

**DIPLOMARBEIT**

# **Qualitätssicherung im Rahmen verteilter Softwareentwicklung**

**von**

**Rafael Sánchez-Moreno Ramírez**

**Eingereicht am 20. April 2007 beim  
Institut für Angewandte Informatik  
und Formale Beschreibungsverfahren (AIFB)  
der Universität Karlsruhe (TH)**

**Referent: Prof. Dr. A. Oberweis  
Koreferent: Prof. Dr. W. Stucky  
Betreuerin: Dipl.-Inform.-Wirt. Stefanie Betz**

**Heimatanschrift:  
Luisenstr. 3  
76137 Karlsruhe  
Rafael.Sanchez-Moreno@gmx.net**

**Studienanschrift:  
Luisenstr. 3  
76137 Karlsruhe**



**Eidesstattliche Erklärung:**

Hiermit versichere ich an Eides Statt, dass ich die vorliegende Diplomarbeit ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt habe. Ich habe alle Stellen, die ich aus den Quellen wörtlich oder inhaltlich entnommen habe, als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Karlsruhe, den 20. April 2007

---

**(Rafael Sánchez-Moreno Ramírez)**







# Inhaltsverzeichnis

<b>1</b>	<b>EINLEITUNG</b>	<b>1</b>
1.1	Motivation	2
1.2	Begriffsdefinitionen	3
1.3	Szenarien und Risiken	5
1.4	Ziel und Überblick dieser Arbeit	7
<b>2</b>	<b>STATE-OF-THE-ART</b>	<b>9</b>
2.1	ISO 9126 Norm	10
2.1.1	Funktionalität	12
2.1.2	Zuverlässigkeit	13
2.1.3	Benutzbarkeit	14
2.1.4	Effizienz	15
2.1.5	Änderbarkeit	16
2.1.6	Übertragbarkeit	17
2.2	Verbesserung der Softwareentwicklung	17
2.2.1	CMMI	18
2.2.2	PSP / TSP: Eine Methode zur Kontinuierlichen Verbesserung	21
2.2.3	ITIL: Der Best-Practice-Ansatz	24
2.2.4	Service Level Agreement (SLA)	26
2.2.5	ISO 9000-Normenfamilie	26
2.3	Softwaremetriken	27
2.3.1	Klassische Metriken	28
2.3.2	Objektorientierte Metriken	29
2.4	Testaktivitäten	30
2.5	Analysewerkzeuge	33
2.5.1	CAST	34
2.5.2	SISSy	35
2.5.3	Sotograph	36
2.6	Produkt- und Prozessverbesserung	37

<b>3</b>	<b>PROZESSRAHMENWERKE IN DER VERTEILTEN ENTWICKLUNG</b>	<b>39</b>
3.1	Neue KPAs für CMMI	39
3.2	ITIL - Der Best-Practice-Ansatz	43
3.3	Zusammenspiel der Prozessrahmenwerke	44
<b>4</b>	<b>VORGEHENSMODELLE UND METHODEN FÜR DIE SOFTWAREENTWICKLUNG</b>	<b>46</b>
4.1	Phasen der Softwareentwicklung	47
4.2	Kurze Geschichte der Vorgehensmodelle	48
4.3	Planungsgetriebene Entwicklung	49
4.3.1	Das Wasserfallmodell	50
4.3.2	Das Spiralmodell	52
4.4	Agile Methoden	54
4.4.1	Extreme Programming (XP)	56
4.5	Entscheidungsmodell von Boehm	58
4.5.1	Projektgröße	60
4.5.2	Sicherheitsrelevante Bedingungen	61
4.5.3	Softwareentwicklungskultur	63
4.5.4	Dynamik	64
4.5.5	Personal	65
<b>5</b>	<b>ENTSCHEIDUNGSMODELL FÜR VERTEILTE PROJEKTE</b>	<b>68</b>
5.1	Angepasste Faktoren	71
5.1.1	Größe	71
5.1.2	Bedingungen für die Qualitätssicherung	73
5.1.3	Kultur der verteilten Teams	75
5.1.4	Dynamik	78
5.1.5	Personal	82
5.2	Organisationsstruktur für Entwicklerteams	88
5.3	Teamaufteilung	90
5.3.1	Teamaufteilung nach den Entwicklungsphasen	91
5.3.2	Teamaufteilung nach der Softwarearchitektur	92
5.4	Zwischen Agilität und Disziplin	94

<b>6</b>	<b>KOLLABORATIVE ENTWICKLUNGSPLATTFORM (CDE)</b>	<b>97</b>
6.1	Teams	98
6.2	Fachwissen und Kommunikation	100
6.3	Reibungspunkte	103
6.4	Aktuelle CDEs	105
6.4.1	MILOS	105
6.4.2	GotDotNet	108
6.4.3	SourceForge.net und SourceForge® Enterprise Edition	110
6.4.4	CollabNet Enterprise Edition	115
6.4.5	CodeBeamer	118
6.5	CDE-Vergleich	120
6.6	„Ideale“ CDE	126
<b>7</b>	<b>KOORDINATION DER TESTAKTIVITÄTEN</b>	<b>129</b>
7.1	Geheimhaltung und Datengröße	130
7.2	Integration	134
7.3	Inspektionen	136
7.3.1	IBIS	138
7.4	Zusammenspiel der Koordinierungsaktivitäten	140
<b>8</b>	<b>VORGEHENSENTWURF BEI VERTEILTEN PROJEKTEN</b>	<b>143</b>
8.1	Die Projektbeispiele	143
8.2	Arbeitsweise der Teams	144
8.3	Analyse der angepassten Faktoren	145
8.4	Vorgehensmodell	148
8.4.1	Prozessrahmenwerk	148
8.4.2	Größe	149
8.4.3	Bedingungen für die Qualitätssicherung	150
8.4.4	Kultur der verteilten Teams	150
8.4.5	Dynamik	153
8.4.6	Personal	153
8.5	CDE	154

<b>9</b>	<b>ZUSAMMENFASSUNG UND AUSBLICK</b>	<b>156</b>
<b>10</b>	<b>LITERATURVERZEICHNIS</b>	<b>161</b>

## Abbildungsverzeichnis

Abbildung 1. Entstehung des Outsourcing-Begriffes [DeBa04] .....	3
Abbildung 2. Informationsfluss in verteilter Softwareentwicklung [SeCS06] .....	5
Abbildung 3. Merkmale und Teilmerkmale des Standards ISO 9126 [Balz01].....	11
Abbildung 4. Reife- und Fähigkeitsgrade in CMMI [Kneu03].....	19
Abbildung 5. Zusammenspiel von CMMI, TSP und PSP() .....	22
Abbildung 6. TSP und PSP erlauben eine ständige Prozessverbesserung().....	23
Abbildung 7. ITIL-Prozesse [BMC06].....	25
Abbildung 8. Die Governance Dashboard von CAST. ....	34
Abbildung 9. Architektur von SISSy .....	36
Abbildung 10. Sotograph-Familie.....	37
Abbildung 11. 24 Neue KPAs und die vier Konzepte [RaKK05] .....	42
Abbildung 12. Zusammenspiel der vorgestellten Prozessrahmenwerke .....	45
Abbildung 13. Wasserfallmodell nach Royce [Royc70] .....	51
Abbildung 14. Spiralmodell [Boeh88].....	53
Abbildung 15. Abgrenzbereiche für Vorgehensmodelle [BoTu03].....	59
Abbildung 16. Polardiagramm von Boehm und Turner [BoTu03].....	60
Abbildung 17. Agile Methoden passen sich besser an kleine Teams [BoTu03].....	61
Abbildung 18. Planungsgetriebene Methoden sind besser für kritische Projekte. Nach [BoTu03] .....	62
Abbildung 19. Projekte befinden sich oft „irgendwo dazwischen“. Nach [BoTu03] .....	63
Abbildung 20. Agile Methoden passen sich sowohl an stabile als auch an hoch dynamische Umgebungen an. Nach [BoTu03] .....	64

## Qualitätssicherung im Rahmen verteilter Softwareentwicklung

---

Abbildung 21. Agile Methoden benötigen eine gute Personalkombination. Nach [BoTu03].....	66
Abbildung 22. Mögliche Bereiche der Risiken eines verteilten Projektes [Karo98].....	70
Abbildung 23. Kommunikationswege in hoch dynamischen Teams [Karo98].....	75
Abbildung 24. Organisation der verteilten Entwicklung [Karo98] .....	91
Abbildung 25. Teamaufteilung nach Entwicklungsphasen [Karo98] .....	92
Abbildung 26. Teamaufteilung nach der Softwarearchitektur [Karo98].....	93
Abbildung 27. Aufgabenverteilung eines Entwicklers [BoBr02].....	98
Abbildung 28. Informelles und formelles Wissen [SeCS06].....	100
Abbildung 29. Fachwissen ist von den Teilnehmern abhängig [SeCS06].....	101
Abbildung 30. Effektivität der Kommunikationsmethoden [Karo98] .....	102
Abbildung 31. Reibungspunkte [BoBr02].....	103
Abbildung 32. Milos Applikationsmodell [RiMa03].....	106
Abbildung 33. Beispiel eines Arbeitsbereiches in GotDotNet.....	109
Abbildung 34. Die SourceForge.net Entwicklungsplattform .....	111
Abbildung 35. Ausschnitt eines Softwareentwicklungsprojektes in SFEE .....	112
Abbildung 36. Architektur von CEE .....	116
Abbildung 37. Projektarbeitsbereich im CollabNet Enterprise Edition (CNEE).....	117
Abbildung 38. CodeBeamer Architektur .....	119
Abbildung 39. Übersicht CodeBeamer .....	120
Abbildung 40. Bereiche einer CDE ([BoBr02], [Four01]).....	126
Abbildung 41. Eigenschaften einer CDE [BoBr02] .....	127
Abbildung 42. Komponenten eines Sicherheitsfiltersystems [WuWZ03] .....	131
Abbildung 43. Komponenten des Ansatzes von [WuWZ03].....	132

Abbildung 44. Inkrementelle Integration [Somm01].....	135
Abbildung 45. Der IBIS-Prozess im Vergleich mit traditionellen Inspektionen [LaMa03] .....	139
Abbildung 46. Zusammenspiel der Koordinierungsaktivitäten.....	142
Abbildung 47. Analyse der kritischen Punkte von AG1 und AG2 .....	147

## Tabellenverzeichnis

Tabelle 1. Zahl der CMMI-Überprüfungen („ <i>Appraisals</i> “). [Ju06] .....	40
Tabelle 2. Faktor Größe des angepassten Entscheidungsmodells.....	73
Tabelle 3. Bedingungen für die Qualitätssicherung des Entscheidungsmodells.....	74
Tabelle 4. Kultur der verteilten Teams im angepassten Entscheidungsmodell.....	78
Tabelle 5. Eigenschaften der Achse Dynamik im angepassten Entscheidungsmodell.....	82
Tabelle 6. Angepasste Personaleigenschaften für das Entscheidungsmodell.....	87
Tabelle 7. Angepasste Faktoren Eigenschaften der Softwareentwicklung für verteilte Projekte .....	87
Tabelle 8. Organisationsstrukturen für Entwicklerteams .....	90
Tabelle 9. Vergleich zwischen SourceForge.net und SFEE .....	114
Tabelle 10. Vergleich der CDEs .....	125
Tabelle 11. Zusammenfassung der Applikationseigenschaften.....	144
Tabelle 12. Prozessverbesserungen der Softwareentwicklungsprojekte.....	155

## 1 Einleitung

Es existiert heute kaum noch ein elektronisches Gerät, das ohne Software auskommt. Die Software wird immer umfangreicher und ausgefeilter. Während zum Beispiel die Bahnverfolgungssoftware für Satelliten der NASA in den Sechzigerjahren circa 220.000 Quellcodezeilen (LOC) besaß, bestand Microsoft Windows NT, eine Anwendung der Neunzigerjahre, aus über 30 Millionen LOC [Thal00]. In den letzten zwei Jahrzehnten haben sich die Anforderungen an die Software und an die Softwareentwicklung stark verändert. Nicht nur, dass die Software immer umfangreicher wird, sondern sie muss durch die so genannte New Economy, die Weiterentwicklung der neuen Technologien, die rasche Entwicklung des Internets als globales Kommunikationsmedium, sowie durch die Globalisierung der Märkte immer schneller fertig gestellt werden können, um überhaupt noch eine Chance im laufenden Wettbewerb zu haben.

Kodak war eine der ersten Firmen weltweit, die sich diesen neuen Anforderungen mit Hilfe verteilter Entwicklung stellte. Sie entschied 1989 einige Geschäftsbereiche nicht mehr intern abzuwickeln, sondern diese auszulagern [LHKP03]. Für die Softwareindustrie hat eine Studie der ITAA<sup>(1)</sup> deutlich gemacht, dass die Auslagerung der Softwareentwicklung, obwohl sie kurzfristig gesehen eine negative Wirkung auf die Wirtschaft zeigt, eine langfristig positive Auswirkung auf die Unternehmen, auf die Beschäftigungsquote des Marktes und vor allem auf die Wirtschaft besitzt [Kola04].

Bedingt durch den größeren Umfang der Software und die potentiellen Möglichkeiten eines globalisierten Marktes, die heute andere Anforderungen an die Softwareindustrie stellen als noch vor zwei Jahrzehnten, muss das Produkt eine gute Qualitätssicherung aufweisen können ([HeMo01], [LHKP03], [Scha06a]). Als die beiden zentralen Begriffe stellen sich die Softwarequalität und die Qualitätssicherung heraus. Unter dem Begriff der Qualitätssicherung werden alle geplanten und systematischen Maßnahmen und Tätigkeiten verstanden, die sicher stellen, dass das Produkt die Qualitätsanforderungen

---

<sup>1</sup> ITAA: Information Technology Association of America  
Quelle: <http://www.ityaa.org/itserv/docs/execsumm.pdf> (Abruf am 10.04.2007)

erfüllt [Balz01]. Die Softwarequalität wird als die Gesamtheit der Merkmale und Merkmalswerte eines Softwareproduktes definiert, die sich auf dessen Eignung beziehen und bestimmte vorher festgelegte oder vorausgesetzte Erfordernisse erfüllen [ISO9126].

### 1.1 Motivation

Über mehrere Jahre hinweg wurde die Softwareentwicklung nur als eine in-house Aktivität betrachtet [LiLL06]. Die Softwareindustrie, genauso wie die Autoindustrie in den Sechzigern, strebt heutzutage allerdings nach der Spezialisierung und Aufsplitterung von Prozessen und Produkten, um die Qualität der Software weiter zu steigern und die Entwicklungszeiten zu beschleunigen [Kola04]. Die Softwareentwicklung ist durch die steigenden Lohnkosten in den Industrieländern mit zunehmend höheren Ausgaben verbunden. Dagegen zeigt die fortschreitende Entwicklung der vergangenen Jahre in den Niedriglohnländern beziehungsweise in den Entwicklungsländern, dass auch mit wenig Kostenaufwand hochwertige Produkte entwickelt werden können [Beck05]. Dabei spielen nicht nur Lohnkosten eine wichtige Rolle, sondern auch die Verfügbarkeit von qualifiziertem Personal und der Zugang zum Internet ([CaAb06], [SeCS06], [HeMo01]).

Einige Faktoren, die dazu beigetragen haben diese Veränderungen festlegen, sind [HeMo01]:

- Die Möglichkeit eine weltweit, günstige und wettbewerbsfähige Software zu produzieren.
- Die potentiellen Geschäftsvorteile durch die Bindung an neue Märkte.
- Die Marktvorteile durch schnelle Bildung von virtuellen Teams.
- Die Möglichkeit einer Softwareentwicklung „rund-um-die-Uhr“, um die Zeit von der Produktentwicklung bis zur Einführung des Produktes auf dem Markt zu reduzieren.

- Die Bereitschaft flexibel genug zu sein, um sich mit anderen Organisationen zusammenzuschließen und so potentielle Marktchancen optimal zu nutzen.

Das Ergebnis ist ein Softwareentwicklungsprozess, der neuerdings an mehreren Standorten geographisch verteilt stattfinden und mehrere Kulturen mit einbeziehen kann. Einen entscheidenden Schritt in diese Richtung hat die Open Source Community bereits erfolgreich durchgeführt [Matl05] und damit bewiesen, dass (Hobby-) Entwickler erfolgreich in einem geographisch verteilten Projekt zusammenarbeiten können.

## 1.2 Begriffsdefinitionen

Das Wort Outsourcing ist ein

*„Kunstwort aus den englischen Wörtern „Outside“, „Resource“ und „Using“, das ganz allgemein die langfristige beziehungsweise endgültige Vergabe von Leistungen an externe Anbieter beschreibt, die bisher selbst erstellt wurden“ [DeBa04] (siehe Abbildung 1)*

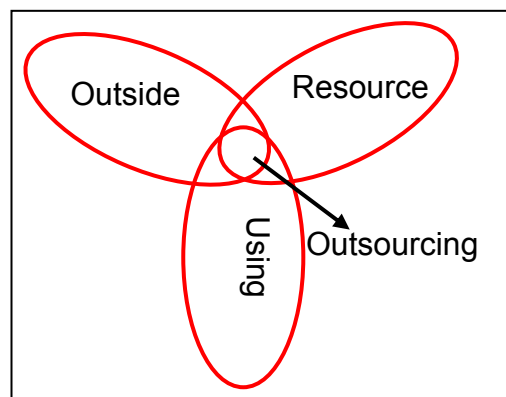


Abbildung 1. Entstehung des Outsourcing-Begriffes [DeBa04]

Outsourcing verfolgt das Ziel Prozesse und Funktionen (Kompetenzen) spezifischer Firmen auszulagern. Diese Auslagerung von Dienstleistungen hat drei geographische Ausprägungen ([CaAb06], [SeCS06], [HeMo01]):

- Onshore Outsourcing: Die Dienstleistungen werden von einer externen Firma jedoch innerhalb des Landes erbracht.
- Nearshore Outsourcing: Die Dienstleistungen werden von einer externen Firma in nächster Nähe des eigenen Marktes erbracht.
- Offshore Outsourcing: Die Dienstleistungen werden von einer externen Firma außerhalb des Landes erbracht.

Es gibt mehrere Begriffe, die die Auslagerung der Softwareentwicklung definieren. Im Englischen wird diese Auslagerung öfters als „Global Software Development“ (GSD) oder „Distributed Software Development“ bezeichnet, während im Deutschen der Begriff „Verteilte Softwareentwicklung“ benutzt wird. Alle Begriffe definieren dieselbe Tätigkeit, nämlich:

*“... [the] Software Engineering that involves the combined efforts of software professionals in different locations separated by significant distances” [TGSC06]*

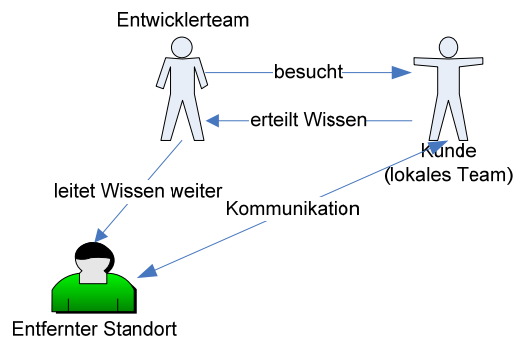
Ein verteiltes Team wird definiert als [HuOc06]:

*„... collection of geographically, organizationally and/or time dispersed individuals who work interdependently, primarily using information and communication technologies, in order to accomplish a shared task or goal. “*

Das Angebot und die Nachfrage technischer Ressourcen und die wachsenden und globalisierten Geschäftsmöglichkeiten machen die Verteilung der Softwareentwicklung somit unvermeidbar, um zuverlässige, effektive und billige Software zu produzieren [Karo98].

### 1.3 Szenarien und Risiken

Ein typisches Szenario für die verteilte Softwareentwicklung, unabhängig von der Art der Auslagerung, wird in Abbildung 2 dargestellt [SeCS06]. Eine Softwarefirma beschließt ihr Softwareprojekt teilweise oder vollständig verteilt zu entwickeln. Dazu wählt diese Firma, in der Abbildung als Kunde dargestellt, einen Partner (entfernter Standort) für die Zusammenarbeit aus. Eine ausgewählte Gruppe (Entwicklerteam) besucht in einer ersten Phase, in der das Team das nötige Fachwissen über das Projekt erlangt, den Kunden vor Ort und zusammen werden die Spezifikationen und Anforderungen definieren. Zusätzlich werden Protokolle und Richtlinien definiert, in denen die Art der Kommunikation und der Entwicklung gelegt werden und darüber gesprochen wird, wie diese zustande kommen sollen.



**Abbildung 2. Informationsfluss in verteilter Softwareentwicklung [SeCS06]**

Nachdem das nötige Fach- und Applikationswissen erlangt wurde, kann den Projektstart ins Leben gerufen werden. Das im Zusammentreffen erworbene Fachwissen unterstützt nun die Gruppe beim Lösen der Aufgaben, was auch dazu dient die Produktivität der Projektteilnehmer zu fördern. Diese Iterationen werden im Laufe der Zeit regelmäßig wiederholt, um das Projektwissen auf dem aktuellen Stand zu halten bis die fertige Software vom Kunden abgenommen wird. Einige Vorteile, die die verteilte Softwareentwicklung mit sich bringt, sind [McCo01]:

- Spezialisierungsmöglichkeiten: Firmen entscheiden sich heutzutage fertige Lösungen zu kaufen, bevor sie die Software selbst entwickeln müssen. Softwarehäuser können sich auf bestimmte Felder der Softwareentwicklung spezialisieren und sich dabei strategische Vorteile verschaffen.
- Mehr Flexibilität bei der Personalbesetzung: Die verteilten Teams können dynamisch zusammengestellt werden.
- Risiken minimieren: Wenn die Applikationsumgebung neu ist oder die Erfahrung in einer bestimmten Technologie nicht groß genug ist, können Risiken minimiert werden, indem die Software von anderen Firmen entwickelt wird.

Wie oben schon erwähnt, hat die Open Source Community vor einigen Jahren die ersten erfolgreichen Schritte in Richtung verteilter Softwareentwicklung unternommen und im Laufe der Jahre bewiesen, dass Barrieren wie Geographie und Kultur überwunden werden können, um gute Software liefern zu können. Nichts desto trotz spielen die Faktoren Kosten und Zeit in einem proprietären Projekt eine wichtigere Rolle als in Open Source Projekte.

Eine Zusammenarbeit geographisch verteilter Teams bedeutet nicht automatisch, dass die Effektivität und Produktivität gesteigert wird [LiLL06]. Die potentiellen Probleme, die in solchen Partnerschaften existieren, sind im Prinzip unabhängig davon ob es sich über Onshore, Nearshore oder Offshore Partnerschaft handelt. Es existieren drei große Bereiche, die bei jedem verteilten Projekt zu Problemen hinführen können ([SeCS06], [Scha06a], [Scha06b], [CaAb06], [CaAb06], [HeMo01], [BCKS01]):

- Kulturelle Unterschiede
- Verschiedene Zeitzonen
- Kommunikationsprobleme

Der ausgewählte Partner kann einem anderen Kulturkreis angehören und sich auf einem anderen Kontinent befinden, wo sich zum Beispiel die Zeitunterschiede als problematisch für die Kommunikation herausstellen können. Eine gute Koordination dieser Bedingungen entscheidet über Erfolg

oder Misserfolg eines Projektes. Ein gut koordiniertes Projekt ist der Schlüssel zum Erfolg. Die Koordination beschäftigt sich mit dem Zusammenspiel zwischen verteilten Teammitgliedern, Prozessen, Informationen und Technologien [Wire06].

