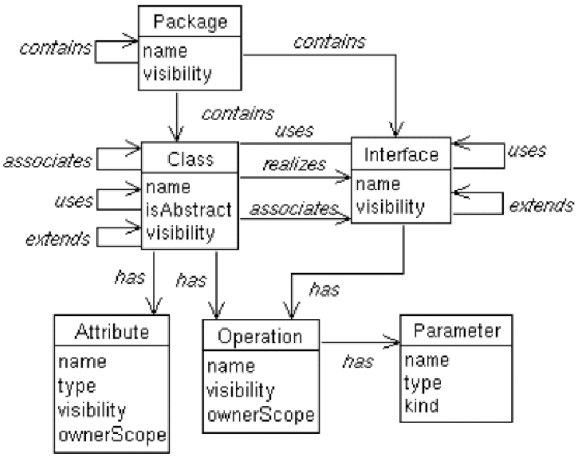

Abbildung 3: zusätzliche Elemente in ODEM

3.3.1 Definition neuer Metriken

Metriken, die sich direkt auf das System beziehen, lassen sich meist aus Metriken anderer Ebenen (Pakete, Klassen) zusammensetzen oder ableiten. Ausnahmen wie *NIH* (Number of inheritance hierarchies), die die Anzahl aller Wurzelklassen enthält, also der Klassen, die von keiner anderen Klasse erben, lassen sich ebenfalls durch die Relationen in ODEM beschreiben. In diesem Fall wird *extends* genutzt:

$$NIH(S) = |\{c \in C : \nexists c' \in C : extends(c, c')\}|. \quad (1)$$

Neben der Anzahl aller Klassen in einem Paket (*NCP*), die mit *contains* formalisiert werden kann, ist für Pakete auch die Tiefe ihrer Verschachtelung ineinander (*DNH*) von interesse. Auch hierfür kann *contains* eingesetzt werden:

$$DNH(p) = DNH(p' : contains(p', p)) + 1. \quad (2)$$

Für Klassen lassen sich eine Vielzahl brauchbarer Metriken finden. So gibt *NLA* (Number of local Associations) die Anzahl lokaler Assoziationen einer Klasse an:

$$NLA(c) = |\{c' \in C : associates(c, c')\}|. \quad (3)$$

Dabei wird nicht berücksichtigt, wenn mehrere Assoziationen zwischen 2 Klassen bestehen. Soll dies in die Bewertung einfließen, so kann hierzu die Gewichtung der Relation genutzt werden.

3.3.2 Formalisierung bestehender Metriken

Eine der bekanntesten Metriksuiten für objektorientierte Systeme wurde 1994 von *Chidamber und Kemerer* veröffentlicht. Die darin enthaltenen Metriken können teilweise

recht einfach in ODEM formalisiert werden, allerdings können auch Probleme auftreten.

Von Interesse ist zum Beispiel *DIT* (Depth of inheritance Tree), die maximale Länge der Vererbungsketten ausgehend von einer Klasse des Systems. Mit den gegebenen Relationen und Mengen ergibt dies:

$$DIT(c) = \begin{array}{l} if(|\{c' \in C : extends(c, c')\}| > 0) \\ \quad then\, max_{c' \in C : extends(c,c')}\{DIT(c') + 1\} \\ \quad else\, 0 \\ endif. \end{array} \tag{4}$$

Eine weitere Metrik ist die Anzahl der Klassen, die direkt von einer angegebenen Klasse abgeleitet werden, erfasst durch die Metrik *NOC*. In ODEM lässt sich diese ebenfalls mit Hilfe der Relation *extends* wie folgt definieren:

$$NOC(c) = |\{c' \in C : extends(c', c)\}|. \tag{5}$$

Da diese Metriken ausschliesslich auf den *extends*-Beziehungen beruhen, lassen sich sie sich also problemlos umsetzen.

Eine der Metriken, die sich nicht problemlos formalisieren lassen, heißt *WMC* (Weighted Methods per Class), die die Summe der Komplexitäten aller Methoden einer Klasse beinhaltet. Innerhalb von ODEM stößt dies jedoch auf mehrere Einschränkungen: Die Komplexität einer Methode hängt von ihrer Implementierung ab und ist somit in diesem Zusammenhang nicht nachvollziehbar. Weiterhin kennt ODEM nur abstrakte Operationen, also keine direkten Methoden. Eine mögliche Formalisierung ist die Definition als Anzahl aller Methoden einer Klasse:

$$WMC(c) = |\{o \in O : has(c, o)\}|. \tag{6}$$

Dies weicht aber stark von der ursprünglichen Definition ab, da u.A. die Komplexität jeder Operation auf 1 festgelegt ist.

Es zeigt sich also, dass ODEM nicht für alle bestehenden Metriken geeignet ist. Dabei könnte das Modell um zusätzliche Elemente und Relationen erweitert werden, um einige dieser Metriken zu erfassen. Dies würde jedoch einen weitaus komplexeren, detailreicheren Entwurf voraussetzen, wie er in der Praxis nur selten zu finden ist. Rein entwurfsspezifische Metriken, die keine Kenntnisse über die Implementierung voraussetzen, bewegen sich allerdings im Rahmen dieses Modells und sind somit schnell und ohne nennenswerte Einschränkungen umsetzbar.

4 QOOD - ein allgemeines Qualitätsmodell

Mit ODEM steht also ein Modell zur Verfügung, welches es ermöglicht, Eigenschaften eines Entwurfs zu messen. Dabei wird aber nicht festgelegt, welche Qualitätsmerkmale (Wartbarkeit, Wiederverwendbarkeit, etc.) die einzelnen Metriken wirklich beein-

flussen. Darüber hinaus sind keine Angaben hinsichtlich der Prioritäten der Qualitätsmerkmale, die in unterschiedlichen Projekten stark variieren können, im Modell vorgesehen. Diese Erweiterungen werden in *QOOD* (**Q**uality Model for **O**bject-**O**riented **D**esign) vorgenommen, ein weiteres Qualitätsmodell, das auf ODEM aufbaut.

4.1 Die Ebenen des Modells

QOOD ist in 3 Ebenen aufgteilt: Die oberste Schicht bilden die sogenannten *Faktoren*, die für die angesprochenen Qualitätsmerkmale stehen, welche es zu bestimmen gilt, um die Qualität des Entwurfs bewerten zu können. Auf die Faktoren beziehen sich wiederum die *Kriterien*, die die zweite Ebene darstellen. Sie ergeben sich aus den Eigenschaften des Entwurfs, die die Faktoren beeinflussen und nach bestimmten Merkmalen ausgesucht werden. *Metriken* bilden die dritte und letzte Schicht, mit denen sich Faktoren und Kriterien quantifizieren lassen. Anders als in ODEM werden hier nicht nur objektive (die sich automatisch über die Mengen und Relationen bestimmen lassen), sondern auch subjektive Metriken eingesetzt, die nach Möglichkeit von Sachverständigen eingeschätzt werden sollten. So lässt sich zum Beispiel die Qualität der vorhandenen Dokumentation (ihre Verständlichkeit, Übersichtlichkeit, usw.) nur zu einem sehr geringen Teil automatisch bestimmen, so dass hier die Bewertung von erfahrenen Mitarbeiter benötigt wird.

So werden mit Hilfe der Metriken zunächst Bewertungsvorschläge für die Kriterien aufgestellt. Diese fliessen dann zusätzlich in die Bewertung der mit den Kriterien verbundenen Faktoren ein, die dann zur Bewertung der Gesamtqualität herangezogen werden. Diese Prozedur wird zunächst auf der Ebene der Klassen angewandt, und danach für alle Pakete wiederholt, wobei die Ergebnisse der Klassen und Interfaces in die Bewertung ihrer Pakete einbezogen werden. Die letzte Iteration der Prozedur erfolgt auf der Systemebene, die wiederum auf die Bewertungsvorschläge der Paketebene zurückgreift. Abbildung 4 verdeutlicht den Zusammenhang von Ebenen und Schichten in diesem Verfahren.

Die einzelnen Schichten von QOOD werden nun noch einmal genauer vorgestellt. Dabei sollen auch speziell ihre Verbindungen untereinander verdeutlicht werden.

4.1.1 Faktoren

Zunächst müssen also die Qualitätsmerkmale bestimmt werden, die für QOOD von Interesse sein könnten und auf die ein Entwurf zu überprüfen ist. Als Quelle für mögliche Merkmale stehen verschiedene Modelle zur Verfügung, wie sie in [Bal98] (S. 257 ff.) beschrieben werden.

Wartbarkeit gibt darüber Auskunft, wie schnell Fehler im System gefunden und behoben werden können, wie leicht es verbessert oder an veränderte Bedingungen angepasst werden kann und wie hoch der Aufwand ist, wenn neue Funktionen hinzugefügt

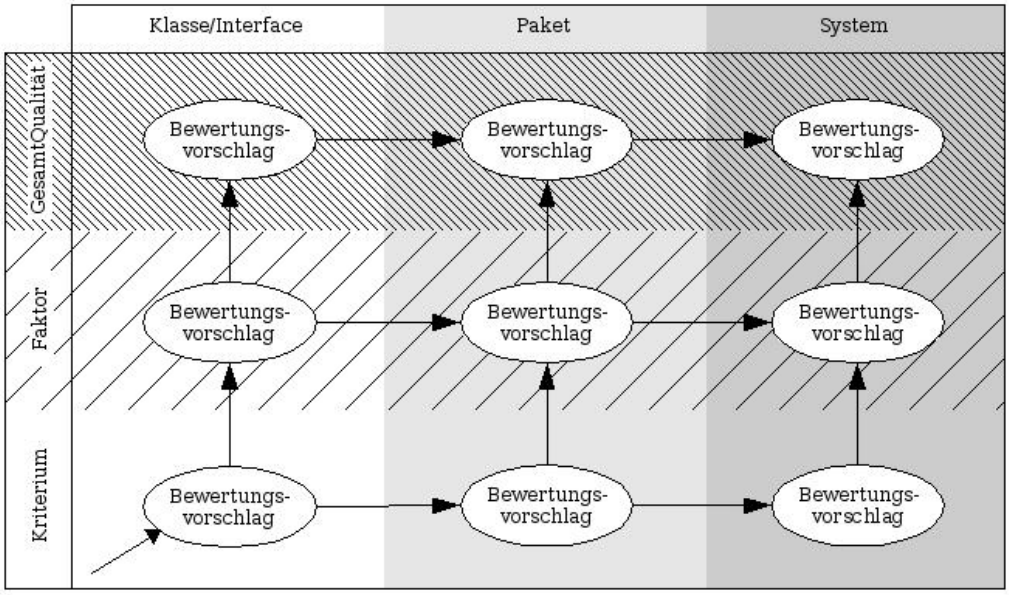

Abbildung 4: Ebenen und Schichten im QOOD-Bewertungsverfahren

werden sollen. Sie spielt in QOOD eine besonders wichtige Rolle, da die Wartung moderner Software heutzutage mehr Kosten verursacht als die Entwicklung selbst. Positiv für die Wartbarkeit ist natürlich, wenn sich ein Entwurf leicht verstehen lässt, so dass die zu ändernden Stellen schnell auffindbar sind. Hierzu trägt auch eine ausführliche Dokumentation bei.

Der Grad der *Wiederverwendung* hält fest, wie stark Redundanzen durch Nutzung bereits erstellter Komponenten vermieden werden oder inwiefern der Entwicklungsaufwand durch Verwendung von Komponenten aus alten Projekten verringert wird. Gerade letzteres kann zu massiven Einsparungen bei den Entwicklungskosten führen. Besonderes Augenmerk liegt hier auf der Wiederverwendung durch Vererbung, die in der Objektorientierung häufig eingesetzt wird.

Wiederverwendbarkeit hingegen bezieht sich auf die Möglichkeit, das entwickelte System oder einige seiner Teile in anderen Systeme zu verwenden, ohne große Änderungen vornehmen zu müssen. Eine zentrale Rolle spielt hier natürlich die Flexibilität der einzelnen Teile, mit der sie in anderen Kontexten eingesetzt werden können. Dazu sollten sie in möglichst geringem Maße von anderen Komponenten abhängen und leicht zu verstehen sein.

Die *Brauchbarkeit* eines Entwurfs lässt erkennen, inwiefern er dazu geeignet ist, die gestellten Anforderungen zu erfüllen oder ein gegebenes Problem zu lösen. Desweiteren spiegelt sie wieder, inwieweit sich der Entwurf überhaupt umsetzen lässt.

Wenn der Integrationstest für das System und seine Komponenten einfach durchzuführen

ist, so spricht man von einer hohen *Testbarkeit*. Ausschlaggebend ist hierfür der Aufwand für das Erstellen und die Durchführung der Testfälle, die benötigt werden, um alle geforderten Funktionen des Systems zu überprüfen. Dabei ist es hilfreich, wenn der zugrundeliegende Entwurf möglichst wenige Komponenten umfasst und diese nur minimal von einander abhängen.

Die *Prüfbarkeit* bezieht sich, anders als die Testbarkeit, nicht auf das System, das aus dem Entwurf entstehen soll, sondern auf den Entwurf selber. Sie gibt an, wie einfach der Entwurf auf Erfüllung der Anforderungen überprüft werden kann. Es muss also nachvollziehbar sein, wie und wo eine Funktionalität im Entwurf geboten wird. Auch hier ist eine möglichst geringe Zahl von Komponenten und ihrer Abhängigkeiten von Vorteil. In diesem Zusammenhang ist auch von Interesse, inwiefern die Prüfung auf mehrere Gruppen aufgeteilt werden kann.

Die Aufnahme weiterer Faktoren in das Modell scheitert an den bereits bekannten Einschränkungen von ODEM: Qualitätsmerkmale wie die Portabilität, die Robustheit oder die Benutzerfreundlichkeit hängen zu stark von Details der Implementierung des Entwurfs ab, die in den von ODEM, und somit auch von QOOD anvisierten Entwürfen nicht verfügbar sind. Damit lassen sie sich in diesem Rahmen nicht zuverlässig bestimmen und werden aus diesem Grund hier nicht weiter beachtet.

4.1.2 Kriterien

Als Eigenschaften des Entwurfs stellen die Kriterien sehr viel speziellere Qualitätsmerkmale dar, als die Faktoren. Da sie diese mit beeinflussen, lassen sie sich als deren Teilaspekte auffassen. Zunächst muss also geklärt werden, welche Kriterien einen Faktor mitbestimmen. Dabei werden folgende Anforderungen zur Auswahl herangezogen: Die *Relevanz* eines Kriteriums für einen Faktor muss möglichst hoch sein, nur die nötigsten Kriterien sollten verwendet werden und ihre Abhängigkeiten untereinander sollten möglichst gering sein, um eine gute *Verständlichkeit* beizubehalten. Ausserdem sollten die Kriterien den Faktor möglichst vollständig beschreiben (*Überdeckung*), dabei geringe Redundanz aufweisen (*Unabhängigkeit*) und sich hauptsächlich mit objektiven Metriken bewerten lassen (*Bewertbarkeit*). Mit den Definitionen der Faktoren wurden bereits Aspekte genannt, die Einfluss auf die Qualitätsmerkmale haben, so dass sich die entsprechenden Kriterien recht schnell identifizieren lassen. Tabelle 1 zeigt das Ergebnis dieser Auswahl.

Die dabei hinaus gewählten Kriterien sollen nun kurz vorgestellt werden. Dabei werden auch bestehende Beziehungen zwischen den einzelnen Kriterien angesprochen.

Allgemeinheit: Besitzt eine Komponente eine hohe Flexibilität bzgl. ihres Einsatzgebietes, so steht dies für eine hohe Allgemeinheit. Wichtig ist dieses Kriterium besonders hinsichtlich der Tatsache, dass die Wiederverwendung einer Komponente nicht immer von Beginn an festgelegt ist.

Dokumentierung: Die Qualität der Dokumentation eines Entwurfes oder einer Kom-

Kriterien	**Faktoren**
Dokumentierung, Einheitlichkeit, Entkopplung, Knappheit, Strukturiertheit, Verfolgbarkeit, Zusammenhalt	Wartbarkeit
(lässt sich nicht in Kriterien zerlegen)	Wiederverwendung
Allgemeinheit, Dokumentiertung, Einheitlichkeit, Entkopplung, Knappheit, Strukturiertheit, Zusammenhalt	Wiederverwendbarkeit
Korrektheit, Realisierbarkeit	Brauchbarkeit
Entkopplung, Knappheit	Testbarkeit
Entkopplung, Knappheit, Verfolgbarkeit	Prüfbarkeit

Tabelle 1: Faktoren und ihre zugehörigen Kriterien

ponente wird in diesem Kriterium festgehalten. Sie wird durch die Vollständigkeit, die eingesetzte Notation und den Aufbau der Dokumente beeinflusst.

Einheitlichkeit: Ein Entwurf sollte einheitlich verfasst sein, also in allen Komponten dem gleichen Stil folgen. Vorgaben für die Namensgebung und die Strukturierung sind hier besonders hilfreich und erleichtern das Verständnis.

Entkopplung: In der Praxis ist der Begriff der *Kopplung* geläufiger, er wird hier aber ersetzt, da alle Kriterien mit hohem Wert positiv bewertet werden sollen und generell eine geringe Kopplung als erstrebenswert gilt. Unter der *Kopplung* versteht man in diesem Zusammenhang die Verbindung von Komponenten in einem Entwurf durch gegenseitiges Aufrufen der Operationen, Vererbung oder Datenabstraktion (Einsatz einer Komponente als Datentyp in einer anderen).

Knappheit: Wenn ein Entwurf mit möglichst wenigen Elementen (z.B. Klassen, Attributen, Beziehungen) die gewünschte Funktionalität bieten kann, so deutet dies auf eine hohe Knappheit hin. Sie erleichtert das Verständnis und die Dokumentation und steht somit in Beziehung mit der *Dokumentierung*. Allerdings steht sie zum Beispiel mit der *Entkopplung* im Gegensatz, da eine geringere Anzahl an Klassen bei gleichbleibender Funktionalität zu einer höheren Kopplung führt.

Korrektheit: Wenn ein Entwurf die geforderten Funktionalitäten bietet, gilt er als korrekt. Dabei sollten diese wie in den Anforderungen beschrieben umgesetzt werden.

Realisierbarkeit: Ist ein Entwurf auf der vorgesehenen Plattform mit den einzusetzenden Mitteln (Programmiersprachen, Werkzeuge, etc.) implementierbar, ohne den geplanten Aufwand zu überziehen, ist er realisierbar.

Strukturiertheit: Lässt sich die Struktur eines Systems schnell überblicken und verstehen, so spricht man von einer hohen *Strukturiertheit*. Die Bildung von Hierarchien im Entwurf tragen zu *Strukturiertheit* bei.

Verfolgbarkeit: Teile des Entwurfs sollten einfach durch die Anforderungen zu begründen sein. Dadurch veringert die *Wiederverwendung* von externen Komponenten die Verfolgbarkeit. Hier besteht also ein widersprüchliche Beziehung zwischen einem

Kriterium und einem Faktor.

Zusammenhalt: Der Grad der Zusammengehörigkeit von Komponenten, die durch ihre Semantik verbunden sind (z.B. Operationen in einer Klasse, Klassen in einem Paket), kann die Verständlichkeit eines Entwurfs stark beeinflussen.

Der Faktor Wiederverwendung nimmt hier eine Sonderrolle ein, denn er lässt sich nicht in Kriterien zerlegen. So zeugt zwar eine hohe Wiederverwendung interner Komponenten von hoher Knappheit, denn mehrere Funktionalitäten greifen in diesem Fall auf die gleichen Komponenten zurück. Verwendet man allerdins externe Komponenten älterer Projekte so veringert dies die Knappheit, da Teile anderer Entwürfe hinzugezogen werden.

4.1.3 Metriken

Die jetzt bekannten Faktoren und zugehörige Kriterien sollen nun über die Metriken quantifiziert werden. Hierzu wird für für jeden Aspekt eine subjektive Metrik auf der Grundlage von Fragebögen, die den Sachverständigen zur Bewertung von Eindrücken dienen, sowie relevanter objektiver Metriken (die in ODEM formalisiert werden) bestimmt.

Für die Bestimmung der objektiven Metriken gelten 3 wichtige Grundsätze: Es sollen möglichst *wenige* Metriken zum Einsatz kommen, die möglichst *einfach* (also nicht aus anderen Metriken aggregiert) sein sollten und den ensprechenden Aspekt möglichst *vollständig* abdecken. So lässt sich zum Beispiel die Metrik *NCP* (die Anzahl aller Klassen in einem Paket) für das Kriterium *Knappheit* bestimmen, welches darüber auch den Faktor *Wartbarkeit* beeinflusst. Für die einzelnen objektiven Metriken wird zunächst ein Schwellenwert, der als Grenze zu dem schlechteren Wertebereich gedacht ist, und eine Toleranz festgelegt, die um den Schwellenwert herum den Übergang von positiv zu negativ bewertetem Wertebereich fließender gestaltet. Daraufhin werden die erhaltenen Werte auf den Wertebereich zwischen 0 und 1 abgebildet (normiert), sie so dem Erfüllungsgrad der Metrik entsprechen. Den Gesamtwert erhält man, indem die Metriken eine Gewichtung erhalten und ein Durchschnittswert gebildet wird. Dieser wird schliesslich mit 9 multipliziert, um ihn in den Wertebereich der subjektiven Metriken (von 1 bis 9) abzubilden.

Fragebögen unterstützen ebenfalls die Bewertung und sorgen darüber hinaus dafür, dass relevante Aspekte nicht vergessen werden. Jede Frage auf einem Fragebogen wird mit 4 weiteren Attributen versehen: Die *Bedingung* gibt an , unter welchen Umständen die Frage sinnvoll anzuwenden ist, die *Antwortskala* gibt den Wertebereich an, aus dem die Antwort gewählt werden soll. Das *Gewicht* gibt die Relevanz der Frage für die Bewertung an und wird in der Regel mit einem Wert von 1 bis 3 belegt. Ob sich die Antwort über ODEM automatisch herleiten lässt, sagt die *Automatisierbarkeit* aus. Weiterhin sollte ein Fragebogen folgenden Anforderungen genügen: Die Fragen sollten für den vorliegenden Aspekt *relevant* und eindeutig zu *entscheiden* sein. Bezogen auf den Aspekt sollten die Fragen ihn *bewertbar* machen und *hinreichend* beschrei-

ben, sich dabei aber möglichst *redundanzfrei* zu einander verhalten. Relevante Fragen ermöglichen also über ihren Wert auf der Antwortskala mit den angegebenen Gewichten die Bildung eines Durchschnittswertes, ähnlich wie die objektiven Metriken.

Von subjektiven Metriken wird gefordert, dass sie sich auf eine eingeschränkte Bewertungsskala beziehen, die aber eine differenzierte Bewertung zulässt. Zusätzlich sollte kein Mittelwert vorhanden sein. Aufgrund des subjektiven Charakters der Metriken sind die Werte eher als Bewertungsvorschlag des jeweiligen Sachverständigen zu verstehen. Dieser kann sein Urteil zum Beispiel aus dem Mittelwert der untergeordneten objektiven und subjektiven Metriken ziehen. Dazu werden zunächst die objektiven Metriken bestimmt und aus ihnen, den subjektiven Metriken und den Ergebnissen untergeordneter Ebenen und Schichten (wie bereits beschrieben) ein Bewertungsvorschlag abgeleitet.

Diese Bewertung lässt sich zum Großteil aber auch automatisieren: Die objektiven Metriken, die ja auf ODEM basieren, lassen sich aufgrund von Klassendiagrammen durch Computern berechnen. Auf Fragebögen werden dann nur die Fragen genutzt, die als automatisierbar gekennzeichnet sind. Diese lassen sich dann ebenfalls über ODEM auswerten. Subjektive Metriken können dann aus den entsprechenden objektiven Metriken, den Werten der Fragebögen und den untergeordneten subjektiven Metriken als Durchschnittswert über die festgelegten Gewichtungen errechnet werden. Dies erleichtert den gesamten Vorgang, lässt allerdings gewisse Attribute des Entwurfs (die zum Beispiel in nicht automatisierbaren Fragen erfragt werden) ausser acht und vernachlässigt die persönlichen (u.U. domänenspezifischen) Erfahrungen möglicher Sachverständiger.