## **1.4 Ziel und Überblick dieser Arbeit**

Diese Diplomarbeit verfolgt den Ansatz, dass die Produktqualität eng an die Prozessverbesserung gekoppelt ist ([Balz01], [HeGr99a], [HeGr99b], [SeCS06]). Um zu beschreiben wie die Qualität der Prozesse der Softwareentwicklung in verteilten Projekten verbessert werden kann, orientiert sich die Diplomarbeit in erster Linie an den aufgeführten Punkten eines von Sengupta, Chandra und Sinha [SeCS06] veröffentlichten Artikels, nämlich die Prozesse und die Metriken der verteilten Entwicklung, die Koordination der Testaktivitäten und die Kollaborative Entwicklungsplattform und Wissensverteilung. Zudem stellt diese Diplomarbeit eine Reihe von Maßnahmen vor, die die Koordinierungsarbeiten in verteilten Projekten unterstützen, dabei aber die jedem Entwicklerteam eigenen Techniken und Gewohnheiten beachtet.

Das folgende Kapitel 2 beschäftigt sich mit dem Thema der Qualitätssicherung. Aktuelle Qualitäts- und Prozessrahmenwerke werden vorgestellt und die Unterschiede zwischen einer Produkt- und Prozessverbesserung aufgezeigt. Das Kapitel macht deutlich, warum die Prozessverbesserung auch in verteilten Projekten eine wichtige Rolle spielt. Im Kapitel 3 werden die im zweiten Kapitel vorgestellten Prozessrahmenwerke (wie das CMMI oder ITIL) an die verteilte Umgebung angepasst. Kapitel 4 handelt von aktuellen Vorgehensmodelle und Methoden der Softwareentwicklung, nämlich der Planungsgetriebenen Softwareentwicklung und den Agilen Methoden. Zudem wird ein Entscheidungsmodell, das zur besseren Unterstützung der Entwicklungsarbeiten und der Entwickler dient, vorgestellt. Dieses Entscheidungsmodell wurde von Boehm und Turner (BoTu03) entwickelt und versucht mit Hilfe von fünf Faktoren eine ausgeglichene Mischung zwischen Agilität und Disziplin zu

finden. Dieses Entscheidungsmodell wird in Kapitel 5 an die verteilte Softwareentwicklung angepasst, indem neue Erkenntnisse aus der verteilten Umgebung hinzugefügt werden.

Lösungsansätze für Kollaborative Entwicklungsplattformen (CDE), die die Zusammenarbeit verteilter Teams gegenüber lokalen Teams besser koordinieren, werden in Kapitel 6 aufgezeigt. Die Weiterleitung von Fachwissen unter den Teammitgliedern spielt dabei eine wichtige Rolle, um den Wissenstransfer zwischen geographisch getrennten Mitgliedern zu ermöglichen. Obwohl die Testaktivitäten der verteilten und der lokalen Projekten sich nicht auffallend unterscheiden [SeCS06], sind mit einer verteilten Umgebung automatisch andere Probleme gegeben, die Koordinationsmaßnahmen der Testaktivitäten erfordern. Diese Probleme werden im Kapitel 7 behandelt, wo Lösungen für den Testdatenschutz, -größe und -integration ausgearbeitet werden. Zusätzlich werden Lösungen für verteilte Inspektionen vorgestellt, um die während des Prozesses der Softwareentwicklung entstandenen Fehler aufzudecken und dadurch lästige Korrekturarbeiten in späteren Phasen zu minimieren.

Alle in dieser Diplomarbeit aufgeführten Vorschläge laufen auf das Ziel hinaus die Verbesserung der Prozessqualität zu erreichen und dienen sowohl für komplett neue Softwareprojekte als auch für Projekte, bei denen der Quellcode schon vorhanden ist. Um das mögliche Zusammenspiel dieser Vorschläge zu verstärken, werden in Kapitel 9 zwei fiktive Projektbeispiele vorgestellt. Diese Projekte analysieren die Risiken, die von zwei verteilten Softwareprojekten ausgehen und machen Vorschläge auf welche Weise der Softwareentwicklungsprozess verbessert werden kann. Abschließen wird die Arbeit mit den Schlussfolgerungen aus den zuvor behandelten Kapiteln und den möglichen Aussichten.

## 2 State-of-the-Art

*„Software-Qualität ist die Gesamtheit der Merkmale und Merkmalswerte eines Software-Produkts, die sich auf dessen Eignung beziehen, festgelegte oder vorausgesetzte Erfordernisse zu erfüllen“ [ISO9126]*

Es gibt verschiedene Ansätze um den Qualitätsbegriff zu definieren [Garv84]. Zum Beispiel werden in [Wong02] mehr als ein Dutzend voneinander abweichende Definitionen für diesen Begriff aufgezählt. Eine Softwarefirma, die der oben genannten Softwarequalitätsdefinition gerecht werden will, beschäftigt sich mit den folgenden Fragen:

- Auf welche Qualitätsmerkmale der Software legt der Kunde Wert?
- Welche Qualitätsstufen, in Abhängigkeit von diesen Qualitätsmerkmalen, sind bei der Softwareentwicklung zu erzielen?

Die Softwarequalität ist eng an die Zufriedenheit des Kunden gebunden und wird von dem Softwaredesign und dem Entwicklungsprozess stark beeinflusst [Karo98]. Diese bestehen aus den Entwicklungstechnologien, der Qualität der Entwickler, Kosten, Zeit- und Termineinhaltung und der Prozessqualität [Somm01]. Mit Hilfe von Qualitätsstandards kann die Softwarequalität spezifiziert, gemessen und analysiert werden [Balz01]. Dabei werden Qualitätsprobleme aufgezeigt, besser verstanden und schließlich Verbesserungen durchgeführt.

In diesem Kapitel wird die Norm ISO 9126 vorgestellt, die aus sechs Qualitätsmerkmalen besteht, die wiederum in Teilmerkmale aufgeteilt werden. Dazu werden das Capability Maturity Model Integration (CMMI), der Personal/Team Software Process (PSP / TSP) und das IT Information Library (ITIL) aufgezeigt. CMMI ist ein sehr verbreitetes Qualitätsmanagementmodell für die System- und Softwareentwicklung [Kneu03], ITIL verfügt über große Akzeptanz im deutschen Raum und der PSP und der TSP sind Methoden, die

die kontinuierliche Verbesserung der Softwareentwickler gewährleisten sollen. Zusätzlich werden einige Metriken aufgezeigt. Mit Hilfe von traditionellen und modernen Metriken können Qualitätsmerkmale gemessen, und so die Qualität der Software bewertet werden. Im letzten Abschnitt werden drei Analysewerkzeuge untersucht, die den produktiven Quellcode analysieren und geben, anhand von vordefinierten Metriken, Auskunft über die Struktur der Software.

### 2.1 ISO 9126 Norm

Die Norm ISO/IEC<sup>(2)</sup> 9126 ist ein international anerkannter Standard für Qualitätssicherung von Software [ISO9126]. Dieses „Factor-Criteria-Metrics-Model“, kurz FCM-Modell, definiert zuerst sechs Qualitätsmerkmale („factors“), die einer benutzerorientierten Sichtweise entsprechen:

- Funktionalität
- Zuverlässigkeit
- Benutzbarkeit
- Effizienz
- Änderbarkeit
- Übertragbarkeit

Diese Qualitätsmerkmale werden dann wiederum in Teilmerkmale („criteria“) untergliedert, die einer softwareorientierten Sicht entsprechen (siehe Abbildung 3). In der Definition des Standards wird ein Vorschlag über 21 Teilmerkmale aufgeführt. Metriken waren in der ersten Version dieses Standards nicht definiert worden. In der aktuellen Version sind externe und interne Metriken definiert ([ISO9126a], [ISO9126b], [ISO9126c], [ISO9126d]).

Zusätzlich zum FCM-Modell existiert auch das GQM-Modell. Dieses Goal-Question-Metric Modell wurde von Basili und Rombach entworfen und definiert

---

<sup>2</sup> IEC: International Electrical technical Commission. URL: <http://www.iec.ch>  
ISO: International Organisation for Standardisation. URL: <http://www.iso.org>

Auswertungsziele zur Erstellung des Qualitätsmodells [Balz01]. Zuerst werden diese Ziele (Goal) in Form von Qualitätsmerkmalen definiert und später davon die Fragestellungen abgeleitet (Question), die die Ziele tatsächlich quantifizieren sollen. Die Metriken, die zur Beantwortung der Fragestellungen helfen, werden zusätzlich abgeleitet. Anschließend muss ein Mechanismus definiert werden, um die Metriken messen, validieren und interpretieren zu können.

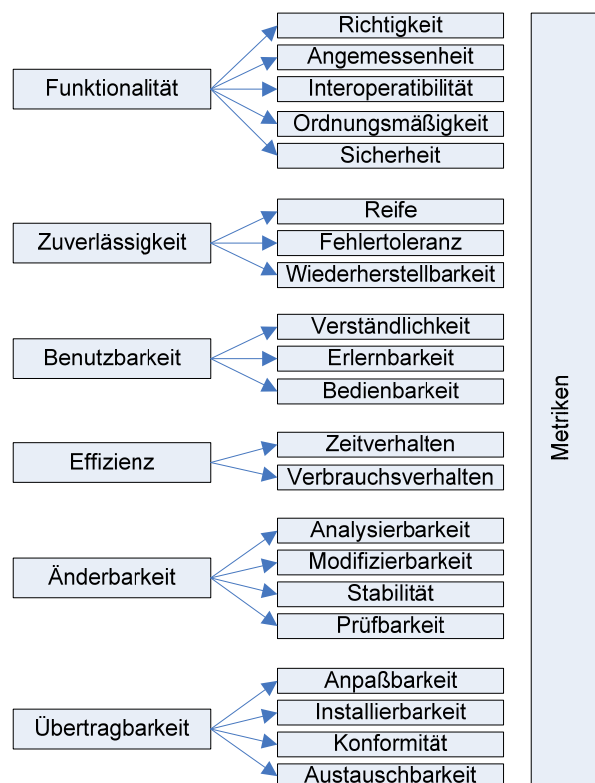


Abbildung 3. Merkmale und Teilmerkmale des Standards ISO 9126 [Balz01]

In einem FCM-Modell müssen nicht unbedingt alle Teil- oder Hauptmerkmale erfüllt beziehungsweise weitere hinzugefügt werden. Jede Firma soll zusammen mit dem Kunden die Wichtigkeit dieser Merkmale für ihr spezifisches Projekt analysieren, um die Kundenwünsche besser zu berücksichtigen. Abhängig vom Projekt, Anwendungsszenario und Art der Software können zusätzlich andere Merkmale definiert und quantifiziert werden. Die erste Version des Standards ISO 9126 gibt keine Auskunft über Metriken, die im FCM-Modell anwendbar sein können. Mögliche Metriken werden im Kapitel 2.3 näher betrachtet.

Im Anschluss werden die Merkmale näher betrachtet, die in der Norm ISO 9126 definiert wurden.

### **2.1.1 Funktionalität**

Die Funktionen mit festgelegten Eigenschaften sind hier zu überprüfen. Die Funktionen erfüllen die vom Kunden im Voraus definierten Anforderungen und liefern die vereinbarten Ergebnisse oder Wirkungen; zum Beispiel die angestrebte Genauigkeit von berechneten Werten. Änderungswünsche des Kunden, die im Laufe des Projektes entstehen, werden erst an zweiter Stelle behandelt, denn die erste und wichtigste Aufgabe eines Systems ist stets seine Funktionalität. Ist diese erst einmal gewährleistet, wird ein Zeitpunkt bestimmt, ab dem neue Wünsche seitens des Kunden gesammelt werden können.

Die Angemessenheit ist die Eignung der Funktionen für spezifizierte Aufgaben, zum Beispiel die aufgabenorientierte Zusammensetzung von Funktionen aus Teilfunktionen. Die Software soll so einfach wie möglich sein. Sie soll nicht mehr (und auch nicht weniger) können als erwartet. Der Entwickler soll sich an dem in der Definitionsphase (siehe Kapitel 4.1) spezifizierten Benutzerprofil orientieren. Es existiert immer ein Kompromiss zwischen einem Optimum an Anpassungsfähigkeit an die Kundenwünsche und der Funktionalität eines Systems. Funktionen, die dieses Teilmerkmal gefährden könnten, werden nicht implementiert, obwohl sie durchaus als sinnvoll erachtet werden können. Um die optimale Eignung einer Funktion sicher zu stellen, bedarf es der Hilfe einer unabhängigen Person. Diese Person bekommt die Argumente des Kunden und des Entwicklers mitgeteilt. Damit soll sie entscheiden, welche Kriterien ausfallen könnten und feststellen welche noch nicht berücksichtigt wurden. Dabei soll nach einem für beide Seiten zufrieden stellenden Mittelweg gesucht werden.

Die Sicherheit liegt in der Eigenschaft der Funktionen, unberechtigte Zugriffe, sowohl versehentlich als auch vorsätzlich, auf Programme und Daten zu verhindern. Daher werden vom Benutzerprofil abhängige Sicherheitsstufen definiert. In der Regel existieren zwei Sicherheitsstufen: die Administratorstufe und die Benutzerstufe. In der Administratorstufe hat der Administrator alle

Rechte inne. Er allein kann Änderungen im System und in der Benutzerdatenbank durchführen und globale so wie lokale Optionen ändern. In der Benutzerstufe kann der einzelne Anwender selbst in der Regel keine Veränderungen am System vornehmen; er nutzt ausschließlich die einmal installierte Anwendung. Das Löschen von anderer Benutzer, Daten oder von Einträgen in Datenbanken ist nicht erlaubt. Ihm sind nur die Leserrechte vorbehalten. Es können allerdings auch mehrere Benutzerprofile existieren, die dann im Voraus definiert werden. Sollte der Wunsch entstehen, zu einem bestimmten Zeitpunkt neue Benutzerprofile zu erstellen, werden diese gesammelt und später diskutiert.

Das Teilmerkmal Interoperabilität wird als die Fähigkeit verstanden, zu vorgegebenen Systemen eine Verbindung herzustellen. Ein wichtiger Punkt in der gesamten Entwicklung ist die Wiederverwendbarkeit von Quellcode (auch von vordefinierten Variablen und Datentypen). Der Entwickler muss sicher sein können, dass die Module, die gerade entwickelt werden, nicht schon vorher implementiert wurden.

### **2.1.2 Zuverlässigkeit**

Dieses Merkmal definiert die Fähigkeit der Software, ihr Leistungsniveau unter festgelegten Bedingungen über einen gewissen Zeitraum bewahren zu können. Die Zuverlässigkeit und die Benutzbarkeit (Abschnitt 2.1.3) werden durch Tests geprüft.

Die Frage nach der Versagenshäufigkeit durch Fehlzustände wird im Teilmerkmal Reife beantwortet. Wichtig ist stets, dass die Stabilität des Systems gewährleistet ist. Der Tester und der Entwickler müssen beide nachprüfen, ob zum Beispiel Laufzeitfehler entstanden sind. Aufrufe von Funktionen, die Eingabe von Variablen und das Öffnen von Masken müssen reibungslos funktionieren.

Unter Fehlertoleranz wird verstanden, dass ein spezifiziertes Leistungsniveau trotz möglicher Softwarefehlern oder Nicht-Einhaltung ihrer spezifizierten

Schnittstelle bewahrt werden kann. Der Tester muss jedoch, die verschiedenen Aspekte der Fehlertoleranz nachprüfen.

Die Wiederherstellbarkeit meint die Fähigkeit, bei einem Versagen das Leistungsniveau wieder herstellen zu können und so die direkt betroffenen Daten wiederzugewinnen. Hierfür sind die dafür benötigte Zeit und der benötigte Aufwand zu berücksichtigen. Die Aufgabe des Testers ist es zum Beispiel zu prüfen, ob im Datenbankbereich die Wiederherstellbarkeit erfüllt ist. Die Dauerhaftigkeit der Einträge soll dadurch sichergestellt werden, dass zum Beispiel gespeicherte Einträge nicht ohne die explizite Erlaubnis vom Benutzer verschwinden oder gelöscht werden können. Nach einem Programmabsturz soll mit der Wiederherstellbarkeit erreicht werden, dass alle alten Einträge immer noch gespeichert bleiben und nur Transaktionen, die nicht zu Ende geführt wurden, verworfen werden.

### **2.1.3 Benutzbarkeit**

Die Benutzbarkeit von Software entscheidet heutzutage oft über Akzeptanz und Erfolg. Eine 100%tige Übereinstimmung zwischen Produktdefinition und fertigem System wird angestrebt. Das erfordert vom Tester, dass er minutiös die Benutzbarkeit des Systems überprüft, und zum Beispiel darauf achtet, ob das System wie gewünscht entwickelt worden ist. Der Tester nimmt dabei die Position des Anwenders ein und versucht das System zu verstehen, die Bedienbarkeit zu überprüfen und auf diese Weise festzustellen, ob ein Standardbenutzer das System beherrschen kann.

Der Aufwand, den der Benutzer aufbringen muss, um das Konzept und die Anwendung zu verstehen, wird unter dem Teilmerkmal Verständlichkeit definiert. Es wird deswegen versucht, die Implementierung von Funktionen so einfach wie möglich zu programmieren. Es darf für den Kunden kein Problem darstellen die Funktionalität der Software sofort zu erkennen. Um dies erreichen zu können, müssen sich nicht nur die Entwickler, sondern auch die Tester und die Nutzer mit der Software beschäftigen.

Auch das Erlernen der Funktionsweise der Software erfordert den notwendigen Aufwand vom Kunden, um die Anwendung beherrschen zu können. Zum Beispiel soll der Benutzer sich mit der Bedienungsoberfläche gut zu Recht finden. Es wird hier erwartet, dass der Benutzer das Konzept versteht und gegebenenfalls Vorschläge für Verbesserungen äußert. Wenn die Software keine Hilfedatei oder ein Benutzerhandbuch zum Nachschlagen bereitstellt, wird es sehr schwer für den Kunden/Benutzer, die Anwendung beziehungsweise das System ordnungsgemäß zu bedienen. Für die Entwickler gilt, dass es sinnvoll und wichtig ist, nicht zu vergessen eine solche Datei oder ein Benutzerhandbuch zu erstellen, denn so können etliche Telefonate und mögliche Fahrtzeiten eingespart werden, die die Kosten der Wartung und Pflege vergrößern. Die Einfachheit und Unkompliziertheit der Software muss gerade durch die Bedienbarkeit gewährleistet sein. Eine leichte Bedienbarkeit wird durch eine einfache Konfiguration der Umgebung erreicht, zum Beispiel durch Schrift- oder Farbanpassungen und soll komplett per Tastatur bedienbar sein und eine Schnellastenkombination für alle Funktionen zur Verfügung stellen.

### **2.1.4 Effizienz**

Die Systemvoraussetzungen von Hard- bzw. Software werden mit diesem Merkmal ermittelt. In jedem Modul spielt die Effizienz eine wichtige Rolle, die besagt, dass zum Beispiel keine langen Antwortzeiten seitens des Systems entstehen dürfen. Implementierte Module werden deshalb ständig mit der Softwarespezifikation verglichen, um sicher gehen zu können, dass die Funktionen korrekt implementiert werden. Der Entwickler muss sich während der Implementierung des Öfteren kritisch hinterfragen und darf nicht aus den Augen verlieren, welche Bedeutung seine Arbeit für die gesamte Entwicklung hat. Nur mit einem kritischen Blick auf die eigene Arbeit wird diese Effizienz gewährleistet.

Die Antwort- und Verarbeitungszeiten sowie der Durchsatz bei der Funktionsausführung sind im Teilmerkmal Zeitverhalten beschrieben. Der Entwickler gewährleistet, dass Berechnungen der Software so optimal wie

möglich durchlaufen und Berechnungszeiten ständig verglichen werden, um eine bessere Effizienz zu erreichen. Da die Software überall einsetzbar sein soll, muss der Entwickler dafür sorgen, dass die Betriebssystemanforderungen nicht zu hoch sind. Dabei sollen die im Markt standardisierten Rechner oder die nach einer Absprache mit dem Kunden spezifizierten Voraussetzungen als Maß betrachtet werden.

### **2.1.5 Änderbarkeit**

Die Änderbarkeit wird als der Aufwand definiert, der zur Durchführung vorgegebener Änderungen notwendig ist. Änderungen können Korrekturen, Verbesserungen oder Anpassungen an veränderte Umgebungen und der Anforderungen und der funktionalen Spezifikationen einschließen.

Nach der Abgabe einer Software ist die entsprechende Wartungs- und Pflegephase sehr wichtig (siehe Kapitel 4.1). Wird ein Produkt nicht ständig verbessert, dann veraltet es sehr schnell. Eine ständige Rücksprache mit dem Kunden ist daher unerlässlich, um das Produkt ständig zu verbessern beziehungsweise zu verbessern. Würden solche Verbesserungen nicht durchgeführt werden, würde die Softwareentwicklung sehr statisch werden und zu schnell altern, was die Konsequenz hätte, dass neue Versionen nicht mehr implementiert werden.

Die Analysierbarkeit wird als der Aufwand definiert, Mängel oder Ursachen von Versagen zu diagnostizieren oder um änderungsbedürftige Teile zu bestimmen. Keine Software ist fehlerfrei. Damit der Kunde schnell zu einer Lösung für seine Probleme kommt, sollen mit jeder Software ein Handbuch, eine FAQ-Datei und eine Hilfedatei mitgeliefert werden. Diese sind in der Menüleiste, unter dem Punkt „Hilfe“ zu implementieren.

Die Modifizierbarkeit meint die Fähigkeit zur Ausführung von Verbesserungen, Fehlerbeseitigung oder Anpassung an Umgebungsänderungen. Nach jedem Update oder Release werden die behobenen Fehler dokumentiert und in einer Art Knowledge Base abgespeichert. Die Pflege und Wartung werden immer ein

wichtiger Punkt sein, ebenso wie die Weiterentwicklung der Software, um den Lernprozess sowohl von Entwicklern als auch von Benutzern ständig zu verbessern.

Eine äußerst positive Eigenschaft der Software ist, dass sie sich flexibel erweitern lässt. Die Grenzen der Software sollen allerdings ganz klar definiert sein. Wenn die vorher definierten Grenzen in Bezug auf den Zeitraum nicht mehr mit der Praxis übereinstimmen und sich die Anzahl der Benutzer oder die Datenbankgröße geändert haben, können weitere Probleme entstehen.

### **2.1.6 Übertragbarkeit**

Der Begriff der Übertragbarkeit wird auch als Portabilität gekennzeichnet und gibt Auskunft darüber wie aufwendig es ist, die Software in eine andere Umgebung (Hardware oder Software) zu übertragen. Die Anforderungen an die Umgebung sollen nicht zu hoch sind, um eine solche Umstellung vorzunehmen.

## **2.2 Verbesserung der Softwareentwicklung**

Die Definition und Standardisierung des Entwicklungsprozesses ermöglicht dem Management das Projekt besser zu kontrollieren und zu steuern [BoTu04b]. Das Capability Maturity Model Integration (CMMI) ist eines der reifsten und meistverbreiteten Qualitätsmodelle im Hinblick auf die Prozessverbesserung [Kneu03] (Abschnitt 2.2.1):

*“Capability Maturity Model<sup>®</sup> Integration (CMMI) is a process improvement approach that provides organizations with the essential elements of effective processes.”<sup>(3)</sup>*

Der „Personal Software Prozess“ (PSP) und der „Team Software Prozess“ (TSP) unterstützen dabei die Kodierungsarbeiten der Entwickler und der

---

<sup>3</sup> Quelle: <http://www.sei.cmu.edu/cmmi/general/general.html> (Abruf am: 17.04.2007)

involvierten Teams (Abschnitt 2.2.2). Der De-facto-Standard, das Best Practice Rahmenwerk „Information Technology Infrastructure Library“ (ITIL) unterstützt das Management von IT-Strukturen [ViGü05] (Abschnitt 2.2.3).

Im Kapitel 3 werden das CMMI-Modell und das ITIL-Rahmenwerk näher untersucht und analysiert, wie gut diese Vorgehensmodelle sich an verteilte Softwareprojekte anpassen lassen.

### 2.2.1 CMMI

Das Capability Maturity Model Integration (CMMI) ist ein Qualitätsmodell<sup>(4)</sup> für die System- und Softwareentwicklung [Kneu03] und beschreibt die Entwicklung und Wartung von Software. CMMI wurde von dem Software Engineering Institute<sup>(5)</sup> (SEI) im Jahr 2000 mit der Version 1.0 verabschiedet. 2002 erschien die Version 1.1 dieses Qualitätsmodells und seit 2006 gibt es die aktuellste Version 1.2. CMMI integriert und fasst vorher entwickelten CMMs verschiedener Disziplinen zusammen. CMMI v1.1 gruppiert folgende Modelle [DHRV05]:

1. Capability Maturity Model for Software (SW-CMM) v2.0 draft C
2. Systems Engineering Capability Maturity Model (SE-CMM)
3. Integrated Product Development Capability Maturity Model (IPD-CMM) v0.98

CMMI ist also eine Weiterentwicklung von SW-CMM, SE-CMM und IPD-CMM, drei für die Softwaretechnik verbreiteten Standards, die mehrere Prozessgebiete einer Organisation verbessern. Die Software-, System-, Prozess- und die Produktentwicklung, sowie der Kauf von Software sind Prozessgebiete, die in CMMI aufgegriffen werden. CMMI richtet ihren Fokus auf die Vorhersagbarkeit und Stabilität eines Projektes durch Standards und ständige Messungen sowie durch die Kontrolle der Abläufe [Kala03].

CMMI wird in sechs Fähigkeitsgrade und fünf Reifegrade unterteilt (siehe Abbildung 4). Die Reifegrade (maturity levels) umfassen eine Menge von

---

<sup>4</sup> In der Literatur wird auch manchmal der Begriff „Referenzmodell“ benutzt.

<sup>5</sup> <http://www.sei.cmu.edu/> (Abruf am: 26.03.2007)

verschiedenen Prozessgebieten, die zu einem bestimmten Fähigkeitsgrad umgewandelt sein müssen. Die stufenförmige Prozessverbesserung der Organisation spiegelt sich in die Reifegrade [Kneu03]. In der Initialen Stufe 1, die jede Organisation automatisch besitzt, stellen sich die Prozesse als chaotisch und wenig definiert dar. Ziel dieser Stufe ist es, in erster Linie Informationen zu der Prozessabwicklung sinnvoll und nachvollziehbar zu dokumentieren. Dadurch werden Soll-Ist Vergleiche durchgeführt, unklare Anforderungen und unvollständige Planungen erkannt und unklare Vorgehensweisen verbessert. In der Stufe 2 werden die Projekte und Prozesse organisiert durchgeführt, wodurch mit der gesammelten Information ein neues, ähnliches Projekt wiederholt werden kann. Die Stufe 3 ist der Schritt, um Informationen von dem einzelnen Projekt zu der gesamten Organisation zu übertragen. Projekte werden allgemein nach einem bestimmten Standardprozess durchgeführt. Dabei findet eine organisationsübergreifende und kontinuierliche Prozessverbesserung statt. In der Stufe 4 wird die Erhebung und die intensive Nutzung von Metriken und Kennzahlen bevorzugt (statistische Prozesskontrolle), um die Grundlage für Entscheidungen bezüglich der Verbesserungsaktivitäten quantitativ festzustellen. In der letzten Stufe werden Prozesse mit den Daten aus der statistischen Prozesskontrolle kontinuierlich weiter entwickelt. Somit können Verbesserungen in der Organisation systematisch ausgewählt und ausgeführt werden.

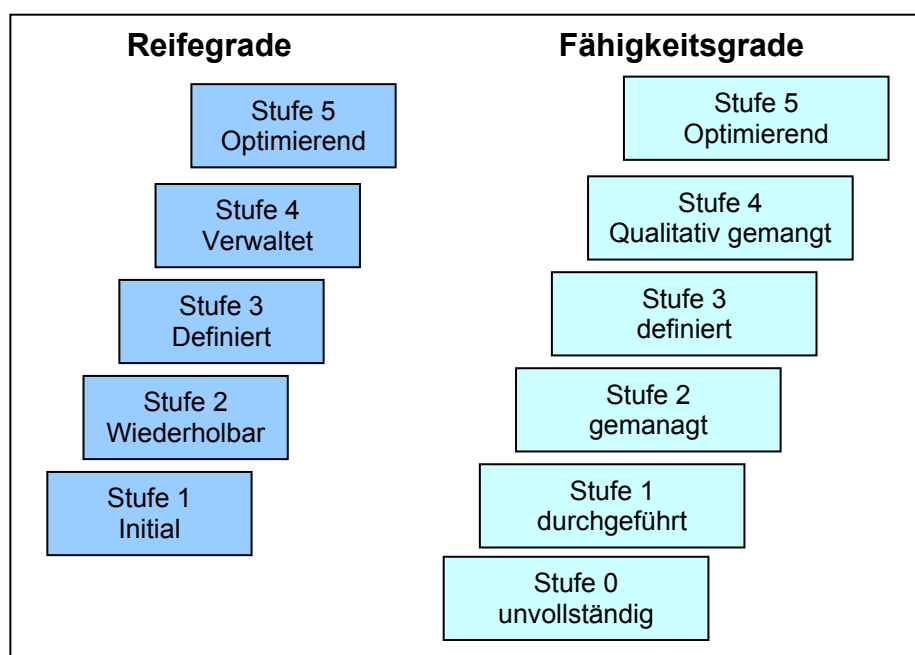


Abbildung 4. Reife- und Fähigkeitsgrade in CMMI [Kneu03]

Zusätzlich zu den stufenförmigen Reifegraden existieren die kontinuierlichen Verbesserungen innerhalb eines Prozessgebietes. Die sogenannten Fähigkeitsgrade (capability levels) [Kneu03] bezeichnen den Grad der Institutionalisierung eines einzelnen Prozessgebietes. Die Stufe 0 ist unvollständig und stellt den Ausgangszustand für alle darauf folgenden Verbesserungen dar. Am Ende der Stufe 1 werden die spezifischen Ziele eines Prozesses erreicht. In der Stufe 2 wird dieser Prozess organisiert durchgeführt. Die nächste Stufe ist die so genannte Definierungsstufe. Hier wird ein Prozess anhand eines angepassten Standardprozesses, verbessert und institutionalisiert. In der vierten Stufe wird der Prozess quantitativ bearbeitet, das heißt er steht unter statistischer Kontrolle. In der letzten Stufe wird dieser Prozess nun mit Hilfe der in der Stufe 4 gesammelten Daten verbessert. Die generischen Ziele (Generic Goals) helfen bei der Institutionalisierung eines bestimmten Prozessgebietes. Als Faustregel gilt: der Fähigkeitsgrad  $n$  ist nur dann erreicht, wenn das generische Ziel  $n$  ( $GG_n$ ) auch erreicht ist. Wenn kein Ziel erreicht wurde, hat die Organisation den Fähigkeitsgrad 0 zu verbuchen [Kneu03].

Die Ziele, die CMMI definiert, werden mit dem Namen „Key Process Areas“ (KPA) definiert. Jede KPA ist eine kleine Komponente der Softwareentwicklung und beinhaltet eine Zusammenfassung von Anforderungen zu einem bestimmten Thema:

*„Each KPA is a small component of software development and includes a cluster of related activities to be performed collectively“.*

*[RaKK05]*

Die KPAs wurden für lokal und nicht für verteilte Projekte definiert [SeCS06]. Nicht alle KPAs lassen sich an die verteilte Softwareentwicklung anpassen. Im Kapitel 3.1 werden 21 neue, angepasste KPAs vorgestellt, die eine verteilte Umgebung besser unterstützen.

## 2.2.2 PSP / TSP: Eine Methode zur Kontinuierlichen Verbesserung

CMMI gibt eine Antwort darauf „was“ Organisationen tun können, um ihre Prozesse zu verbessern. Aus diesem Qualitätsmodell wird aber nicht ersichtlich, wie diese Verbesserungen erreicht werden können. Der Personal Software Process (PSP)<sup>(6)</sup>, der ebenfalls von SEI entwickelt wurde, gibt Antwort auf diese Frage. Während PSP für den einzelnen Entwickler gedacht wurde, wurde zusätzlich das Team Software Process (TSP)<sup>(7)</sup> für die im Projekt involvierten Teams entwickelt [Hump97].

PSP hilft den Entwicklern die Schwankungen des Entwicklungsprozesses zu verringern. Ziel ist es, eine Verbesserung der eigenen Planung und Qualität zu erreichen, indem Fehler und Defekte besser abgeschätzt und korrigiert werden. Die Produktivität, Defekthäufigkeit und Qualität der Ergebnisse können mit Hilfe von Werkzeugen ermittelt werden. Die gewonnenen Daten helfen zukünftige Projekte vorhersagbarer zu machen.

Eine interessante Sammlung von Werkzeugen für PSP wurde von der Universität Karlsruhe (IPD Tichy)<sup>(8)</sup> zusammengestellt. Werkzeuge wie PPLog, evalpsp oder evalpplog helfen den Entwicklern ihre Arbeit zu verbessern.

Diese kontinuierliche Verbesserung wird gewährleistet durch Aktivitäten wie:

- Quellcodezeilen zählen. Hinzugefügte, neue und geänderte Lines of Code (LOC) werden ermittelt.
- Zeit- und Defektprotokoll erfassen. Die Zeit für die verschiedenen Entwicklungsphasen, für das Entfernen von Fehlern und die Unterbrechungen der Arbeit wird protokolliert.
- Fehlerklassifikationen erstellen. Jeder Fehler wird durch drei Argumente klassifiziert: Einführphase, Defekttyp und Defektgrund.

---

<sup>6</sup> URL: <http://www.sei.cmu.edu/tsp/psp.html> (Abruf am 26.03.07)

<sup>7</sup> URL: <http://www.sei.cmu.edu/tsp/tsp.html> (Abruf am 26.03.07)

<sup>8</sup> URL: <http://www.ipd.uka.de/PSP> (Abruf am 28.03.07)

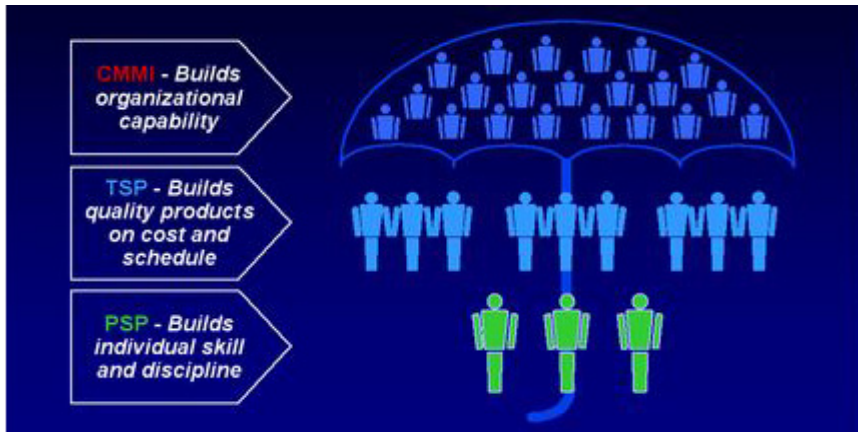


Abbildung 5. Zusammenspiel von CMMI, TSP und PSP(9)

PSP ist für den einzelnen Entwickler gedacht. Dabei soll aber jeder Entwickler in der Lage sein, das eigene Wissen in das gesamte Team einzubringen. Ein Projektteam stellt die kleinste operationelle Einheit in einem Unternehmen dar [Hump99]. TSP wird zwischen CMMI und PSP aufgebaut (siehe Abbildung 5):

*„Perhaps the most powerful consequence of the TSP is the way it helps teams manage their working environment.” [Hump99]*

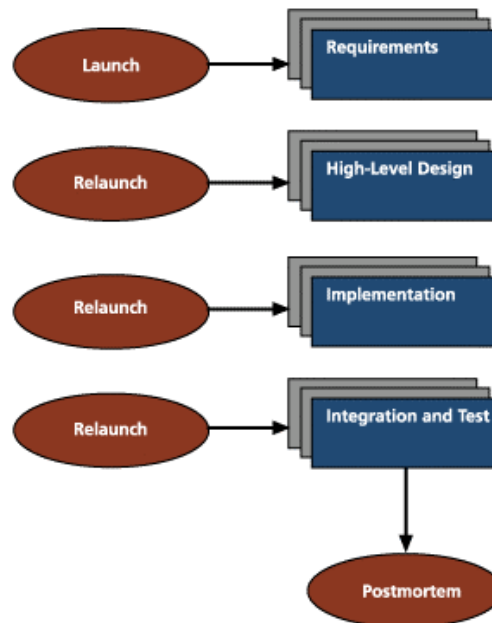
TSP ist folglich ein Handbuch für Entwickler, die ihre Entwicklungsarbeiten in Teams ausführen. Diese Methode begleitet Entwickler durch die verschiedenen Projektphasen und gibt Auskunft darüber wie die Teams sich selbst steuern und wie sich die Mitglieder effektiver verhalten können. Entwickler gestalten dadurch das Projekt selbst, in dem sie eigene Pläne und Prozesse gestalten. PSP und TSP eignen sich insbesondere auch für kleine Teams [SeMo04].

Watts Humphrey definiert vier Projektphasen, die er für jedes Projekt verwendet (mehr dazu im Kapitel 4.1) [Hump00]:

1. die Anforderungs-,
2. die High-Level Design-,
3. die Implementierungs- und
4. die Integrations- und Testphase.

<sup>9</sup> Quelle: <http://www.sei.cmu.edu/videos/watts/DPWatts.mov> (Abruf am 26.03.07)

PSP und TSP unterstützen Teams und Entwickler mit Anweisungen, Standards, Prozessen und Skripten damit eine möglichst disziplinierte und effektive Arbeit geleistet werden kann. Jedes Projekt startet mit einem so genannten „Launch“ (siehe Abbildung 6). In diesem Treffen werden Ziele und Strategien vereinbart, die Arbeit organisiert, Rollen verteilt, Risiken identifiziert und Pläne und Termine für das gesamte Projekt besprochen.



**Abbildung 6. TSP und PSP erlauben eine ständige Prozessverbesserung(10)**

Während der Durchführung des Projektes und am Ende jeder Phase findet ein so genanntes „Relaunch“ statt. Diese periodischen Treffen dienen dazu neue, detaillierte Pläne zu erarbeiten und Termine zu vereinbaren. Am Ende des Projektes findet eine so genannte „Post mortem“ Analyse statt, um die Ergebnisse zu untersuchen. Dieses Vorgehen lässt sich auch für kleine und Start-Up Unternehmen adaptieren, die in verteilten Projekten involviert sind [SeMo04]. Der PSP kann immer für die kontinuierliche Verbesserung der verschiedenen Teams genutzt werden, unabhängig davon, ob die Teams verteilt sind oder nicht.

<sup>10</sup> Quelle: <http://www.sei.cmu.edu/videos/watts/DPWatts.mov> Abruf am 26.03.07

### 2.2.3 ITIL: Der Best-Practice-Ansatz

Der IT Infrastructure Library (ITIL) ist ein verbreiteter Standard für das IT Service Management (ITSM) und wurde vom „IT Department of the British Government“ (OGC) entwickelt [OfGC00]. ITIL hat in den letzten Jahren zunehmend an Wichtigkeit gewonnen und wird heutzutage auch vorzugsweise in deutschen Unternehmen eingesetzt [HoLS03]. Die Koordinierungs- und Beratungsstelle der Bundesregierung für Informationstechnik in der Bundesverwaltung (KBSt) hat diesbezüglich 2006 ihr Informationsangebot im Bereich ITIL erweitert<sup>(11)</sup>.

ITIL verfolgt das Ziel, das IT-Management zu verbessern. Das von der OGC definierte Rahmenwerk ist in Deutschland stark vertreten. Mit der Service-Unterstützung und der Service-Verfügbarkeit werden Geschäftsbereiche mit potentiell Verbesserungbedarf identifiziert. Verantwortungsgebiete und Risiken können zusätzlich identifiziert werden und dementsprechend an einen externen Dienstleister angepasst werden.

ITIL beschreibt die Aktivitäten, die Dokumente, die Rollen und die wichtigsten Erfolgskennzahlen (kurz KPA - Key Performance Indicators), die in der Organisation beachtet werden sollen, um ein verbessertes IT-Management zu gestalten [HoZB05a]. Das Ziel dieses Ansatzes ist nicht nur eine reine Prozessverbesserung zu erreichen, sondern viel mehr eine Reihe von strukturierten Methoden anzubieten, mit denen aktuelle IT-Services identifiziert und verstanden werden können [Kirk05].

Das IT Service Management bezeichnet die Gesamtheit von Maßnahmen und Methoden, die nötig sind, um die bestmögliche Unterstützung von Geschäftsprozessen durch eine Organisation zu erreichen [VaVP06]. Für das IT Service Management existieren zwei zentrale Bereiche ([OfGC00], [VaVP06], [ViGü05]):

---

<sup>11</sup> Quelle: [http://www.kbst.bund.de/SharedDocs/Meldungen/2007/03\\_\\_12\\_\\_ITIL\\_\\_neues\\_\\_in\\_\\_der\\_\\_Bundesverwaltung.html](http://www.kbst.bund.de/SharedDocs/Meldungen/2007/03__12__ITIL__neues__in__der__Bundesverwaltung.html) (Abruf am 28.03.2007)

- Service Support (Service-Unterstützung) mit
  - Incident Management,
  - Problem-Management,
  - Konfigurations-Management,
  - Change Management und
  - Release Management
- Service Delivery (Service-Verfügbarkeit) mit
  - Service Level Management,
  - Verfügbarkeits-Management,
  - Finanz-Management,
  - IT-Service Kontinuitäts-Management und
  - Kapazitätsmanagement.

Abbildung 7 zeigt die zehn Prozesse, die diese zwei Bereiche umfassen. Selbst große Unternehmen besitzen oftmals nicht alle Ressourcen um diese zehn Prozesse zu implementieren. Kleine und Mittelständische Unternehmen können dennoch im Bereich der Service-Unterstützung merkliche Verbesserungen erzielen [BMC06].

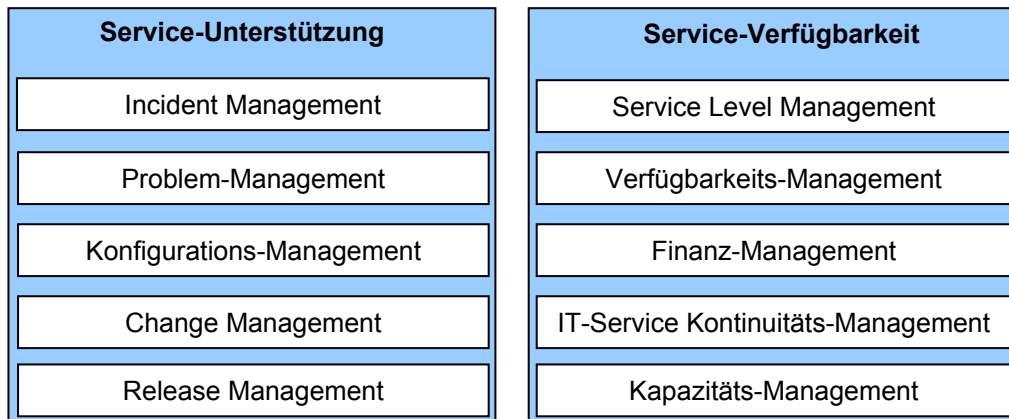


Abbildung 7. ITIL-Prozesse [BMC06]

## 2.2.4 Service Level Agreement (SLA)

Ein Service Level Agreement, kurz SLA, definiert eine Vereinbarung zwischen Auftraggeber und Dienstleister. Eine SLA stellt eine vertragliche Grundlage für die Erbringung einer Dienstleistung gegenüber den Auftraggeber dar [ViGü05] und kann beispielweise die Reaktionszeit in einem bestimmten Fall oder der Umfang und Schnelligkeit einer Bearbeitung näher definieren ([ViGü05], [VaVP06]):

*„A written agreement or contract between the customers and the IT provider which documents the agreed service levels for an IT service”*

In Deutschland ist der Begriff SLA durch ITIL bekannt geworden und wird im Bereich des „Service Level Management“ (siehe Abbildung 7) entwickelt, überwacht und gepflegt. Zusätzlich schaffen SLAs eine gewisse Transparenz der Preis/Leistungsverhältnis für Kunden und Dienstleister, denn der Auftraggeber hat die Möglichkeit zwischen mehrere Ebenen der vereinbarten Dienstleistung auszuwählen.

## 2.2.5 ISO 9000-Normenfamilie

ISO 9000<sup>(12)</sup> ist eine von ISO herausgegebene Familie von Normen, die internationale Qualitätsmanagementsysteme (QM-Systeme) vereinheitlicht. Diese Normenfamilie legt für Auftraggeber und Lieferanten einen Rahmen zur Qualitätssicherung sowohl für materielle als auch für immaterielle Produkte fest. Dieser Rahmen ist von allgemeiner, übergeordneter und organisatorischer Art [Balz01].

Die Struktur dieser Norm beinhaltet Begriffsbestimmungen (ISO 8402), Einführungen und Leitfäden zur Auswahl und Anwendung der Normen ISO 9001, 9002, 9003 und 9004 bezüglich der Qualitätssicherung. Wichtigste

---

<sup>12</sup> URL: <http://praxiom.com/> (Abruf am 03.04.2007)

Aufgaben dieser Norm sind die Darlegung der Qualitätssicherung gegenüber Dritter und die Verbesserung oder der Aufbau eines QM-Systems [Kneu95].