4.2 Ableiten eines spezifischen Modells

Bis jetzt wurde QOOD als allgemeines Modell beschrieben mit allen sinnvoll einsetzbaren Faktoren und Kriterien. Wie aber schon eingangs erwähnt soll QOOD auch ermöglichen, projektabhängige Prioritäten und Preferenzen hinsichtlich der Qualitätsmerkmale zu Berücksichtigen. Dazu wird das Modell spezifiziert, indem, basierend auf den Anforderungen an die Qualität, die relevanten Aspekte ausgewählt und die Gewichtungen festlegt werden, die sich in vielen Ebenen des Modells finden lassen. Hier spielt auch häufig die Qualitätspolitik des ausführenden Unternehmens eine große Rolle. Die Spezifizierung des Modells läuft dann wie folgt ab:

1. Auswahl der zu quantifizierenden Faktoren
2. Gewichtung der Faktoren
3. Auswahl der Kriterien für jeden Faktor
4. Gewichtung der Kriterien
5. Auswahl der objektiven Metriken für jedes Kriterium

6. Bestimmen von Gewichtung, Schwellenwerten und Toleranzen der Metriken

7. Auswahl der Fragen aus den Fragebögen für jedes Kriterium

8. Gewichtung der Fragen

9. Gewichtung der einzelnen Bewertungsvorschläge

Generell gelten beim Entwurf Knappheit, Entkopplung und Zusammenhalt als die wichtigsten Kriterien, sollten also bei der Auswahl beachtet und mit einem entsprechenden Gewicht versehen werden. Dies sollte allerdings projektbezogene Qualitätsanforderungen nicht überschatten. Mehrfaches auftretten und einfließen von Metriken und Kriterien in die Berechnung ist zulässig und durch erhöhen der Gewichte leicht zu berücksichtigen, sollte aber mit Blick auf die Qualitätsanforderungen kritisch gesehen werden, um ihnen nicht ungewollt eine zu hohe Bedeutung zukommen zu lassen. Eine Automatisierung des Vorgangs ist denkbar, wobei hier die Möglichkeit besteht, schon zu Beginn nur automatisch bestimmbare Metriken und Fragen auszuwählen.

5 Zusammenfassung

Mit ODEM und QOOD stehen also 2 Qualitätsmodelle zur Verfügung, die die Bewertung der Qualität von objektorientierten Architekturentwürfen in UML ermöglichen. Zwar können sie nicht alle Qualitätsmerkmale abdecken, was mit dem Bezug auf die reinen Entwürfe zusammenhängt, allerdings sind Erweiterungen in den Qualitätsmodellen denkbar, um weitere Aspekte erfassen zu können. Dies würde jedoch voraussetzen, dass die zugrundeliegenden Entwürfe mit weiteren Informationen ausgestattet werden.

Ein wichtiger Faktor ist auch die Automatisierung: Wenn nur wenig Zeit für Maßnahmen bzgl. der Qualität zur Verfügung steht, kann eine automatische Bewertung trotzdem eingesetzt werden. Zwar schränkt die Automatisierung die Anzahl der überprüfbaren Aspekte wie beschrieben noch weiter ein, liefert darüber hinaus jedoch sinnvolle und brauchbare Ergebnisse. Viele der Maßnahmen gegen die so aufgedeckten Schwachstellen im Entwurf lassen sich ebenfalls durch QOOD bewerten. Damit ermöglicht das Modell zusätzlich, dass Empfehlungen für Entwurfsprobleme festgelegt werden können.

ODEM und QOOD stellen also keine allumfassende Lösung für die Bewertung von objektorientierten Entwürfen dar, bieten aber bereits in der vorgestellten Form umfangreiche Möglichkeiten. So bleibt abzuwarten, wie Anwendungsfälle in der Praxis verlaufen und welche Weiterentwicklungen und Erweiterungen der Qualitätsmodelle sich als sinnvoll erweisen.

Literatur

[Bal98] Helmut Balzert. *Lehrbuch der Software-Technik Band 2: Software-Management, Software-Qualitätssicherung, Unternehmensmodellierung*. Spektrum Verlag, 1998.

[OMG99] OMG. UML 1.3 Specifications, 1999.

[Rei01] Ralf Reissing. Towards a Model for Object-Oriented Design Measurement. Technical report, Institute of Computer Science, University of Stuttgart, 2001.

[Rei02] Ralf Reissing. Bewertung der Qualität objektorientierter Entwürfe. Dissertation, Institute of Computer Science, University of Stuttgart, 2002.

Testen von Web-Anwendungen

Maren Wiese

Inhaltsverzeichnis

1 Einführung

Thema dieser Seminararbeit ist das Testen von Web-Anwendungen. Eine Webanwendung kann allgemein definiert werden, als ein Softwaresystem, das auf Technologien und Standards des Webs beruht und web-spezifische Ressourcen bereitstellt (vgl. S.2 in [Kap03]).

Das Testen webbasierter Anwendungen stellt besondere Anforderungen an den Testprozess und die Tester. Selbst erfahrene Tester von traditionellen Anwendungen müssen zunächst die Unterschiede zwischen Web-Anwendungen und traditionellen Anwendungen, sowie zwischen deren unterschiedlichen Architekturen und Technologien verstehen, um ihre Erfahrungen auf das Testen von Web-Anwendungen anwenden zu können. Obwohl eine Vielzahl von traditionellen Testverfahren auch auf webbasierte Anwendungen angewendet werden können, existieren viele technische webspezifische Gegebenheiten, die beim Testen von Web-Anwendungen berücksichtigt werden müssen.

In Abschnitt 2 werden zunächst beispielhaft einige Unterschiede zwischen traditionellen und webbasierten Anwendungen aufgelistet. Daraufhin werden in Abschnitt 3 allgemeine Testmethoden und Techniken aufgeführt, um dann in Kapitel 4 webspezifische Testkategorien zu behandeln, die beim Testen von Web-Anwendungen beachtet werden sollten. Kapitel 5befasst sich mit Testautomatisierung und Testwerkzeugen.

2 Unterschiede zwischen traditionellen und webbasierten Anwendungen

Web-Anwendungen basieren auf einer Client/Server-Architektur. Diese benötigt ein Netzwerk von mindestens zwei PC's. Einem Client, der Dienste abfragt und einem Server, der die erwünschten Dienste zur Verfügung stellt. Im Folgenden sollen einige Unterschiede zwischen webbasierten und traditionellen Client/Server-Systemen festgehalten und erläutert werden. Dabei unterscheiden wir zwischen Anwendungen, die auf dem Client ablaufen und solchen, die auf dem Server abgelegt sind.

2.1 Clientseitige Anwendungen

Clientseitige Anwendungen in einem traditionellen System sind plattformabhängig und müssen somit für jedes unterstützte Betriebssystem entwickelt und getestet werden. Im Gegensatz dazu operiert der Webclient innerhalb einer Browserumgebung. Web-Anwendungen werden als HTML Code (HyperText Markup Language) mit eingebetteten Elementen wie Skripten, Java Applets, XML Elementen (eXtensible Markup Language) oder Cascading Style Sheets (CSS) vom Browser übersetzt und dargestellt. Daher müssen Webanwendungsanbieter nicht das Zusammenspiel mit einem Betriebssystem berücksichtigen. Im Gegenzug dafür müssen sie aber sicherstellen, dass die Anwendung mit den verbreitetsten Browsern und in deren verschiedensten Versionen darstellbar ist.

Bei Web-Anwendungen handelt es sich außerdem um ereignisgesteuerte Anwendungen. Ereignisse sind dabei Aktionen des Nutzers, wie etwa der Mausklick auf einen Button oder das Eingeben von Daten in ein Formular. Ein Nutzer kann auf einen Link klicken und neue Dialogboxen oder eine Anwendung, die auf dem Client ausgeführt wird (z.B den Acrobat Reader), öffnen. Die Ereignisbehandlung kann auf dem Server oder direkt auf dem Client durch Skriptaufrufe, die in den HTML Code eingebettet sind, erfolgen. Um effektive Testfälle erstellen zu können, müssen die Tester verstehen, wo die Ereignisse (auf dem Server oder auf dem Client) behandelt werden.

Durch das Internet wird eine Web-Anwendung nicht nur ein paar Mal, sondern von vielen tausend Benutzern zur selben Zeit genutzt. Das System muss also gewährleisten, dass mehrere gleichzeitig geöffnete Anwendungsinstanzen nicht vertauscht werden, d.h. es muss den Überblick über die Daten und die User der zugehörigen geöffneten Instanzen behalten und darf diese nicht verwechseln, indem sie beispielsweise Daten an die falsche Instanz sendet.

2.2 Serverseitige Anwendungen

Serverseitige Anwendungen interagieren nicht direkt über User Interfaces mit dem Benutzer, sondern über Kommunikationsprotokolle. Sie werden nicht explizit wie eine

Client-Anwendung aufgerufen, sondern bleiben immer aktiv, um einen angeforderten Dienst anzubieten. Wenn eine solche serverseitige Anwendung versagt, weiss man selten, welche Ursachen den Fehler ausgelöst haben. Um nachvollziehen zu können, wodurch Fehler verursacht wurden, müssen serverseitige Ereignisse in Log-Dateien aufgezeichnet werden. Welche Probleme auf der Serverseite zu beachten sind und worauf dort geprüft werden sollte, wird in Abschnitt 4.2 näher betrachtet.

2.3 Webarchitekturen

2.3.1 Statische Web-Anwendungen

Statische Web-Anwendungen legen Webseiten in Form von statischen HTML-Dateien auf einem Webserver ab. Der Webserver macht seine Inhalte für Clients zugänglich. Im Falle einer Anfrage auf Basis des HTTP-Protokolls werden diese Webseiten als Antwort an den Web Client geschickt.

2.3.2 Datenbankbasierte Web-Anwendungen

Ist die Web-Anwendung datenbasiert, wird die Verwendung eines Datenbanksystems zur effizienten Verwaltung dieser Daten unumgänglich. Dieses liegt auf einem sogenannten Datenbankserver. Durch die Einrichtung eines Datenbankservers wird die Trennung von Inhalt und Präsentation erreicht. Die Anbindung an die Datenbank kann dabei entweder serverseitig oder clientseitig stattfinden (siehe Abbildung 1). Die meisten Websysteme verwenden relationale Datenbankserver. Der Einsatz eines Datenbankservers zieht ein Reihe von Datenbanktests mit sich, die in dieser Seminararbeit jedoch außer Acht gelassen werden und bei Bedarf in [Ngu03] nachgeschlagen werden können.

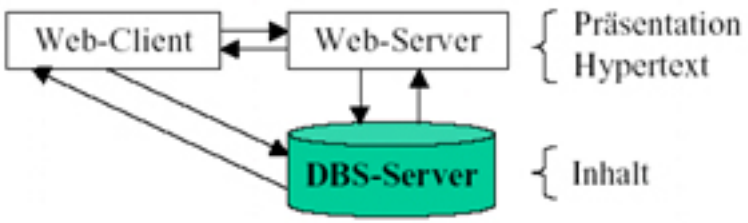

Abbildung 1: Architektur einer datenbankbasierten Web-Anwendung

2.3.3 Anwendungslogikorientierte Web-Anwendungen

Stabilitätsprobleme, sowie mangelnde Performanz von Web-Anwendungen haben zum Einsatz von Anwendungsservern geführt. Dabei wird Anwendungsslogik vom Webserver auf den Anwendungsserver ausgelagert. Der Webserver reicht dabei nur noch die Anfragen der Clients an den Anwendungsserver weiter und leitet dessen zurückgegebene Ergebnisse an den Client weiter (siehe Abbildung 2).

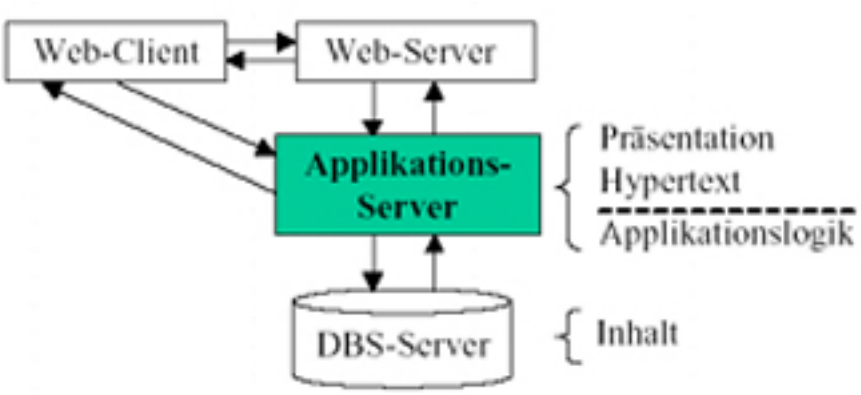

Abbildung 2: Architektur einer anwendungslogikorientierten Web-Anwendung

3 Testgrundlagen

Da davon ausgegangen werden kann, dass die Leser dieser Arbeit bereits über Kenntnisse im Testen von Software verfügen und Ihnen daher die allgemeinen Testgrundlagen und Begriffe bekannt sein sollten, werden im Folgenden nur einige Techniken für die Testfallauswahl kurz eingeführt. Es wird auf weiterführende Literatur verwiesen (siehe [Wal90]).

3.1 Testfallerstellung

Zur zentralen Aufgabe des Testens von Anwendungen gehört die Testfallbestimmung. Dies gilt für traditionelle Softwaretests, genauso wie für webspezifische Tests. Bei der Testfallerstellung handelt es sich um einen Prozess, bei dem Sollwerte für eine Testmenge bestimmt werden, die dann mit den tatsächlichen Ergebnissen (den Ist-Werten) nach Programmdurchlauf verglichen werden. Stimmen Soll und Ist-Werte nach dem Test nicht überein, war der Test erfolgreich, d.h er konnte einen Fehler aufdecken. Dabei müssen Testfälle möglichst repräsentativ, fehlersensitiv, redundanzarm und ökonomisch sein. Ziel des Testens ist es also, mit einer möglichst kleinen Auswahl von

Testfällen möglichst viele Fehler aufzudecken. Darin liegt die Schwierigkeit der Testfallerstellung.

Gegenwärtig kennen wir zwei Gruppen von Methoden zur Testfallermittlung, nämlich Black-Box-Methoden und White-Box-Methoden. Bei Black-Box-Tests werden die Testfälle aus der Programm- oder Komponentenspezifikation abgleitet. Black-Box-Tests werden auch als funktionale Tests bezeichnet, da der Tester sich nur mit der Funktionalität und nicht mit der Implementierung einer Anwendung beschäftigt. Das heisst Black-Box-Tests kommen ohne Zugang zum Quelltext aus und leiten ihre Testfälle allein aus der Programmspezifikation ab (vgl. S. 450 [Som01]). Zu den wichtigsten Black-Box-Methoden gehören die Äquivalenzklassenbildung, die Grenzwertanalyse und die Ursache-/Wirkungsgraphmethode (vgl. [Wal90]).

Bei White-Box-Tests muss vorausgesetzt werden, dass die Struktur des Testobjekts bekannt ist. Daher wird der White-Box-Test auch als struktureller Test bezeichnet. Testfälle werden aus der Softwarestruktur und der Implementierung abgleitet.

Um effektive Testfälle für webbasierte Anwendungen zu erstellen, greifen auch Nguyen, Johnson und Hackett auf die Methoden der Äquivalenzklassenbildung und Grenzwertanalyse zurück. Zusätzlich benennen sie Methoden wie die Zustandsübergangsanalyse (vgl. [Ngu03] S. 63-65) und die Use Case Analyse (vgl. [Ngu03] S. 66-74).

Bei der Äquivalenzklassenbildung werden Eingabe- und Ausgabedaten in gültige und ungültige Äquivalenzklassen eingeteilt. Das Programm wird dann mit Repräsentanten dieser Klassen ausgeführt. Diese sollten so ausgewählt werden, dass sie möglichst viele noch nicht abgedeckte gültige Äquivalenzklassen oder genau eine noch nicht abgedeckte ungültige Äquivalenzklasse abdecken. Die Grenzwertanalyse ergänzt die Äquivalenklassenbildung und basiert auf dem Wissen, dass an den Grenzen von Wertebereichen gehäuft Fehler auftreten. Hier werden Testfälle ermittelt, mit denen alle Grenzwerte einer Äquivalenzklasse getestet werden.

Bei der Zustandsübergangsanalyse werden alle möglichen Zustände einer Anwendung identifiziert, d.h für jeden Testfall werden Anfangszustand, Zustandsübergang auslösende Ereignisse, die resultierenden Ergebniszustände und der Endzustand definiert. Danach wird ein Zustandsdiagramm generiert, dass alle Beziehungen zwischen den möglichen Zuständen, Ereignissen und Aktionen der Anwendung abbildet. Mit Hilfe von Wertetabellen werden anhand des Zustanddiagramms Testfälle abgeleitet.

4 Webspezifische Testkategorien

Fehler lassen sich in verschiedene Klassen einteilen, anhand derer sich Testkategorien ableiten lassen, die eine bestimmte Klasse von Fehlern aufdecken sollen. Nguyen, Johnson und Hackett listen eine Reihe von Testkategorien auf [Ngu03], die beim Testen von Web-Anwendungen besonders beachtet werden sollten und beschreiben, welche Testdesign-Kriterien sich aus den Bedingungen für Web-Anwendungen ergeben. Diese werden in den folgenden Abschnitten beschrieben.

4.1 Funktionale Tests

Durch funktionales Testen wird überprüft, ob die Anwendung, die in der Spezifikation an sie gestellten Anforderungen zuverlässig und zufriedenstellend erfüllt. Funktionale Tests finden auf verschiedenen Komplexitätsebenen statt.

4.1.1 Functional Acceptance Simple Tests

Functional acceptance simple Tests oder kurz FASTs laufen auf der niedrigsten Ebene einer Anwendung ab und prüfen, ob allgemeine Basisfunktionalitäten korrekt implementiert wurden. In Webumgebungen wird mit FAST-Tests zum Beispiel geprüft, ob Inhalte, Thumbnails, Bilder und Image Maps korrekt verlinkt sind oder ob grundlegende Kontrollfunktionen (wie Vorwärts- und Rückwärts navigieren, Ein- und Auszoomen oder das Neuladen von Inhalten) funktionieren und ob sich Daten mit Hilfe von Formularen eingeben, ändern oder löschen lassen. Zudem wird überprüft, ob Benutzerprofile und Zugangsdaten eingerichtet werden können und ob andere Schlüsselfunktionen wie der login/logout-Prozess, Emailbenachrichtigungen und Suchfunktionen funktionieren.

Mit Hilfe von FAST-Tests werden grundlegende Fehler wie fehlende Bilder, fehlende oder falsche Verlinkungen, überholte oder fehlende Inhalte aufgedeckt, die leicht zu beseitigen sind. Solche Tests werden in der Praxis selten manuell durchgeführt, da sie leicht automatisierbar sind. Die bekanntesten Tools sind HTML-Validatoren und Link-Checker. Sie suchen automatisch nach schlechter Verlinkung, toten Links und Rechtschreibfehlern. Diese regelbasierten Analysewerkzeuge lesen den implementierten Code ein und vergleichen diesen mit der sprachenspezifischen Syntax und deren Regeln. Dabei versuchen sie, Abweichungen von Normen oder Vorschriften aufzudecken (vgl. Kapitel 5).

4.1.2 Task-Oriented Functional Testing

Task-oriented functional tests (TOFTs) überprüfen, ob die in der Anforderungsspezifikation geforderten aufgabenorientierten Funktionalitäten von der Anwendung erfüllt und realisiert werden. Hierzu wird ein Liste mit allen zu prüfenden Anforderungen aufgestellt, indem die Spezifikation bis aufs kleinste zerpflückt wird und alle geforderten Anforderungen herausgeschrieben werden. Mit Hilfe dieser Liste werden dann Testfälle ermittelt, die kontrollieren, ob die Anforderungen eingehalten wurden.

4.1.3 Forced Error Tests

Forced error tests (FETs) produzieren absichtlich Fehlersituationen und prüfen so die Fehlerbehandlung des Systems. Ziel ist es, alle Fehlerfälle zu finden, die noch nicht berücksichtigt wurden.

Angenommen ein Eingabeformular für Text soll getestet werden und in der Spezifikation wird gefordert, dass nur Buchstaben akzeptiert werden dürfen. Indem ein Folge von Zahlen eingegeben wird, wird absichtlich ein Fehlerfall erzeugt und dessen Behandlung getestet. Es sollten so viel fehlerauslösende Ereignisse wie möglich ermittelt werden. Hierzu sollten sich Tester mit den Systementwicklern beraten, Fehlernachrichten sammeln und die Spezifikationsdokumente durchsuchen. Ist eine vollständige Liste mit Fehlerauslösern zusammengestellt worden, wird mit diesen Testfällen überprüft, ob das System den Fehler überhaupt erkennt und falls ja, wie das System reagiert.

Um die Fehlerbehandlung beurteilen zu können, sollte man feststellen, ob das System eine angemessene Fehlerbehebung durch den Nutzer erlaubt (wie z.B. das erneute Ausfüllen eines einzelnen Feldes, ohne das gesamte Formular neu ausfüllen zu müssen) oder ob das System nach einem Absturz uneingeschränkt weiter arbeitet. Außerdem sollte man prüfen, ob eine Fehlermeldung auf dem Bildschirm ausgegeben wird, ob aus dieser zu erkennen ist, wodurch der Fehler verursacht wurde, ob darin Schritte zur Behebung des Fehlers beschrieben werden und ob gegebenfalls ein akustischer oder visueller Trigger ausgelöst werden sollte.

4.2 Serverseitige Tests

Wie bereits erwähnt bestehen Websysteme genauso wie Client/Server-Systeme aus einem Netzwerk, das Clients und Server miteinander verbindet. Als Dienstanbieter, der Dienste zur Verfügung stellt, muss der Server bzw. dessen Software und Hardware unabhängig vom Client getestet werden.

Webtransaktionen können aus zwei Hauptgründen fehlschlagen: (1) der gewünschte Request erreicht nicht sein Ziel oder (2) der Request erreicht zwar das Ziel, erhält aber nicht das Geforderte. Erreicht der Request nicht sein Ziel, kann das zwei Gründe haben: Es liegt entweder ein Verbindungsproblem oder ein Serverproblem vor. In den folgenden zwei Unterkapiteln werden verbindungsrelevante Aspekte behandelt.

4.2.1 Time-Outs

Time-Outs entstehen aufgrund nicht erfolgter Antworten des Servers und verursachen den Abbruch laufender Transaktionen. Dies führt nicht nur zu einer lästigen Unterbrechung, die ein neues Einloggen des Nutzers verlangt, sondern eventuell auch zu Datenintegritätsproblemen. Tritt zum Beisppiel ein Time-Out auf, während ein Client Daten an eine Anwendung sendet, kann es vorkommen, dass die Daten auf dem Weg zum Datenbankserver verloren gehen und dort nicht gespeichert werden. Bei Fortsetzen der Session geht der Nutzer davon aus, dass seine Daten in der Datenbank aktualisiert wurden, was jedoch nicht zutrifft.

Beim serverseitigen Testen muss daher die maximal erlaubte Verzögerung für Antwortzeiten bestimmt werden und das System daraufhin getestet werden, ob es jederzeit

die geforderten Antwortzeiten erreicht bzw. mit der entsprechenden Fehlermeldung angemessen reagiert (siehe auch Stresstesten in Kapitel 4.5). Erhält die Datenbank eine sehr hohe Anzahl an Requests, kann es zu verzögerten Antwortzeiten kommen und einige Requests gehen dabei eventuell ganz verloren. Zudem kann es passieren, dass der Datenbankserver komplett ausfällt und kein Feedback in Form von Meldungen an Web- bzw. Applikationsserver geben kann. Um zu Testen, wie jede einzelne Komponente (d.h. Datenbank, Webserver, Apllikationsserver, Browser und Clients) reagiert, wenn ein Time-Out auftritt, sollte man prüfen, ob ein nicht beantworteter Request dazu führt, dass der Applikationsserver eine bedeutungslose oder irreführende Fehlermeldung zurückgibt oder ob der Applikationsserver die Verbindung zum Client trennt. Eventuell trennt auch die Applikation alle Verbindungen zur Datenbank. In einem anderen Fall kann die Anwendung vielleicht die Verbindung nicht mehr trennen und belegt deshalb Ressourcen oder es kommt dazu, dass die Applikation abstürzt und deshalb nicht mehr auf weitere Requests antworten kann.

Ob diese Reaktionen im konkreten Fall akzeptierbar sind oder nicht, hängt davon ab, wie kritisch die Anwendung ist, wie sicher die Daten sein müssen und wie fehlertolerant der Client ist.

4.2.2 Speicherung von Zuständen

Websysteme veranlassen den Besucher, HTML-Dokumente zu laden, von einer Seite zu einer anderen zu navigieren oder Daten einzugeben. Häufig ist es dabei nötig, dass das System einen bestimmten Zustand dokumentiert, um sich an bestimmte Aktionen des Anwenders zu erinnern. Beispielsweise muss die Warenkorbfunktion eines Online Shops bereits ausgewählte Produkte speichern. Die meisten Applikationen merken sich solche Zustandsinformationen mit Hilfe von Cookies, Session ID’s oder IP-Adressen. Cookies sind kleine Textdateien, die zur Speicherung von Serverdaten (wie z.B. der Session ID) auf dem Client genutzt werden können. Cookies werden von Webservern erzeugt und im Header von HTTP-Responses zum Client übertragen. Der Web-Browser speichert den Cookie beim Client und überträgt den Cookie fortan mit jedem Request zum Server, von dem der Cookie erzeugt wurde.

Um zustandsbezogene Probleme zu testen, müssen Informationen, die verschiedene Zustände beschreiben, geändert oder entfernt werden. Dabei sollte geprüft werden, wie das System auf folgende Szenarien reagiert:

- Es sind Zustandsinformationen verloren gegangen, die für die nächste Session benötigt werden.
- Informationen sind veraltet, z.B. Kreditkarteninformationen wie das Gültigkeitsdatum
- Es existieren gleiche Daten unterschiedlicher Benutzer wie Namen oder Geburtsdaten.
- Fehlende Rückmeldung des Clients

- Informationen wurden nicht an den richtigen Client gesendet

4.2.3 Ressourcen

Unter Ressourcen versteht man sämtliche der Anwendung zur Verfügung stehende Mittel, um Instruktionen korrekt ausführen zu können. Dazu gehören RAM, Speicherplatz, CPUs, Bandbreite, offene Verbindungen, usw. Ressourcenbedingte Probleme können aufgedeckt werden, indem folgende Punkte untersucht werden: Wie interagiert die Anwendung mit dem Betriebssystem, bezüglich der Vergabe von Ressourcen? Wie reagiert die Anwendung bei unzureichenden Ressourcen? Kann die Anwendung mit verteilten Ressourcen (z.B mit mehreren CPUs) arbeiten?

Um die Ressourcenverwaltung verfolgen zu können, sollten Monitoring-Werkzeuge verwendet werden.

4.2.4 Failovers

Bei einem Failover handelt es sich um eine Ausfallsicherung, die bei Ausfall oder Wartung des primären Systems den Betrieb auf ein redundantes Stand-By-System umleitet. Eine solche Sicherung ist vor allem bei 'mission-critical' Anwendungen, d.h bei Anwendungen die 24 Stunden und 7 Tage die Woche verfügbar sein müssen, Pflicht. Failover-Tests versetzen das System in einen Zustand, der den Ausfall auslöst. Dabei müssen Software-Failovers (Ausfall der Anwendung), genauso wie Hardware-Failovers (Ausfall der Hardware) simuliert werden. Mit diesen Tests soll geprüft werden, ob die Failover-Konfiguration wie erwartet arbeitet, d.h. ob Requests bei einem Failover weitergeleitet werden (und zwar ohne Verlust der Session), ob der ausgefallene Server automatisch wieder hochfährt und ohne Leistungsverlust weiter arbeitet und wie lange es dauert bis er mit dem Betrieb fortfahren kann.

4.2.5 Multithreads

Threads sind kleinste Kontrollflüsse, die von einem Rechner ausgeführt werden können. Eine Applikation führt oft mehrere Threads parallel aus, um ihre Effizienz zu steigern. Multithreading bedeutet demnach, dass mehrere Aufgaben gleichzeitig von der Applikation ausgeführt werden. Ein Webserver kann beispielsweise so entworfen werden, dass er mehrere Requests in einzelne Threads packt und somit schnellere Antwortzeiten erreicht. In Abbildung 4 generiert der Webserver aus einer Preisanfrage für ein bestimmtes Produkt vier Threads. Jedes einzelne Thread führt eine unabhängige Dantenbanksuche durch. Da alle Threads parallel und nicht sequentiell ausgeführt werden, kann der Request schneller beantwortet werden.

Bei mehreren Threads kann es allerdings zu Ressourcenproblemen kommen, die in einem Deadlock enden. Ein Deadlock beschreibt hier einen Zustand von Prozessen,

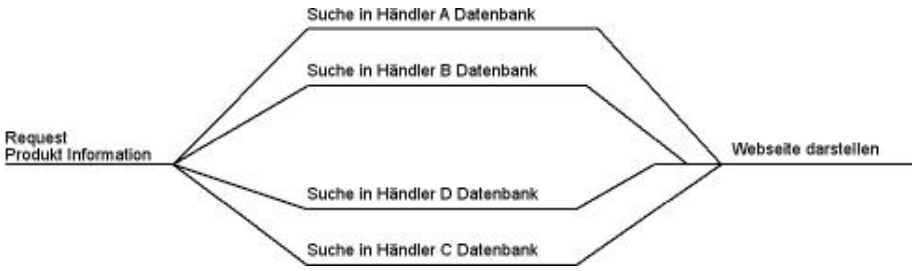

Abbildung 3: Beispiel: Multithreads

bei dem mindestens zwei Threads untereinander auf Betriebsmittel warten, die dem jeweils anderen Thread zugeteilt sind. Man stelle sich vor, dass ein Request drei Threads (T1, T2, T3) verlangt. Jeder einzelne Thread benötigt zwei Ressourcen. T1 benötigt (R1,R2), T2 benötigt (R2,R3) und T3 braucht (R1,R3). Dann besitzen T1,T2 und T3 jedoch nur eine Ressource, während die andere von einem anderen Thread blockiert wird, so dass nicht weiter gearbeitet werden kann. Wenn alle Prozesse warten bis die verlangte Ressource wieder frei ist, ensteht eine klassische Deadlock Situation.

Um Multithreading Probleme zu finden, muss ein eizelner Testlauf oft tausend mal wiederholt werden, bevor das Problem tatsächlich auftritt.

4.3 Testen mit Hilfe von Skripten

Das Verwenden von Skripten bietet eine Reihe von Vorteilen. Es ermöglicht zum Beispiel ein Testen ohne Nutzung einer Benutzeroberfläche, die eventuell noch nicht entwickelt wurde. Der Tester kann somit direkt mit dem Server kommunizieren. Das Ausführen von Skripten spart Zeit und gewährleistet, dass Instruktionen bei jeder Neuausführung des Skripts auf gleiche Art und Weise ausgeführt werden.

Zu den bekanntesten Skriptsprachen gehören Perl, VBScript, JavaScript, awk, sed und Python, wobei jede dieser Sprachen Stärken und Schwächen hat. Perl, zum Beispiel, ist für seine Text- und Datenverarbeitung und den TCP/IP Service Support bekannt, was gute Voraussetzungen für das Web-Testen sind. Wer mehr über Skriptsprachen erfahren möchte, findet dazu ausreichend Literatur und Informationen im Internet.

Im Folgenden sollen Aufgaben des Web-Testens, bei denen Skripte sinnvoll eingesetzt werden können, beschrieben werden. Dabei handelt es sich nicht um alle möglichen Einsatzgebiete im Testvorgang, sondern nur um einige Beispielszenarien (vgl. [Ngu03] Kapitel 13).

a) System-Administration: Einfache Skripte sparen Zeit und Aufwand bei der Systempflege der Testumgebung. Beispielsweise beim Zurücksetzten eines Servers oder um den Datenbankserver für den nächsten Testdurchlauf durch Löschen oder Modifizieren von Records aufzuräumen. Das Vorbereiten der Testumge-

bung ist eine notwendige Prozedur, die ständig wiederholt werden muss. Alte Log-Dateien müssen gelöscht und Konfigurationsdateien in das aktuelle Testverzeichnis kopiert werden. Durch ein paar Skriptzeilen werden solche administrativen Aufgaben, die sonst bei manueller Durchführung leicht vergessen werden könnten, automatisch und zuverlässig ausgeführt.

b) Direktes Testen des Servers: Wie oben bereits angedeutet, kann der Server unter Verwendung von Skripten direkt und ohne Umwege über ein Client Interface, dessen Möglichkeiten oft begrenzt sind, getestet werden. Zum Beispiel um eine Konfigurationsdatei, die auf einem Webserver liegt und Daten wie User-Namen und Emailadressen enthält, zu testen. Beim Start des Webservers wird diese Datei geöffnet. Um zu Testen wie der Server reagiert, wenn die Konfiguration fehlerhaft sind, muss dieser mit vielen verschiedenen Konfigurationen gestartet werden. Ein Skript lässt den Webserver dann mit diesen Konfigurationsdateien der Reihe nach laufen.

c) Anwenden der Applikation ohne Benutzeroberfläche: Es ist wichtig, so früh wie möglich mit dem Testen der Anwendung zu beginnen. Allerdings kann es vorkommen, dass das User Interface nicht früh genug vorliegt. Um trotzdem ohne User Interface testen zu können, schicken Skripte Requests auf direktem Wege an den Server. VBScript oder Perl ermöglichen Funktionsaufrufe.

d) Log-Dateien und Reports: Log-Dateien beinhalten Informationen,die beschreiben wie sich das System verhält, wer es nutzt und was er damit tut. Log-Dateien zeichnen Requests, Warnmeldungen, besuchte Webseiten, Anzahl und IP-Adressen der Nutzer und viele andere Daten auf, die in späteren Testphasen (z.B. beim Stress- oder Lasttesten) ausgewertet werden. Da diese Log-Dateien allerdings sehr lang und unübersichtlich sind, werden Skripte vefasst, um diese nach bestimmten Daten oder Meldungen, die potentielle Probleme kennzeichen, zu durchsuchen.

Auf der folgenden Webpage können nützliche Skripte für das Testen von Web-Anwendungen heruntergeladen werden: *http://www.hotscripts.com/Tools_and_Utilities/Debugging_and_Testing/*.

4.4 Testen der Sicherheit von Web-Anwendungen

Sicherheit spielt eine zentrale Rolle bei Web-Anwendungen, weil sie über deren erfolgreichen Einsatz entscheidet. Die Anzahl der Benutzer hat sich mit der Einführung des WWW exponentiell vergrößert. Damit haben Angriffe auf die Privatsphäre, der Missbrauch von persönlichen Informationen oder die missbräuchliche Nutzung von Computerressourcen viel weitreichendere Auswirkungen als noch vor zehn Jahren. Anleitungen zum Hacken geben sogar Laien die Möglichkeit, unzureichend geschützte Server zu knacken oder zum Absturz zu bringen. Die Sicherheitsproblematik bei Web-Anwendungen wird dadurch verschärft, dass der Benutzer oft nicht die Möglich-

keit hat, einen Angriff als solchen zu erkennen, oder angemessen darauf zu reagieren. Er akzeptiert alles was im Browser passiert als rechtmässiges Verhalten.

4.4.1 Was bedeutet Sicherheit?

Das Testen der Sicherheit ist ein umfangreiches Gebiet. Es geht dabei beispielsweise um Fragestellungen im Zusammenhang mit den Quälitätsmerkmalen von Sicherheit. Dazu gehören:

Vertraulichkeit: Wer darf auf welche Daten zugreifen? Wer darf welche Daten verändern oder löschen? Mit Vertraulichkeit wird der Schutz von Daten gegenüber Kenntnisnahme unbefugter Dritter bezeichnet.

Autorisierung: Wie und wo werden Rechte verwaltet? Werden Daten überhaupt verschlüsselt? Wie werden Daten verschlüsselt?

Authentifikation: Wie authentifiziert sich der Benutzer, wie der Server? Daten sind authentisch, wenn gewährleistet ist, dass sie von dem Absender stammen, der vorgegeben ist. Für einen externen Angreifer muss es praktisch unmöglich sein, Dokumente unter falschem Namen einzustellen.

Zurechenbarkeit: Wie werden Zugriffe protokolliert?

Integrität: Wie wird gewährleistet, dass die Daten bei der Übermittlung nicht verändert wurden? Sind Daten integer, so sind sie unversehrt, unverfälscht, korrekt und nicht durch einen Dritten erstellt oder manipuliert worden.

Für jede einzelne Funktion muss getestet werden, ob sie die einzelnen Merkmale (Vertraulichkeit, Integrität, usw.) entsprechend den an sie gestellten Anforderungen erfüllt. Wir sprechen von Attacken, wenn ein Außenstehender versucht, verheerenden Schaden anzurichten oder Informationen zu stehlen. Eine Attacke besteht aus drei Phasen: dem Erfassen von Informationen über das System, gefolgt vom Scannen des Systems, um eine Lücke aufzufinden, und dem Attackieren, d.h das aktive Einbrechen in das System. Nur wenn man den Ablauf von Angriffen versteht, kann man Strategien entwickeln, um sich vor solchen zu schützen. Eine Sicherheitslücke kann schwerwiegende Folgen mit sich bringen: finanziellen Verlust, Aufdeckung sensibler Daten oder sogar persönliche Schäden. Die Aufgabe des Testers besteht darin, sich in die Rolle eines Eindringlings zu versetzten und alle Systemlücken zu finden und zu eliminieren.

4.4.2 Schutzmaßnahmen

Im Folgenden werden einige klassische Methoden und Technologien aufgeführt, die eingesetzt werden, um Systeme zu schützen.

Authentifizierung durch Passwörter: Ein Passwort ist eine geheime Zeichenfolge, die der Identifikation eines Computer-Nutzers oder Anwendung dient. Nur demjenigen, der das Passwort kennt, wird Zugriff auf den betreffenden Rechner oder Dienst gewährt. Die Benutzerkennung soll den Zugang unberechtigter Personen verhindern.

Verschlüsselung: Die Verschlüsselung von Daten dient deren Schutz vor unbefugter Einsichtnahme oder Manipulation auf ihrem Weg durch das Internet. Das Ver- und Entschlüsseln von übermittelten Nachrichten und Daten geschieht mit Hilfe von Schlüsseln.

Zertifikate: Digitale Zertifikate sind das elektronische Gegenstück zu einem Ausweis. Sie ordnen ihrem Inhaber eindeutig einen öffentlichen Schlüssel (public key) und damit eine digitale Signatur zu. Diese elektronische Unterschrift sichert das damit unterzeichnete Dokument vor Verfälschung auf seinem Weg durch das Internet. Mit Hilfe des Zertifikats kann der Client überprüfen, ob die Antwort tatsächlich vom gewünschten Server stammt

SSL: Abkürzung für Security Sockets Layer. SSL ist ein allgemeines Sicherungsprotokoll insbesondere zur Kommunikation von Webbrowsern mit Webservern. Das Verfahren basiert auf öffentlichen Schlüsseln.

Firewalls: Firewalls schützen ein lokales Netz vor unbefugten Zugriffen aus dem Internet und erlauben angeschlossenen Rechnern den Zugang in das Internet.

4.4.3 Webspezifische Sicherheitsaspekte

Beim Testen der Sicherheit von Web-Anwendungen sollte davon ausgegangen werden, dass der Eindringling hohe Fähigkeiten, Glück und viel Zeit besitzt. Anliegen dieses Kapitels ist es, aufzuführen welche Fragestellungen berücksichtigt werden müssen, um Sicherheitslücken aufzudecken, Schwächen zu eliminieren und Attacken vorzubeugen oder zu erkennen.

Dazu gehört zum Einen die Frage nach der Zugangskontrolle. Wer hat berechtigten Zugriff auf die Anwendung? Gibt es verschiedene Klassen von Benutzern mit unterschiedlichen Zugriffsrechten? Hat die Zielanwendung ausreichend Zugriffsrechte für den Server? Solche Rechte sollten übrigens auf das minimal Nötigste beschränkt werden.

Zum Anderen muss die Frage, welche Ressourcen geschützt werden sollen, gestellt werden. Der als 'denial-of-service' bekannte Angriff hat nicht zum Ziel, Daten zu stehlen oder zu manipulieren, sondern das System, durch wiederholte Requests an den Server (d.h durch Überlasten des Systems), zum Absturz zu bringen. Andere Angriffe auf Ressourcen sind das Herunterfahren und Neustarten oder das Abspeichern sehr großer Dateien.

Zudem muss festgelegt werden, welche Daten privat behandelt werden müssen. Es muss geprüft werden, wie gut und sicher die Anwendung bzw. die Webseite Nutzerda-

ten verwaltet. Wird der Benutzer zum Beispiel darüber informiert, wie seine Angaben gesammelt und genutzt werden?

Eine andere Art von Angriff besteht darin, Bibliotheken oder andere notwendige Bestandteile der Applikation durch modifizierte Versionen zu ersetzten. Um dies zu verhindern sollten Hintertüren, die von Entwicklern zuvor eingebaut wurden, um schneller Fehler beseitigen zu können, in der Produktionsversion des Systems aufgespürt und geschlossen werden.

Gelegentlich treten unerwartete Ausnahmefälle (Exceptions) ein, die ein Programm berücksichtigen sollte. Der Tester muss sich die Frage stellen, wie Exceptions korrekt behandelt werden sollen. In manchen Fällen, soll das System leise versagen, den Ausnahmefall melden oder den Befund in einer Log-Datei speichern. Ein solcher Fehler kann zwar durch ein geringfügiges Problem ausgelöst werden, allerdings können auch Angriffe von außen Grund für eine Exception Situation sein.

Beim Testen von Id's und Passwörtern, muss der Mißbrauch von Superuser-Rechten geprüft werden. Der Benutzer darf immer nur über geringste Zugriffsrechte verfügen, so dass er mit dem System arbeiten, aber keinen Schaden anrichten kann. Zudem muss gewährleistet sein, dass Passwörter, die zum Verbindungsaufbau von Anwendungen zum Server gespeichert werden müssen, ausreichend versteckt und geschützt sind.

Das Ändern von Werten in Cookies kann Hackern den Zugriff auf Accounts ermöglichen, der ihnen nicht erlaubt ist. Das Stehlen des Cookies gewährleistet ihm den Zugang ohne ID und Passwort. Er kann sich somit alle Zugangsrechte des vermeintlichen Nutzers aneignen und mißbrauchen. Da Cookies einfach zu erzeugen sind und auf jedem Client gespeichert werden, ist es relativ einfach diese zu manipulieren.

Buffer-Overflows gehören zu den verbreitetsten Sicherheitslücken aktueller Software, welche sich über das Internet ausnutzen lassen. Im Wesentlichen können bei einem Buffer-Overflow durch Designfehler im Programm zu große Datenmengen in einen unterdimensionierten Speicherbereich geschrieben werden, wodurch ungewollt Informationen im Speicher überschrieben werden. Das Ziel bei einem Buffer-Overflow-Test ist es, aufzudecken, wann das Senden zu grosser Mengen von Daten ein unerwartetes Systemverhalten auslöst. Neben einer sehr grossen Datenmenge sollten auch Grenzwerte getestet werden, nämlich Daten derselben Grösse wie der Buffer oder Datenmengen, die ein Byte größer oder kleiner sind als der verfügbare Speicherplatz.

Dies waren nur einige Testaspekte und es gibt noch ein Vielzahl weiterer Teststrategien, um die Systemsicherheit von webbasierten Anwendungen zu prüfen. Da dieses Thema allerdings, wie oben bereits erwähnt, sehr umfangreich ist, verweise ich den interessierten Leser auf das Kapitel 18 in [Ngu03]. Dort werden viele Anregungen und Guidelines gegeben.

4.5 Leistungstests

Ein erfolgskritisches Qualitätsmerkmal von Web-Anwendungen ist die Performanz oder Leistung des Systems. Web-Anwendungen müssen auf ihr Leistungsvermögen beim Ausführen von kritischen Funktionen während normaler und sehr hoch frequentierten Nutzungsperioden getestet werden. Wegen der Homogenität und der Vielzahl potentieller Benutzer, muss beantwortet werden, ob das System die prognostizierte Arbeitslast aushält und dabei mit akzeptablen Antwortzeiten dienen kann. Falls nicht, muss festgehalten werden, an welchem Punkt sich das Systemverhalten verschlechtert.

Als Kennzahlen für die Leistung werden typischerweise Antwortzeit, Durchsatz oder Auslastung des Systems in Abhängigkeit der zu verarbeitenden Last angegeben. Die Antwortzeit wird dabei als jene Zeit definiert, die zwischen dem Abschicken einer Anfrage an das System und dem Erhalten einer Rückmeldung vergeht. Der Durchsatz gibt an, wie viele Anfragen pro Zeiteinheit fertig gestellt werden können. Die Auslastung besagt, zu wie viel Prozent das System beschäftigt war.

Das Testen der Performanz ist ein Prozess, bei dem Informationen gesammelt und analysiert werden, um vorherzusagen, wann bestimmte Lasten die Systemressourcen ausschöpfen. Dabei gehen Last- und Stress-Tests oft mit dem Performanztest einher. Sie basieren auf dem gleichen Prinzip. Es werden mehrere Anfragen gleichzeitig von simulierten Benutzern an die zu testende Web-Anwendung gesandt und Antwortzeit sowie Durchsatz gemessen. Trotzdem verfolgen sie unterschiedliche Testziele. Beim Lasttest wird verifiziert, ob das System die geforderten Antwortzeiten und den geforderten Durchsatz erbringt. Dazu werden zunächst Lastprofile ermittelt (d.h. welche Art von Zugriffen, wie viele pro Tag, mit welchen Spitzenzeiten, wieviele Transaktionen pro Sitzung), dann die Ziele für die Antwortzeiten und den Durchsatz (d.h. unter Normalzuständen sowie zu Spitzenzeiten, mit minimalen, maximalen und durchschnittlichen Werten) ermittelt. Anschließend werden die Tests anhand der Lastprofile durchgeführt und die tatsächlichen Antwortzeiten und Durchsätze gemessen. Beim Auswerten der Ergebnisse werden eventuelle Engpässe (bottlenecks) indentifiziert und beseitigt. Beim Stresstest wird verifiziert, ob das System in Stresssituationen kontrolliert reagiert. Durch Extrembedingungen, wie zum Beispiel durch unrealistisches Überlasten oder durch stark schwankende Belastungen werden derartige Stresssituationen simuliert. Ziel des Tests ist es, festzustellen, ob das System jederzeit die geforderten Antwortzeiten und den geforderten Durchsatz erreicht bzw. ob es mit einer entsprechenden Fehlermeldung angemessen reagiert.

Das grundlegende Vorgehensmodell bei der Leistungsanalyse kann in folgende drei sich wiederholende Phasen untergliedert werden:(1) die Plannungsphase, (2) die Testphase und (3) die Analysephase. Während der Plannungsphase muss zunächst das zu untersuchende System und danach die erwarteten Kennzahlen definiert, System und Testanforderungen gesammelt, Arbeitslast definiert, Metriken ausgewählt und Testpläne entworfen werden. Während der Testphase werden die Tests durchgeführt und Daten gesammelt. In der Analysephase werden die Ergebnisse analysiert und Optimierungsvorschläge diskutiert, die erneut getestet werden müssen.