Die Vorteile dieser Normenfamilie sind unter anderem die Lenkung der Aufmerksamkeit des Managements auf die Probleme der Qualitätssicherung und die Verstärkung des Qualitätsbewusstsein. Einige Nachteile bestehen in der erhöhten Bürokratie und mangelnden Flexibilität. Eine ISO-Zertifizierung kann mit CMMI Stufe 3 verglichen werden. CMMI bietet aber zusätzlich Hilfe bei Technik, Prozessdefinition und Metriken.

## 2.3 Softwaremetriken

Eine Softwaremetrik definiert, wie eine Kenngröße eines Softwareprodukts oder eines Softwareentwicklungsprozesses gemessen wird [Balz01]. Um die im Abschnitt 2.1 definierten Qualitätsmerkmale (und dabei auch die Teilmerkmale) beurteilen zu können, sind Softwaremetriken notwendig, mit deren Hilfe bestimmte Eigenschaften eines Produktes gemessen und interpretiert werden. Auf diese Weise können Aussagen über die Qualität eines Softwareproduktes getroffen werden. Sie ermöglichen zusätzlich das Auffinden kritischer Teile des entwickelten Systems und können sogar konkrete Hinweise auf Verbesserungsansätze liefern, wenn sie in einem geeigneten Rahmen zur Anwendung kommen [Glob05]. In [IEEE04] wird eine Metrik folgendermaßen definiert:

*„A function whose inputs are software data and whose output is a single numerical value that can be interpreted as the degree to which software possesses a given attribute that affects its quality.“*

Mathematisch gesehen wird eine Metrik folgendermaßen definiert [Fent94]:

Ist  $m$  eine Metrik, die eine bestimmte Eigenschaft  $E$  misst, so ist es wünschenswert, dass die gemessenen numerischen Werte  $m(P_1)$  und  $m(P_2)$  die empirische Ordnung  $\prec$  zwischen  $P_1$  und  $P_2$  wiedergeben, oder formal ausgedrückt:

$$m(P_1) < m(P_2) \Leftrightarrow P_1 \prec P_2$$

Die erste Version des Standards ISO 9126 gibt keine Auskunft über Metriken, die im FCM-Modell (siehe Abschnitt 2.1) anwendbar sein können. Der überarbeitete Standard ISO 92126 definiert Metriken, die aber nicht formal evaluiert und validiert wurden [AICK05]. Im Folgenden werden Metriken vorgestellt, die in zwei Gruppen unterteilt sind: auf der einen Seite werden die klassischen Metriken zusammengefasst, auf der anderen Seite stehen die modernen objektorientierten Metriken.

### 2.3.1 Klassische Metriken

Klassische Metriken sind zum Beispiel das Messen der Anzahl der Quellcodezeilen (Lines of Code, kurz LOC), die Anzahl der vorhandenen Fehler oder die so genannte benötigten Personentage.

- **Lines of Code (LOC).** Die Metrik LOC misst die Anzahl der Zeilen im Quellcode eines Programms. Für viele ist diese Metrik notwendig, da die Arbeitsmenge der Entwickler in LOC/Tag bei festen Bedingungen gemessen werden kann. Wenn die Umgebungsbedingungen konstant bleiben, kann diese Metrik im Voraus geschätzt werden. Sonst ist diese Metrik aber ungenau oder irreführend, wenn zum Beispiel die LOC zwei verschiedener Programmiersprachen verglichen werden.
- **Fehleranzahl.** Diese Metrik ist maßgeblich für die Produktivität und Qualität der Prozesse, in dem Defekte der Software klassifiziert werden.

Da in den meisten Fällen eine Defektklassifikation von Menschen durchgeführt wird, verfügt diese Metrik über eine begrenzte Genauigkeit aber einem hohen Nutzwert an Information, weil die gefundenen Defekte starken Einfluss auf die Qualität und Produktivität der Software ausüben. Die Art und Genauigkeit der gefundenen Fehler bleibt aber subjektiv und die reale Nützlichkeit der Metrik deswegen sehr fraglich.

- **Personentage.** Bei dieser Metrik wird der Aufwand an Arbeitszeit gemessen. Die Wichtigkeit dieser Metrik liegt darin, dass die Projektkosten öfters darauf basieren, denn Kosten-, Zeitbedarf- und Produktivitätsschätzungen werden mit dieser Metrik berechnet. Diese Metrik ist aber schwer übertragbar wenn zum Beispiel Entwickler in mehreren Projekten oder Organisationen gleichzeitig arbeiten.

### 2.3.2 Objektorientierte Metriken

Die Eigenschaften der objektorientierten (OO) Entwicklung, wie etwa die Abstraktion in Klassen, die Vererbung, die Kapselung und die Polymorphie werden von den klassischen Metriken nur unzulänglich berücksichtigt [Glob05]. Chidamber und Kemerer definierten in einer Studie [ChKe94] sechs verschiedene Metriken, die später von Basili, Briand und Melo [BaBM96] validiert wurden.

- **Gewichtete Methoden pro Klasse (Weighted Methods per Class - WMC):** dient dazu den Aufwand zur Entwicklung und Wartung vorherzusagen. Dabei wird die die Komplexität einer Klasse  $C$  mit  $n$  Methoden gemessen, indem jeder Methode ein Gewicht  $c_i$  zugeordnet und über die Methoden summiert wird:

$$WMC(C) = \sum_{i=1}^{|n|} c_i$$

- **Tiefe im Vererbungsbaum (Depth of Inheritance Tree - DIT):** ist die maximale Länge des Pfades von der Klasse bis zur Wurzel ihres Vererbungsbaums. Es wird immer der azyklische Vererbungsbaum genommen.
- **Zahl von Unterklassen (Number of Children - NOC):** ist die Zahl der direkten Unterklassen einer Klasse.
- **Kopplung zwischen Objektklassen (Coupling Between Object Classes - CBO):** zählt die Anzahl anderer Klassen, deren Methoden oder Instanzvariablen von Klasse  $C$  benutzt werden.
- **Antwortmenge einer Klasse (Response For a Class - RFC):** ist die Gesamtanzahl verschiedener Methoden, die innerhalb einer Klasse direkt aufgerufen werden. Die Methoden werden bis zur ersten Aufruftiefe zurückverfolgt.
- **Mangel an Zusammenhang zwischen Methoden (Lack of Cohesion in Methods - LCOM):** ist ein Maß für den Mangel an Kohäsion oder Bindung zwischen den Klassen. Diese Metrik ist definiert durch die Differenz zwischen die Menge  $S$  der Methodenpaare einer Klasse, die eine gemeinsame Instanzvariable verwenden, und der Anzahl der Paare  $N$ , die das nicht tun:  $LCOM = \max(|N| - |S|, 0)$ .

## 2.4 Testaktivitäten

Es gibt keine Software, deren Code perfekt ist. Das Testen der Software ist sozusagen ein Muss-Kriterium für jeden, der eine Mindestqualität seines Produktes anbieten will. Es gibt genügend Beispiele, die belegen, warum das Testen so wichtig ist, zum Beispiel die Katastrophe der Ariane 5 Rakete [Thal00]. Die Rakete explodierte 40 Sekunden nach dem Start und verursachte so Schaden in Höhe von mehr als 500 Millionen US-Dollar. Eine Datenbibliothek einer alten Raketenvariante wurde wieder verwendet, ohne

vorher sorgfältig getestet worden zu sein. Dadurch erfolgte ein Überlauffehler, der die Selbstzerstörung der Rakete ausgelöst hat.

Das Testen wird folgendermaßen definiert:

*"... the process of exercising or evaluating a system or system component by manual or automated means to verify that it satisfies specified requirements or to identify differences between expected and actual results ..."*<sup>(13)</sup>

Mit der Hilfe der oben stehenden Definition, kann das Testen folgendermaßen erläutert werden:

*„Testen ist das Ausführen eines Programms, mit der Absicht (möglichst viele) Fehler zu finden. Ein Test besteht aus einem Satz von Eingabedaten zusammen mit dem jeweils erwarteten Ergebnis (SOLL). Beim Durchführen eines Tests wird das Programm mit den Eingabedaten ausgeführt und das Ergebnis (IST) mit dem erwarteten Ergebnis verglichen.“*[Gies05]

Obwohl die meisten Entwickler wissen, dass Testen ein wichtiger Bestandteil des Programmierens ist, setzen sie sich des Öfteren über die Notwendigkeit des Testens hinweg. Mögliche Argumente dafür sind:

- Die Implementierung von Tests kostet zu viel Zeit.
- Es dauert zu lange um den Code zu testen.
- Der Entwickler schreibt das Testen seines eigenen Codes nicht seinem Aufgabenbereich zu.
- Zudem kursiert die Auffassung: „Der Entwickler wird für das Kodieren bezahlt, nicht für das Testen“.

Wird auf das Testen verzichtet, dann wird der Code unzuverlässig. Die Wahrscheinlichkeit doch einen Fehler in der Implementierung zu finden, wird

---

<sup>13</sup> ANSI/IEEE Standard 729, 1983

größer und die Fehlerbeseitigung erhöht die Kosten. Der Testprozess ist unabhkömmlich, er liefert die Antwort auf zwei wichtige Fragen:

1. „Wird die richtige Software entwickelt?“
2. „Wird die Software richtig entwickelt?“

In den letzten 30 Jahren hat sich der Wichtigkeit des Testprozesses geändert [Beck05]. Während es in den frühen Siebziger Jahren keine Unterschiede zwischen Testen und Debuggen gab, wurden am Ende der Siebziger zunehmend die Unterschiede zwischen diesen beiden Prozessen erkannt. Viele Tests waren von Nöten, um sicher zu gehen, dass die Software tatsächlich funktionierte. Die Gefahr lag in der benötigten Testzeit für das Entfernen eines einzelnen Fehlers. Die Testphase wurde extrem teuer und die Firmen hatten bereits eine zuverlässige Software, aber diese ließ sich nicht verkaufen. Mit der Zeit wurden mathematische Schätzmodelle eingeführt, um den Prozentsatz unentdeckter Fehler besser abschätzen zu können. Das Risiko wurde auf ein akzeptables Niveau verringert. Heutzutage wird das Testen nicht mehr als Tätigkeit gesehen, sondern vielmehr als eine Einstellung, die zu leicht-testender Software mit geringfügigem Restrisiko führt. Tests, als Teil der Softwarequalitätssicherung, vermitteln Vertrauen in die Funktionalität und Zuverlässigkeit einer Software.

Um ordnungsgemäße Test durchführen zu können, gibt es einige Richtlinien, die beachtet werden sollten [HuTh03]. Ein guter Test lässt sich:

- automatisch,
- vollständig,
- wiederholbar,
- unabhängig und
- professionell

ausführen. In verteilten Softwareentwicklungsprojekten wird die Quellcodeentwicklung dezentral durch verteilte Teams durchgeführt. Aktuelle Testtechniken und -verfahren, (wie Blackbox Tests, Glassbox Tests, Unit Tests und so weiter [Gies05]) die der Qualitätssicherung von in-house Software-

entwicklungsprojekten dienen, werden in verteilten Softwareentwicklungsprojekten weiterhin unverändert angewandt ([BCKS01], [SeCS06]). Der dezentrale Ansatz der Softwareentwicklung weckt allerdings neue Fragen bezüglich des Koordinierungsbedarfs des Testens für verteilte Teams und den Auftraggeber. Auf diese Fragen wird in Kapitel 7 eingegangen.

## 2.5 Analysewerkzeuge

Analysewerkzeuge dienen einer statischen Analyse der Quellcodestruktur, in dem der Quellcode mit Hilfe eines Metamodells in eine Datenbank eingelesen wird. Das Metamodell ermittelt Verzeichnisse, Quellcodedateien, Pakete, Klassen, Referenzen zwischen Klassen, Methoden und Attribute des eingelesenen Codes. Anschließend werden die ermittelten Daten in einem Systemmodell abgelegt, das werkzeugabhängig ist. Durch Anfragen an das Systemmodell können Analysen durchgeführt werden, die das Ziel der Verbesserung der Quellcode-Qualität verfolgen. Die Bildschirmdarstellung der durchgeführten Analysen kann sich je nach Benutzerrolle unterschiedlich anpassen. Während ein Projektleiter wichtige Kennzahlen benötigt, interessiert sich zum Beispiel ein Entwickler für die Analyse auf der Codeebene. Die Ergebnisberichte werden in der Regel mit Hilfe eines Browsers über ein Webinterface betrachtet [SiSM06].

Mit den analysierten Daten kann ein Know-how Transfer auf neue Entwickler schneller erfolgen oder Arbeitsverläufe besser kontrolliert werden. Letztlich geben Analysewerkzeuge Auskunft über die Wartbarkeit des Quellcodes. Analysen auf Modul- oder Dateiebene geben zusätzliche Hilfe für Entwickler und Softwarearchitekten. Das Aktualisierungsintervall der Quellcodestruktur kann die Aktualität der Analysen gefährden. CAST, Sotograph und SISSy sind Analysewerkzeuge, die nach der oben aufgeführten Architektur arbeiten. CAST (Abschnitt 2.5.1) und Sotograph (Abschnitt 2.5.3) sind kommerzielle Werkzeuge, während SISSy (Abschnitt 2.5.2) eine kostenfreie Lösung darstellt.

## 2.5.1 CAST

Der Schwerpunkt von CAST Application Intelligence Platform<sup>(14)</sup> liegt auf der Ebene der Unternehmensführung mit einem bewussten Einsatz der statischen Codeanalyse. Die analysierten Ergebnisse werden in der webbasierten Governance Dashboard (siehe Abbildung 8) graphisch dargestellt. In diesem Dashboard können die verschiedenen Projektteilnehmer, vom Management bis zum Entwickler, auf die gewünschte Information zugreifen. CAST analysiert eine große Vielzahl von Programmiersprachen, was der größte Vorteil dieser Lösung ist.

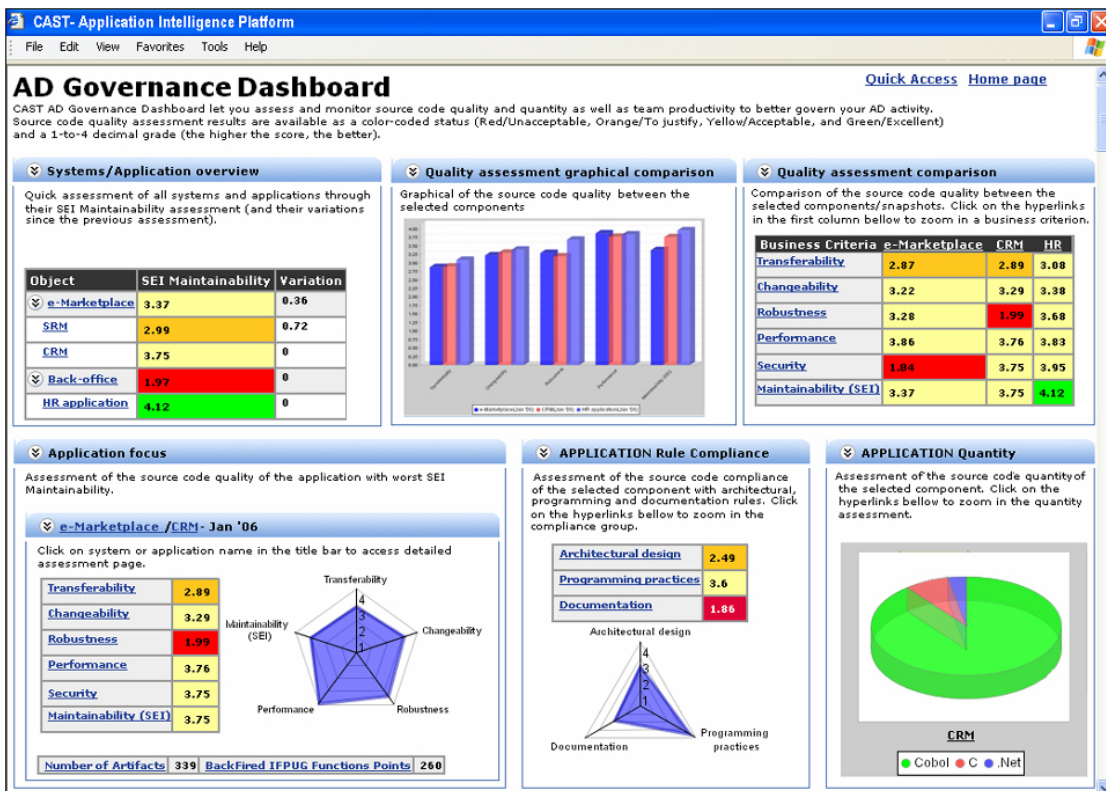


Abbildung 8. Die Governance Dashboard von CAST. <sup>(15)</sup>

CAST wird mit eigenen, vorkonfigurierten Qualitätsindikatoren geliefert, die aber an die firmeninternen Kategorien und Konventionen angepasst werden können. Qualitätsmerkmale, die in der Norm ISO 9126 definiert sind, werden zusätzlich analysiert. Dieses Werkzeug erlaubt sämtliche Vergleiche zwischen verschiedenen Applikationen und Versionen des Projektes. In verteilten Umgebungen kann der Auftraggeber somit die Qualität der Arbeiten der

<sup>14</sup> URL: <http://www.castsoftware.com/> (Abruf 06.04.2007)

<sup>15</sup> Quelle: <http://www.castsoftware.com/> (Abruf 06.04.2007)

verteilten Teams nachprüfen und durch die automatisierte Analyse des produzierten Quellcodes die Effektivität der Arbeiten im Team verbessern.

CAST ist eine Lösung, die für große und mittelgroße Projekte gedacht ist. Die Mindestanzahl für die Einführung dieses Werkzeuges liegt zwischen 15 und 20 Entwickler. Zukünftige Versionen von CAST werden auf der einen Seite mehr Programmiersprachen und Technologien unterstützen, auf der anderen Seite eine bessere Analyse des Codes entwerfen.

### **2.5.2 SISSy**

Das Werkzeug SISSy<sup>(16)</sup> (Structural Improvement of Software Systems) wurde am Forschungszentrum Informatik (FZI) in Karlsruhe entwickelt und unterstützt die Programmiersprachen Java, C++ und Delphi. Dieses Werkzeug verwendet je nach Programmiersprache so genannte Faktenextraktoren (Fact extractors), die das zugehörige Systemmodell und Artefakte aus den Quellcodedateien extrahiert [SiSM06] (siehe Abbildung 9). Die Qualitätsmerkmale Wartbarkeit, Erweiterbarkeit, Wiederverwendbarkeit, Verständlichkeit, Robustheit und Zuverlässigkeit des Quellcodes können dadurch analysiert werden.

Ein großer Nachteil von SISSy ist, dass dieses Werkzeug keine grafische Benutzerschnittstelle unterstützt, sondern es kann nur über die Kommandozeile gesteuert werden kann. Die Entwickler dieser Lösung sehen darin den Vorteil, dass dadurch eine einfache Integration in ein vorhandenes Portal oder in die Build-Umgebung erfolgen kann. Zukünftige Entwicklungen von SISSy werden unter anderem weitere Programmiersprachen wie zum Beispiel C# unterstützen.

---

<sup>16</sup> URL: <http://sissy.fzi.de/> (Abruf am 07.04.2007)

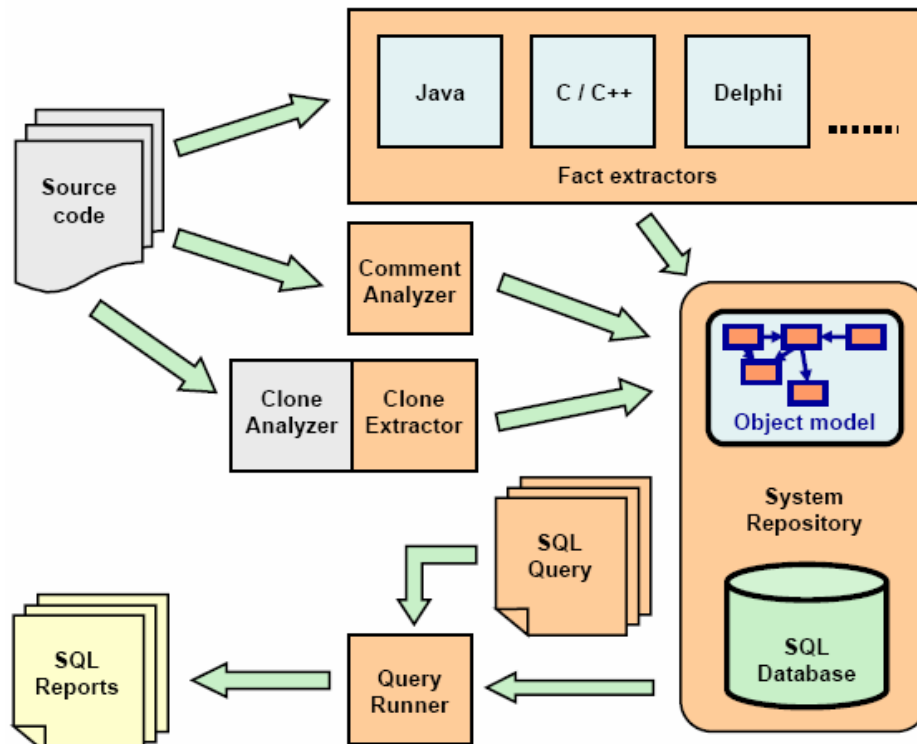


Abbildung 9. Architektur von Sissy<sup>(17)</sup>

### 2.5.3 Sotograph

Sotograph ähnelt dem Konkurrenzwerkzeug CAST. Dieses kostenpflichtige Werkzeug wurde von der Firma Software-Tomography GmbH<sup>(18)</sup> 2003 entwickelt und ermöglicht die Analyse von Java, C#, C und C++ Quellcodedateien. Das Werkzeug kann auch verschiedene Software-Versionsstände vergleichen und dadurch Entwicklungstrends aufzeigen. Dies ermöglicht eine zeitsparende, projektbegleitende Qualitätsbewertung im Hinblick auf Wartbarkeit, Erweiterbarkeit und Verständlichkeit eines Systems [SiSM06].

Die Sotograph-Familie (siehe Abbildung 10) beinhaltet neben dem Hauptprodukt Sotograph die Zusatzprodukte Sotoweb und Sotoreport. Diese Werkzeuge stellen die Qualitätsdaten aus dem Sotograph-Repository in anderer Form und Aufbereitung bereit. Während Sotoreport ein Reportgenerator ist, der automatisiert und regelmäßig Berichte zur Situation der

<sup>17</sup> Quelle: <http://sissy.fzi.de/> (Abruf am 07.04.2007)

<sup>18</sup> URL: <http://www.software-tomography.de/index.html> (Abruf am 07.04.2007)

Softwarequalität und der -architektur liefert, liefert das webbasierte Werkzeug Sotoweb aktuelle Softwarequalitätsdaten für das gesamte Projektteam.