5 Testautomatisierung

Das Testen von umfangreichen Anwendungen kann kaum ohne die Unterstützung von Testwerkzeugen durchgeführt werden. Denn erst durch den Einsatz von Werkzeugen können die häufig zu wiederholenden Tests innerhalb der kurzen Entwicklungszyklen und des engen zeitlichen Rahmens durchgeführt werden. Automatisierte Tests sparen Zeit beim Herstellen der Vorbedingungen, beim Ausführen und beim Überprüfen der Ergebnisse. Mit ihrer Hilfe können große Mengen von Eingabedaten und -kombinationen getestet werden. Testdurchführungen können exakt reproduziert und mit geringem Aufwand wiederholt werden. Der Einsatz von Werkzeugen bringt gegenüber der manuellen Testausführung jedoch nur dann einen Vorteil, wenn sich der höhere Initialaufwand für die Automatisierung durch die häufige Wiederholung der Tests rentiert. Dies gilt insbesondere für die stark iterative Entwicklung und die evolutionäre Weiterentwicklung von Web-Anwendungen. Dabei werden die Tests zumindest in jeder Iteration erneut ausgeführt und in der evolutionären Weiterentwicklung werden bestehende Teile der Web-Anwendung und somit auch bestehende Tests wiederverwendet.

Nguyen, Johnson und Hackett listen eine Reihe von Werkzeugen für das Testen von Web-Anwendungen auf (vgl. [Ngu03] Kapitel 21). Davon werden hier nur einige genannt.

Hier eine kleine Auswahl von kostenlosen Validatoren und Link-Checkern: Dr. Watson (*watson.addy.com*), Xenu's Link Sleuth (*home.snafu.de/tilman/xenulink.html*), Watchfire Linkbot Pro (*www.watchfire.com*)

Load-und Performance-Testwekzeuge ermöglichen es, Requests an den Server zu simulieren und Leistungsmetriken, Antwortzeiten und Datendurchsatz zu verfolgen. Für die Leistungsanalyse können diese Informationen tabellarisch oder graphisch dargestellt werden. Dazu gehören: Empirix Eload (*www.rswsoftware.com*), Loadtesting.com Portent (*www.loadtesting.com*).

Es existieren auch Werkzeuge, die das Testen der Web-Anwendung auf Sicherheit unterstützen. Dazu zählen unter anderem: Surfingate Firewalls (*www.finjan.com*) und Netscan Tools (*www.nwpsw.com*).

6 Zusammenfassung

Zusammenfassend können wir feststellen, dass das Testen von Web-Anwendungen über das Testen von klassischen Softwaresystemen hinaus geht. Es gelten zwar vergleichbare Anforderungen an die fachliche Korrektheit einer Anwendung, jedoch ergeben sich aus der Nutzung einer Web-Anwendung durch heterogene Benutzergruppen auf einer Vielfalt von Plattformen besondere Herausforderungen an das Testen. Dazu gehören ebenfalls Konfigurations- und Kompatibilitätstests sowie User Interface Tests, die leider in dieser Seminararbeit aus Platzgründen nicht behandelt werden konnten.

Da zudem die späteren Benutzerzahlen schwer abzuschätzen sind und da Antwortzeiten im Internet zu den entscheidenden Erfolgsfaktoren zählen, müssen diese bereits früh getestet werden. Darüber hinaus sind Qualitätsmerkmale wie Benutzbarkeit, Erreichbarkeit, Browser-Kompatibilität, Sicherheit, Aktualität und Effizienz entscheidende Eigenschaften für den Erfolg von Webanwendungen.

Literatur

[Kap03] Gerti Kappel. *Web Engineering*. dpunkt Verlag, 2003.

[Ngu03] Hung Q. Nguyen. *Testing applications on the web*. Wiley Publishing, 2003.

[Som01] Ian Sommerville. *Software Engineering*. Addison-Wesley, 2001.

[Wal90] Ernest Wallmüller. *Software-Qualitätssicherung in der Praxis*. Carl Hanser Verlag München Wien, 1990.

Test-Management

Michael Becher

Inhaltsverzeichnis

1 Einführung

Das Testen von Software ist wichtig, das ist bekannt. Wenn bei einem Software-Projekt sichergestellt werden soll, dass das Testen wirklich effektiv ist, dann kann ein Test-Management als eigene Organisationseinheit neben dem Projekt-Management eingesetzt werden. Angesiedelt auf derselben Ebene wie das Projekt-Management, kümmert es sich um den reibungslosen Ablauf derjenigen Teile des Projekts, die mit dem Testen zu tun haben. Vorgestellt wird das Test-Management in folgender Reihenfolge: Nach einer kurzen theoretischen Betrachtung in Abschnitt 2 geht es in Abschnitt 3 um die reibungslose Integration der Testaktivitäten auf Mitarbeiterebene. In Abschnitt 4 wird gezeigt, welche organisatorischen Aufgaben anfallen, um die Effektivität des Testens zu gewährleisten.

Besondere Aufmerksamkeit erhält anschließend der Vorgang der Fehlerbeseitigung in den Abschnitten 5 und 6. Denn ein vorhandener Fehler kann zwar entdeckt werden, das Wissen über dessen Vorhandensein aber im Zusammenspiel zwischen allen Beteiligten verloren gehen. Wichtig ist daher ein funktionierendes Fehler-Management, weil die beste fachliche Methode zur Fehlerentdeckung keinen Nutzen hat, wenn durch mangelndes Wissen über den Prozess des Testens der Fehler entweder gar nicht behoben wird oder erst durch aufwändiges Suchen in Dokumenten die Meldung darüber wiederentdeckt werden muss.

Ausgangspunkt für den vorliegenden Text ist „Basiswissen Softwaretest“ [SL03]. Hier wird neben dem fachlichen Wissen u.a. über statischen und dynamischen Test auch das organisatorische Wissen zum Thema Test-Management behandelt.

2 Zur Theorie des Test-Managements

Zu Beginn werden zwei allgemeine Punkte besprochen. In Abschnitt 2.1 geht es um mögliche Arten, den Software-Test durchzuführen, und in Abschnitt 2.2 um die Rollen, die beim Software-Test eingenommen werden und die dafür benötigten Qualifikationen.

2.1 Aufgabenteilung zwischen Entwicklung und Test: Testmodelle

Es gibt verschiedene Möglichkeiten, wie die Aufgabenteilung zwischen Entwicklung und Test aussehen kann. Der Nutzen verglichen mit den Kosten ist abhängig von der Größe des Projekts. Im einzelnen sind das in aufsteigender Reihenfolge:

1. Geschriebene Programmteile werden vom *Entwickler selbst* getestet. Der organisatorische Aufwand ist dabei am geringsten. Allerdings besteht die Gefahr, dass der Entwickler seine eigenen Fehler übersieht.
2. Ein immer wechselndes *anderes Mitglied des Entwicklungsteams* testet die Programmteile. Dadurch gibt es eine zweite Meinung und der Test ist unbefangener, da nicht die eigenen Fehler gefunden werden müssen.
3. Ein *dediziertes Mitglied des Entwicklungsteams* führt alle Testarbeiten für die vom Team entwickelten Programmteile durch.
4. Entwicklung und Test werden getrennt. Es gibt *dedizierte Testteams*, die die Testarbeit erledigen.
5. Die Testarbeiten werden in eine *separate Organisation* verlagert, sind also unabhängig vom Projektteam. Das kann eine firmeninterne separate Testabteilung sein oder ein externer Testdienstleister.

In den ersten drei Fällen kann es passieren, dass der Test nicht in dem Maße durchgeführt wird, wie es eigentlich notwendig wäre. So etwa, wenn sich die Entwickler als reine Entwickler begreifen und den Test von Software nur als eine destruktive und wenig kreative Aktivität ansehen.

In den letzten beiden Modellen werden diejenigen, die den Test ausführen, nicht nur überzeugt sein vom Sinn ihrer Arbeit, sondern auch die dafür erforderlichen Methoden professioneller anwenden können als in den ersten drei Modellen.

2.2 Rollen beim Test und Qualifikation der Mitarbeiter

Bei der Durchführung des Software-Tests gibt es verschiedene Rollen, die ausgefüllt werden müssen. An oberster und damit verantwortlicher Stelle im Testteam steht der *Test-Manager*. Er arbeitet auf derselben Ebene wie der Projekt-Manager und ist verantwortlich für die Testplanung und -steuerung und alle organisatorischen Aufgaben (s. Abschnitt 4).

Die ausführenden Rollen im Software-Test sind Testdesigner, Testautomatisierer, Testadministrator und Tester. Der *Tester* ist derjenige Mitarbeiter, der den Test tatsächlich durchführen, dokumentieren und gegebenenfalls Fehlerberichte erstellen wird. Für diese Rolle ist qualifiziert, wer das weiter unten beschriebene Foundation Certificate erworben hat. Der *Testadministrator* ist verantwortlich für Installation und technisch

einwandfreien Betrieb der Testumgebung (entspricht der Ebene des Systemadministrators, mit Spezialkenntnissen zu den Testwerkzeugen). Der *Testautomatisierer* mit Grundlagenwissen in allen Bereichen und Spezialwissen in Testautomatisierung ist dafür zuständig, den Testern mehr Arbeit in weniger Zeit zu ermöglichen. Das Aufgabengebiet des *Testdesigners* ist das Erstellen der Testfälle, weshalb er vertieftes Wissen über Testmethoden und (formale) Testspezifikation mitbringen muss.

Zur Aufteilung der Mitarbeiter auf die Rollen ist es hilfreich, auf der Grundlage von formalen Qualifikationen entscheiden zu können. Um den Qualifikationsstand von Software-Testern vergleichbar zu machen, wurde deshalb in einer europäisch internationalen Initiative vom britischen *Information Systems Examinations Board* [ISE] ein Qualifizierungsprogramm mit drei Stufen entwickelt. Die erste Stufe endet mit dem *Foundation Certificate* [CTF], das vom Umfang her die Grundlagen zum Software-Test enthält und neben Fachwissen auch vermittelt, warum Disziplin und strukturiertes Vorgehen beim Software-Test erforderlich sind. Diese Grundlagen sind der Horizont des Buches [SL03], das ein Lehrbuch über die Anforderungen zum Erlangen des Foundation Certificate ist.

Die zweite Stufe ist das *Practicioner Certificate* [CTP], das seit 2002 erworben werden kann und vertiefte Kenntnisse zum Software-Test nachweist. Die dritte Stufe ist noch in Planung.

Es ist noch anzumerken, dass die (verantwortlichen) Rollen des Testdesigners und des Testmanagers praktische Erfahrung erfordern, die in den ausführenden Rollen des Testadministrators, Testautomatisierers oder Testers erst einmal gesammelt werden muss. Von der formalen Qualifizierung her ist dafür das Practicioner Certificate gedacht. Des weiteren muss ein Testteam eine gewisse Mindestgröße haben. Wenn zu wenige für den Test qualifizierte Mitarbeiter vorhanden sind, dann kann der gesamte Software-Test durch einen externen Dienstleister durchgeführt werden. Das hat den Vorteil, dass Schulungskosten für die eigenen Mitarbeiter sinken, während die Professionalität des Tests zunimmt.

3 Das Test-Management im Verhältnis zur Entwicklung

Hier geht es um die konkrete Durchführung der Testaktivitäten im Gegensatz zu den organisatorischen Aktivitäten, die in Abschnitt 4 vorgestellt werden. In den Abschnitten 3.1 und 3.3 werden die Schnittstellen zwischen Entwicklung und Test beschrieben und in Abschnitt 3.2 diejenigen Dokumente, deren Inhalt den gesamten Testprozess von Anfang bis Ende dokumentiert und damit nachvollziehbar macht.

3.1 Einbetten des Testens in den Entwicklungsprozess

Das Testen ist eine Tätigkeit, die organisatorisch nicht erst *nach* der Entwicklung beginnt. Vielmehr ist Testen als Überprüfung des gerade Entwickelten ein zentraler Be-

standteil eines jeden Projekts[1]. In den frühen Phasen des Projekts besteht die Aufgabe des Test-Managements darin, die eigentlichen ausführenden Tätigkeiten vorzubereiten. Wenn implementierte und testbereite Programmteile dann zu fortgeschrittener Projektzeit vorliegen, müssen alle vorbereitenden Tätigkeiten abgeschlossen sein, damit der dann zunehmende Zeitdruck nicht zum Scheitern der Testarbeiten führt. Stattdessen soll der Test, wohlgeplant und professionell ausgeführt, die Qualität der entwickelten Software aufzeigen oder Ansätze zur Verbesserung geben.

Die frühen Tätigkeiten sind das Erstellen eines Testkonzepts und einer Testspezifikation (s. nächster Abschnitt). Hält sich der Testmanager dabei an (inhaltlich vollständige!) Vorlagen, so ist sichergestellt, dass dabei alle für einen erfolgreichen Software-Test nötigen Aspekte beachtet werden. Auf der Grundlage dieser Dokumente kann das Testteam mit dem Aufbau der für den Test benötigten Infrastruktur beginnen. Die frühen Projektphasen sind auch die richtige Zeit, um für den Erfolg notwendige Schulungen durchzuführen. Dazu muss der Kenntnisstand der Enwickler zum Software-Test herausgefunden und gegebenenfalls angehoben werden. Denn wie im weiteren noch gezeigt wird, ist das Wissen der Entwickler um die *genauen* organisatorischen Abläufe im Software-Test existenziell wichtig für eine erfolgreiche Durchführung der Testarbeiten, und deshalb auch Themengebiet des Test-Managements. Schulungen müssen unter Umständen auch im Testteam selbst durchgeführt werden, etwa weil es neu zusammengestellt wurde, oder einzelne Mitarbeiter ausgetauscht oder dem Team neu zugeteilt wurden.

Die Bedeutung des Testens während des Entwicklungsprozesses wird besonders deutlich beim Betrachten formalisierter Prozessmodelle wie etwa dem V-Modell, bei dem jeder Stufe der Entwicklung eine Teststufe zugeordnet ist. Wird bei der Entwicklung nach einem solchen Modell vorgegangen, dann ist das Testen automatisch integraler Bestandteil des Projekts und muss nicht erst ausführlich begründet werden.

3.2 Testkonzept, Testspezifikation und Testprotokoll

Wird das Testen wie hier vorgestellt in einer eigenen Organisationseinheit betrieben, dann müssen die im folgenden angesprochenen Dokumente zwingend erstellt werden. Für die Planung des Testens ist das das *Testkonzept* (engl. *test plan*)[2] gedacht, in dem die Fragen zu Umfang, Vorgehensweise, Ressourcen und Zeitplanung für das Testen geklärt werden. So etwa bei personellen Ressourcen die Größe des Testteams und mögliche erwartete Spezialkenntnisse, und bei technischen Ressourcen, ob spezielle Werkzeuge oder Hilfsmittel bereitgestellt oder selbst entwickelt werden müssen und wie lange das dauern würde. Dabei geht es auch um die *Teststrategie*, in der der gesamte Testaufwand im Großen auf die einzelnen Systemteile aufgeteilt wird, basierend auf der Einschätzung, wie schwer sich verbleibende Fehler in den jeweiligen Systemteilen

[1] In der Theorie! Wie ein reales Projekt-Management über Umfang und Art der Testarbeiten entscheidet, ist eine andere Sache. Für diese Ausarbeitung wollen wir aber nur das Beste annehmen!

[2] Der deutsche Begriff *Testplan* ist dem englischen Begriff *test schedule* zugedacht, ein Dokument, das die konkrete zeitliche Planung der Testdurchführung festlegt (Zuordnung der Testfälle zu Testern und Festlegung des Durchführungszeitpunktes). Siehe dazu das Glossar in [SL03].

auswirken. Außerdem gehört in das Testkonzept auch eine zeitliche Priorisierung der Tests. Hier wird der mögliche Zeitdruck zum Ende eines (jeden) Projekts beachtet und die Tests zeitlich so geordnet, dass die kritischen Systemteile zuerst getestet werden.

Die *Testspezifikation* enthält die auf der Basis der im Testkonzept festgelegten Teststrategie die eigentlichen Testfälle und die Begründung für deren Auswahl. Dabei wird unterschieden zwischen *logischen* Testfällen, die etwa aus der Anforderungsspezifikation erstellt werden, und den *konkreten* Testfällen, die erst später im Entwicklungsprozess definiert und unverändert ausgeführt werden können. Die Spezifikation aller Testfälle enthält selbstverständlich auch die Vorbedingungen und die Sollresultate.

Nach Abschluss eines Testlaufs wird das *Testprotokoll* als schriftlich festgehaltenes Ergebnis erstellt. Hieraus muss hervorgehen, welche Teile wann, von wem, wie intensiv und mit welchem Ergebnis getestet wurden. Bei entsprechender Werkzeugunterstützung kann es auch automatisch erstellt werden.

3.3 Soziale Kompetenzen bei der Arbeit in Teams

In einer idealen Welt funktioniert die Zusammenarbeit zwischen dem Entwicklungs- und dem Testteam vollkommen reibungslos, dieser Abschnitt wäre damit überflüssig. In der Realität der Software-Entwicklung geht es jedoch um die Zusammenarbeit zwischen (ganz normalen) Menschen. Insbesondere beim Zusammenspiel von Entwicklern und Testern liegt ein großes Spannungspotenzial, denn hier geht es darum, Fehler zu finden. Fehler, die Menschen gemacht haben, und die ihnen vor Augen geführt werden. Die Rollenverteilung ist dabei festgelegt: Die Entwickler machen die Fehler und die Tester finden sie. Ganz abgesehen von der fachlichen Qualifikation aller Mitarbeiter hängt die Effizienz des Testprozesses auch von gewissen, im folgenden beschriebenen, sozialen Kompetenzen im Umgang der Mitarbeiter miteinander ab. Im Rahmen des Test-Managements ist dafür Sorge zu tragen, dass die Zusammenarbeit unter Beachtung dieser Aspekte abläuft.

Generell können Spannungen vermieden werden durch den Aufbau von Verständnis der Arbeit des jeweiligen anderen. Das mag (wie vielleicht häufiger in dieser Ausarbeitung) wie eine Selbstverständlichkeit klingen. Dabei muss aber beachtet werden, dass nicht immer jeder Mitarbeiter an einem Projekt von vorneherein eine so umfassende Vorbildung hat, dass er nicht nur seine eigene Rolle im Projekt souverän beherrscht, sondern auch Grundwissen bzw. Verständnis für die anderen Rollen und Tätigkeiten im Projekt hat! Der Prüfungsinhalt des Foundation Certificate enthält (da es ja um das Testen geht) keine Aussagen zur Software-*Entwicklung*, sondern lediglich zur Einbettung des Testens in die Entwicklung. Andererseits ist es auch so, dass Mitarbeiter in die Rolle des Software-Entwicklers treten und überhaupt keine formale Qualifikation dafür haben[3]. Diese haben natürlich das Wort Software-Test schon einmal gehört, was dabei genau passiert und wie mit den Ergebnissen konkret umzugehen ist, ist dann aber eventuell nicht bekannt, insbesondere wenn es um das Verständnis eines zusammen-

[3] Also die Quereinsteiger, die damals wie heute von dem zahlenmäßigen Mangel an richtig qualifizierten Fachkräften und dem mangelnden Wissen der Entscheider über eine richtige Qualifikation profitieren

hängenden Ansatzes wie hier im Test-Management geht. Ob diese Arten des “entwicklungsfremden” Testers und des “testfremden” Entwicklers tatsächlich eine relevante Größe für oder wider den Erfolg des Projekts sind, muss im Einzelfall entschieden und es muss gegebenenfalls durch Schulungen, Seminare o.ä. gegengesteuert werden.

Eine Auswahl an weiteren Punkten, die helfen, Spanungen zu vermeiden: *Diplomatisches Geschick* (etwa beim Überbringen oder Formulieren der Fehlermeldung), damit verbunden auch *Teamfähigkeit* (auch wenn das “Team” nur ein virtuelles ist, weil etwa der Test räumlich getrennt von der Entwicklung stattfindet und sich die Mitarbeiter kaum persönlich kennen). *Exaktheit* ist immens wichtig, um Missverständnisse schon vorsorglich zu vermeiden und keine ungerechtfertigten Fehlermeldungen zu verbreiten, *Kreativität* ist immer wichtig, da es sich bei den Testobjekten um Dinge handelt, zu denen sich nur wenige Menschen vorher Gedanken gemacht haben. Dazu gehören dann auch die Bereitschaft, *scheinbare Tatsachen zu hinterfragen* und die Fähigkeit, sich schnell *in neue Sachverhalte einzuarbeiten*.

4 Das Test-Management im Verhältnis zum Projekt-Management

Hier geht es jetzt um die organisatorischen Aufgaben im Test-Management. Also überwachen und steuern in Abschnitt 4.1 und in den folgenden Abschnitten Aktivitäten, die mit dem Projekt-Management zusammen besprochen werden müssen.

4.1 Den Test überwachen und steuern

Nehmen wir den besten Fall für das Test-Management an: Alle in den anderen Abschnitten beschriebenen Punkte werden beachtet bzw. sind als Wissen bei allen Mitarbeitern vorhanden. Es gibt dann also ein funktionierendes Konfigurations-Management und Fehler-Management, dem Test-Management stehen ausreichend große Ressourcen zur Verfügung. In diesem Fall muss sich das Test-Management *nicht* herumschlagen mit den vielen kleinen Störungen, die andernfalls passieren können, wie nicht genau spezifizierten Testobjekten, vielen fehlerhaften Einträgen in der Fehlerdatenbank, nicht beachteten Vorgehensweisen in allen Stufen der Entwicklung und so weiter.

Vielmehr können dann Tätigkeiten ausgeführt werden, die dem Management entsprechen, nämlich den Testzyklus zu überwachen und zu steuern. Einige ausgewählte Tätigkeiten stehen in diesem Abschnitt.

Zentraler Indikator für die aktuelle Qualität der bisher entwickelten Software ist die Fehlerdatenbank. Im Idealfall stehen hier die meisten der vorhandenen Fehler und Abweichungen, da sie durch den gut organisierten Testprozess fast alle gefunden worden sind. Durch die verschiedenen Parameter und Kommentare, die zu einer Fehlermel-

dung gehören[4], stehen eine Vielzahl an Analysemöglichkeiten zur Verfügung, auch um Statistiken zu erstellen, z.B. der zeitliche Verlauf von offenen Problemen über verschiedene Testzyklen. So können neben den grundlegenden Fragen nach der bloßen Anzahl der offenen (schweren) Probleme auch weitergehende Fragen beantwortet werden, z.B. wie die Tendenz dieser Anzahl. Steigt die Anzahl der offenen (oder auch der gefundenen) Probleme mit jedem Testzyklus, so ist das definitiv ein schlechtes Zeichen.

Eine weitere Aufgabe des Test-Managements ist die Zusammenarbeit mit dem Projekt-Management. Ein gutes Arbeitsklima und gegenseitiges Verständnis für die Arbeit des jeweils anderen ist auf dieser Ebene unerlässlich, da das Test-Management meistens der Überbringer von schlechten Nachrichten ist. Das Klima ist deshalb so enorm wichtig, damit die eigentlichen Sachfragen schonungslos besprochen werden können. Wenn etwa das Projekt kurz vor seinem (zeitlichen) Abschluss steht, die offenen Fehler sich aber noch häufen und die Software definitiv noch nicht auslieferungsfähig ist, dann wird klar, was gemeint ist.

Kurz erwähnt werden soll noch, dass alle Analysen auch für das Test-Management intern benutzt werden können. Einerseits kann so der Testprozess in einem kontinuierlichen Verbesserungsprozess verfeinert werden, andererseits können Fehlplanungen (die natürlich auch im Test-Management auftreten können) erkannt und geeignete Gegenmaßnahmen ergriffen werden.

4.2 Konfigurations-Management

Ein funktionierendes Konfigurations-Management ist immer dann wichtig, wenn - wie hier - ein guter Software-Test gewünscht ist. Aus mehreren aktuellen Versionen der Projektdateien (den *Konfigurationsobjekten*) werden ein oder mehrere Testobjekte zusammengestellt (etwa durch Kompilieren). Die Version jedes Konfigurationsobjekts muss beim Benutzen des Testobjekts bekannt sein, nur so ist genaues Arbeiten beim Testen überhaupt erst möglich und eine detaillierte Analyse des Fehlers, wenn z.B. Fehler in der Testumgebung nicht in der Entwicklungsumgebung reproduzierbar sind.

Die wichtigsten Grundregeln im Umgang mit dem Konfigurations-Management sind, dass *jede* Datei, die zum Projekt gehört, unter die Kontrolle des Systems gestellt wird und dass bei *jeder* Änderung ein *präziser* Kommentar dokumentiert, was bei der Änderung geschehen ist. Das ermöglicht das Verfolgen des Status einer Datei und des Projekts generell, und das Nachvollziehen der bisher geleisteten Arbeit. Wird ein Testobjekt erstellt, dann werden die dazu gehörenden Projektdateien während des Tests *nicht verändert* (keine neuen Funktionen und keine sonstigen Änderungen), andernfalls verliert der Test seinen Sinn.

Um die Effektivität des Kofigurations-Management zu prüfen, kann im Rahmen des Test-Managements ein *Konfigurationsaudit* erfolgen, bei dem das Einhalten der Grundregeln geprüft wird. In größeren Projekten besteht zudem die Möglichkeit, einen ei-

[4] die natürlich alle korrekt gesetzt sind, weil wir ja von Idealfall ausgehen

genen Plan für das Konfigurations-Management zu erstellen. Grundlage dafür können etwa die Normen IEEE 828 oder IEEE 1042 sein.

Die Anforderungen an das Konfigurations-Management selbst sind etwa die *Versionenverwaltung* mit Katalogisieren, Speichern und Wiederabrufen eines Konfigurationsobjekts und die *Konfigurationsverwaltung*, die darauf aufbauend alle Konfigurationsobjekte, die ein Teilsystem bilden, zusammen verwalten kann. Gut geeignete Open Source-Werkzeuge dafür sind CVS [CVS] als stabiler Klassiker und Subversion [Sub] als Neuentwicklung der jüngeren Zeit. Neben dem Benutzen von aktuellen Technologien ist Subversion angetreten, die häufig geäußerten (berechtigten) Kritikpunkte an CVS zu beseitigen. So beachtet es auch Verzeichnisse in der Versionskontrolle, ebenso Kopieren oder Umbenennen von Dateien oder Verzeichnissen.

4.3 Nutzen und Bedeutung von Standards

Es sollte eine generelle Einsicht sein, dass Standards in fast allen Bereichen des Arbeitslebens nützlich sind (ein kleiner Hinweis darauf ist allein schon die bloße Anzahl an vorhandenen Standards). Für die hier betrachtete Formalisierung des Software-Tests werden in diesem Abschnitt einige Hinweise auf deren spezielle Nützlichkeit gegeben.

Grundsätzlich beschreibt ein Standard den Stand der Technik in einem bestimmten Bereich zur Zeit seiner Definition. Wer sich also bei der Durchführung seiner Arbeit an Standards hält, sichert damit fast automatisch zwei Dinge: Zum einen sinkt die Notwendigkeit, das Rad neu zu erfinden, da Vorgehensweisen und zu erstellende Dokumente definiert oder zumindest umschrieben sind. Damit einhergehend sinkt zum anderen auch die Wahrscheinlichkeit, dass wichtige Punkte vergessen oder einfach nicht beachtet werden.

Ein weiterer Punkt ist insbesondere dann interessant, wenn zwar der Testmanager weiß, war er tut, das Projekt-Management aber gar nichts vom Umfang der dafür benötigten Ressourcen hält. Sollten die Punkte, die der Test-Manager für wichtig hält, in einem Standard oder einer Norm stehen, so kann das sehr gut als (fast unanfechtbare) Argumentationsgrundlage für die Notwendigkeit der verlangten Ressourcen dienen. Ganz konkret: Standards geben Rückendeckung gegenüber ignoranten oder unwilligen Projekt-Managern, wenn es darum geht, die gedankliche Basis seiner Ideen als nicht subjektiv darzustellen, sondern als notwendig zum effizienten Software-Test.

Es gibt noch einen anderen Aspekt: Es gibt Standards, die zwar definiert, aber nie oder nur von wenigen benutzt wurden, etwa weil die praktische Umsetzung zu komplex ist, oder das Einhalten des Standards ohne wirtschaftliche Bedeutung. Doch meistens gibt es Standards, die erfolgreich von vielen anderen eingesetzt werden, was die eigene Argumentationsposition verbessert, da die Umsetzung des Konzeptes nachweislich funktioniert. Sich an diesen zu orientieren, ist fast nie verkehrt, wohingegen bei kaum benutzten Standards erst einmal eine Bewertung nützlich sein könnte. In der Software-Entwicklung viel benutzt sind etwa Prozessmodelle wie CMM oder Bootstrap, deren Namen man zumindest schon einmal gehört hat. Für den Software-Test gelten etwa

IEEE 829 oder IEEE 1028, aber sogar im branchenneutralen ISO 9001 ist vom Testen die Rede. Weitere Quellen für Standards, die möglicherweise eingehalten werden sollten, sind firmeninterne Dokumente oder Richtlinien, Branchenstandards, die sich sehr speziell auf bestimmte Produkte oder Produktkategorien beschränken, oder bewährte Vorgehensweisen („best practices“), die nicht formal standardisiert, aber nachweislich bewährt sind.

Darüber hinaus hilft das Beachten von Standards bei Rechtsstreitigkeiten, wenn etwa der Kunde grundsätzlich nicht zufrieden ist mit dem gelieferten Produkt. So kann das Projekt-Management nachweisen, dass bei der Entwicklung nach dem Stand der Technik gearbeitet wurde.

5 Fehler-Management I: Modell

In diesem Abschnitt werden die Parameter definiert, die den allgemeinen Begriff "Fehler" bestimmen. Verwaltet wird ein Fehler in der *Fehlerdatenbank* (oder auch Fehler-Management-Werkzeug oder engl. bug tracking system). Bei jedem Eintragen eines Fehlers in die Fehlerdatenbank muss jedem dieser Parameter ein sinnvoller Wert zugewiesen werden. Das führt einerseits dazu, dass schon beim Eintragen strukturierte Gedanken über die Art des Fehlers gemacht werden. Andererseits sind so viel differenziertere Aussagen über die Gesamtzahl der Fehler im Projekt möglich. Die Aussage, dass im Programm (wohl) noch einige Fehler seien, weicht - bei richtiger Anwendung - einer Entscheidung über den Status des Programms, die auf einer viel breiteren Grundlage getroffen wurde, die in einem festen Schema verschriftlicht ist.

5.1 Fehlermeldung

Es gibt einige grundlegende Parameter, die unabhängig von der Art der erstellten Software zu jeder Fehlermeldung gehören, und eine genaue *Identifikation* erlauben. Es sollte selbstverständlich sein, diese zusammen mit dem Fehler zu speichern, um bei Bedarf die Möglichkeit zu haben, Rückschlüsse auf den Fehler zu ziehen. Es beginnt mit so banalen Dingen wie dem Datum der Eintragung, der Identifikation des Eintragenden (meistens des Testers selbst) und einer eindeutigen Identifikation der Meldung in der Fehlerdatenbank, bei der es sich meistens um eine global vergebene Nummer handelt. Diese Parameter werden von den aktuellen Werkzeugen automatisch vergeben, was sicherstellt, dass sie vorhanden sind. Gerade in diesem Fall muss jedem, der mit der Fehlerdatenbank mehr arbeitet als nur Fehlermeldungen einzutragen, die Existenz dieser Parameter bewusst sein, um diese sinnvoll nutzen zu können.

Weiter geht es bei der Identifikation mit denjenigen Parametern, die aus dem wohlstrukturierten Testprozess entstehen. Dann ist es einfach, Parameter wie die *Bezeichnung des Testobjekts* und seiner genauen *Version* anzugeben. Das ist aber nicht selbstverständlich: Wird vom Projekt-Management zwar eine Fehlerdatenbank aufgesetzt, gleichzeitig aber das Testen nicht formal geplant (schließlich ist die Fehlerdatenbank

ein Werkzeug, dessen Einsatz man sieht, und ein Testplan „nur“ ein Dokument, dessen Fehlen nicht auf den ersten Blick auffällt), dann sind Testobjekt und seine Version gar nicht definiert. Damit verlassen die Fehlermeldungen ihre bestmögliche Ebene an Aussagekraft, und die Gefahr wächst, dass wieder einige Erkenntnisse jedes Mal neu gewonnen werden müssen, die sich durch bessere Strukturierung des Testens ganz natürlich ergeben hätten.

Nach der Identifikation erfolgt die *Klassifikation* des Fehlers (s. nächsten Abschnitt) und abschließend die eigentliche *Problembeschreibung*. Hier wird nun in Fließtext all das beschrieben, was durch die anderen Parameter noch nicht abgedeckt wurde.

5.2 Fehlerklassifikation

Bei der *Fehlerklassifikation* geht es wie bei der Identifikation um das Setzen der jeweiligen Parameter aus einer wohldefinierten Menge. Auch hier ist der Vorteil, dass die Ausdrucksmöglichkeiten des Eintragenden genau auf das sinnvolle Maß beschränkt sind, dass zur Klassifikation des Fehlers wirklich benötigt wird.

Bei der Klassifikation gibt es - wie auch bei der Identifikation - zwei Arten von Parametern: Diejenigen, die auch im ungeordneten Testprozess ohne Probleme gesetzt werden können (das sind Klasse und die Priorität), und diejenigen für einen geordneten Testprozess (Anforderung, Fehlerquelle).

Die beiden Parameter Fehlerklasse und Fehlerpriorität ergänzen sich gegenseitig. Die *Fehlerklasse* beschreibt die *Schwere* des Fehlers aus Sicht des (späteren) Anwenders: Der Wertebereich kann vom Systemabsturz über “normale” funktionale Abweichungen bis zu geringfügigen Abweichungen und Schönheits”fehlern” (z.B. Rechtschreibung) gehen. Fehlerdatenbanken können das Test-Management darin unterstützen, indem sie die Definition des Wertebereichs beim Einrichten eines Projekts frei wählbar machen. Die Fehlerklasse ist neben der eigentlichen Anzahl an Fehlern in einem Testobjekt der wichtigste Parameter, um Aussagen über seinen Fortschritt zu berichten (s. Abschnitt 4).

Die *Fehlerpriorität* ist (fast) ganz unabhängig von der Schwere. Hier werden Überlegungen einbezogen, die mit dem Software-Lebenszyklus zu tun haben: Ist der Arbeitsablauf des Anwenders durch den Fehler blockiert, so muss ein *Patch* erstellt werden, der außerhalb der normalen Auslieferungstermine fertiggestellt und ausgeliefert werden muss. Eine Stufe darunter ist die Priorität *Update*, die den Fehler bis zur nächsten Produktversion behoben sehen will (bzw. bis zur nächsten Version des Testobjekts). Die unterste Stufe ist hierbei *Offen*. Die Entscheidung über die Priorität liegt meistens nicht beim Eintragenden (Tester), sondern beim Produkt- oder Projektmanagement. Auch diese Werte wird eine gute Fehlerdatenbank nicht fest vorgeben, sondern projektspezifisch einstellbar machen!

6 Fehler-Management II: Umsetzung

Was wird nach der Eingabe der Fehlermeldung geschehen? Wie wird insbesondere sichergestellt, dass der gefundene Fehler tatsächlich *behoben* wird, idealerweise sogar in einem Prozess, der geringstmöglichen organisatorischen Aufwand erfordert bei garantierter Effektivität?

Grundlage dafür ist, dass die technische Umsetzung des Fehler-Managements von *allen* Projektmitarbeitern gut verstanden wird. Denn das Beseitigen von Fehlern ist ein Prozess, bei dem die Verantwortlichkeit für den jeweils nächsten Schritt zwischen dem Test- und dem Entwicklungsteam wechselt. Als erste Konsequenz für den Testmanager folgt daraus, dass er für den effizienten Prozess der Fehlerbeseitigung auch den Kenntnisstand der *Entwickler* zum Fehler-Management beachten und wie schon erwähnt gegebenenfalls für Schulungen Sorge tragen muss.

In größeren Projekten werden die in Abschnitt 6.2 beschriebenen Rollen manchmal nicht von Einzelpersonen, sondern von Gremien (engl. boards) ausgefüllt. Entscheidungen fallen dann im Konsens aller beteiligten Interessensgruppen. Die Bearbeitung einer einzelnen Fehlermeldung dauert dadurch natürlich entsprechend länger, je nachdem, wie häufig diese Gremien zusammenkommen. Das ist bei der zeitlichen Planung der Testarbeiten mit in Betracht zu ziehen!

6.1 Werkzeuge zum Fehler-Management

Fehler-Management kann nicht manuell erfolgen, Werkzeugunterstützung ist zwingend notwendig. Es gibt wie im Konfigurations-Management auch hier gute Open Source-Werkzeuge, die erfolgreich in vielen Projekten eingesetzt werden.

Ein bekannter Vertreter ist *Bugzilla* [Bug], die Fehlerdatenbank des Mozilla-Projekts. Es ist gut geeignet für große Projekte, hat aber den Nachteil, dass die einstellbaren Parameter nur global und nicht projektspezifisch vergeben werden können. Diese Einschränkung kann durch parallele Installationen des Systems behoben werden, was aber nicht unerheblichen administrativen Mehraufwand erzeugt.

Ein Projekt, das genau diesen größten Mangel beheben will, ist *Scarab* [Sca]. Es läuft nach Aussagen der Entwickler schon recht stabil, ist aber noch nicht in einer für Produktionsumgebungen freigegebenen Version erschienen. Hier liegt der Fokus nicht nur auf Fehlern, sondern es wird generell von Artefakten (engl. *artifact*) als verwalteten Einheiten gesprochen, was auch Erweiterungswünsche einbezieht und andere Möglichkeiten eröffnet für eine geregelte und nachvollziehbare Kommunikation zwischen allen Projektbeteiligten, die unabhängig von der eigentlichen Fehlerbeseitigung abläuft.

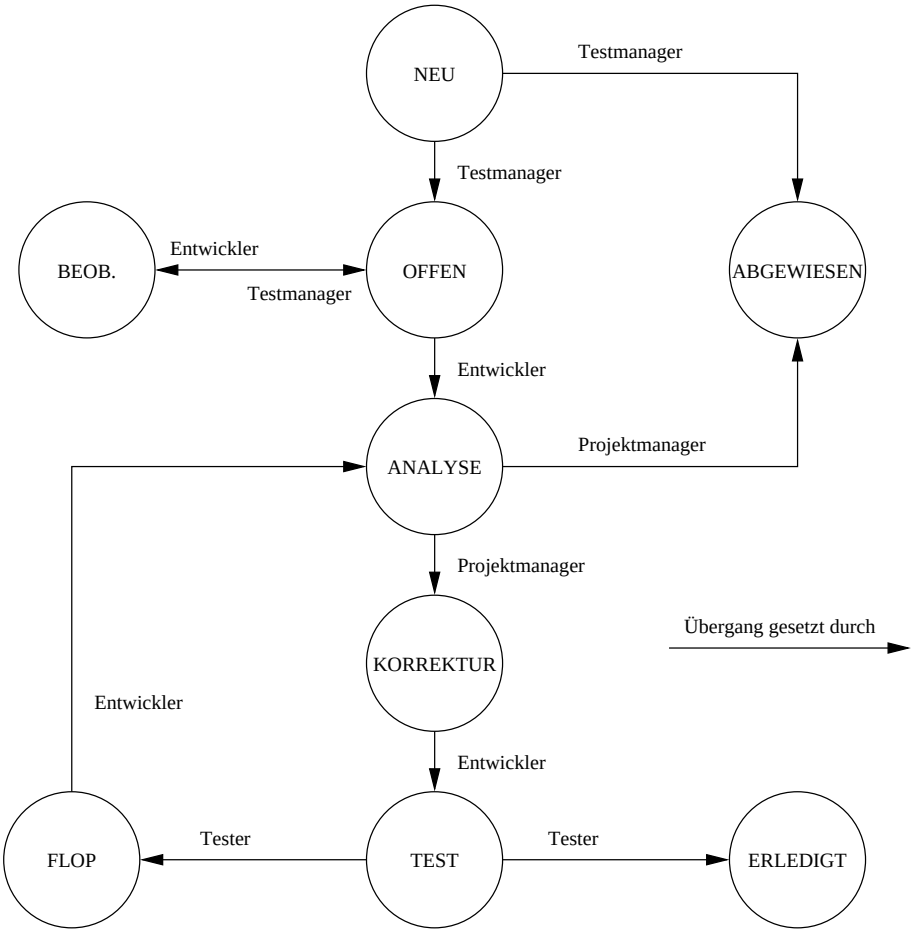

Abbildung 1: Fehlerstatus Modell (in leicht veränderter Form aus [SL03, S. 162])

6.2 Fehlerbeseitigung im Zusammenspiel aller Beteiligten

Der vollständige Ablauf ist in Abb. 1 dargestellt. Ist er einmal in den Köpfen aller an der Fehlerbeseitigung beteiligten Mitarbeiter fest verankert, dann ermöglicht dieser Ablauf effiziente Fehlerbeseitigung und stellt Zeit zur Verfügung, um weitere Probleme in Angriff zu nehmen. Er wird im folgenden ausführlich beschrieben. Es ist anzumerken, dass der Ablauf nicht zwangsweise in dieser Vollständigkeit durchgeführt werden muss, sondern einige Schritte auch weggelassen werden können (s. nächsten Abschnitt).

Wird beim Testen ein Fehler entdeckt, so trägt der Tester in die Fehlerdatenbank eine Fehlermeldung ein, mit Parametern wie in Kapitel 5 beschrieben. Die Fehlermeldung erhält den Status NEU. Der nächste Schritt liegt beim Testmanager. Er setzt ändert den Zustand auf OFFEN, wenn alle festgelegten Parameter richtig gesetzt wurden, die Meldung keinen Unsinn enthält und auch keinen schon berichteten Fehler enthält (“Dublette”, duplicate). Ist alles in Ordnung, so wählt der Testmanager den zuständigen Entwickler, und weist ihm die Meldung zu. Andernfalls setzt er den Status auf ABGE-

WIESEN und dokumentiert dabei den Grund, warum dieser Fehler unberechtigt ist.

Fängt der zugewiesene Entwickler mit der Bearbeitung der Fehlermeldung an, so versucht er zuerst einmal, das geschilderte Problem nachzuvollziehen. Kann er es weder nachvollziehen noch ausschließen, so wird das Bearbeiten der Meldung verschoben, der Status auf BEOBACHTUNG gesetzt. Nun liegt es wieder in der Hand des Testmanagers, die Meldung bei neuen Erkenntnissen wieder auf OFFEN zu setzen. Ist die Meldung nachvollziehbar, beginnt der Entwickler mit der Bearbeitung, fügt seine Analysen und Kommentare an die Meldung an und zeigt das Ende mit dem Status ANALYSE an, wodurch die Meldung dem Projektmanager zugewiesen wird.

Der Projektmanager kann jetzt an Hand der Analyse entscheiden, was mit dem Fehler weiter passieren soll: Es kann etwa die Fehlerpriorität geändert werden (z.B. nach unten, wenn der Korrekturaufwand als zu hoch eingeschätzt wird) oder unter Umständen auch in dieser Phase noch der Fehler als ABGEWIESEN deklariert werden[5], selbstverständlich mit entsprechend angefügter Begründung! Ansonsten geht der Status auf KORREKTUR und die Meldung wieder an den zuständigen Entwickler.

Der Entwickler korrigiert den Fehler und setzt danach den Status auf TEST. Es ist dabei organisatorisch sicherzustellen, dass der Entwickler die Art der Korrektur und die Version der bearbeiteten Dateien in der Fehlermeldung dokumentiert, damit die Fehlerbehebung auch im Nachhinein nachvollziehbar bleibt!

Der Tester steht nun an der letzten Stelle im Prozess der Fehlerbearbeitung: Im nächstmöglichen Testzyklus versucht er, die Beseitigung des Fehlers zu verifizieren und wiederholt dabei mindestens den fehleraufdeckenden Test, nach Möglichkeit noch weitere. Kann er bestätigen, dass der Fehler mit den von ihm durchgeführten Tests nicht mehr auftritt, so wird er dieses Ergebnis dokumentieren und die Fehlermeldung mit dem Zustand ERLEDIGT schließen. Wichtig dabei ist, dass nur der *Tester* den Zustand ERLEDIGT setzen darf! Im unerfreulichen Fall der Nichtbeseitigung setzt er den Zustand FLOP, wodurch der Fehler wieder dem Entwickler zugewiesen wird. Dieser muss nun weitere Analysen durchführen.

An dieser Stelle sollte klar sein, dass die oben beschriebene Fehlerbehandlung viel Zeit kostet. Zeit, die unter Umständen nicht in allen (weil kleinen) Projekten zur Verfügung steht. In diesem Fall können einige der Schritte übersprungen werden, d.h. Zustände zusammengefasst werden. Bei jeder Vereinfachung des Modells werden jedoch Abstriche gemacht in Bezug auf die zu erwartende Qualität des Fehler-Managements (bzw. der Fehlerbeseitigung)! Deshalb ist es enorm wichtig, dass *jede* Entscheidung über eine Beschneidung des Prozesses getroffen wird erstens im Bewusstsein des damit erkauften Nachteils und zweitens in Kenntnis des *vollständigen* Modells. Wenn also der Entscheider (also Projekt-, Test- oder sonstiger Manager) darüber unsicher ist, etwa weil er wenig praktische Erfahrung damit hat, dann kann dieser kritische Teil des Testens das gesamte Entwicklungsergebnis negativ beeinflussen.

[5] Es ist ja das erste Mal, dass der *Projekt*manager die Meldung sieht. Falls ihm weitergehende Informationen als etwa dem Testmanager vorliegen, dann wird die Meldung auch nach der Analyse verworfen.

Steht dem Projekt nun also wenig Zeit zur Verfügung[6] und setzen wir präzise arbeitende Mitarbeiter voraus, so können die Zustände NEU und OFFEN zusammengefasst werden: Der Eintragende trägt nun selber dafür Sorge, keinen Unsinn und auch keine Doppelmeldungen einzutragen. Auch kann er selber den zuständigen Entwickler auswählen. Dafür müssen allerdings die jeweiligen Entwicklungskompetenzen genau festgelegt und allen bekannt sein. Der Zustand ABGEWIESEN wird dann nicht vom Testmanager, sondern vom ausgewählten Entwickler gesetzt.

Auch die Zustände ANALYSE und KORREKTUR können zusammenfallen, wenn dem Entwickler die Kompetenz zugestanden wird, selbst zu entscheiden, ob ein Fehler nach der Analyse noch behoben werden muss oder ABGEWIESEN wird. Verläuft der Test nach dem naiven Testmodell, dann kann auch der Zustand TEST in die Kompetenz des Entwicklers fallen. Diese Entscheidung sollte aber, wie die anderen hier beschriebenen auch, gut überlegt sein.

7 Zusammenfassung

Wir haben gesehen, dass sehr viele Aspekte beachtet werden müssen, wenn es darum geht, Software effizient zu testen durch den Aufbau der eigenen organisatorischen Abteilung des Test-Managements. Viele der hier beschriebenen Vorgehensweisen in voller Größe umzusetzen bedarf einer gewissen Projektgröße, damit es sich rentiert. Trotzdem wurde versucht, auch Ansätze zur Einsparung von Ressourcen für mittlere und kleine Projekte zu geben. Wichtig dabei ist, dass jede Einschränkung gegenüber dem vollständigen Ansatz wohlüberlegt und die Gesamtheit auch bekannt sein muss.

Von enormer Wichtigkeit für das Test-Management ist die Fehlerdatenbank. Nach den Plänen und Spezifikationen, die für den Software-Test erstellt worden sind, spielt sich hier im Zusammenwirken von Entwicklungs- und Testteams der Hauptteil der Fehlerbeseitigung ab. Bei richtigem Anwenden der Parameter, die zu einer Fehlermeldung gehören, bietet sich dem Test-Management auch ein weites Feld zum Erstellen von Statistiken und Analysen.