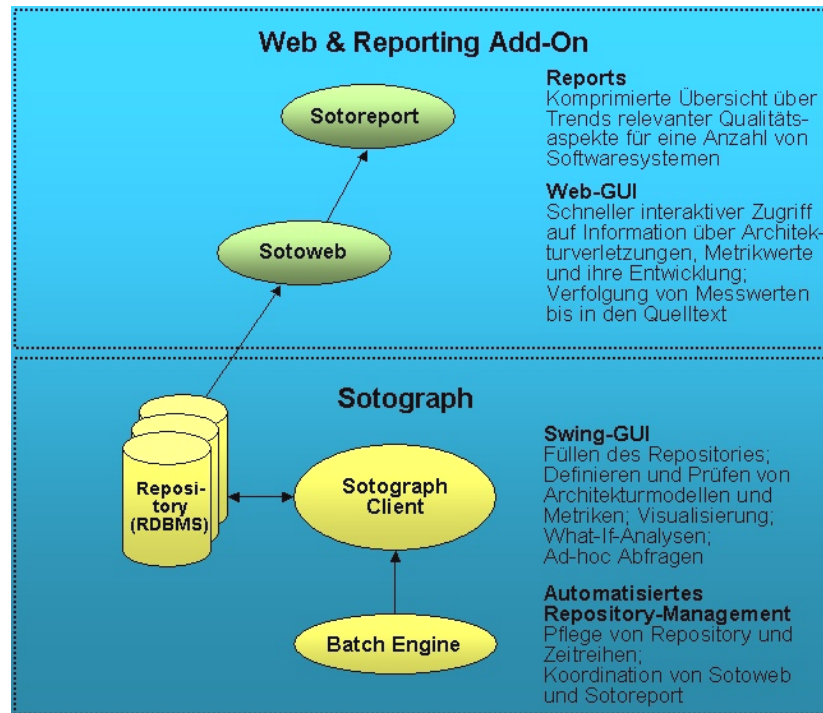


Abbildung 10. Sotograph-Familie<sup>(19)</sup>

## 2.6 Produkt- und Prozessverbesserung

Die drei vorgestellten Analysewerkzeuge beschäftigen sich mit dem verfügbaren Quellcode eines bestimmten Projektes. Diese Werkzeuge helfen die Software zu verbessern. Die Qualität des verfügbaren Quellcode wird dabei gemessen und analysiert, um punktuelle Verbesserungen durchführen zu können. Bei diesen Werkzeugen muss aber der Quellcode bereits existieren und ist daher der Ausgangspunkt der Analyse. Wenn aber ein komplett neues Projekt entstehen soll, können diese Werkzeuge keine Aussage darüber treffen, wie die Produktqualität für das neue Projekt verbessert werden kann. Obwohl die vorgestellten Normen und Metriken einige Kritik einstecken müssen [AICK05], sind diese im Laufe der Jahre reifer geworden und haben seine Wichtigkeit im Projekt behauptet.

<sup>19</sup> Quelle: <http://www.software-tomography.de/index.html> (Abruf am 07.04.2007)

Um die Qualität der Software zu verbessern und eine bessere Unterstützung der Prozessqualität zu erreichen, wird empfohlen die Planbarkeit, die Transparenz und die Kontrollierbarkeit des Prozesses zusätzlich zu unterstützen [Balz01]. Prozesse innerhalb eines verteilten Projektes verbessern sich oft nur innerhalb eines einzelnen Teams an einem bestimmten Standort, das heißt diese Verbesserungen finden nur lokal statt und verbessern dabei nicht den Gesamtprozess des Projektes. Die Effektivität und Erfolg der Koordinierungsarbeiten wird durch die Fähigkeit Entscheidungen zu treffen bestimmt, um zukünftige Probleme zu minimieren [HeGr99b]. Mit einer besseren Koordinierung der Prozesse werden kürzere Wege zwischen den Teams aufgebaut, Entwickler werden untereinander mehr integriert und Best Practices überall verbreitet [HeGr99a].

### 3 Prozessrahmenwerke in der verteilten Entwicklung

Bei der Entwicklung von Software reicht es nicht nur eine individuelle Sicht der Softwareentwicklungskomponenten zu haben und diese zu bewerten. Es soll viel mehr der komplette Entwicklungsprozess durch alle Ebenen der Organisation betrachtet [CuKO92] und mit einem Standard für den Softwareentwicklungsprozess unterstützt werden [Balz01]. Für das Management sind standardisierte Prozesse der Schlüssel zu einer besseren Projektübersicht. Der Vorteil einer standardisierten Prozesssteuerung liegt in der Wiederholung und Standardisierung der Tätigkeiten.

Im letzten Kapitel wurden unter anderen das CMMI und ITIL vorgestellt (siehe Kapitel 2.2). Qualitätsmodelle wie das CMMI sind in einer Zeit entstanden, in der die Projekte hauptsächlich vor Ort durchgeführt wurden. Um CMMI an die verteilte Softwareentwicklung zu adaptieren, wurden neue KPAs<sup>(20)</sup> entwickelt und diese an die vorhandene Struktur des Qualitätsmodells angepasst. ITIL hat dabei den Vorteil, dass das Rahmenwerk sich sowohl an die Arbeit mit internen als auch externen Dienstleistern anpassen lässt [Kirk05], was bedeutet, dass dieser Ansatz auch für verteilte Softwareprojekte skalieren kann.

#### 3.1 Neue KPAs für CMMI

CMM Modelle und die dazu gehörenden KPAs wurden in seiner ursprünglichen Version für lokale und nicht für verteilte Projekte definiert [SeCS06]. Die Umgebung der Projekte hat sich im Laufe der Jahre geändert und nicht alle KPAs lassen sich jetzt an die verteilte Softwareentwicklung anpassen. Der gegenseitige Wissensaustausch und die Wissensverteilung der verteilten, sozialen Netzwerke werden von traditionellen CMMs mit den aktuellen KPAs zum Beispiel nicht unterstützt [RaKK05]. Die Definition von neuen KPAs für die verteilte Softwareentwicklung soll aufgestellt werden.

---

<sup>20</sup> Eine KPA ist eine kleine Komponente der Softwareentwicklung und beinhaltet eine Zusammenfassung von Anforderungen zu einem bestimmten Thema (siehe Abschnitt 2.2.1)

Heutzutage existieren zahlreiche Unternehmen, die CMMI eingeführt haben und erfolgreich damit werben. Im asiatischen Raum ist CMMI sogar sehr verbreitet ([Ju06], [Matl05]). Wenn die Entwicklung der letzten Jahre der Länder wie China, Indien oder USA betrachtet wird, wird festgestellt, dass die Benutzung von CMMI stark zugenommen hat, was nicht unbedingt an der Qualität der Ergebnisse gekoppelt ist (siehe Abschnitt 3.3). Tabelle 1 zeigt die Entwicklung dieser Länder. Die chinesische Regierung subventioniert zum Beispiel bis zu 50% der Zertifizierungskosten, was auch anhand eines schnellen Wachstums der Zertifizierungen dieses Landes zu sehen ist [Ju06].

Land	März 2002	März 2004	März 2005
China	18	152	243
Indien	153	330	387
USA	1498	1838	1947

Tabelle 1. Zahl der CMMI-Überprüfungen<sup>(21)</sup> („Appraisals“). [Ju06]

Um verteilte Umgebungen besser zu unterstützen, hat die SAP AG mit der Unterstützung von der University of Michigan 24 neue KPAs definiert [RaKK05]. Ein Ziel dieser Zusammenarbeit waren die verteilten Geschäfts- und Projektziele der verschiedenen Teilnehmer zu regeln. Zusätzlich sollten Best Practices der verteilten Entwicklung identifiziert und standardisiert und Richtlinien für die Weiterleitung von gemeinsamem Wissen und informeller Kommunikation erarbeitet werden.

Diese neuen 24 KPAs wurden in drei Ebenen und in 4 Konzepte aufgeteilt (siehe Abbildung 12). Die drei Ebenen sind kompatibel mit den CMMI-Reifegraden: Initialgrad, Konsolidierungsgrad und hoher Produktivitätsgrad.

---

<sup>21</sup> Aktuelle Zahlen sind unter der folgende Adresse zu finden:  
<http://www.sei.cmu.edu/appraisal-program/profile/profile.html> (Abruf am 26.03.07)

Die vier Konzepte umfassen folgende vier Eigenschaften [RaKK05]:

1. Zusammenarbeit und Kollaboration, um Geschäfts- und Projektziele zu definieren („collaboration readiness“).
2. Gemeinsame Interessen, um das vorhandene Wissen über verteilte Standorte weiterzuleiten. Dabei wird das gemeinsame Wissen über das Produkt, die Firma und der verschiedene Arbeitsstil ausgetauscht („mutual knowledge“).
3. Arbeitskopplung, um die Mechanismen der Arbeit aufzuteilen und zu verteilen. („coupling in work“).
4. Bereitschaft eine neue Technologie, eine neue Entwicklungsumgebung, eine neue Infrastruktur, neue Tools oder Werkzeuge einzusetzen („Technology readiness“).

Um die KPAs zu validieren, wurden Gespräche und Fragebögen bei fünf Unternehmen durchgeführt. Diese fünf ausgewählten Unternehmen hatten bereits Erfahrungen in der verteilten Softwareentwicklung gesammelt [RaKK05]. Die Experten wurden gefragt in welcher Ebene der Prozessverbesserung die KPAs eingestuft werden sollten. Nur in drei Fällen gab es zusätzlich Untersuchungsbedarf, alle anderen KPAs wurden von den Unternehmen bestätigt. Die in Abbildung 11 eingekreisten KPAs, nämlich „Managerial training“ und „Nurturing and leveraging core competencies“, wurden weiterhin verwendet [RaKK05]. Die dritte KPA „Top-management communication channels“ wurde aus der Liste entfernt, da die Abweichung zwischen dem Ansatz und der Einschätzung der Befragten zu groß war<sup>(22)</sup>:

*“However, three KPAs - managerial training, top management communication channels, and nurturing and leveraging core competencies - didn't meet the acceptable cut-off level of 75 percent.” [RaKK05]*

---

<sup>22</sup> In Abbildung 11 ist diese KPA mit einem Kreuz markiert.

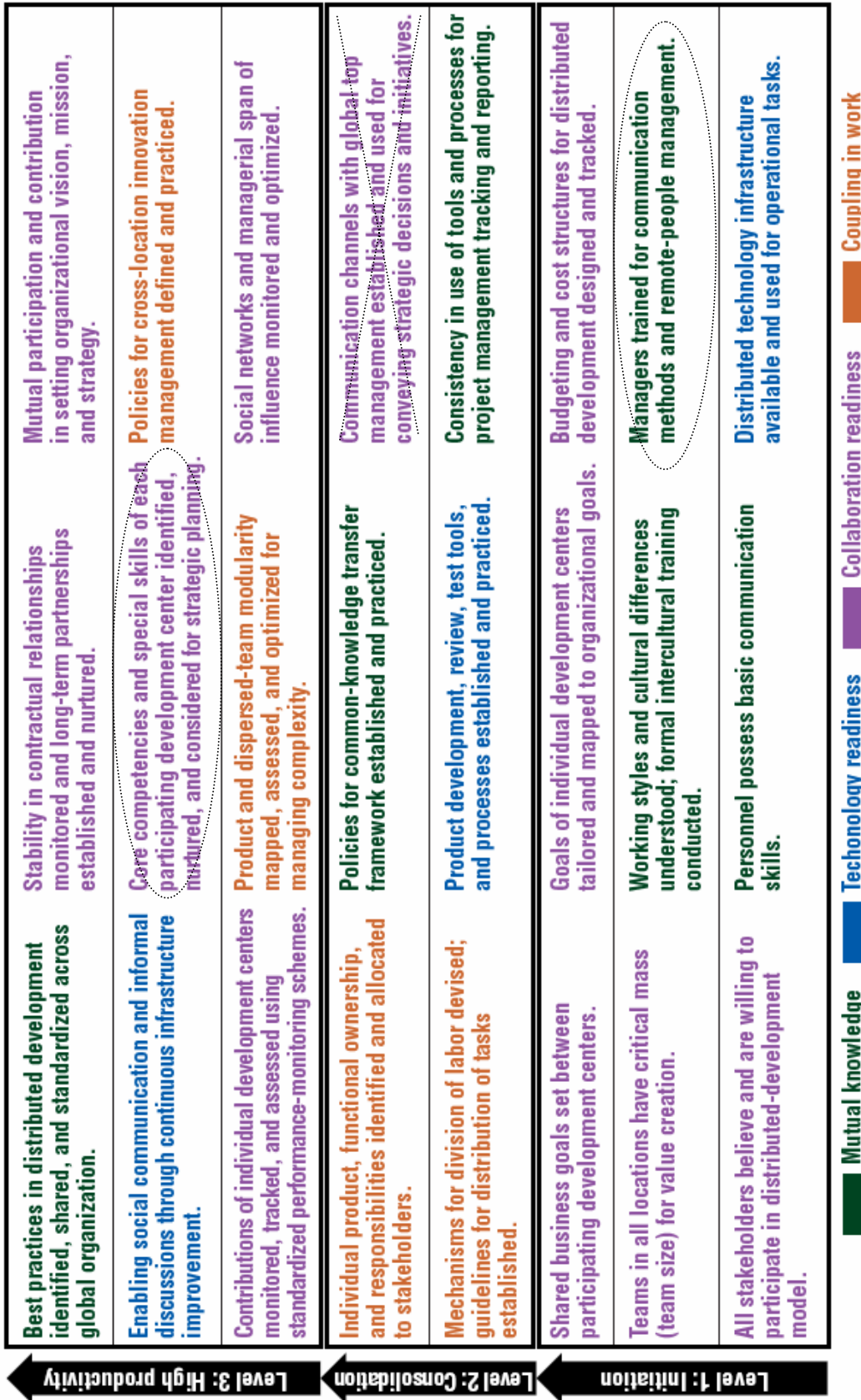


Abbildung 11. 24 Neue KPAs und die vier Konzepte [RaKK05]

### 3.2 ITIL - Der Best-Practice-Ansatz

ITIL hilft Geschäftsbereiche zu identifizieren und zu entscheiden, welche dieser Bereiche tatsächlich zu den Kernkompetenzen gehören und welche nicht. Wenn ITIL in einem verteilten Softwareprojekt eingesetzt wird, müssen zuerst auf der Auftraggeberseite die Kernkompetenzen des Unternehmens identifiziert werden (siehe dazu Abschnitt 4.5.2, 5.1.2 und 7.1). In einer zweiten Phase werden eine Reihe von Bedürfnissen anhand von gesammelten Dokumenten und Empfehlungen ausgearbeitet. Bereiche, die nicht Kern des Geschäftes sind, können dann gegebenenfalls ausgelagert werden. Ziel ist es, die nötigen Entscheidungen so zu treffen, so dass der Auftragnehmer mindestens genauso gute Lösungen wie bei einer internen Abteilung des Auftraggebers liefern kann. ([HHXJ06], [Kirk05]).

Ein Vorteil von ITIL ist, dass das Rahmenwerk sich sowohl an die Arbeit mit internen als auch externen Dienstleistern anpassen lässt [Kirk05], was bedeutet, dass dieser Ansatz auch für verteilte Softwareprojekte skalieren kann. ITIL versucht nicht nur die Qualität der Arbeit zu verbessern, sondern auch die Arbeit durch eine verbesserte Kundenorientierung effizienter zu gestalten, um die Qualität der Dienstleistungen zu erhöhen [HoZB05b]. Die Standardisierung, kontinuierliche Verbesserung und Automatisierung der Prozesse durch ITIL erhöht zusätzlich die Effizienz der Arbeiten. Wenn verschiedene Abteilungen gleiche Prozesse ausführen, kann ineffiziente Arbeit in einer bestimmten Abteilung einfacher identifiziert werden. Mit ITIL wird die Transparenz und Kompatibilität der Prozesse durch eine ständige Prozessdokumentation und -kontrolle erreicht.

ITIL ist sehr vorteilhaft für die Verbesserung von Qualität, erfordert aber stark involvierte Mitarbeiter. Der Wille von Mitarbeitern und Abteilungen sich an den neuen Standard anzupassen, kann zu Ineffizienz des Rahmenwerkes führen. Mitarbeiter glauben immer fest daran eine gute Arbeit zu leisten und verstehen die Einführung von ITIL als eine persönliche Kritik an ihre Arbeit und sehen zuerst die erhöhte bürokratische Arbeit und die fehlende Individualität mehr als eine Bremse als einen Vorteil ([HoZB05a], [HoZB05b]).

### 3.3 Zusammenspiel der Prozessrahmenwerke

Bei jedem Projekt ist es sehr wichtig eine gute Mischung aus Disziplin, Planung und Agilität zu finden [BoTu03]. In Kapitel 4.5 wird diesbezüglich das Entscheidungsmodell von Boehm und Turner erklärt. Da nicht nur Dokumente, sondern auch Quellcode erzeugt werden soll, ist es immer wichtig die Produktqualität und die Produktivität der Softwareentwicklung zu verbessern, um die strategischen Ziele erreichen zu können. Mit der Benutzung eines Prozessrahmenwerkes wird die Softwareentwicklung an einen vordefinierten Prozess angepasst.

Viele Firmen streben heutzutage nach dem höchsten Reifegrad eines gegebenen Prozesses als Qualitätsmerkmal. Dieses Verhalten ist aber kein Ersatz für die vorhandene Erfahrung. Zum Beispiel verfügt Indien über eine gute Marketing Kampagne bezüglich des Einsatzes von CMMI. Mehr als die Hälfte der Firmen weltweit, die die Stufe 5 erreicht haben, haben ihren Sitz in Indien [Matl05], was nicht unbedingt an der Qualität der Ergebnisse gekoppelt ist:

*„ A study of 89 level-5 companies [...] found that, on average, the code produced by level-5 firms had higher error rates than those of non-CMM-rated firms.”<sup>(23)</sup>*

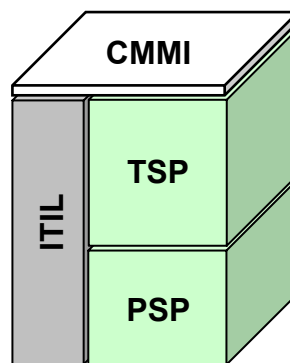
CMMI v1.1. beschäftigt sich mit der Prozessverbesserung der gesamten Organisation. Dieses Qualitätsmodell wurde in erster Linie für lokale Projekte gedacht, die einen Planungsgetriebenen Ansatz verfolgen. Um verteilte Projekte besser zu unterstützen sind einige Lösungen entwickelt worden. Als eine Alternative wurden 24 neue KPAs vorgestellt, die die Prozesse verteilter Teams unterstützen [RaKK05]. Die aktuelle CMMI Version, nämlich CMMI v1.2. (CMMI-DEV)<sup>(24)</sup> gibt Auskunft über die Unterstützung verteilter Projekte. Diese Version ist aber sehr neu (August 2006) und noch nicht sehr verbreitet.

---

<sup>23</sup> Christopher Koch, “Bursting the CMM Hype,” CIO, 1 Mar. 2004.  
URL: <http://www.cio.com/archive/030104/cmm.html>. (Abruf am 29.03.2007)

<sup>24</sup> URL: <http://www.sei.cmu.edu/publications/documents/06.reports/06tr008.html>  
(Abruf am 29.03.2007)

Wenn ein Softwareprojekt verteilt entwickelt wird, kann es vorkommen, dass verschiedene Standards eingesetzt werden oder ein externer Dienstleister sogar kein Standard benutzt. In solchen Fällen müssen Kompromisse eingegangen werden, um die Unterstützung der Prozesse zu gewährleisten. Eine Studie zeigt [PiAP06], dass die Benutzung verschiedener Softwarekonfigurationssysteme Probleme bei der Quellcodeentwicklung verursachen können. Ein lokales Team hatte ein funktionierendes Softwarekonfigurationssystem, ein anderes verteiltes Team aber nicht. Viele Stunden gingen für die langen Synchronisierungszyklen verloren. Eine Lösung ist, Prozesse und Standards von allen Projektteilnehmern zusammenzuführen, um die Arbeiten für alle verständlicher zu machen.



**Abbildung 12. Zusammenspiel der vorgestellten Prozessrahmenwerke**

Abbildung 12 zeigt ein mögliches Szenario für die Benutzung von den vorgestellten Prozessrahmenwerken. Während CMMI das gesamte Unternehmen in Betracht zieht und für große Unternehmen gedacht ist, konzentriert sich ITIL in der Unterstützung des IT-Managements. ITIL kann nicht nur für große Unternehmen vom Nutzen sein. Auch kleine und mittelständische Unternehmen (KMU), bei denen ein Standard wie CMMI zu teuer oder nicht praktikabel scheint, können ITIL einsetzen. Solche Unternehmen arbeiten nur bestimmte Bereiche des Ansatzes aus, zum Beispiel im Bereich der Service-Unterstützung können erhebliche Verbesserungen erreicht werden [BMC06]. PSP und TSP konzentrieren sich zusätzlich auf die einzelnen Entwickler beziehungsweise auf die Teams und können immer für die kontinuierliche Verbesserung der verschiedenen Teams genutzt werden.

## 4 Vorgehensmodelle und Methoden für die Softwareentwicklung

Im Laufe der letzten dreißig Jahre sind mehrere Vorgehensmodelle und Methoden entwickelt worden, die den Prozess der Softwareentwicklung beschreiben und unterstützen. Dieser komplexe, kontinuierliche und iterative Prozess der Softwareentwicklung wird von Barry Boehm [Boeh88] definiert als

*„... ein Verfahren, das zur Feststellung der Ordnung der einzelnen involvierten Phasen bei der Softwareentwicklung und zur Festlegung der Übergangskriterien von einer Phase zur Nächsten dienlich ist“.*

Mithilfe Vorgehensmodelle und Methoden werden innerhalb eines jeden Projektes die zwei folgenden Fragen beantwortet:

1. „Was soll als nächstes getan werden?“
2. „Wie lange soll es getan werden?“

Vorgehensmodelle und Methoden helfen Projekte eine gewisse Ordnung einzuhalten. Die Entwicklungsphasen sind die Leitlinien für jedes Projekt, damit die Abwicklung eines Projektes sinngemäß erfolgt. In diesem Kapitel werden Vorgehensmodelle und Methoden vorgestellt und es wird gezeigt welche Änderungen diese in den letzten Jahren durchlaufen haben. Die Entstehung dieser Vorgehensmodelle und Methoden wird erläutert und die zwei meist verbreiteten Richtungen werden vorgestellt [LaBa03], nämlich die bekannte Planungsgetriebene Softwareentwicklung und die etwas neueren Agilen Methoden. Es wird untersucht, wie gut sich diese Vorgehensmodelle und Methoden an die verteilte Welt anpassen lassen. Am Ende dieses Kapitel wird ein Entscheidungsmodell vorgestellt, das besagt, auf welche Weise beide Konzepte vereint werden können, um zu höchster Effektivität beziehungsweise Qualität im Rahmen der Softwareentwicklung zu gelangen.

## 4.1 Phasen der Softwareentwicklung

Helmut Balzert [Balz01] definiert sechs Phasen der Softwareentwicklung:

1. Die Planungsphase
2. Die Definitionsphase
3. Die Entwurfs- oder Designphase
4. Die Implementierungsphase
5. Die Abnahme- und Einführungsphase
6. Die Wartungs- und Pflegephase

In der Planungsphase wird das Produkt definiert und ausgewählt. Eine Durchführbarkeitsstudie, mit Untersuchungen der Ziele und ökonomischen Faktoren, wird gemacht. Das Ergebnis dieser Phase ist das Lastenheft, das die Projektkalkulation, der Projektplan, die Produktfunktionen, -daten und -leistungen und die Qualitätsanforderungen beinhaltet. In einem Pflichtenheft werden die Zielbestimmungen, Produkteinsatz, -umgebung, -funktionen, -daten und -leistungen fest gehalten. Dazu werden noch die Qualitätsanforderungen, die Benutzeroberfläche, die nichtfunktionale Anforderungen, die globalen Testszenarien und die Entwicklungsumgebung definiert. Nicht zu vergessen in diesem Pflichtenheft sind die Ergänzungen und ein Glossar oder Begriffslexikon als Anhang um Begriffe klar definiert zu haben.

In der Definitionsphase wird nach dem „was wird gebaut?“ gefragt. In der Entwurfsphase folgt nun das „wie wird es gebaut?“. Es wird eine Softwarearchitektur beziehungsweise das erste Design entwickelt, die die funktionalen und nichtfunktionalen Produkthanforderungen sowie allgemeine und produktspezifische Qualitätsanforderungen erfüllt und die Schnittstellen zur Umgebung versorgt [Balz01]. In der Implementierungsphase wird die Software programmiert, dokumentiert und getestet aufgrund vorgegebener Spezifikationen der Systemkomponenten. In dieser Phase wird versucht, dass die Entwickler sich selbst kontrollieren und eigenen Entwicklerichtlinien verfolgen, um die Zusammenarbeit zu vereinfachen und aus eigenen Fehlern lernen zu können.

In der Abnahme- und Einführungsphase wird die fertig gestellte Software abgenommen und beim Anwender eingeführt und in Betrieb genommen. Das Gesamtprodukt inklusive Dokumentation wird an den Auftraggeber übergeben. Dabei wird ein Abnahmetest durchgeführt. In diesem Test wird geprüft, ob die in dem Pflichtenheft definierten Qualitätsmerkmale erfüllt worden sind. Die Software wird beim Kunden installiert und Benutzer werden geschult. Die Wartungs- und Pflegephase zeichnet sich durch die korrektiven Tätigkeiten aus, um Anpassungen, Erweiterungen, Korrekturen, Verbesserungen oder Aktualisierungen der Software durchzuführen.

### 4.2 Kurze Geschichte der Vorgehensmodelle

Die oben definierten Phasen sind in jedem Softwareprojekt vorhanden, es ist aber üblich Projektphasen miteinander zu verschmelzen oder parallel zu führen. Nichts desto trotz verfolgen alle Phasen den oben genannten Pfad: Planung, Definition, Entwurf, Implementierung, Testen, und Einsatz und Wartung. Dieses Vorgehen für die Softwareerstellung wurde in den siebziger Jahren formal definiert und es entstand die Planungsgetriebene Softwareentwicklung. Royce [Royce70] definierte 1970 ein Phasenmodell, um die Softwareentwicklung zu unterstützen. Das Royce-Modell entwickelte sich in das bekannte Wasserfallmodell (siehe Abschnitt 4.3.1) weiter, das als eine einfache, verallgemeinerte Version des Royce-Modells gilt. Das Spiralmodell (siehe Abschnitt 4.3.2) entstand einige Jahre später als eine Verbesserung des Wasserfallmodells. Die Kritik an dem Modell von Royce war die geringe Flexibilität des Ansatzes:

*“All of us, as far as I can remember, thought waterfalling of a huge project was rather stupid or at least ignorant of the realities... I think what the waterfall description did for us was make us realize that we were doing something else, something unnamed except for “software development.”<sup>(25)</sup>*

---

<sup>25</sup> Gerald M. Weinberg. [LaBa03]

Parallel dazu entwickelte sich ein anderes Vorgehensmodell: die inkrementelle und iterative Softwareentwicklung (IIE). Die ersten offiziellen Beschreibungen dieser parallelen Entwicklung kommen aus den sechziger Jahren [LaBa03].

*“The fundamental idea was to build in small increments, with feedback cycles involving the customer for each.” [Wein80]*

IIE gewann mit der Zeit mehr Akzeptanz, da dieser Ansatz die Vermeidung eines sequentiellen Ablaufs darstellte. Nichts desto trotz blieb das Wasserfallmodell parallel auf dem Markt als ein allgemein anerkannter Standard für Softwareprojekte. Das amerikanische Verteidigungsministerium (Department of Defense) hatte Jahrelang dieses Modell unterstützt, änderte aber 1987 seine Standards für die Softwareentwicklung. Zu dem traditionellen Wasserfallmodell wurden zusätzlich Ansätze angenommen, die die IIE unterstützten. Anfang der neunziger Jahre wurden die Agilen Methoden definiert, die im Abschnitt 4.4 näher erklärt werden. Diese Methoden bauen auf der IIE auf. Der heutzutage bekannteste „Nachfolger“ dieser Methoden ist Extreme Programming (XP). Die Agilen Methoden werden heutzutage als der neue und viel versprechende Weg für die moderne Softwareentwicklung gesehen.

### **4.3 Planungsgetriebene Entwicklung**

Die Planungsgetriebene Softwareentwicklung wird als der traditionelle Weg gesehen, Software zu entwickeln [BoTu04b]. Dieser Ansatz wurde während des Kalten Krieges definiert, um die fehlende Planung der Softwareentwicklung zu bekämpfen. Militärische Projekte brauchten Standards für die Softwareproduktion, da Projektkomponenten von verschiedenen Firmen geliefert wurden. Es wurden Modelle entwickelt, die als Basis die Phasen der Softwareentwicklung besaßen. Für die Hardwareprogrammierung funktionierte dieses Vorgehen gut, aber für die Software, ein von Natur aus inhärentes Produkt, galt das nicht (immer). Um eine Phase abschließen zu können, wurden vollständig ausgearbeitete Dokumente gebraucht, um mit der nächsten Phase

weiter machen zu können. Zum Beispiel wurde ein vollständiges und ausgearbeitetes Pflichtenheft benötigt, um mit der Entwurfsphase zu starten. Für einige Softwarebereiche, wie Übersetzerbau oder für sichere Betriebsumgebungen, war dieser Ansatz erfolgreich. Es gab aber einige andere Bereiche, wie Applikationen für Endbenutzer, bei denen dieser Ansatz, bedingt durch seine Starrheit, Probleme verursachte [Boeh88].

Dieser Ansatz wurde trotzdem als Standard vom amerikanischen Verteidigungsministerium verabschiedet, und wurde sowohl vom Militär als auch von großen Firmen wie IBM, Hitachi oder Siemens benutzt, da sie die Software als Prozess für formale mathematische Spezifikation und Verifikation sahen. Die beiden wichtigsten Vertreter dieses Ansatzes sind das Wasserfall- und das Spiralmodell, die im Folgenden erklärt werden.

### **4.3.1 Das Wasserfallmodell**

Das Wasserfallmodell gehörte zu den Pioniermodellen der Softwareentwicklung und wurde 1970 von Winston Royce definiert. Er entwickelte, im Kontext verschiedener Regierungsprojekte der sechziger und siebziger Jahre, ein einfaches Modell für einfache Projekte. Royce empfiehlt in seinem Artikel [Royc70], dass ab der Designphase, auch als Entwurfsphase bekannt, circa ein Drittel des Projektaufwands für ein Pilotprojekt investiert werden soll, wenn das Projekt als neu eingestuft wird. Das Pilotprojekt hilft zum Beispiel eventuelle Lücken des Entwurfes aufzudecken, somit bekommt der Kunde nicht die erste, sondern die zweite Version des Produktes, das heißt eine verbesserte Version, geliefert. Damit wollte Royce Fehler und Probleme der ersten Version beheben, und die Qualität der zweiten Version steigern. Diese Idee zeigt eine Annäherung an die Thematik der inkrementellen und iterativen Softwareentwicklung, bei der mehrere Iterationen während der Entwicklung vorkommen, mehr dazu im Abschnitt 4.4. Das Royce-Modell wurde aber nicht richtig verstanden und umgesetzt und mit der Zeit mutierte es in Richtung des aktuellen und starren Wasserfallmodells [LaBa03].

Wie im Abschnitt 4.1 definiert wurde, besteht ein Softwareentwicklungsprozess aus den Phasen: Planung, Definition, Entwurf, Implementierung, Testen, Einführung und Wartung. Im Wasserfallmodell wird für jede Phase eine Menge von Dokumenten benötigt, wie z. B. das Lastenheft, das Pflichtenheft, ein Kalkulationsplan und Benutzerhandbücher. Die ursprüngliche Idee von Royce wird in Abbildung 13 dargestellt. Hier wird gezeigt, dass erst die zweite Version der Software als Endversion benutzt werden sollte. Eine Art Prototyp sollte zuerst erstellt und auf Nützlichkeit und Funktionalität geprüft werden. Sofern in einer Phase Probleme auftreten, ist es möglich schrittweise eine Phase zurückzugehen, um die Probleme dort zu beheben. Die Phasen überschneiden sich aber nicht, das heißt eine Phase darf erst verlassen werden, wenn diese sich als vollständig abgeschlossen erweist, was der Nachteil von diesem Softwareentwicklungsmodell ist.

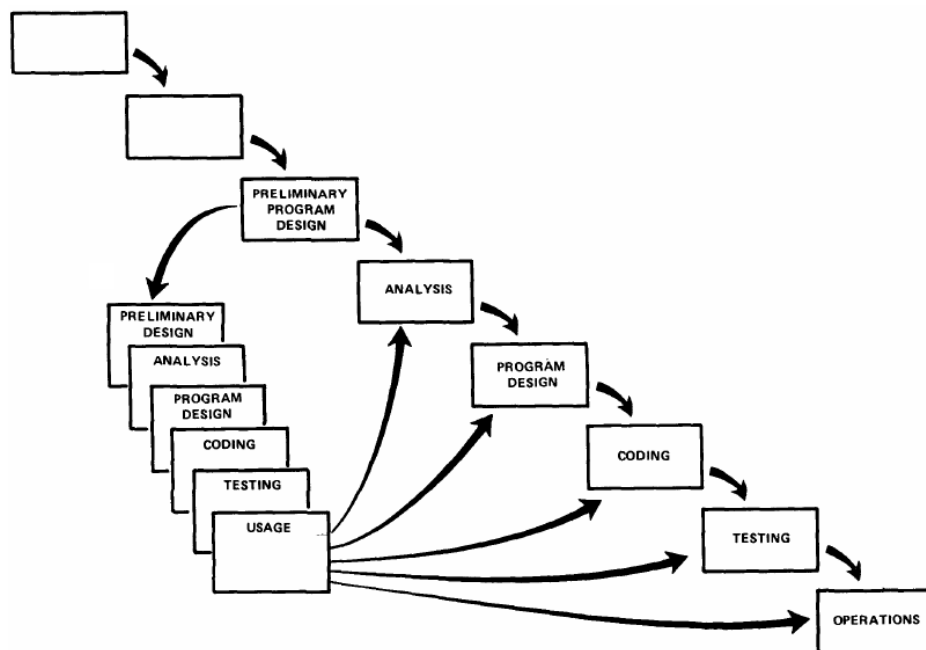


Abbildung 13. Wasserfallmodell nach Royce [Royc70]

Die größte Kritik am Wasserfallmodell ist, dass die Realität der alltäglichen Projekte nicht widerspiegelt wird [Wein80]. Im Modell gibt es zwischen Entwurf und Abgabe keine handfesten Ergebnisse, mit denen der Status des Projektes verfolgt werden kann. Es ist bis zur Abnahme des Produktes nicht sicher, ob die Kundenwünsche richtig verstanden wurden und ob das korrekte, passende Produkt entwickelt wurde. Es ist schwer bereits am Anfang eines Projektes die genauen Spezifikationsangaben bezüglich der Anforderungen

vom Kunden zu haben. Projektteilnehmer, die die Anforderungen an die Software schreiben, könnten nicht das nötige Wissen besitzen, um eine präzise Spezifikation zu schreiben. Manchmal sind es auch die Kunden, die nicht genau wissen, was sie erreichen wollen, aber eine grobe Idee haben, wie die Software funktionieren soll. Wenn die Produktspezifikation in der Planungsbeziehungweise Definitionsphase nicht korrekt erstellt wurde, ist das Modell in dieser Hinsicht nicht flexibel genug. Spezifikationsprobleme werden erst recht spät bemerkt und die Korrektur der Probleme erfordert eine große Investition, die meistens nicht im Voraus geplant ist. Wenn etwas falsch definiert worden ist, wird ein schlechtes Produkt kodiert und geliefert.

Das Wasserfallmodell funktioniert allerdings gut, wenn eine stabile Produktdefinition vorliegt und wenn die fachlichen Methoden verstanden worden sind. Das Modell ist auch geeignet für Wartungsarbeiten von existierenden Produkten und für die Systemportierung auf eine neue Plattform [McCo01].

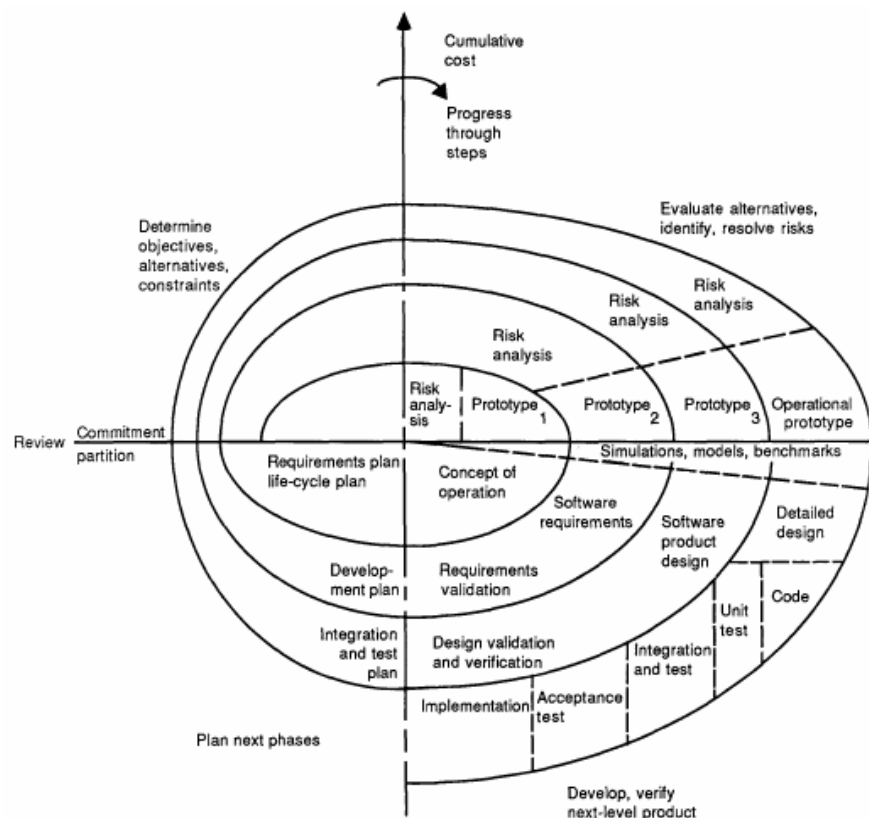
### **4.3.2 Das Spiralmodell**

Das Spiralmodell entstand als ein Verbesserungsvorschlag des Wasserfallmodells und sieht in der Dokumentation des Projektes keinen Schwerpunkt mehr. Dieses Risiko gesteuertes Modell basiert auf dem Versuch die verschiedenen Ausprägungen des Risikos im Projekt zu minimieren, indem ein bestimmtes Projekt in kleinere Teilprojekte untergliedert wird, so dass die Risiken für einen Projektabbruch minimiert werden.

Die verschiedenen Ausprägungen eines Risikos sind zum Beispiel schlecht definierte Anforderungen, eine falsch verstandene Architektur, eine falsch eingesetzte Technologie oder Performanceprobleme. Das Spiralmodell versucht diese Risiken zu minimieren, um sicher zu gehen, dass das was erreicht wurde tatsächlich korrekt ist. Erst dann kann die nächste Iteration angestoßen werden. Jede Iteration verfügt über folgende sechs Schritte, die projektabhängig anpassbar sind [McCo01]:

1. Ziele, Alternativen und Einschränkungen festlegen.
2. Risiken identifizieren und Lösungen erarbeiten.
3. Alternativen evaluieren.
4. Teilprojekte für die aktuelle Iteration entwickeln und verifizieren, dass diese auch korrekt sind.
5. Die nächste Iteration planen.
6. Ansätze für die nächste Iteration festlegen.

Abbildung 14 stellt das Modell in graphischer Form dar. Die radiale Dimension oder das Radialmaß zeigt die kumulierten Kosten auf, die bei jeder Iteration erzeugt werden. Das Winkelmaß stellt den Fortschritt des Projektes am Ende einer Iteration dar.



**Abbildung 14. Spiralmodell [Boeh88]**

Das Spiralmodell wird aufgrund einer Hypothese oder einer Projektidee über die Realisierung eines Softwareproduktes angestoßen. Diese Hypothese wird auf ihre Machbarkeit und Realisierbarkeit hin getestet und gegebenenfalls auch wieder verworfen oder verändert. Die angenommenen Hypothesen werden fortgesetzt und in verschiedene Teilprojekte untergliedert. Jedes Teilprojekt wird

am Ende der jeweiligen Iteration geprüft und seine Risiken analysiert und bewertet. Die Spirale gilt dann als beendet, wenn das Endprodukt vom Kunden abgenommen wird.

Der größte Vorteil des Modells basiert darauf, dass wenn im Laufe des Projektes die Kosten für die Produktentwicklung steigen, sich gleichzeitig die Risiken verringern, da diese während des Projektablaufes in kleinere Einheiten unterteilt worden sind. Diese Eigenschaft erleichtert die Managementkontrolle, da immer am Ende einer Iteration geprüft wird, ob Korrekturen erforderlich sind und somit eine frühzeitige Warnung der Risiken sichergestellt werden kann.

Die Kompliziertheit des Modells ist dessen größte Gefahr. Dieses Modell verlangt sehr viel Aufmerksamkeit von Seiten des Management, denn es ist nicht immer einfach, Ziele und Meilensteine zu definieren, die besagen, wann eine Iteration als erledigt gilt und wann mit der nächsten Iteration fortgefahren werden kann. Um Risiken als tatsächliche Risiken einzustufen, benötigt das Risikomanagement einige Projekterfahrungen auf diesem Gebiet.

### **4.4 Agile Methoden**

In den letzten zwei Jahrzehnten haben sich die Anforderungen an die Software und an die Softwareentwicklung stark verändert. Die so genannte New Economy, die Weiterentwicklung der neuen Technologien, die rasche Entwicklung des Internets als globales Kommunikationsmedium und die Globalisierung der Märkte haben bewirkt, dass Softwareprojekte immer schneller fertig sein müssen, um überhaupt noch eine Chance im laufenden Wettbewerb haben zu können. Dadurch darf allerdings die Qualität der Ergebnisse auf keinen Fall beeinträchtigt werden ([HeMo01], [LHKP03], [Scha06a]).

Neue Methoden versuchen sich an die verschiedenen Änderungen im Projekt anzupassen. In einigen Projekten wird heutzutage viel Flexibilität und schnelles Arbeiten der Entwickler bei der Problemlösung gesucht und manchmal auch

vorausgesetzt. Das Problem der Planungsgetriebenen Entwicklung ist, dass die Ergebnisse häufig erst spät im Projektablauf deutlich werden und dann in manchen Fällen nicht mehr den Kundenwünschen entsprechen. Der Markt verändert sich ständig, so dass immerzu neue Anforderungen notwendig sind. Der Begriff „Chaordisch“ stammt aus dem Englischen „chaordic“<sup>(26)</sup> und steht für das Zusammenspiel von Chaos und Ordnung in einer besonderen Organisationsform, bei der weder Chaos noch Ordnung dominieren. Die Agilen Methoden versuchen das Chaos der neuen globalisierten Welt durch die Ordnung der Prozessgestaltung zu unterstützen.

Die Agilen Methoden besitzen eine lange Entstehungsgeschichte, obwohl sie erst in den letzten Jahren durch das „Agile Manifest“<sup>(27)</sup> an Popularität gewonnen haben. Die ersten Versuche gab es schon in den sechziger Jahren mit Beginn der inkrementellen und iterativen Softwareentwicklung [LaBa03]. Die im Jahr 2001 von Kent Beck ins Leben gerufene „Agile Alliance“<sup>(28)</sup> soll als eine persönliche Einstellung bei der Softwareentwicklung, also als eine Philosophie verstanden werden, dass erfüllt werden soll.

Die Agilen Methoden besitzen Eigenschaften einer IIE. Sie sind iterativ, inkrementell und selbstorganisiert. Kurze Iterationen sind üblich in Projekten bei welchen Benutzer und Entwickler zusammen arbeiten, um das informelle Wissen stark unterstützen zu können und aufzubauen. Diese Methoden gehen davon aus, dass Anforderungen sich ständig ändern und sehen deshalb kurze Iterationen, um der Verfehlung des Kundenwunsches vorzubeugen. Dabei werden ständig Integrations- und Regressionstests durchgeführt. Bei der Planung und der Validierung der Prozesse wird der Kunde immer mit einbezogen, um dadurch das korrekte Produkt rechtzeitig liefern zu können.

Dieser Ansatz setzt voraus, dass stark motivierte Mitarbeiter im Team vorhanden sind, die die Fähigkeit besitzen das informelle Wissen zu erfassen und weiterzuleiten. Wenn große und komplexe Projekte vorhanden sind,

---

<sup>26</sup> Hock, Dee. Birth of the Chaordic Age. McGraw-Hill Professional, 1999.

<sup>27</sup> Agile Manifesto (2001): Manifesto for Agile Software Development.  
Quelle: <http://www.agilemanifesto.org> (Abruf am: 27.03.07)

<sup>28</sup> <http://www.agilealliance.org/> (Abruf am: 27.03.07)

werden zusätzlich zu den Agilen Methoden Planungsgetriebene Entwicklungsmodelle zur Unterstützung benötigt [BoTu04a].

### 4.4.1 Extreme Programming (XP)

Unter den vielen Vertretern der Agilen Methoden verspricht insbesondere der Ansatz von Extreme Programming (XP) große Vorteile gegenüber Projekten, die konventionelle Techniken einsetzen. Der Verwaltungsaufwand wird weniger, die Entwicklungszeiten werden kürzer und die Produktqualität und Produktivität erhöhen sich [MüPa03]. XP wurde in den letzten Jahren vielfach von den Empirikern untersucht, um an Ansätzen und Theorien dieser Methode weiterzuentwickeln oder zu kritisieren. Diverse Studien versuchen etwas Klarheit in diesem Gebiet zu schaffen ([Nose98], [WKCJ00], [NaWo01], [Müll04], [WiHN93], [MüPa04], [Sanc05], [Müll06]).

XP wurde entwickelt um die Kosten/Zeit-Verhältnisse während der Softwareentwicklung zu minimieren. Eine zentrale Technik von XP ist die Benutzung der Paarprogrammierung während der gesamten Softwareentwicklung. Die Grundidee, die hinter dem Begriff der Paarprogrammierung liegt, ist die Zusammenarbeit von zwei Entwicklern, die vor einem Rechner sitzen mit nur einem Monitor, einer Maus und einer Tastatur und zusammen Software entwickeln. Ein Argument, das für die Paarprogrammierung spricht, ist, dass die Entwickler mehr Spaß an der Zusammenarbeit haben und sehr gut voneinander profitieren können, wenn sie als Paar arbeiten. Sie finden Lösungen, die sie alleine nicht gefunden hätten.