[6] also begrenzte personelle Ressourcen, weil es etwa ein kleineres Projekt ist

Literatur

[Bug] Bugzilla. http://www.bugzilla.org.

[CTF] Foundation certificate. http://www1.bcs.org.uk/DocsRepository/00900/913/docs/Syllabus.pdf.

[CTP] Practicioner certificate. http://www1.bcs.org.uk/DocsRepository/00900/913/docs/PractSyll.pdf.

[CVS] CVS - Concurrent Versioning System. http://www.cvshome.org.

[ISE] Information Systems Examination Board (ISEB). http://www.iseb.org.uk.

[Sca] Scarab Issue Management System. http://scarab.tigris.org.

[SL03] A. Spillner and T. Linz. *Basiswissen Softwaretest*. dpkunkt.verlag, 2003.

[Sub] Subversion. http://subversion.tigris.org.

Strategien und Prinzipien des Abnahmetests

Uschi Rick

Inhaltsverzeichnis

1 Einleitung

Bevor ein Produkt in den Betrieb aufgenommen wird, muss untersucht werden, ob es die zu Beginn der Systementwicklung festgelegten Anforderungen erfüllt. Ein solcher Abnahmetest dient im Rahmen einer Auftragsarbeit dazu, die Qualität des Produktes zu testen und zu prüfen, ob alle vertraglich geregelten Anforderungen erfüllt werden. Der Auftraggeber nimmt das Produkt gegenüber dem Auftragnehmer ab. Hierbei ist es sinnvoll, methodische Verfahren zu benutzen, die sowohl günstig als auch aussagekräftig sind. Dieser Grundsatz wird aber in vielen Projekten verletzt, indem Maßnahmen getroffen werden, die keinem klaren Konzept genügen, und so kommt es zu unnötiger Zeit- und Geldverschwendung. Dem kann man entgegenwirken, indem methodisch Abnahmekriterien auf Basis der Anforderungen erstellt werden.

Ziel dieser Seminararbeit ist es, die grundlegene Methodik des Abnahmetests vorzustellen und zu erläutern, um dem Leser auf diese Weise einen thematischen Überblick zu bieten. Hierbei wird der Fokus auf der SOPHISTischen Vorgehensweise zur Entwicklung von Abnahmekriterien und -tests liegen. Die Unternehmen *SOPHIST Gesellschaft für innovatives Software-Engineering mbH* und *SOPHIST Technologies Gesellschaft für innovative Informationstechnologien mbH* sind seit Jahren im Bereich Requirements Engineering und Requirements Management tätig. Sie haben aufgrund ihrer Erfahrungen schon existierende Vorgehensmodelle um neue Aspekte ergänzt. Im zweiten Abschnitt wird nun das SOPHISTische Vorgehensmodell für die Anforderungsanalyse, das Object Engineering, dargestellt und erklärt. Im nächsten Abschnitt wird der Begriff der Abnahmekriterien zuerst formal definiert und anschließend noch einmal in zwei Typen unterteilt. Darauf aufbauend können dann in Abschnitt 4 die Merkmale qualitativ hochwertiger Kriterien festgelegt werden. In den folgenden Abschnitten 5 und 6 werden sowohl allgemeine Hilfestellungen als auch konkrete Verfahren zur Entwicklung von Abnahmekriterien beschrieben. In Abschnitt 7 wird schließlich dargestellt, wie ein Testszenario definiert ist und wie es aus den Abnahmekriterien sinnvoll abgeleitet werden kann. Abschließend werden im letzten Abschnitt noch einmal die wichtigsten Punkte zusammengefasst.

2 Grundlagen des Object Engineering

Es existieren viele unterschiedliche Vorgehensmodelle, da sie jeweils für einen Kontext konzipiert wurden. In diesem vorgegebenen Rahmen können die Modelle auch sinnvoll eingesetzt werden. Die Schwierigkeit bei der Wahl eines Modells besteht darin, dass alle spezifischen Rahmenbedingungen eines Projektes berücksichtigt werden müssen, um ein optimales Vorgehen zu finden. Man unterscheidet Vorgehensmodelle unter anderem nach ihrer Vorgehensweise. So gibt es beispielsweise iterative Vorgehen, die es ermöglichen, erreichte Ergebnisse zu untersuchen und zu verbessern, indem schon durchlaufene Schritte wiederholt werden. Inkrementelle Modelle erstellen ein System etappenweise, indem stetig in jeder Etappe neue Funktionalitäten integriert werden. Als Basis und Strukturierungselement für weiteres Vorgehen können auch An-

wendungsfälle benutzt werden. Ein solches anwendungsfallgetriebenes Vorgehen kann zusammen mit den beiden erstgenannten Modellen angewandt werden. Das agile Vorgehen arbeitet ergebnisorientiert: Es wird immer diejenige Aktivität durchgeführt, die minimale Risiken birgt und gleichzeitig vorhandene Chancen am sinnvollsten ausnutzt. Man kann aber Vorgehensmodelle nicht nur nach methodischem Vorgehen sondern auch nach Abstraktionsgrad unterscheiden. Das sogenannte V-Modell ist zum Beispiel ein sehr abstraktes Modell, so dass es in vielen verschiedenen Projekten angewendet werden kann, egal ob es sich um große Bauprojekte oder überschaubare Softwareprojekte handelt. Konkretere Hilfestellungen für die Entwicklung von Software bietet hingegen das Vorgehensmodell des Rational Unified Process , es ist allerdings nur für den objektorientierten Softwareentwicklungszyklus konzipiert. Die Schwachstelle beider Modelle liegt in der Anforderungsanalyse, da nicht detailliert genug beschrieben wird, wie Anforderungen methodisch gefunden, formuliert und geprüft werden können. An dieser Stelle ergänzt nun das SOPHISTische Objekt Engineering (OE) die vorhandenen Vorgehensmodelle und nutzt zusätzlich agile Ansätze und solche des Extreme Programming. Es definiert eine systematische, ingenieursmäßige und flexible Vorgehensweise und legt dazu folgende aufeinander abgestimmte Komponenten fest:

- die einzelnen Aktivitäten, die ausgeführt werden können
- die Personen, die verantwortlich für die Aktivitätenausführung sind
- die Abhängigkeiten zwischen den Aktivitäten
- die erstellten Produkte
- die Methoden, die zur Durchführung der Aktivitäten benutzt werden

Abbildung 1 veranschaulicht die einzelnen Aktivitäten und die resultierenden Produkte. Hierbei ist keine Reihenfolge der Bearbeitung vorgeschrieben und nicht alle angegebenen Aktivitäten und Produkte müssen ausgeführt, bzw. erstellt werden, sondern lediglich diejenigen, die tatsächlich benötigt werden. Weiterhin sollen Produkte nicht ausführlicher als nötig erarbeitet werden, da sie sonst zu statisch gegenüberÄnderungen werden.

3 Abnahmekriterien

3.1 Funktion und Definition

Abnahmekriterien haben zwei grundlegene Funktionen:

- Sie können genutzt werden, um zu testen, ob ein System die gestellten Anforderungen erfüllt.
- Sie dienen der Überprüfung der Qualität von Anforderungen.

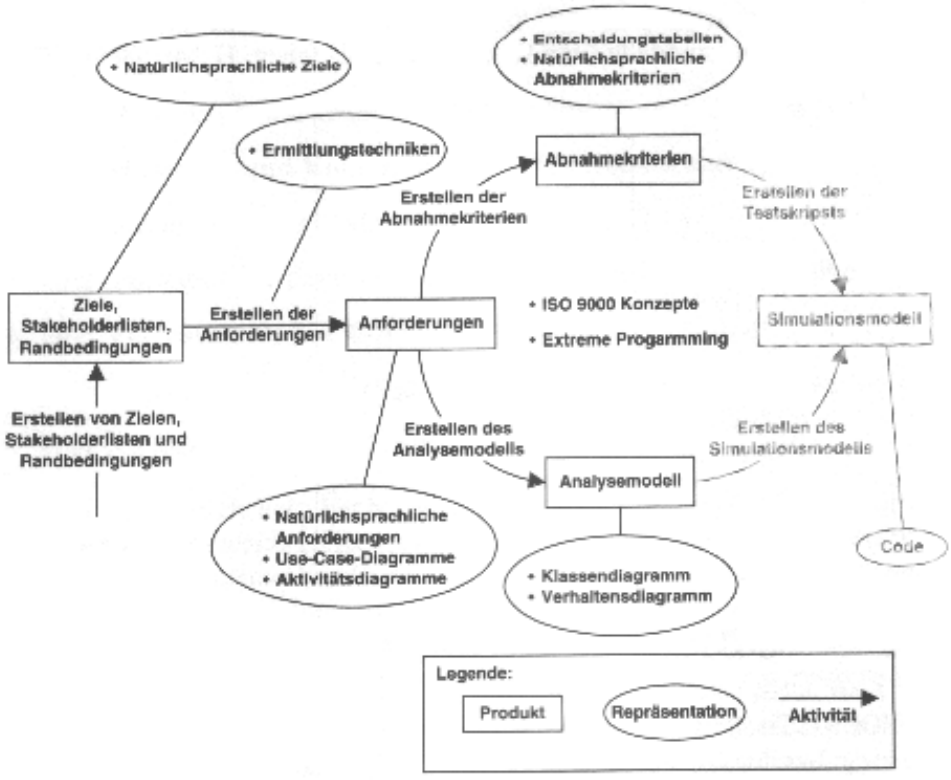

Abbildung 1: Vorgehensmodell Object Engineering

Abnahmekriterien sollten gleichzeitig mit den Anforderungen erstellt werden, damit gefundene Fehler und Unstimmigkeiten noch rechtzeitig und ohne großen Kostenaufwand behoben werden können. Es wird geprüft, ob die Anforderungen komplett sind, und gleichzeitig werden konkrete Beispiele für eine abstrakte Anforderung spezifiziert. Auf diese Weise reduzieren Abnahmekriterien die Gefahr von Mißverständnissen, erhöhen die Verständlichkeit der Anforderungen und stellen somit eine effektive Methode zur Qualitätssicherung dar. Die formale Definition eines Abnahmekriteriums ist wie folgt gegeben:

Def.: Ein *Abnahmekriterium* ist eine Anweisung für den Test bezüglich einer Anforderung (oder eines Anforderungsteils), die die Prüfung und Bewertung des erstellten Produktes oder durchgeführten Prozesses gegenüber dieser Anforderung (oder des Teils) beschreibt. Quelle: [1, S. 294]

Es wird also eine Vorschrift festgelegt, wie ein Prüfling auf Spezifikationstreue getestet werden kann, und zwar immer genau für eine Anforderung, bzw. einen Anforderungsteil. Es können folglich mehrere Abnahmekriterien für eine Anforderung definiert werden. Tabelle 1 zeigt ein erstes, mögliches Abnahmekriterium eines Bibliothekssystems. Im folgenden werden sich die Beispiele immer auf ein zu erstellendes Bibliothekssystem beziehen.

Ausgangssituation	Der Administrator ist im Bibliothekssystem angemeldet.
Ereignis	Der Administrator startet einen Report mit Ausgabe auf dem Drucker.
Erwartetes Ergebnis	Das Bibliothekssystem hat den vom Administrator gewählten Report nach maximal 20 Sekunden auf dem Drucker ausgegeben.

Tabelle 1: ein erstes Abnahmekriterium

3.2 Typen von Abnahmekriterien

Abnahmekriterien werden anhand zweier Dimensionen klassifiziert. Zum einen unterscheidet man sie anhand ihres Formalisierungsgrades, zum anderen anhand ihres Abstraktionsgrades. Es ergeben sich die folgenden Typen von Abnahmekriterien:

- natürlichspachlich und abstrakt
- formalisiert und abstrakt
- natürlichsprachlich und konkret
- formalisiert und konkret

3.2.1 Natürlichsprachliche Abnahmekriterien

Um die Übersichtlichkeit zu gewährleisten, müssen natürlichsprachliche Abnahmekriterien in einer standardisierten Form dargestellt werden. Diese gliedert sich in drei Teile:

1. die Ausgangssituation des Tests
2. das Ereignis des Tests
3. das erwartete Ergebnis des Tests

Im ersten Teil wird der zur Durchführung des Testkriteriums nötige Zustand des Prüflings und der Testumgebung beschrieben. Diese Angaben findet man meistens innerhalb der Anforderungen in Form von Bedingungen, Funktionalitätseinschränkungen o.ä. (z.B.: Nur wenn ...., dann... oder Ist ...., soll das System...). Der zweite Teil beinhaltet jenes Ereignis, welches für das in der Anforderung spezifizierte Verhalten verantwortlich ist: z.B. eine Nutzereingabe, das Eintreffen einer Meldung an einer Schnittstelle oder ein zeitlich orientiertes Ereignis (z.B.: Wenn der Nutzer ... drückt, dann soll das System... oder Beim Eintreffen einer Meldung soll das System..). Das erwartete Ergebnis stellt den Soll-Zustand des Prüflings nach dem Test dar. Es entspricht der Nachbedingung der Anforderung (z.B.: ...soll das System.... machen oder

...soll das System eine Meldung versenden). Das folgende Beispiel verdeutlicht den Erstellungsprozess natürlichsprachlicher Anforderungen auf Basis der Anforderung. Folgende Anforderung sei gegeben:

Anforderung Wenn ein Nutzer innerhalb eines Ausleihvorgangs einen Bibliothekskunden auswählt, soll das Bibliothekssystem diesem Nutzer den Namen, die Adresse und den Kontostand des Kunden anzeigen.

Die Tabelle 2 zeigt ein mögliches, erstes (noch zu abstraktes) natürlichsprachliches Abnahmekriterium.

Ausgangssituation	Ein Nutzer ist im Bibliothekssystem angemeldet. Es ist mindestens ein Bibliothekskunde erfasst.
Ereignis	Im Rahmen eines Ausleihvorgangs wählt der Nutzer einen Kunden aus.
Erwartetes Ergebnis	Das Bibliothekssystem zeigt dem Nutzer, der den Kunden im Rahmen des Ausleihvorgangs auswählt, den Namen, die Adresse und den augenblicklichen Kontostand des ausgewählten Kunden an.

Tabelle 2: natürlichsprachliches Abnahmekriterium

Wenn man nun konkrete Testdaten ergänzt, entsteht ein in Tabelle 3 dargestelltes natürlichsprachliches, konkretes Abnahmekriterium.

Ausgangssituation	Der Bibliothekar *Sokrates* ist im Bibliothekssystem angemeldet. Es ist der Bibliothekskunde *Protagoras* mit der Adresse *Philosophenweg 42, Athen* erfasst und dieser hat augenblicklich zwei Leihgegenstände ausgeliehen.
Ereignis	Im Rahmen eines Ausleihvorgangs wählt Bibliothekar *Sokrates* den Kunden *Protagoras* aus.
Erwartetes Ergebnis	Das Bibliothekssystem zeigt dem Bibliothekar *Sokrates* den Namen *Protagoras*, die Adresse *Philosophenweg 42, Athen* und den augenblicklichen Kontostand von zwei an.

Tabelle 3: natürlichsprachliches, konkretes Abnahmekriterium

In diesem Fall deckt ein Abnahmekriterium die gesamte Anforderung ab, dies muß aber, wie bereits erwähnt, nicht immer der Fall sein.

3.2.2 Formalisierte Abnahmekriterien

Formalisierte Abnahmekriterien haben auch eine standardisierte Struktur. Sie werden nach [1] als Entscheidungstabellen mit dem in Tabelle 4 dargestellten Aufbau.

Liste der Bedingungen	Bedingungszeiger Kombination aller Werte für die Bedingung Die Zellen einer Zeile enthalten die möglichen Werte der jeweiligen Bedingung. Bei mehreren Zeilen entspricht jede Spalte einer Bedingungskombination.
Liste der Aktionen	Aktionszeiger Belegung der Bedingungen mit Aktionen

Tabelle 4: Struktur formalisierter Abnahmekriterien

Hierbei beschreiben die Bedingungen mögliche Zustände des Prüflings, bevor eine Aktion ausgeführt werden kann, ähnlich der Ausganssituation bei natürlichsprachlichen Abnahmekriterien. Mit Aktion ist diejenige Aktion gemeint, welche in der Anforderung für die entsprechende Bedingungskombination festgelegt wurde. Es gibt zwei festgelegte Symbole, um die Aktionszeiger zu repräsentieren: (x) (die Aktion sollte ausgeführt werden) und (-) (die Aktion sollte nicht ausgeführt werden). Mit Hilfe dieser festgelegeten Form wird die zu prüfende Testvorschrift und das erwartete Ergebnis beschrieben. An folgendem Beispiel wird deutlich, wie man ein solches formalisiertes Abnahmekriterium aus einer Anforderung ableitet, und zwar zunächst in Form eines abstrakten und dann in der Form eines formalisierten Abnahmekriteriums. Es ist wichtig, dass die Übersichtlichkeit der Tabelle gewährleistet bleibt. Aus diesem Grund ist es nicht sinnvoll, alle möglichen Bedingungen zu testen, sondern es werden nur solche Bedingungen getestet, die fachlich relevant sind. Alle anderen nicht relevanten Bedingungen werden im Schritt der sogenannten Konsolidation zusammengefasst. Ein Beispiel soll das Vorgehen verdeutlichen. Folgende Anforderung sei gegeben:

Anforderung Das Bibliothekssystem soll ausschließlich dem Administrator die Möglichkeit bieten, neue Leihgegenstände von externen Datenmedien in den Datenbestand des Bibliothekssystems importieren zu können. Falls während des Importierens ein semantischer oder syntaktischer Datenfehler auftritt, soll das System dem Administrator für einzelne zu importierende, neue Leihgegenstände den jeweiligen Fehler (Fehlernummer und -beschreibung) auf dem Bildschirm und auf dem Drucker ausgeben.

Die Tabellen 5, 6 und 7 zeigen die entstehenden Abnahmekriterien. Aufgrund der Komplexität formalisierter Abnahmekriterien ist es oft der Fall, dass es äußerlich mehr natürlichsprachliche als formalisierte Abnahmekriterien zu einer Anforderung gibt.

1. Nutzer ist Administrator	j						n					
2. Importieren neuer Daten	j			n			j			n		
3. Fehler aufgetreten	se	sy	n	se	sy	n	se	sy	n	se	sy	n
1. Übertragung möglich	-	-	x	-	-	-	-	-	-	-	-	-
2. Bildschirmausgabe Fehlernummer	x	x	-	-	-	-	-	-	-	-	-	-
2. Bildschirmausgabe Fehlerbeschreibung	x	x	-	-	-	-	-	-	-	-	-	-
2. Druckausgabe Fehlernummer	x	x	-	-	-	-	-	-	-	-	-	-
2. Druckausgabe Fehlerbeschreibung	x	x	-	-	-	-	-	-	-	-	-	-

Legende: j = ja, n = nein, se = semantisch, sy = syntaktisch

Tabelle 5: formalisiert abstraktes Abnahmekriterium 1

1. Nutzer ist Administrator	j		n	
2. Importieren neuer Daten	j		n	-
3. Fehler aufgetreten	se oder sy	n	-	-
1. Übertragung möglich	-	x	-	-
2. Bildschirmausgabe Fehlernummer	x	-	-	-
2. Bildschirmausgabe Fehlerbeschreibung	x	-	-	-
2. Druckausgabe Fehlernummer	x	-	-	-
2. Druckausgabe Fehlerbeschreibung	x	-	-	-

Legende: j = ja, n = nein, se = semantisch, sy = syntaktisch

Tabelle 6: formalisiert abstraktes Abnahmekriterium 2 (nach Konsolidation)

4 Bewertungsgrundlagen für Abnahmekriterien

Es ist sinnvoll, Bewertungsgrundlagen für Abnahmekriterien zu definieren, damit ihre Qualität beurteilt werden kann. Hierzu werden drei Qualitätsmerkmale formuliert:

1. *Testbarkeit*
   Die Testbarkeit setzt sich aus den drei Merkmalen Durchführbarkeit, Messbarkeit und Reproduzierbarkeit zusammen. Das heißt, dass der Test anhand der Abnahmekriterien überhaupt realisierbar sein muß und der Testnutzen in einem vernünftigem Verhältnis zu dem benötigten Aufwand steht. Wenn zum Beispiel ein System laut Anforderung für 1000 Kunden erstellt werden soll, es aber nur für 10 Kunden gedacht ist, so ist zu überdenken, ob sich das Installieren der 1000 Kunden für den Test überhaupt lohnt. Logischerweise ist es auch nicht sinnvoll, Funktionen für Anforderungen zu testen, deren Ausführung nicht sichtbar wird.

1. Nutzer ist	Administrator Aristoteles				Bibliothekar Sokrates oder Bibliothekar Gorgias
2. neue Daten	j		n		-
3. Fehler aufgetreten	se	sy	n	-	-
1. Übertragung möglich	-	-	x	-	-
2. Bildschirmausgabe: Nr. 17, semantischer Fehler	x	-	-	-	-
2. Bildschirmausgabe: Nr. 18, syntaktischer Fehler	-	x	-	-	-
2. Drucker: Nr. 17, semantischer Fehler	x	-	-	-	-
2. Drucker: Nr. 18, syntaktischer Fehler	-	x	-	-	-

Legende: j = ja, n = nein, se = semantisch, sy = syntaktisch

Tabelle 7: formalisiert konkretes Abnahmekriterium 3

Damit in einem erneuten Test nach einer Produktausbesserung die Testergebnisse mit den alten leichter verglichen werden können, müssen Abnahmekriterien reproduzierbar sein. Deshalb dürfen Zufallsgeneratoren im Test nur bei der Initialisierung verwendet werden.

2. *Vollständigkeit*
   Vollständigkeit bedeutet an dieser Stelle, dass mit Hilfe der Abnahmekriterien genau die in den Anforderungen geforderten Funktionalitäten geprüft werden können. Es dürfen folglich keine zusätzlichen Leistungen in den Abnahmekriterien getestet werden und keine Teilanforderungen ausgelassen werden.

3. *Minimalität*
   Die Abnahmekriterien müssen minimal sein, da ansonsten der Testaufwand erheblich steigen würde.

5 Strategien zur Entwicklung von Abnahmekriterien

5.1 Allgemeine Hilfestellungen

Im folgenden werden eine Reihe allgemeiner Ratschläge beschrieben, die beim Suchen und Erstellen von Abnahmekriterien hilfreich sind.

- Abnahmekriterien sollten innerhalb der Anforderungsanalyse verfasst werden. Auf diesem Weg können Anforderungsfehler, die während des Erstellens der

Abnahmekriterien gefunden werden, direkt behoben werden und verursachen keine allzu großen Kosten.

- Abnahmekriterien sollten erst geschrieben werden, wenn die Anforderungen eine gewisse Reife besitzen. Ansonsten würden deren Verbesserungen anhand der Abnahmekriterien zu viel Zeit und Kosten in Anspruch nehmen. Weiterhin ist es sinnvoll, fachlich zusammengehörige Anforderungspakete zu betrachten und als Abnahmekriterien zu verfassen.

- Es sollte genau darauf geachtet werden, alle Aspekte der Anforderung, aber nicht mehr, in den Abnahmekriterien abzubilden.

- Man sollte die Art des Abnahmekriteriums der Art der Anforderung anpassen. Das bedeutet zum Beispiel, dass zu einfach strukturierten Anforderungen keine Testszenarien entworfen werden müssen oder dass im Falle einfach aufgebauter Daten ruhig abstrakte Kriterien erstellt werden können. Falls es sich um komplizierte, verschachtelte Anforderungen handelt, sollten hingegen formalisierte Abnahmekriterien erarbeitet werden. Weiterhin ist es hilfreich, zunächst die funktionalen Anforderungen und erst später die Anforderungen an eine mögliche Benutzeroberfläche zu betrachten .

- Für kritische Anforderungen sollte man eher mehr Abnahmekriterien formulieren, um sicher zu gehen, dass die Korrektheit der beschriebenen Funktionalität getestet wird.

- Abnahmekriterien sollten von Fachleuten geschrieben werden, die nicht die Anforderungen verfasst haben, damit sogenannte Betriebsblindheit ausgeschlossen werden kann. Die Prüfung der Abnahmekriterien wiederum sollte jemand übernehmen, der die Anforderungen mit erstellt hat, damit untersucht werden kann, ob die ursprüngliche Bedeutung der Anforderungen abgebildet wurde.

- Die Abnahmekriterien sollten als Bestandteil in den Vertrag zwischen Arbeitnehmer und Arbeitgeber aufgenommen werden, so dass beide Parteien an einen festgelegten Rahmen für die Tests gebunden sind.

5.2 Konkrete Verfahren

Man findet in der Literatur verschiedene Verfahren zur Testfallermittlung, die in der Praxis hauptsächlich für Modultests verwendet werden. Einige dieser Verfahren, die sogenannten White-Box-Tests, arbeiten strukturorientiert. Das heißt, dass diese den Prüfling unter Berücksichtigung seiner internen Logik und Struktur untersuchen. Der Aufbau von Lösungen ist aber in guten Anforderungen freigestellt, so dass es nicht sinnvoll ist, diesen im Abnahmetest zu prüfen. In diesem Fall ist es besser, den Prüfling und die Anforderung als schwarze Kästen zu betrachten, in welche man nicht blicken kann, so dass nur die Ein- und Ausgaben nach außen sichtbar sind. Einen Test nach diesem ergebnisorientierten Prinzip bezeichnet man als Black-Box-Test. In der Literatur werden hierzu einige konkrete Verfahren beschrieben, die man zum Entwickeln

von Testfällen nutzen kann. Wie man diese zur Ermittlung von guten Abnahmekriterien verwenden kann wird nun im folgenden beschrieben. Falls mehrere Eingaben und Bedingungen die Ausgabe beeinflussen, so müssen alle Kombinationen aus Bedingungen und Eingaben getestet werden. Aus Kostengründen werden veschiedene, sich ergänzende Verfahren benutzt, um die Anzahl der Abnahmekriterien so weit wie möglich zu reduzieren und gleichzeitig weder Abdeckung noch Gültigkeit des Tests einzuschränken. Diese Verfahren werden nun einzeln dargestellt.

1. *Funktionsabdeckung*
   Mit Abnahmekriterien, die mit dieser Methode gefunden wurden, kann man nachweisen, dass der Prüfling die geforderten Funktionen oder Eigenschaften besitzt und dass diese auch ausführbar sind. Es wird hier nur das Normalverhalten des Produktes untersucht, so dass die weiter unten beschriebenen Methoden ergänzend angewandt werden sollten, um auch die Fehlerfälle in Form von Abnahmekriterien zu berücksichtigen. Das folgende Beispiel soll die Methode der Funktionsabdeckung erläutern. Folgende Anforderung sei gegeben:

   Anforderung Das Bibliothekssystem soll dem Nutzer die Möglichkeit bieten, Bibliothekskunden und Leihgegenstände permanent im Bibliothekssystem löschen zu können.

   Es ist also zum einen gefordert, daß man Bibliothekskunden löschen kann, und zum anderen ist gefordert, daß man Leihobjekte löschen kann. Zusätzlich wird gefordert, daß die Löschvorgänge permanent sind. Um nun diese Funktionalitäten abzudecken, benötigt man zwei Abnahmekriterien. Die beiden untenstehenden in den Tabellen 8 und 9 abgebildeten Kriterien leisten dies.

Ausgangssituation	Ein Nutzer ist im Bibliothekssystem angemeldet. Im Bibliothekssystem ist mindestens ein Kunde erfasst.
Ereignis	Ein Nutzer löscht einen erfassten Bibliothekskunden aus dem Bibliothekssystem.
Erwartetes Ergebnis	Nach einem Systemstart befindet sich der gelöschte Bibliothekskunde nich mehr im Bibliothekssystem.

Tabelle 8: Abnahmekriterium 1 nach Funktionsabdeckungsmethode

2. *Äquivalenzklassenbildung*
   Mit Hilfe dieser Methode wird versucht, Eingabewerte, die identisches Verhalten oder identische Eigenschaften des Prüflings verursachen, in Klassen zusammenzufassen. Auf diese Weise können viele Fehler gefunden werden, gleichzeitig müssen aber nur minimal viele Abnahmekriterien erstellt werden. Bei der Bildung der Äquivalenzklassen müssen neben den gültigen Eingabewerten auch ungültige berücksichtigt werden, damit auch das Verhalten für unzulässige Eingaben untersucht werden kann. Häufig ist es auch sinnvoll, Äquivalenzklassen für Ausgabewerte zu bilden.

Ausgangssituation	Ein Nutzer ist im Bibliothekssystem angemeldet. Im Bibliothekssystem ist mindestens ein Leihobjekt erfasst.
Ereignis	Ein Nutzer löscht ein erfasstes Leihobjekt aus dem Bibliothekssystem.
Erwartetes Ergebnis	Nach einem Systemstart befindet sich das gelöschte Leihobjekt nicht mehr im Bibliothekssystem.

Tabelle 9: Abnahmekriterium 2 nach Funktionsabdeckungsmethode

Die Methode der Äquivalenzklassenbildung eignet sich besonders, um Bedingungen und Ausprägungen für formalisierte Abnahmekriterien zu finden. Ein Beispiel soll die Anwendungsweise der Methode an dieser Stelle verdeutlichen.

Anforderung An jedem Dienstag um 8:00 Uhr und unter der Bedingung, dass:

(a) das letztmögliche Rückgabedatum eines Leihobjektes um mehr als eine Woche überschritten wurde, soll das Bibliothekssystem eine erste Mahnung für einen Bibliothekskunden bezüglich eines Leihobjektes drucken;

(b) das letztmögliche Rückgabedatum eines Leihobjektes um mehr als zwei Wochen überschritten wurde, soll das Bibliothekssystem eine zweite Mahnung für einen Bibliothekskunden bezüglich eines Leihobjektes drucken;

(c) das letztmögliche Rückgabedatum eines Leihobjektes um mehr als drei Wochen
überschritten wurde, soll das Bibliothekssystem eine Meldung an den Geschäftsführer bezüglich des Leihobjektes eines Bibliothekskunden erstellen.

Zur Erstellung der Abnahmekriterien analysiert man zunächst die möglichen Eingabewerte, Ausgabewerte und Bedingungen.

Eingabewerte letztmögliches Rückgabedatum eines Leihobjektes

Ausgabewerte erste Mahnung, zweite Mahnung, Meldung an den Geschäftsführer

Bedingungen (a) letztmögliches Rückgabedatum um mehr als eine Woche überschritten : erste Mahnung;

(b) letztmögliches Rückgabedatum um mehr als zwei Wochen überschritten : zweite Mahnung;

(c) letztmögliches Rückgabedatum um mehr als drei Wochen überschritten : Meldung an den Geschäftsführer;

Nun müssen die Wertebereiche in Äquivalenzklassen unterteilt werden. Es wird also für diejenigen Eingabewerte, welche gemäß der Anforderung gleiches Verhalten des Bibliothekssystems verursachen, eine Äquivalenzklasse gebildet.
Dies wird in Tabelle 10 dargestellt. (IRD = letztmögliches Rückgabedatum)

Äquivalenzklasse	Wertebereich
1	$\leq$ IRD + eine Woche
2	$>$ IRD + eine Woche und $\leq$ IRD + zwei Wochen
3	$>$ IRD + zwei Wochen und $\leq$ IRD + drei Wochen
4	$>$ IRD + drei Wochen

Tabelle 10: Äquivalenzklassen der Äquivalenzklassenbildungsmethode

Aus jeder Äquivalenzklasse wird jetzt ein beliebiger Repräsentant gewählt, wie in Tabelle 11 angeführt.

Äquivalenzklasse	Wertebereich
1	zwei Tage vor IRD
2	IRD + 11Tage
3	IRD + 16 Tage
4	IRD + 32 Tage

Tabelle 11: Repräsentanten der Äquivalenzklassenbildungsmethode

Zu jeder Äquivalenzklasse kann nun ein Abnahmekriterium erstellt werden, wobei der jeweilige Repräsentant der Ausgangssituation und das jeweilige geforderte Verhalten dem erwarteten Ereignis entsprechen. Die Tabellen 12, 13, 14 und 15 enthalten die entstehenden Kriterien.

Ausgangssituation	Das letztmögliche Rückgabedatum eines Leihobjektes wird in zwei Tagen erreicht sein.
Ereignis	Es ist Dienstag, 8:00.
Erwartetes Ergebnis	Das Bibliothekssystem druckt keine Mahnung.

Tabelle 12: Abnahmekriterium 1 nach Äquivalenzklassenbildungsmethode

Ausgangssituation	Das letztmögliche Rückgabedatum eines Leihobjektes ist um 11 Tage überschritten.
Ereignis	Es ist Dienstag, 8:00.
Erwartetes Ergebnis	Das Bibliothekssystem druckt für den entsprechenden Bibliothekskunden eine erste Mahnung bezüglich des überfälligen Leihobjektes.

Tabelle 13: Abnahmekriterium 2 nach Äquivalenzklassenbildungsmethode

Ausgangssituation	Das letztmögliche Rückgabedatum eines Leihobjektes ist um 16 Tage überschritten.
Ereignis	Es ist Dienstag, 8:00.
Erwartetes Ergebnis	Das Bibliothekssystem druckt für den entsprechenden Bibliothekskunden eine zweite Mahnung bezüglich des überfälligen Leihobjektes.

Tabelle 14: Abnahmekriterium 3 nach Äquivalenzklassenbildungsmethode

Ausgangssituation	Das letztmögliche Rückgabedatum eines Leihobjektes ist um 32 Tage überschritten.
Ereignis	Es ist Dienstag, 8:00.
Erwartetes Ergebnis	Das Bibliothekssystem erstellt eine Meldung an den Geschäftsführer bezüglich des überfälligen Leihobjektes eines Bibliothekskunden.

Tabelle 15: Abnahmekriterium 4 nach Äquivalenzklassenbildungsmethode

3. *Grenzwertanalyse*
   In diesem Verfahren testet man die Grenzen einer Äquivalenzklasse, da dort erfahrungsgemäß häufig Probleme auftreten. Man wählt hier nicht einen beliebigen Repräsentanten einer Klasse aus sondern solche, die genau an den Grenzen dieser liegen. Anhand des obigen Beispiels wird nun die Methode erläutert.

Äquivalenzklasse	Wertebereich	Werteauswahl
1	$\leq$ IRD + eine Woche	IRD + 7 Tage
2	$>$ IRD + eine Woche und $\leq$ IRD + zwei Wochen	IRD + 8 Tage, IRD + 14 Tage
3	$>$ IRD + zwei Wochen und $\leq$ IRD + drei Wochen	IRD + 15 Tage, IRD + 21 Tage
4	$>$ IRD + drei Wochen	IRD + 22 Tage

Tabelle 16: Äquivalenzklassen/Repräsentanten der Grenzwertanalysemethode

Die in Tabelle 16 dargestellten Werte müssen aus den Klassen ausgewählt werden, um das Verhalten des Bibliothekssystems an den Grenzen der Äquivalenzklassen zu überprüfen. Für jeden ausgewählten Wert wird ein Abnahmekriterium formuliert, wobei auch hier wieder der jeweilige Repräsentant der Ausgangssituation und das jeweilige geforderte Verhalten dem erwarteten Ereignis entsprechen. Die Tabellen 17, 18, 19, 20, 21 und 22 enthalten die jeweiligen Abnahmekriterien.

Ausgangssituation	Das letztmögliche Rückgabedatum eines Leihobjektes ist um 7 Tage überschritten.
Ereignis	Es ist Dienstag, 8:00.
Erwartetes Ergebnis	Das Bibliothekssystem druckt keine Mahnung.

Tabelle 17: Abnahmekriterium 1 nach Grenzwertanalysemethode

Ausgangssituation	Das letztmögliche Rückgabedatum eines Leihobjektes ist um 8 Tage überschritten.
Ereignis	Es ist Dienstag, 8:00.
Erwartetes Ergebnis	Das Bibliothekssystem druckt für den entsprechenden Bibliothekskunden eine erste Mahnung bezüglich des überfälligen Leihobjektes.

Tabelle 18: Abnahmekriterium 2 nach Grenzwertanalysemethode

Ausgangssituation	Das letztmögliche Rückgabedatum eines Leihobjektes ist um 14 Tage überschritten.
Ereignis	Es ist Dienstag, 8:00.
Erwartetes Ergebnis	Das Bibliothekssystem druckt für den entsprechenden Bibliothekskunden eine erste Mahnung bezüglich des überfälligen Leihobjektes.

Tabelle 19: Abnahmekriterium 3 nach Grenzwertanalysemethode

Ausgangssituation	Das letztmögliche Rückgabedatum eines Leihobjektes ist um 15 Tage überschritten.
Ereignis	Es ist Dienstag, 8:00.
Erwartetes Ergebnis	Das Bibliothekssystem druckt für den entsprechenden Bibliothekskunden eine zweite Mahnung bezüglich des überfälligen Leihobjektes.

Tabelle 20: Abnahmekriterium 4 nach Grenzwertanalysemethode

4. *Intuitive Abnahmekriterien-Ermittlung*
   Ziel dieser Methode ist es, die bereits gefundenen Abnahmekriterien um neue zu ergänzen, die erfahrungsgemäß kritische Punkte testen. Solche Punkte können sein: Standardfehler, fehlerverdächtige Situationen oder Funktionen und Teilsysteme, die in der Vergangenheit schon Probleme bereitet haben. Außerdem sollten übliche Fehlerquellen wie z.B. lange oder leere Zeichenketten als Eingabe, ungültige Datumswerte und Richtlinienverletzungen usw. getestet werden.

Ausgangssituation	Das letztmögliche Rückgabedatum eines Leihobjektes ist um 21 Tage überschritten.
Ereignis	Es ist Dienstag, 8:00.
Erwartetes Ergebnis	Das Bibliothekssystem druckt für den entsprechenden Bibliothekskunden eine zweite Mahnung bezüglich des überfälligen Leihobjektes.

Tabelle 21: Abnahmekriterium 5 nach Grenzwertanalysemethode

Ausgangssituation	Das letztmögliche Rückgabedatum eines Leihobjektes ist um 22 Tage überschritten.
Ereignis	Es ist Dienstag, 8:00.
Erwartetes Ergebnis	Das Bibliothekssystem erstellt eine Meldung an den Geschäftsführer bezüglich des überfälligen Leihobjektes eines Bibliothekskunden.

Tabelle 22: Abnahmekriterium 6 nach Grenzwertanalysemethode

Als Beispiel dieser intuitiven Ermittlung von Abnahmekriterien wird nun eine bereits bearbeitete Anforderung um ein zusätzliches Abnahmekriterium ergänzt.

Anforderung Das Bibliothekssystem soll ausschließlich dem Administrator die Möglichkeit bieten, neue Leihgegenstände von externen Datenmedien in den Datenbestand des Bibliothekssystems importieren zu können. Falls während des Importierens ein semantischer oder syntaktischer Datenfehler auftritt, soll das System dem Administrator für einzelne zu importierende, neue Leihgegenstände den jeweiligen Fehler (Fehlernummer und -beschreibung) auf dem Bildschirm und auf dem Drucker ausgeben.

Tabelle 23 zeigt ein entstandenes, zusätzliches Abnahmekriterium zu dieser Anforderung.

Ausgangssituation	Der Administrator *Aristoteles* ist im Bibliothekssystem angemeldet. Auf einem externen Medium sind keine Leihgegenstände verfügbar (nicht nur keine neuen).
Ereignis	Der Administrator *Aristoteles* startet die Übertragung von Daten auf dem externen Medium in das Bibliothekssystem.
Erwartetes Ergebnis	Das Bibliothekssystem führt keine Übertragung durch, gibt keinen Fehler aus und beendet den Vorgang.

Tabelle 23: Abnahmekriterium nach intuitiver Abnahmekriterien-Ermittlung

6 Strategien zur Entwicklung von Testszenarien mit Hilfe von Abnahmekriterien

Nach der Formulierung der Abnahmekriterien sind diese entweder direkt bei den jeweiligen Anforderungen oder als eigenständiges Dokument abgelegt. In einem Test werden aber normalerweise typische Arbeitsabläufe, die noch einmal in kleinere Arbeitsschritte unterteilt sind, getestet. Aus diesem Grund ist die vorliegende Sortierung der Abnahmekriterien für den Test gänzlich ungeeignet. Die Abnahmekriterien müssen noch einmal in der Reihenfolge der jeweiligen Geschäftsvorfälle sortiert werden. Hierzu verwendet man sogenannte Testszenarien, die folgendermaßen definiert sind.

Def.: Ein *Testszenario* ist eine zeitlich geordnete Sammlung von Abnahmekriterien, die als Leitfaden für einen Test herangezogen werden kann. Quelle: [1, S. 323]

In einem solchen Testszenario wird jedem Abnahmekriterium ein eindeutiger Platz in der neuen Sortierung zugeordnet. Es muß angegeben werden, um welches Abnahmekriterium es sich handelt (Nummer und kurze Beschreibung). Weiterhin wird zu jedem Kriterium eingetragen, ob das erwartete Ergebnis eingetroffen ist (OK, nicht OK) und im Fehlerfall noch zusätzlich eine kurze Fehlerbeschreibung mit Angabe des Fehlergrades. Den Fehlergrad bestimmt der Testverantwortliche und legt zudem fest, welche Auswirkung ein Auftreten eines Fehlers eines bestimmten Fehlergrades auf den Testverlauf und auf das Testergebnis hat (z.B. Testabbruch). Weiterhin muß definiert werden, wie viele Fehler eines Fehlergrades auftreten dürfen bis bestimmte Auflegen erfolgen oder bis die Abnahme verweigert wird. Falls das Testszenario in Papierform vorliegt, muß zusätzlich der Tester und das Datum mit Uhrzeit eingetragen werden. Bei Verwendung eines Tools können diese beiden Angaben elektronisch verwaltet werden. Wie ein Testszenario für die durchführbaren Arbeitsschritte mit Videoleihobjekten des Bibliothekssystems aufgebaut sein könnte, zeigt das Beispiel in Tabelle 24.

Es ist wichtig, zu verwalten, wie Anforderungen, Abnahmekriterien und Testszenarien miteinander verknüpft sind, damit eventuelle Änderungen in allen Dokumenten vorgenommen werden. Um im Test nicht für jedes Abnahmekriterium neue Ausgangssituationen erzeugen zu müssen, ist es hilfreich, das erwartete Ergebnis des vorherigen Abnahmekriteriums als Ausgangssituation zu verwenden. Weiterhin ist es häufig der Fall, dass zum Zeitpunkt des Tests immer noch nicht-testbare Anforderungen existieren. Diese sollten dann in Testszenarien eingeordnet werden, so dass man sie dennoch testen kann. Ähnlich wird mit Randbedingungen, bzw. geforderten Nebeneffekten des Produkts verfahren. Diese sogenannten Constraint-Anforderungen werden parallel zu einer funktionalen Anforderung getestet.

7 Zusammenfassung

Das Erstellen von Abnahmekriterien auf Basis der Anforderungen ist eine sinnvolle Methode, um die Qualität eines Produktes oder eines Prozesses zu gewährleisten.

Test-schritt	Ab-nahme-kri-terium	Kurzbeschreibung des Abnahme-kriteriums (Ereignis/ erwartetes Ergebnis)	Ergebnis des Tests (Fehlergrad) und Beschreibung	Datum/ Uhrzeit	Tester
1	331	Video-Leihobjekt erfassen	OK	7.1.2000/ 9.12	AG
2	78	Bezeichnung des Video-Leihobjektes modifizieren	OK	7.1.2000/ 9.13	AG
3	3	Video-Leihobjekt verleihen	nicht OK (3) Das System hat die Eingabe des Leihobjektes verweigert.	7.1.2000/ 9.42	AG
4	577	Video-Leihobjekt mahnen	OK	7.1.2000/ 9.47	AG
5	17	Video-Leihobjekt zurückgegeben	nicht OK (1) Das System vermerkte das zurückgegebene Leihobjekt als noch immer entliehen.	7.1.2000/ 9.51	AG

Tabelle 24: Testszenario: Video-Leihobjekte

Abnahmekriterien werden nach Formalisierungsgrad und Abstraktionsgrad unterschieden, wodurch folgende Typen von Kriterien entstehen:

- natürlichsprachlich und abstrakt
- formalisiert und abstrakt
- natürlichsprachlich und konkret
- formal und konkret

Abnahmekriterien müssen bestimmten qualitativen Merkmalen genügen. Sie müssen testbar, vollständig bezüglich der Anforderungen und minimal sein, damit der Testaufwand in einem vernünftigen Rahmen bleiben kann. Um die Abnahmekriterien zu finden, werden Black-Box- und White-Box-Verfahren genutzt. Zum Auffinden von ergebnisorientierten Abnahmekriterien eignen sich folgende Black-Box-Verfahren: Funktionsabdeckung, Äquivalenzklassenbildung, Grenzwertverfahren und intuitive Abnahmekriterien-Ermittlung. Die Formulierung von Testszenarien eignet sich, um den zeitlichen Ablauf des Produktes oder des Prozesses zu untersuchen. Hierzu werden mehrere Abnahmekriterien in eine Ordnung gebracht, die einem typischen Arbeitsablauf entspricht.

Literatur

[1] Chris Rupp. “Requirements-Engineering und -Management“. Hanser Verlag, 2001.

Zentrale Metriken für Projektleitung und Controlling

Johanna Rauchenberger

Inhaltsverzeichnis

1 Einleitung

Diese Ausarbeitung stellt verschiedene Metriken vor, die besonders zur Planung, Steuerung und Kontrolle auf Ebene von Controlling und Projektleitung für ein bestimmtes Produkt oder einen bestimmten Prozess in der Softwareentwicklung verwendet werden können.

2 Definition von Metriken und ihre zentrale Bedeutung

Eine Metrik definiert, wie eine Kenngröße der Software oder deren Entwicklung, die in der statistischen Auswertung eine quantitative Aussage erlaubt, gemessen wird [Tha00]. Ziel ist es, verschiedene Merkmale eines Softwareprodukts zu messen, d.h. ihnen einen numerischen Wert zuzuordnen. Dadurch ist es möglich Erfahrungen zu quantifizieren, so dass man Sachverhalte, die man ahnt oder auch aus Erfahrung weiß mit Messwerten belegen kann. Zudem lässt sich eine Aussage über den qualitativen Zustand machen, vor allem, wenn in dem Unternehmen bereits Vergleichszahlen aus abgeschlossenen Projekten existieren.

Prozessmetriken sollen zudem insbesondere dazu dienen, den Entwicklungsprozess überschaubar und bewertbar zu machen. Außerdem sollen Probleme, die sich auf den Zeit- und Budgetplan auswirken können, frühzeitig erkannt werden, um mit Gegenmaßnahmen verhindert werden zu können. Dadurch bieten sie dem Controlling und der Projektleitung eine Entscheidungs- bzw. Handlungsgrundlage.

Die vorgestellten Metriken stellen eine Auswahl von in der Praxis verwendeten Metriken dar. Um eine aussagekräftige Beurteilung des Produktes oder Prozesses zu erhalten, müssen Metriken ausgewählt werden, die den speziellen Bedürfnissen des jeweiligen Prozesses oder Produktes innerhalb eines Unternehmens gerecht werden. Zudem ist eine eingehende Analyse der Messergebnisse erforderlich, damit die erhaltenen Daten sinnvoll weiterverwendet werden können und nicht zu Fehlinterpretationen führen.

3 Aufgaben von Projektleitung und Controlling

In diesem Abschnitt werden die wichtigsten Aufgaben sowohl der Projektleitung als auch des Controllings vorgestellt. Das Controlling ist ein Teilbereich der Unternehmensführung, dem die Steuerung und Kontrolle des Unternehmensgeschehens durch die Bereitstellung geeigneter Informationen obliegt [www03]. Das Controlling unterteilt sich in strategisches und funktionales Controlling, Auftrags-, Ressourcen-, Risiko-, Maßnahmen- und Finanzcontrolling.