Die sich fast verdoppelnden Personalkosten für die Entwickler machen diesen Ansatz allerdings sehr kontrovers. Zwei Entwickler werden, anstatt eines Einzelnen, für dieselbe Tätigkeit bezahlt. Es existiert bereits ein Ökonomisches Modell von Müller und Padberg zur Bewertung von XP-Projekten [MüPa03] die weitere Antworten zur Wirtschaftlichkeit dieser Ansätze geben. Dieses Modell gilt als einzigartig, da gewisse Variationen mehrerer zentraler XP-Kenngrößen erlaubt sind, da für jedes Projekt die Parameter neu bestimmt werden können. Das Ergebnis des Modells zeigt, dass ein Projektleiter XP wählen sollte, wenn

der Marktdruck hoch ist, da die Programmierpaare viel schneller arbeiten als Einzelentwickler, allerdings gilt das nur, wenn genügend Entwickler für die Programmierung in Paaren vorhanden sind.

Andere Studien haben gezeigt, dass Paare eine Aufwärmphase, eine so genannte „Warm-up“ Phase brauchen, um richtig effektiv zusammen arbeiten zu können [Müll04]. Diese Einarbeitungskosten müssen natürlich auch im Projektplan mitberechnet werden, was wiederum ein Gegenargument für den Einsatz der Paarprogrammierung liefern könnte.

Zu den quantitativen und qualitativen Ergebnissen der Paarprogrammierung lassen sich noch keine endgültigen Rückschlüsse ziehen. Die quantitativen Ergebnisse sind sehr verschieden und zeigen dabei, dass:

- Paare schneller sind und mehr Vertrauen in die eigenen Lösungen haben [Nose98],
- Paare schneller sind, weniger Fehler machen und teurer sind [WKCJ00],
- Paare nicht schneller als Einzelentwickler sind [NaWo01] und
- Paare gleichwertig zur Einzelprogrammierung mit Durchsichten sind [Müll04].

Zu den qualitativen Ergebnisse lässt sich sagen, dass:

- die Paarleistung nicht alleine vom Leistungsniveau der Individuen abhängt ([WiHN93], [Nose98]),
- die Paarleistung von der Stimmung im Paar bestimmt sein könnte [MüPa04] und
- Paare mehr Vertrauen in die eigene Lösung und mehr Spaß an der Arbeit des Projektes haben ([WiHN93], [Nose98]).

Die Paarprogrammierung wurde lange Zeit mit dem Satz „Paare machen weniger Fehler als Einzelprogrammierer“ unterstützt. Die Gegner der Paarprogrammierung haben wiederum damit argumentiert, dass „Paare mehr kosten“. In neuesten Studien ([Sanc05], [Müll06]) ist untersucht worden, in welchen Szenarien Paare im Vergleich zu Einzelentwicklern besser einsetzbar

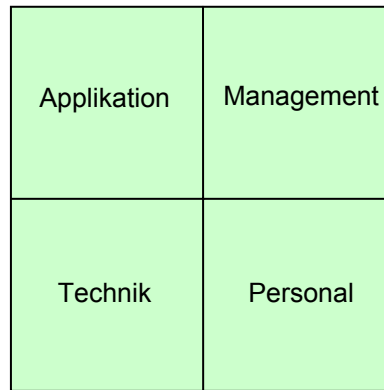
sind. In diesen Studien scheinen Paare bei einfachen Problemen produktiver zu sein als Einzelentwickler. Das heißt gleichzeitig, dass Paare leichter die einfachen Fehler finden. Paare scheinen auch bessere Ergebnisse für Probleme liefern zu können, bei denen der Lösungsvorschlag ohne große Schwierigkeiten als korrekt angenommen werden kann (so genannte „Heureka-Probleme“).

### 4.5 Entscheidungsmodell von Boehm

Es existiert bisher keine Studie, die endgültige Rückschlüsse ziehen kann, welches Vorgehensmodell oder Methode die Bessere ist. Die Planungsgetriebene Entwicklung und die Agilen Methoden scheinen in erster Linie nicht kompatibel zu sein und stehen in direktem Konkurrenzkampf statt sich gemeinsam weiterzuentwickeln. Alltägliche Projekte und Prozesse befinden sich immer irgendwo dazwischen. Sie benutzen keine reine Methode der einen oder anderen Art, sondern eine Mischung aus beiden [TuFR02]. Boehm und Turner ([BoTu03], [BoTu04a], [BoTu04b]) haben einen Ansatz entwickelt, dessen oberstes Ziel es ist, eine Mischung beider Methoden zu definieren und das Projekt dementsprechend anzupassen. Die Agilität des einen und die Disziplin des anderen Ansatzes werden in einigen Bereichen verglichen und der beste Ansatz ausgewählt. Boehm definiert vier Bereiche, die die Eigenschaften jedes Projektes abgrenzen:

1. Applikation,
2. Management,
3. Technik und
4. Personal.

Diese, in Abbildung 15 graphisch dargestellte Eigenschaften, sind auch unter anderem in der Untersuchung von [PaLa04] analysiert worden.



**Abbildung 15. Abgrenzbereiche für Vorgehensmodelle [BoTu03]**

Es gibt zusätzlich drei Eigenschaften eines Projektes, die von Bedeutung sind [BoTu03]:

- Zielsetzung des Projektes,
- Projektgröße und
- Projektumgebung.

Je nach Zielsetzung, Größe und Umgebung passt der eine oder andere Ansatz besser zu den Wünschen und Anforderungen des Kunden und des Projektes.

Zusätzlich zu den vier Abgrenzbereichen für Vorgehensmodelle, haben Boehm und Turner fünf kritische Faktoren definiert, die auf diesen vier Bereichen aufbauen. Mit Hilfe derer lassen sich Entscheidungskriterien für ein bestimmtes Projektmerkmal herausarbeiten und eine Entscheidung treffen, welche Eigenschaften sich für ein bestimmtes Projekt am Besten bewähren. Die Stabilität eines Projektes wird mit der Ordnung und Disziplin der Planungsgetriebene Softwareentwicklung verglichen, während die Agilen Methoden in Zusammenhang mit hoch dynamischen Umgebungen gebracht werden. Die fünf entwickelten Faktoren sind [BoTu03]:

1. die Projektgröße,
2. die Bedingungen, die über die Sicherheit im Projekt entscheiden
3. die Kultur der Entwickler und Kunden,
4. die Dynamik des Projektverhaltens und
5. die personelle Situation in Team.

Diese Faktoren haben Boehm und Turner durch ihr Diagramm (Abbildung 16) dargestellt. Das Entscheidungsmodell wird im Kapitel 5 zusätzlich an die verteilte Entwicklung angepasst, in dem für jeden Faktor, die dazu passenden Eigenschaften eines verteilten Projektes analysiert werden.

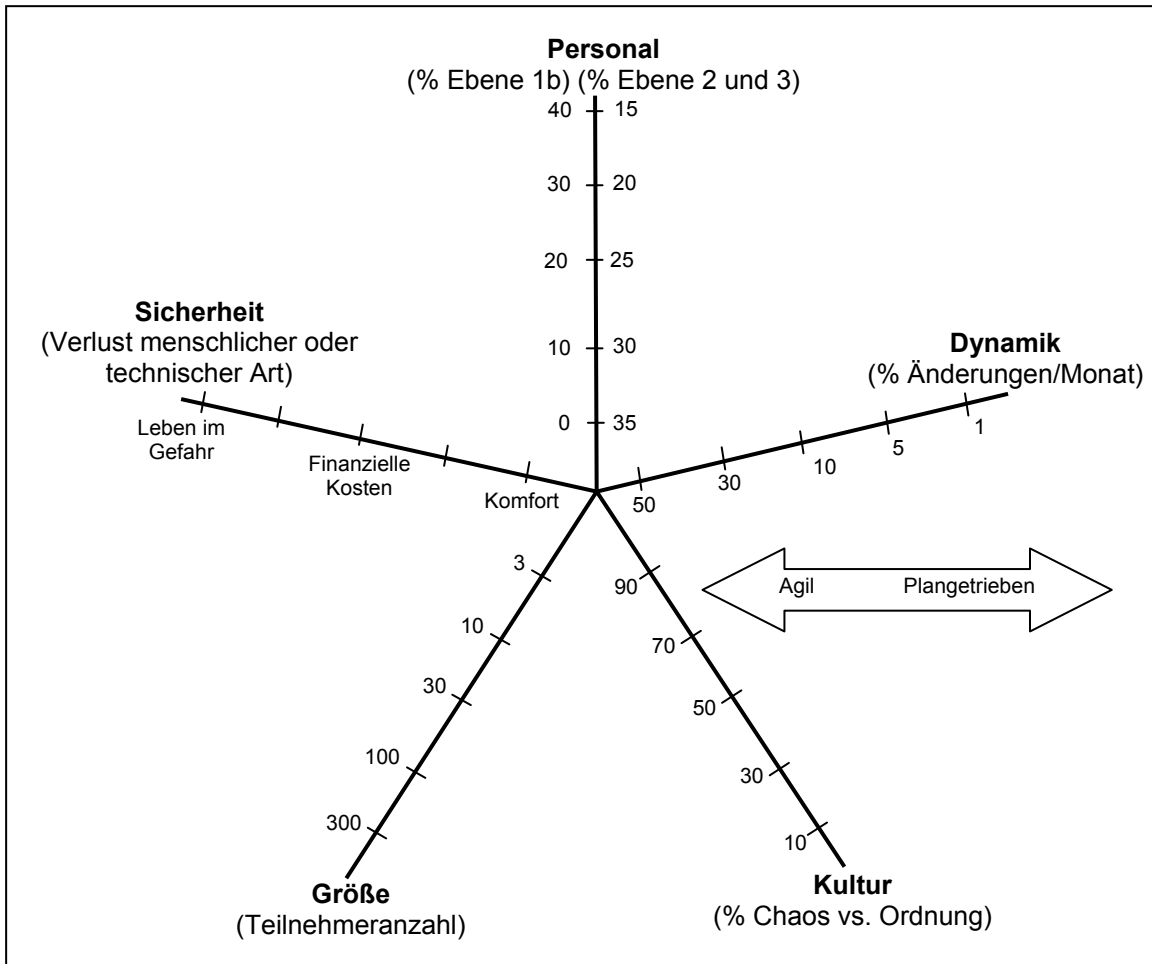


Abbildung 16. Polardiagramm von Boehm und Turner [BoTu03]

#### 4.5.1 Projektgröße

Der Faktor Größe basiert auf den Eigenschaften der Projektgröße der von Boehm und Turner definierten Bereiche [BoTu03]. Mit steigender Teilnehmeranzahl wird ein Projekt als zunehmend schwerer eingestuft. Applikationen, die eine kleine oder mittelgroße Teilnehmergruppe benötigen, werden von den Agilen Methoden unterstützt. Projekte, die über eine erhöhte Agilität besitzen, passen sich besser an kleine Teams an, da diese sich schneller an Änderungen der Umgebung anpassen und dementsprechend reagieren können. Die Grenze für die Projektskalierbarkeit der Agilen Methoden

wird durch die Abhängigkeit der notwendigen, informellen Kommunikation dieser Methoden erreicht. Es wird gesagt, dass maximal vierzig Mitglieder eine gewünschte und anzustrebende Gruppengröße bei agilen Projekten sei [BoTu03]. Durchsichten (Reviews), Meetings und persönlichere Formen der Kommunikation tragen dazu bei, Kommunikationsengpässe dieses Ansatzes zu reduzieren. Mit großen Entwicklerteams ist es dagegen besser, eine detaillierte Planungs-, Definitions- und Entwurfsphase vorzuziehen. Projekte, die planungsgetrieben entwickelt werden, skalieren dadurch besser, haben aber, durch die erhöhte Standardisierung der prozesslastigen Verfahren, Schwierigkeiten mit kleineren Gruppen. Abbildung 17 stellt dieses Verhalten graphisch dar.

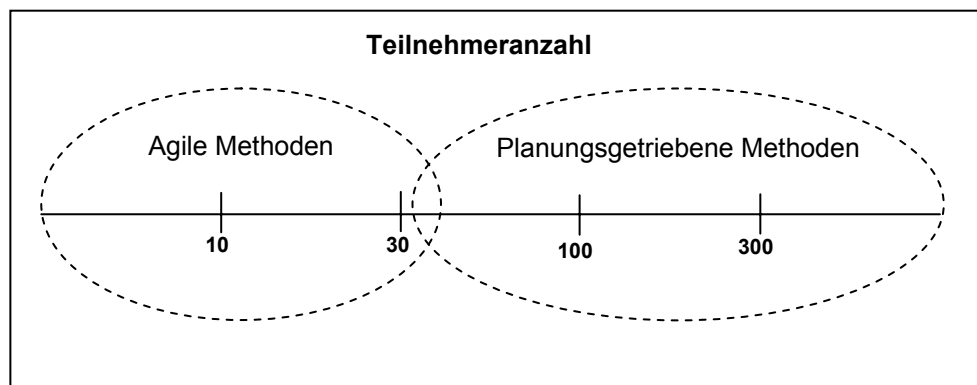


Abbildung 17. Agile Methoden passen sich besser an kleine Teams [BoTu03]

### 4.5.2 Sicherheitsrelevante Bedingungen

Die kritischen Merkmale eines Projektes umfassen die technischen Eigenschaften der Software. Diese werden als der Verlust menschlicher oder technischer Art bezeichnet, der durch den Einfluss von Softwaredefekten zustande kommen kann [BoTu03]. Agile Methoden erreichen schnell ihre Grenze in Anbetracht dieser kritischen Merkmale aufgrund ihres einfachen Aufbaus von Design und Dokumentation. Die Entwickler verfolgen das Motto „immer das einfachste Design verwenden, das die aktuellen Tests besteht“. Das vorhandene Design, das gerade nicht mehr benötigt wird, muss von der Applikation entfernt werden.

Dieser Ansatz skaliert aber schlecht, wenn bestimmte Anforderungen zukünftig in das System integriert werden müssen und trotzdem entfernt werden. Diese Anforderungen werden nicht betrachtet, solange sie nicht gebraucht werden. Die kurzen Iterationen erlauben das Erkennen von Änderungswünsche und die Anpassung, Korrektur oder Verbesserung des Produktes. Die wichtigsten Änderungen werden kollaborativ zwischen dem Kunden (oder Auftraggeber) und den Entwicklern auf der Auftragnehmerseite, zum Beispiel im monatlichen Rhythmus, festgelegt.

Die sicherheitsbedingte Qualitätssicherungsmaßnahmen von Agilen Methoden haben aber noch nicht bewiesen, dass sie sicher genug sind [BoTu03], dennoch existieren einige agile Techniken, die versuchen den Code sicher zu machen. So zum Beispiel die Test-First Programmierung, welche dabei hilft sich im Voraus Gedanken über die Software zu machen, um den Code sicherer machen zu können.

Planungsgetriebene Methoden haben mit einem architekturbasierten Design mehr Chancen in einem kritischen Bereich zu bestehen. Formale Spezifikation, großer Testumfang und andere formelle Analysen und Evaluierungstechniken werden bereits im Voraus geplant. Dieses Verhalten verursacht aber bei nicht so komplexen Projekten mehr Engpässe. Nichts desto trotz, wenn das Projekt über Leben oder Tod entscheidet und sehr dringlich zu behandeln ist, ist eine Anwendung dieser Techniken von Nöten.

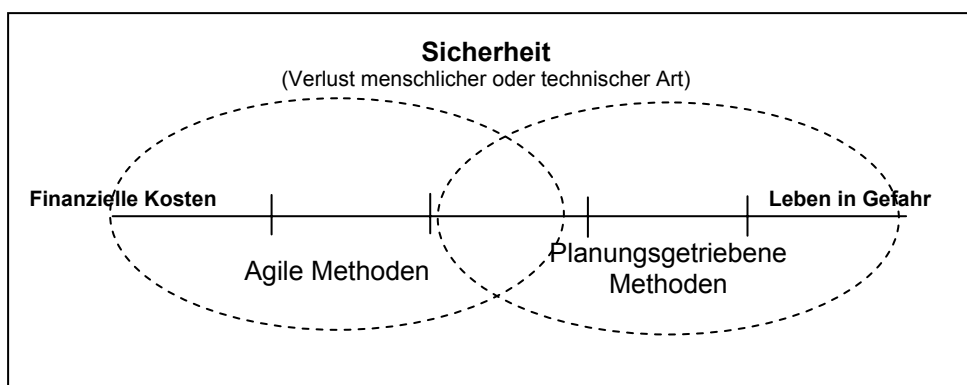


Abbildung 18. Planungsgetriebene Methoden sind besser für kritische Projekte. Nach [BoTu03]

Abbildung 18 zeigt die Spielräume beider Ansätze auf. Zu Beginn des Projektes wird den Grad der Vorhersagbarkeit, die zu erreichen ist, gezielt gesucht. In der Planungs- und Definitionsphase wird diesbezüglich eine präzise Analyse und genaue Definition der Funktionalitäten benötigt, um die Software dementsprechend robust gestalten zu können. Dieses Verhalten kann jedoch überbewertet werden, wenn es sich um kleine oder schnell wechselnde Applikationen handelt. Hierfür müssen einige agile Ansätze an das Planungsgetriebene Projekt adaptiert werden. Andererseits, wenn ein Projekt auf Agilen Methoden aufbaut, kann es sich von formalen Techniken unterstützen lassen, um das Vertrauen und die Qualität kritischer Codestücke der Software zu gewährleisten.

### 4.5.3 Softwareentwicklungskultur

Im Kapitel 4.4 wurde der Begriff „chaordic“ vorgestellt. Teilnehmer, die über mehr Freiheit und Entscheidungsmöglichkeit verfügen, arbeiten im so genannten „Chaos“, um die zu lösende Situationen genauer zu definieren und abzuarbeiten. Jeder wird als ein wichtiger Bestandteil des Projektes geschätzt und es wird angenommen, dass jeder Einzelne sich für den Projekterfolg einsetzen wird. Auf der anderen Seite (siehe Abbildung 19) stehen klare Leitlinien, festgestellte Abläufe und definierte Rollen, die synonym für die „Ordnung“ stehen. Ein Projekt und dessen Team bewegen sich also zwischen diesen beiden Extremen, wobei ein Team, das hauptsächlich in einem der beiden Bereiche, also im „Chaos“ oder „Ordnung“ arbeitet, schwer vom Gegensatz zu überzeugen ist.

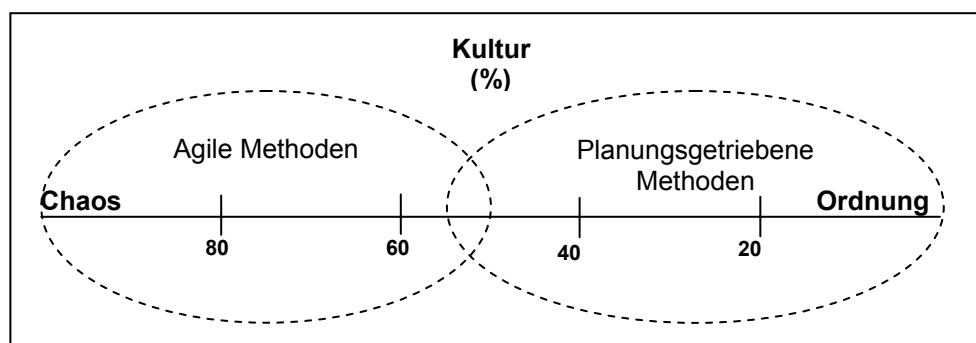


Abbildung 19. Projekte befinden sich oft „irgendwo dazwischen“. Nach [BoTu03]

#### 4.5.4 Dynamik

Abbildung 20 zeigt, wie die Dynamik eines Projektes von den Agilen Methoden besser gemeistert wird, im Vergleich zu den Planungsgetriebenen Ansätzen. Durch ein einfaches Design und ein ständiges Refactoring passen sich die ersteren sowohl an stabile als auch an hoch dynamische Umgebungen an. Auf der anderen Seite erlauben die Planungsgetriebenen Ansätze nur eine kleine Anzahl von Änderungen im Monat und brauchen detaillierte Arbeitspläne und ein klar vorgegebenes Produktdesign.

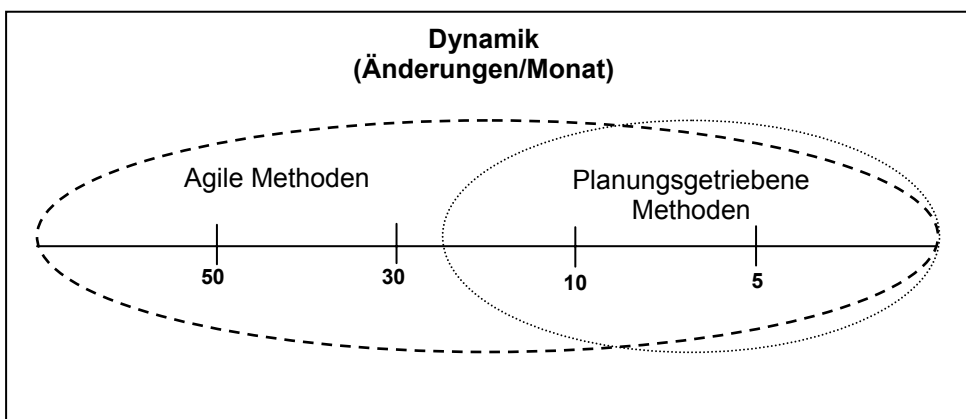


Abbildung 20. Agile Methoden passen sich sowohl an stabile als auch an hoch dynamische Umgebungen an. Nach [BoTu03]

Die Agilen Methoden passen sich an turbulente, hoch-änderbare Umgebungen an. Die Anforderungen werden dabei nicht im Voraus genau spezifiziert sondern ad-hoc festgestellt und definiert. Diese Spontaneität kann sich allerdings schnell in eine Problemsituation verwandeln, wenn Teammitglieder sich nur auf die aktuelle Projektaufgabe oder nur auf einen Teil des Projektes konzentrieren, dabei aber das Gesamtprojekt und -konzept vernachlässigen und den Überblick verlieren ([BoTu03] und Abschnitt 4.5.2). Für die Planungsgetriebenen Methoden werden die Anforderungen im Voraus festgestellt und festgesetzt und gelten für die gesamte Projektdauer. Projekte, die Planungsgetrieben entwickelt werden, erlauben keine großen Änderungen der Anforderungen im Laufe des Projektes, können aber gegebenenfalls eine starke und quantitative Prozessverbesserung durch eine gute Organisation und Struktur unterstützen.

Das informelle Verhalten der Agilen Methoden hängt von der jeweiligen Teamgröße des Projektes ab. Je größer die Gruppe und das Projekt an sich sind, desto schwieriger ist es auch, dieses Verhalten beizubehalten. Planungsgetriebene Projekte können mit Hilfe von Prozess- und Produktplänen eine gewisse Kontrolle über die Teams und das Projekt erreichen. Solche Prozesspläne, die den Projektablauf und die zu erreichenden Meilensteine definieren, und Produktpläne mit den nötigen Anforderungen und Produktfunktionen sind für eine erfolgreiche Projektsteuerung von Bedeutung.