- Das strategische Controlling befasst sich mit Markt- und Wettbewerbsbeobachtungen und dem Abgleich des Konzeptes mit der Unternehmensrealität.

- Das Auftragscontrolling mit der systematischen Erfassung des Auftragseingangs, um frühzeitig Markt- und Kundenveränderungen zu erkennen.

- Das Ressourcencontrolling versucht, Unterauslastungen bei Mitarbeitern und Maschinen aufzudecken, um Leerlaufkosten und Arbeitsspitzen abzufedern.

- Das Risikocontrolling erfasst und bewertet die zu erwartenden Belastungen aus Gewährleistung, Rechtsstreitigkeiten, Vertragsstrafen etc. Bei Auslandsaktivitäten außerdem Währungs- sowie politische Risiken.

- Das funktionale Controlling dokumentiert und bewertet die Produktivität und Qualität verschiedener Unternehmensfunktionen anhand geeigneter Kennzahlen.

- Das Maßnahmencontrolling verfolgt alle vereinbarten Maßnahmen durch Terminplanung, Benennung von Verantwortlichen und Ergebnisverfolgung und erfasst diese systematisch.

- Das Finanzcontrolling ist für die Bilanz, Gewinn-und-Verlust-Rechnung, Finanzierung und Liquiditätsplanung und -steuerung zuständig, um die Liquidität und die Kapitalbasis festzustellen.

Die hier angeführten Metriken dienen dem funktionalen Controlling, wobei sie Auswirkungen auf das Finanz- und Maßnahmencontrolling haben können.

Die Projektleitung befasst sich mit der Organisation und Durchführung eines einzelnen Projektes und ist verantwortlich für die Aufgabenverteilung innerhalb des Projektteams, die Termineinhaltung, sowie für die Koordination von verschiedenen Mitarbeitern oder Projektteams. Außerdem werden von der Projektleitung Maßnahmen beschlossen, wenn das Controlling Schwachstellen aufgezeigt hat. Diese Maßnahmen werden dann sowohl von der Projektleitung als auch vom Controlling bewertet. Daher ist es sinnvoll, dass eine enge Zusammenarbeit zwischen Controlling und Projektleitung stattfindet.

4 Produktmetriken

Zunächst wird hier eine Auswahl an Produktmetriken vorgestellt, die dazu dienen sollen, einem Softwareprodukt Kenngrößen zuzuordnen und es somit bewertbar und überschaubar, insbesondere für Projektleitung und Controlling, zu machen. Es sollen aufgrund der gesammelten Daten einerseits Aussagen über die Qualität und den Fortschritt des Produkts möglich sein, andererseits Planung und Kontrolle des Zeit- und Budgetplans realistisch ausführbar sein.

4.1 Externer Stressfaktor

Der externe Stressfaktor beschreibt die Verbindung eines Moduls zu seiner Umwelt und lässt sich durch die Formel

$$S_e = e_1 * (inflow * outflow) + e_2 * (fanin * fanout)$$

berechnen, wobei

e_1 und e_2	Wichtungsfaktoren sind,
$inflow$	die Zahl der Daten, die von übergeordneten oder untergeordneten Modulen zu dem betrachteten Modul fließen,
$outflow$	die Zahl der Daten, die von dem betrachteten Modul zu übergeordneten oder untergeordneten Modulen fließen,
$fanin$	die Zahl der Module, die das betrachtete Modul aufrufen können,
$fanout$	die Zahl der Module, die von dem betrachteten Modul aus aufgerufen werden können.

Das Ziel dieser Metrik ist es, den Entwurf des zu entwickelnden Softwareproduktes genau zu untersuchen bevor mit der Implementierung begonnen wird. Nach der Berechnung des externen Stressfaktors sollten alle Module, die vom Durchschnitt abweichen untersucht werden. Unter Umständen ist es sinnvoll ein sehr komplexes Modul aufzuspalten oder den Entwurf nochmal zu überdenken. Auch wenn keine Änderungen vorgenommen werden und die Komplexität des Moduls beibehalten wird, kann man solche Module in den Reviews besonders beachten und die Implementierung eines solchen Moduls einem erfahrenen Programmierer zuweisen. Dabei muss die Projektleitung veranlassen, dass die entsprechenden Mitarbeiter von anderen Aufgaben befreit werden und/oder die komplexen Module wirklich besonders behandelt werden [Tha00].

4.2 Interner Stressfaktor

Die Metrik zur Bestimmung des internen Stressfaktors beruht auf der Auswertung der häufigsten Fehlerursachen. Auswertungen von Fehlern haben gezeigt, dass die Bereiche Aufrufen von Prozeduren, Routinen oder Unterprogrammen, Anweisungen, die komplexe Datenstrukturen manipulieren, und Ein- und Ausgabe jeder Art besonders betroffen sind. Diese Tatsache wird in der Formel

$$S_i = C_1 * (CC) + C_2 * (DSM) + C_3 * (IO)$$

berücksichtigt, wobei

C_1, C_2, C_3 Wichtungsfaktoren sind mit $C_1 = C_3 = 1,0$ und $C_2 = 2,5$,

CC Central Calls, das sind alle Aufrufe anderer Funktionen, Prozeduren oder Routinen mit der Ausnahme von Bibliotheksroutinen,

DSM die Benutzung oder das Referenzieren komplexer Datentypen, z.B. Zeiger, Felder oder Records,

IO die Zahl der Zugriffe auf externe Geräte oder logische Einheiten, also Lese- oder Schreibzugriffe auf Dateien, Ausgabe auf den Bildschirm oder den Drucker sowie Eingaben von der Tastatur.

Ähnlich wie beim externen Stressfaktor sollen auch hier Schwachpunkte oder besonders fehleranfällige Module herausgearbeitet werden. Dazu berechnet man den internen Stressfaktor für jedes Modul des Entwurfs und kennzeichnet die besonders komplexen. Es kann nötig sein, einen Teil des Entwurfs zu verändern oder auch nur die gekennzeichneten besonders zu beachten. Hier liegt der Verantwortungsbereich der Projektleitung, die veranlassen muss, durch Bereitstellung von Zeit, Kapazitäten und Ressourcen, dass eine besondere Betrachtung der gekennzeichneten Module möglich ist [Tha00].

4.3 Produktivität

Diese Kennzahl berechnet sich aus der Anzahl der entwickelten Codezeilen des Produktes, geteilt durch den zur Produktentwicklung nötigen Aufwand, gemessen in Personentagen (LOC/Personentage). Der Aufwand wird von Beginn des Entwurfs bis zur ersten Kundenauslieferung gemessen. Er umfasst alle Entwicklungsaktivitäten, sowie die Aufwände für die Tests, die Reviews, das Projektmanagement, die Programm- und Benutzerdokumentation. Allerdings sollte sich die Projektleitung im Vorfeld überlegen wie wiederverwendete Software berücksichtigt wird. Eine zusätzliche Schwierigkeit bei der Messung der Produktivität ergibt sich aus der Unkenntnis der entwickelten Codezeilen des Produktes in den frühen Phasen des Projektes.

Diese Metrik ist für die Projektleitung und das Controlling besonders interessant, da bei einem Soll-/Istvergleich zu einem Zeitpunkt der Entwicklungsstand des Produktes ziemlich genau einschätzbar ist. Zudem können Daten aus früheren Projekten die Projektleitung und das Controlling während der Planung des Projektes unterstützen, um einen möglichst exakten Zeitplan für die Realisierung des Projektes aufzustellen.

Es ist auch ein Indiz für die Veränderung des Prozesses nach eingeführten Maßnahmen zur Steigerung der Qualität. Dadurch sollte sich die Produktivität erhöhen, auch wenn die Arbeitsintensität umgelagert wird, z.B. von Fehlerbeseitigung auf Tests. Im gesamten dient die Produktivität der Überwachung und Kontrolle der Prozessqualität [KM93].

4.4 Schwere der Fehler

Diese Metrik ordnet die gefundenen Fehler einer Software bezüglich der Schwere der Auswirkungen in Kategorien ein. Schwere der Auswirkung meint dabei die möglichen Folgen bei Auftreten eines Fehlers. Beispielsweise könnte die in Tabelle 1 dargestellte Einteilung in Fehlerklassen vorgenommen werden.

Fehlerklasse	Merkmale des Fehlers
1	Schwerwiegende Fehler, zum Beispiel verbunden mit Systemabsturz
2	Minderschwere Fehler, zum Beispiel Verlust einer Teilfunktion
3	Leichte Fehler wie Tippfehler, unklare Beschreibungen

Tabelle 1: Einteilung in Fehlerklassen

Für die Projektleitung kann diese Metrik eine wichtige Kenngröße über den Fertigstellungsgrad und die Qualität der Software geben. Die Anzahl der Fehler in Klasse 1 sollte in der Regel 10% deutlich unterschreiten. Andernfalls scheint es erhebliche Mängel zu geben, deren Behebung insgesamt sehr aufwendig und teuer werden könnte. Dadurch wiederum könnte es zu einem Nichteinhalten des Budget- und Zeitplans kommen, wodurch unter Umständen das gesamte Projekt gefährdet wäre.

Ein hoher Anteil an Fehlern in Klasse 1 macht zudem den qualitativen Zustand deutlich. Denn viele Fehler, die die Systemfunktion beeinträchtigen, lassen auf eine mäßige Qualität des momentanen Produktes schließen. Auch für den Fertigstellungsgrad ist diese Metrik aussagekräftig, denn wie bereits erwähnt kann die Fehlerbeseitigung von Fehlern besonders der Klasse 1, aber auch der Klasse 2, sehr zeitaufwendig sein und den gesamten Entwicklungsprozess verzögern.

Es ist also Aufgabe der Projektleitung und des Controllings die Fehlerklassen zu beobachten. Sollte es bei einer angemessenen Verteilung der Fehler innerhalb der Klassen zu Zeitdruck im Entwicklungsprozess kommen, kann von der Projektleitung zunächst die Bearbeitung der Fehler aus Klasse 1 und 2 gefordert werden. Denn, obwohl alle Fehler beseitigt werden sollten, ist die Auslieferung der Software an den Kunden eventuell möglich, wenn sich noch einige Schönheitsfehler der Klasse 3 im Produkt befinden.

4.5 Fertigstellungsgrad der Module

Für die Projektleitung und das Controlling bietet es sich an, die Anzahl der fertiggestellten Module genau zu verfolgen. Ein fertiggestelltes Modul ist bereits implementiert, getestet und befindet sich in der Bibliothek der Software. Um einen groben Überblick zu erhalten, kann man zunächst einmal einen Graph erstellen, der die Anzahl über die Zeit repräsentiert. Dazu trägt man auf die x-Achse die Zeit, beispielsweise in Wochen, und auf die y-Achse die absolute Anzahl der fertiggestellten Module. Die erste Kurve zeigt in Abbildung 1 den Idealverlauf und die zweite Kurve

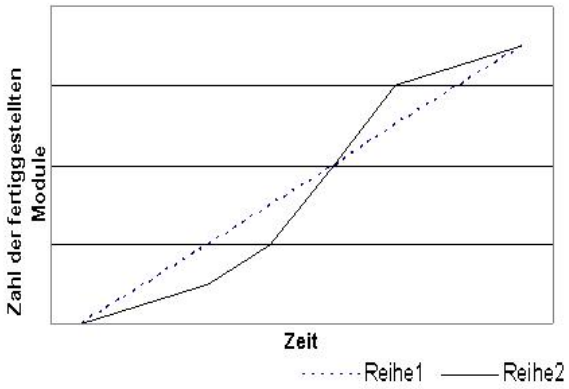

Abbildung 1: Fertiggestellte Module

einen beispielhaften tatsächlichen. Grundsätzlich sollte die tatsächliche Kurve in der Implementierungsphase kontinuierlich ansteigen. Um einen genauen Einblick in den Fertigstellungsgrad zu bekommen, kann man die Sollzahlen mit den Istzahlen zu jedem bestimmten Zeitpunkt vergleichen. Voraussetzung dafür ist selbstverständlich eine genaue Planung durch Projektleitung und Controlling im Vorfeld, um überhaupt aussagekräftige Sollzahlen nach dem Fortschreiten einer bestimmten Zeit vorliegen zu haben.

Drohen bei der Fertigstellung einzelner Module Schwierigkeiten, können rechtzeitig Maßnahmen getroffen werden die Ressourcen- und Zeitplan betreffen, die diesen Verzögerungen entgegenwirken. Vorteil dieser Metrik ist eine frühzeitige Erkennung von Problemen bei der Fertigstellung von Modulen bei stetigem Soll-/Istvergleich und entsprechend guten Daten.

5 Prozessmetriken

Im folgenden Abschnitt werden einige Prozessmetriken beschrieben. Prozessmetriken sollen dazu dienen den Prozess auf lange Sicht zu verbessern und damit auch indirekt die Qualität des Produktes durch einen effizienteren Prozess zu erhöhen. Mit Hilfe der Auswertungen können die Projektleitung und das Controlling erkennen, ob eingeführte Maßnahmen die gewünschten Wirkungen erzielen und somit sinnvoll waren oder sich eher gegenteilig oder neutral ausgewirkt haben, was auch auf eine Fehlinterpretation der Messergebnisse oder voreilige Schlüsse zurückzuführen sein könnte.

5.1 Entwicklung der Fehlermeldungen

Mit der Metrik über die Entwicklung der Fehlermeldungen soll sowohl die Zahl der neuen als auch die der abgearbeiteten Fehlermeldungen kontrolliert werden.

Dazu trägt man auf die x-Achse eines Koordinatensystems die Zeit, zum Beispiel in Wochen, und auf die y-Achse die Anzahl der kumulierten Fehlermeldungen ab. Dabei wird eine Kurve mit den entsprechenden Werten für die „offenen“ Fehlermeldungen und eine weitere für die abgearbeiteten Fehlermeldungen eingetragen.

Im Idealfall folgt die Kurve der abgearbeiteten Fehlermeldungen der Kurve der „offenen“ Fehlermeldungen in einiger zeitlicher Versetzung (Abbildung 2). Kommt es zwischen den Kurven zu einer andauernden Auseinanderdriftung (Abbildung 3), muss die Projektleitung eingreifen, nach den Ursachen suchen und gegebenenfalls Veränderungen des aktuellen Prozesses und/oder dauerhafte Maßnahmen für nachfolgende Projekte treffen.

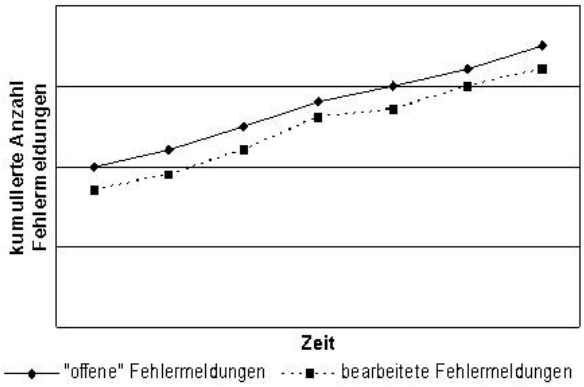

Abbildung 2: Fehlermeldungen über die Zeit

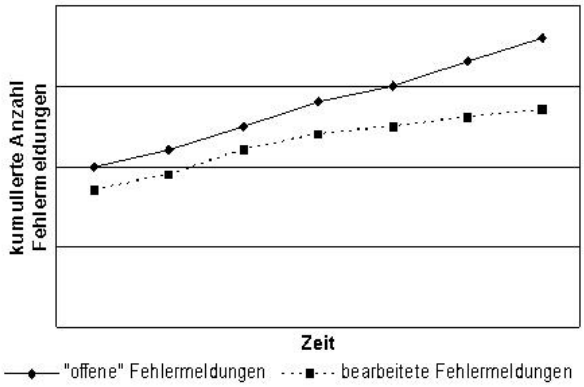

Abbildung 3: Ungünstige Entwicklung der Fehlermeldungen über die Zeit

Ursachen für Auseinanderdriftungen der beiden Kurven können zum Beispiel sehr komplexe Fehler sein, deren Behebung sehr viel Zeit in Anspruch nimmt, mangelnde Kapazitäten oder auch fehlende Kompetenz. Es kann auch Urlaub oder Krankheit von Mitarbeitern der Grund für die schleppende Abarbeitung der Fehlermeldungen sein. Die Projektleitung muss bei der Ursachenforschung viele Möglichkeiten in Betracht ziehen und sollte nicht voreilig handeln.

5.2 Laufender Durchschnitt

Der laufende Durchschnitt bezieht sich auf die Bearbeitungszeit von Fehlermeldungen. Dazu bildet man beispielsweise wöchentlich das arithmetische Mittel über eine bestimmte Anzahl von Fehlermeldungen und trägt dieses in einem Koordinatensystem wie in Abbildung 4 dargestellt ein.

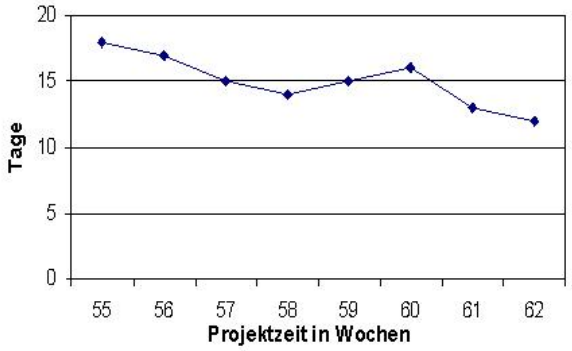

Abbildung 4: Laufender Durchschnitt der Bearbeitungszeit von Fehlermeldungen

Da nur eine festgelegte Anzahl an Fehlermeldungen berücksichtigt wird, fällt eine alte Fehlermeldung raus, sobald eine neue hinzukommt. Anhand des resultierenden Graphen kann man die Entwicklung der Bearbeitung von Fehlermeldungen und die Auswirkung eventuell getroffener Maßnahmen verfolgen. Auf die Dauer der Bearbeitungszeit haben verschiedene Faktoren Einfluss, unter anderem die Projektgröße, die Zahl der unmittelbar in der Softwareentwicklung tätigen Mitarbeiter, die Zahl der Mitarbeiter im Konfigurationsmanagement und der Qualitätssicherung und die technische Schwierigkeit der Aufgabenstellung.

Wie man in Abbildung 4 erkennen kann, zeigt hier der Durchschnitt über die Bearbeitungszeit der Fehlermeldungen eine positive Entwicklung. Sollte dies nicht der Fall sein, so sollte die Projektleitung eventuell gemeinsam mit dem Controlling nach geeigneten Maßnahmen zur Reduktion der Bearbeitungszeit von Fehlermeldungen suchen. Zunächst sollte dabei gründlich nach den Ursachen gesucht werden. Möglich Gründe wären zum Beispiel Krankheit, Urlaub, Unfähigkeit von Mitarbeitern, Fehler an Schnittstellen, für die sich kein Projektteam verantwortlich fühlt oder Zeitmangel. Aufgabe der Projektleitung ist es in diesem Fall durch eine neue Koordination und Organisation des Projektteams veränderte Situationen zu berücksichtigen.

5.3 Fehler pro Phase

Diese Metrik ordnet die Fehler in zwei unterschiedlichen Weisen ein, zum einen nach ihrer Entstehung und zum anderen nach ihrer Entdeckung. Beide Metriken können der Projektleitung und dem Controlling dazu dienen, den gesamten Entwicklungsprozess

von Software zu verbessern. Zunächst soll hier die Einteilung nach der Entstehung oder Einführung von Fehlern untersucht werden. Anschließend dann die Einteilung nach ihrer Entdeckung.

Betrachtet man nun zunächst die Entstehung, so wird versucht jeden gefundenen Fehler in die entsprechende Projektphase, zum Beispiel Spezifikation, Dokumentation, Entwurf und Kodierung, einzuordnen. Ist der prozentuale Anteil einer Phase sehr hoch oder hat sich im Vergleich zu anderen Projekten verschlechtert oder verbessert, lassen sich Rückschlüsse auf die Qualität des Prozesses ziehen.

Für die Projektleitung und das Controlling dienen diese Zahlen als Maßstab, um beispielsweise den Entwicklungsprozess zu verändern und um diese Veränderung zu überwachen. So können in einzelnen Phasen zusätzliche Maßnahmen, wie Reviews, Inspections, Tests oder Prototyping eingeführt werden. Dabei dienen dann wiederum Zahlen aus Projekten nach der Einführung dieser zusätzlichen Maßnahmen als Kontrolle für die Wirksamkeit. Auch können Ressourcen in kritischen Phasen gezielt eingesetzt werden, was allerdings ebenfalls von der Projektleitung geplant und organisiert werden muss. Alle zusätzlichen oder veränderten Maßnahmen müssen im Vorfeld von Projektleitung und Controlling sinnvoll in den gesamten Entwicklungsprozess miteinbezogen werden, damit es nicht zu unerwarteten Engpässen im Budget-, Zeit- oder Ressourcenplan kommt.

Mit den Zahlen der Entstehung eines Fehlers korrelieren die Zahlen der Entdeckung eines Fehlers. Wird ein Großteil der Fehler erst in der Integration oder im Systemtest gefunden, sollten Projektleitung und Controlling die Wirksamkeit der Modultests oder vergleichbarer Tests im Entwicklungsprozess genauer untersuchen. Generell gilt, je später im Prozess ein Fehler in der Software gefunden wird, desto aufwendiger und teurer ist seine Behebung. Durch gezielte Betonung von zum Beispiel Modultests und dem Einsatz von zusätzlichen Ressourcen oder z.B. Mitarbeiterschulungen, ließen sich die Fehlerbeseitigungskosten gegebenenfalls senken, auch wenn dafür an anderer Stelle im Entwicklungsprozess zusätzliche Mittel zur Verfügung gestellt werden müssten. Allerdings muss insbesondere die Projektleitung ausreichend Zeit und Kapazitäten für z.B. umfangreiche Modultests oder weitere Reviews oder Inspections zur Verfügung stellen.

Die durch die Fehler pro Phase-Metrik gewonnen Daten lassen in Projekten, die nach Einführung von Veränderungen im Entwicklungsprozess im Vergleich zu den Daten vergangener Projekte Rückschlüsse und eine genaue Analyse der Wirksamkeit dieser Veränderungen zu. Die beiden Metriken über die Fehler per Phase mit der entsprechenden Einteilung nach Entstehung und Entdeckung des Fehlers geben einen Einblick in den Qualitätszustand der einzelnen Projektphasen und der Wirksamkeit von Veränderungen.

6 Zusammenfassung

Insgesamt kann man also erkennen, dass Metriken die Möglichkeit bieten den Entwicklungsprozess eines Softwareproduktes zu messen und somit zu beurteilen.

Für das Controlling bieten sich solche Kennzahlen zur Überwachung und Dokumentation eines Prozesses an, anhand derer die Notwendigkeit von Veränderungen erkannt und diese Veränderungen kontrolliert werden können. Wichtig dabei ist die kontinuierliche Datenerfassung, um einen aktuellen Überblick über das Projekt und den Prozess zu erhalten. Für die Projektleitung dienen diese Metriken ebenfalls zur Überwachung des Fortschritts und der Qualität des Produktes. Sie ist außerdem dafür zuständig, Ursachen für eventuelle Missstände zu finden und diese zu beseitigen. Dadurch kann ein termingerechtes und qualitativ hochwertiges Produkt entwickelt werden. Zudem kann der Entwicklungsprozess kontinuierlich verbessert und überwacht werden, so dass qualitativ gute Software Zufallsprodukt bleibt.

Literatur

[KM93] D.J.Paulish K.H. Möller. *Software-Metriken in der Praxis*. R.Oldenburg Verlag, 1993.

[Som01] I. Sommerville. *Software Engineering*. Addison Wesley, 2001.

[Tha00] G. E. Thaller. *Software-Metriken einsetzen, bewerten, messen*. Technik Verlag, 2000.

[www03] www.wissen.de. *Controlling*. www.wissen.de, 2003.