### 4.5.5 Personal

Auf der Personalachse kommt es zu einem gegenteiligen Verhalten im Vergleich mit den Eigenschaften, die auf der Dynamik-Achse ersichtlich wurden. Boehm und Turner [BoTu03] haben für diese Achse fünf Personalebene entwickelt, um die Fähigkeiten und das Verständnis der Entwickler zu definieren. Diese Ebenen sind:

- -1,
- 1B,
- 1A,
- 2 und
- 3.

Wie in Abbildung 21 gezeigt wird, funktionieren hier die Planungsgetriebenen Ansätze an beiden Ende der Achse gut; Agile Methoden brauchen im Gegensatz dazu eine bessere Personalkombination, da der Erfolg des Projektes deutlich vom Personal abhängt und ein agiles Team mit ausreichenden Qualitäten gebraucht wird. Wenn die Gruppe gut ist, kann sie viel leisten, doch ist sie dies nicht, läuft sie Gefahr, ein schlechtes Ergebnis zu präsentieren. Bei Planungsgetriebenen Projekten bekommen die Entwickler zusätzliche Hilfe von der erstellten Architektur und den vorgegebenen Ablaufplänen.

Teammitglieder der Ebene -1 sollten schnell identifiziert werden können, da sie nicht mit dem Agilen Ansatz oder mit den Planungsgetriebenen Methoden korrekt umzugehen wissen, ihnen werden so neue Arbeiten zugewiesen. Entwickler der Ebene 1B sind Entwickler, die durchschnittlich und unterdurchschnittlich gut sind und über eine geringe Programmiererfahrung verfügen. Entwickler dieser Ebene sind fleißige Entwickler, die einfache Entwicklungsaufgaben in stabilen und gut strukturierten Situationen beherrschen. Entwickler, die auf Ebene 1A eingestuft werden, können gut in agilen Projekten arbeiten, wenn es genügend andere Entwickler gibt, die die Ebene 2 abdecken und diese das Team unterstützen.

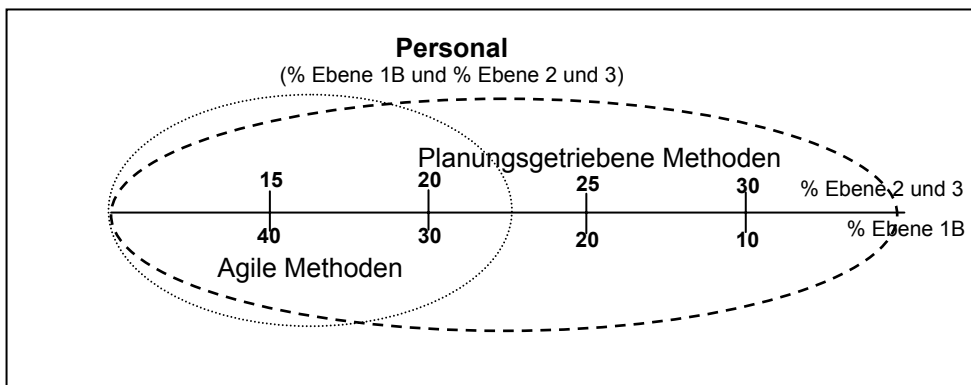


Abbildung 21. Agile Methoden benötigen eine gute Personalkombination. Nach [BoTu03]

Es gibt eine Faustregel, die besagt, dass fünf Teilnehmer der Ebene 1A für einen Teilnehmer der Ebene 2 nötig sind [BoTu03]. Entwickler, die der Ebene 2 angehören, können kleine, sowohl agile als auch planungsgetriebene Projekte leiten und haben bereits Erfahrung mit solchen Projekten gesammelt. Sie brauchen dann die Hilfe von Entwicklern, die der Ebene 3 angehören, wenn das Projekt größer oder sehr neu ist. Nur einige wenige Entwickler haben die Fähigkeit die Ebene 3 zu erreichen, diese können ein noch nie da gewesenes Projekt leiten.

Damit die Agilen Methoden erfolgreich arbeiten können, benötigen sie eine kritische Masse an so genannten Level 2 oder Level 3 Experten (siehe Abbildung 21), um die Anwesenheit von Entwicklern der Ebene 1B im Team zu kompensieren. Die kritische Masse bei planungsgetriebenen Projekten aus der Ebene 2 oder 3 ist nur am Anfang des Projektes wichtig, das heißt bei der Projektdefinition und Spezifikation. Später werden diese Entwickler nicht mehr

so häufig benötigt, sind sogar während der Zeit der Programmierarbeiten völlig überflüssig, wenn nicht das Projekt als hoch dynamisch gilt. Planungsgetriebene Projekte können in der Regel gut mit Entwicklern aus der Ebene 1B auskommen.

## 5 Entscheidungsmodell für verteilte Projekte

Wenn Software verteilt entwickelt werden soll, existieren einige Aspekte, die bei einer firmeninternen (in-house) Softwareentwicklung nicht vorhanden sind. Getrennte und vor allem geographisch verteilte Teams, die mehreren Kulturen mit einbeziehen können, benötigen gesonderte Aufmerksamkeit, damit die Distanz und die kulturellen Unterschiede den Gesamterfolg eines Projektes nicht gefährden. Sowohl Planungsgetriebene Entwicklungsmodelle als auch Agile Methoden müssen an diese verteilten Umgebungen angepasst werden. Einige Probleme der verteilten Softwareentwicklung sind ([Karo98], [BoTu03], [PaLa04]):

- die Abhängigkeit der verteilten Entwicklerteams,
- die Koordinationsschwierigkeiten zwischen diesen,
- die Schwierigkeiten bei der Verteilung der Arbeit und Aufgaben an die verschiedenen Teams,
- die von verteilten Entwicklerteams falsch implizierten Annahmen oder
- die immer wieder erwähnten Kommunikationsprobleme.

Verteilte Teams besitzen im Voraus selten genaue Informationen bezüglich der Anforderungen oder Spezifikationen des zu liefernden Produktes (siehe dazu Kapitel 4.4). Diese Informationen sind am Anfang eines Projektes manchmal schwer zu bekommen oder sind nicht vollständig definiert worden. Die Kooperation und Kommunikation zwischen Auftraggeber und -nehmer ist dann während der gesamten Softwareentwicklung notwendig und unentbehrlich. Auftragnehmer müssen verstehen, was sie entwickeln sollen und immer wieder Rückmeldungen (Feedback) vom Auftraggeber bekommen, um Missverständnisse zu vermeiden.

Verteilte Projektumgebungen lassen vermuten, dass die Agilen Methoden für solche Umgebungen besser geeignet sind als die Planungsgetriebenen Ansätze [PaLa04], insbesondere wenn ein Projekt als komplett neu eingestuft wird und wenn Unklarheit, Ungewissheit und Unvorhersagbarkeit im Projekt

herrschen. Leider existieren nicht viele empirische Erfahrungsberichte über das Zusammenspiel der Agilen und Planungsgetriebenen Entwicklungsmethoden mit der Auslagerung von Projekten, um diese Annahme bestätigen zu können oder zu widerlegen.

Das im Kapitel 4.5 beschriebene Entscheidungsmodell von Boehm und Turner analysiert die Projektrisiken, um die Vorgehensmodelle und Methoden näher zu bringen. Dieses Entscheidungsmodell stützt sich auf die Fähigkeiten der Entwicklungsteams und deren Teilnehmern, um die Projektumgebung und die Organisationsstrukturen besser verstehen zu können ([BoTu04a], [BoTu04b]). Das Ziel hier ist, dieses Entscheidungsmodell an die verteilten Eigenschaften eines Projektes und deren Umgebung anzupassen, um die interpersonellen Probleme zu minimieren damit verteilte Teams bedeutende Beiträge zum Erfolg des Projektes leisten. Das Entscheidungsmodell wird angepasst und mögliche Organisationsstrukturen der Teams werden aufgezeigt, die sich an bestimmte Projekt- und Personaleigenschaften anpassen lassen. Es wird zusätzlich gezeigt, welche Aufteilungsmöglichkeiten für die Teams existieren, um verteilte Teams sinnvoll strukturieren zu können ohne dabei die Verteilung der Aufgaben im Projekt und die zu lösenden Problemstellungen unbeachtet zu lassen.

Im Fall eines verteilten Softwareprojektes wird von virtueller Organisation gesprochen [Karo98]. Die virtuelle Organisation existiert ausschließlich um eine bestimmte Leistung zu erbringen, sei es eine Softwarelösung, ein Stück Hardware oder der Support eines Systems. Eine virtuelle Organisation kommt dann zustande, wenn Teile der Projektaufgaben nicht vor Ort durchgeführt werden, alle involvierten Standorte aber gemeinsam zusammen arbeiten, um den Erfolg des Projektes zu gewährleisten.

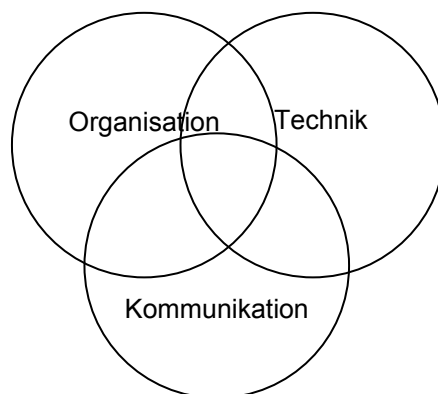
Diese Art der Organisation erhöht den Kommunikationsbedarf und die Abhängigkeit zwischen den involvierten Teams. Die Kommunikationswege sind in solchen Fällen selten synchron, zum Beispiel kann sich ein Entwicklungsteam in Sydney, Australien befinden, während das Wartungsteam in Sao Paulo, Brasilien seine Arbeiten durchführt. Der Zeitonenunterschied beträgt in diesem Fall dreizehn Stunden, wenn in Sao Paulo der Arbeitstag anfängt, sind die Teammitglieder in Sydney nicht mehr am Arbeitsplatz. Mit

Hilfe einer CDE<sup>(29)</sup> soll der Kommunikationsbedarf auf den aktuellsten Stand gebracht werden (siehe Kapitel 6).

Walter Karolak zählt in [Karo98] Risiken auf, die in jedem Projekt, insbesondere in verteilten Projekten, existieren (siehe Abbildung 22):

1. Organisation
2. Technik
3. Kommunikation

Diese drei Kategorien sind eng miteinander gekoppelt und manchmal ist es schwer, diese voneinander zu trennen. Die Organisationsrisiken haben mit der Rollenverteilung und die Autoritätsvergabe im Projekt zu tun. Unklare Verteilungen im Projekt verursachen Verspätungen und einen erhöhten Aufwand bei der Produkterwicklung.



**Abbildung 22. Mögliche Bereiche der Risiken eines verteilten Projektes [Karo98]**

Falsch angewandte Methoden und Werkzeuge für die Softwareentwicklung sind die Risiken, die die Technik verursachen können. Die falsche Benutzung der Softwaretechniken kann zu irreparablen Problemen führen. Eine gute Kommunikationsinfrastruktur kann Missverständnisse und falsche Erwartungen minimieren.

---

<sup>29</sup> CDE: Kollaborative Entwicklungsplattform (siehe Kapitel 6)

## 5.1 Angepasste Faktoren

Herbsleb [HePB05] hat zusätzliche Erfahrungen von neun verteilten Siemens-Projekten gesammelt. Obwohl die hier vorgestellten Überlegungen empirisch nicht bewiesen worden sind, werden die Siemens-Projekte, die von Karolak aufgezählte Risikobereiche [Karo98] und die fünf kritischen Faktoren von Boehm und Turner [BoTu03] trotzdem als Ausgangspunkt für die Anpassung des Entscheidungsmodells an verteilte Projekte genutzt. Im Folgenden werden Überlegungen für verteilte Projekte an die fünf kritischen Bereiche von Boehm und Turner angepasst.

### 5.1.1 Größe

Der unautoritäre Ansatz der Agilen Methoden unterstützt eine schnelle Produktentwicklung dank kurzen Iterationen. Mit der nötigen Erfahrung und dem nötigen Wissen kann ein Team Entscheidungen darüber treffen, welche Aktivitäten dem Projekt mehr Wert bringen können. Die Projekte werden, durch die vielen Änderungen im Laufe des Projektes, zunehmend spontaner. Wenn es bei den Agilen Methoden zu Änderungen kommt oder die Zielsetzungen sich anders als erwartet entwickeln, ist es wichtig eine Strategie auszuarbeiten, wie mit den damit verbundenen Problemen leichter umgegangen werden kann, anstatt nur eine starre Lösungsstrategie zu verfolgen [Beck05]. Der Fokus der Planungsgetriebenen Methoden liegt auf der Vorhersagbarkeit und Stabilität der Projekte. Ein Projekt kann sich dennoch instabil verhalten und unerwartete Vorkommnisse können auftreten. Der Vorteil dieses Planungsgetriebenen Ansatzes, ist die erhöhte Sicherheit, die solche Methoden mit sich bringen. Dank einer guten Dokumentation, Firmenstandards und gut definierten Strukturen, sollen Projekte, die auf einer starken Sicherheit basieren, ohne Schwierigkeiten gut gelingen (siehe Abschnitt 4.5.1).

### Kontrolle und Sichtbarkeit

Eine Erkenntnis der virtuellen Organisation ist, dass die Sichtbarkeit und die Kontrolle über alle Phasen des Projektes für den Auftraggeber verloren gehen können [HePB05]. Diese Problematik verschärft sich in virtuellen

Organisationen mit steigender Anzahl der Teams. Je mehr Teams in der Softwareentwicklung involviert sind, desto geringer werden die Sichtbarkeit und die Kontrolle über diese Teams. Die Benutzung einer CDE hilft dem Management auf der Auftraggeberseite bei der Kontrolle der Ergebnisse, Lieferungen und der Synchronisation der Abgaben [BoBr02]. Anderenfalls kann der Auftraggeber nur darauf hoffen, dass der Dienstleister die Termine einhält, die Qualität der Lieferungen sichert, und dass die Probleme zwischen Teams schnell gelöst werden. Für kleine Projekte könnte eine kostenfreie CDE genügen, um die Übersicht zu behalten. Für große Projekte muss dennoch eine proprietäre Lösung eingesetzt werden. Es ist abzuraten, die CDE ständig zu wechseln, die ausgewählte CDE soll einmalig angeschafft werden und als Standardwerkzeug für alle Projekte benutzt werden (siehe Kapitel 6)

### **Partnerwahl**

Die Auswahl des Auftragnehmers soll gut überlegt sein, da die Teams im Laufe des gesamten Projekts untereinander interagieren müssen, als ob sie vor Ort arbeiten würden. Große oder wichtige Projekte sollen mit Dienstleistern durchgeführt werden, die dem Auftraggeber bekannt sind, die über ein vollständiges Domänenwissen, ausgereifte Technikenkenntnisse, flexible Arbeitskultur und gute Infrastruktur verfügen [Karo98]. Der Auftraggeber muss sorgfältig untersuchen, über welche Erfahrungen die möglichen Auftragnehmer besitzen. Mit Hilfe von Fragebögen und Testszenarien soll untersucht werden wie viel Domänenwissen der mögliche Dienstleister besitzt, wie viel Erfahrung die Entwickler aufweisen oder welche Technologien und Methodologien unterstützt werden. Entscheidend ist, dass der Auftragnehmer nicht die komplette Kontrolle des Produktes behält.

### **Zentrale Autorität**

Der Auftraggeber muss über genügend Verantwortungen und Zuständigkeiten verfügen, um zum Beispiel zukünftige, interne Wartungsarbeiten selbständig durchführen zu können. Somit wird eine gewisse Unabhängigkeit vom Dienstleister gewährleistet. Für technische und geschäftliche Konflikte kann eine zentrale Autorität als letzte Instanz im Streitfall dienen [HePB05]. Es muss aber darauf geachtet werden, dass diese Person die Kommunikationswege nicht bremst. Diese Wege sollen nicht nur auf eine Person fokussiert werden,

sonst wird die Dynamik der Kommunikation beeinträchtigt ([HMFG00], [OHRG04]). Tabelle 2 fasst die drei oben genannten Punkte zusammen.

Größe	Anpassung	Wenige Teams / Einfache Projekte	Viele Teams / Komplizierte Projekte
	Kontrolle und Sichtbarkeit	Eine kostenfreie CDE genügt für die Teamarbeit	Eine proprietäre CDE sollte verwendet werden.
	Partnerwahl	Eine Bekanntschaft aus anderen Projekten ist nicht zwingend erforderlich	Mehrere Projekte des Auftraggebers durchgeführt. Gutes Domänenwissen ausgereifte Technikenntnisse Flexible Arbeitskultur Gute Infrastruktur
	Zentrale Autorität für Probleme	Die Dynamik und die informelle Kommunikation könnnten beeinträchtigt werden	Hilfe, als letzte Instanz im Streitfall

Tabelle 2. Faktor Größe des angepassten Entscheidungsmodells.

### 5.1.2 Bedingungen für die Qualitätssicherung

Beim Testen einer Applikation geht es darum [BoTu03]:

- 1.) zu validieren, ob die Kundenspezifikation korrekt ist und
- 2.) zu verifizieren, ob die Entwickler das Produkt korrekt implementiert haben.

#### Testverfahren

Für verteilte Teams sind Design- und Code-Reviews eine Gelegenheit, um diese Aspekte der Entwicklung noch einmal nachzuprüfen [Karo98]. Die Reviews (siehe Kapitel 7.3) helfen den verteilten Teams sofortiges Feedback über die aktuelle Arbeit zu bekommen. Durch dieses Feedback können die Entwickler sich immer wieder neu motivieren, da die mitgeteilte Information über die Arbeit sehr motivierend ist, wenn die gelieferten Ergebnisse gut waren. Der Auftraggeber hat damit etwas in der Hand, um einen Projekterfolg oder Misserfolg feststellen zu können und zu entscheiden, ob die Entwickler der verteilten Standorte über genügend Know-how verfügen.

Das Problem der Planungsgetriebenen Entwicklung liegt darin, dass das Testen erst sehr spät durchgeführt wird. Die Entwicklung erfolgt zu Beginn des

Projektes ohne eine Überprüfung von außen wodurch Probleme erst spät erkannt werden. Es kann vorkommen, dass die in der Planungs- und Definitionsphase erstellte Dokumentation sich in der Testphase als nicht mehr gültig erweist, wenn zu viele Änderungen stattgefunden haben. In Planungsgetriebenen Projekte lassen sich die Testdefinitionen einfacher auskoppeln. Die von einem Team erstellten Definitionen werden von einem anderen Team an einem anderen Standort getestet. Die Auslagerung der Testaktivitäten kann dennoch verursachen, dass für das Testteam nur die Spezifikationszufriedenheit wichtig ist, während das Entwicklungsteam das funktionale Ziel und die Kundenzufriedenheit in erster Linie von Bedeutung sind. Dieses Verhalten erschwert die Aufgabe des Managements, die Kontrolle über das Projekt zu haben. Agile Projekte dagegen, werden durch das inkrementelle Kodieren, die Paarprogrammierung, Review-Techniken und automatisierte Tests unterstützt. Diese Techniken garantieren, dass die Anforderungen testbar sind, ohne große Dokumentationen erstellen zu müssen. Fehlende Erfahrung bei den Entwicklern führt zusätzlich zu einer ungeeigneten Testabdeckung.

**Know-how Verlust**

Wenn sich eine Firma entscheidet, ein Projekt verteilt zu entwickeln, muss sie sich im Klaren darüber sein, dass je nach Schwierigkeitsgrad sämtliches Domänenwissen, Know-how und Information der Firma beim Auftragnehmer landen [HePB05]. Damit der Auftragnehmer nicht als Konkurrent sondern als Partner gesehen wird, müssen die so genannten Kernkompetenzen auf der Auftraggeberseite bleiben. Wenn vertrauenswürdige Informationen weiter gegeben werden müssen, dann muss gewährleistet werden, dass der Auftragnehmer diese Informationen sorgfältig bewahrt und nicht weitergeben darf.

Sicherheits-relevante Bedingungen	Anpassung	Wenige Teams / Einfache Projekte	Viele Teams / Komplizierte Projekte
	Testverfahren	Inkrementelles Kodieren, Paarprogrammierung, Review-Techniken, Automatisierte Tests	Testphase nach der Implementierungsphase. Testdefinition und -ausführung voneinander trennbar
	Know-how Verlust	Keine große Gefahr die Kernkompetenz zu verlieren.	Gefahr, dass Kernkompetenzen verloren gehen. Konkurrenzgefahr zwischen Auftragnehmer und -geber

Tabelle 3. Bedingungen für die Qualitätssicherung des Entscheidungsmodells.

### 5.1.3 Kultur der verteilten Teams

#### Kommunikation und Skalierbarkeit

Die Kommunikation zwischen Teams ist eines der komplexesten Themen der verteilten Softwareentwicklung. Während Agile Methoden einen informellen Wissensaustausch unterstützen, lassen sich die Planungsgetriebenen Methoden durch eine explizite, dokumentierte Kommunikation erkennen [BoTu03]. Agile Methoden unterstützen die bidirektionale Kommunikation zwischen Teams, die synchron gestaltet wird. Diese persönliche, Mensch-zu-Mensch Kommunikation wird zum Beispiel mittels Meetings oder mit Hilfe der Paarprogrammierung unterstützt, bringt jedoch von Natur aus zwei Probleme mit sich. Das erste Problem ist, dass das kommunizierte Wissen immer personenabhängig ist. Verlässt ein Mitglied das Team, geht mit ihm auch das Wissen verloren. Das zweite Problem hat mit der Anzahl der Teammitglieder zu tun: je mehr Mitglieder ein Team besitzt, desto komplizierter ist die Aufrechterhaltung aller Kommunikationswege.

Der Informationsfluss ist in hoch dynamischen Teams proportional zum Quadrat der Anzahl der verteilten Teammitglieder, nämlich  $\frac{n(n-1)}{2}$  [Karo98].

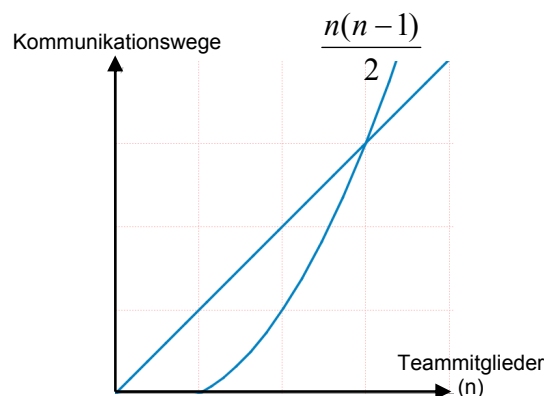


Abbildung 23. Kommunikationswege in hoch dynamischen Teams [Karo98]

Kleine Gruppen besitzen wenige Kommunikationswege, aber je größer die Mitgliederanzahl in verteilten Teams ist, desto komplizierter ist auch die Aufrechterhaltung aller möglichen Kommunikationswege (siehe Abbildung 23). Die Skalierbarkeit der Kommunikation zwischen den Teams und zwischen den

Mitgliedern in einem Team wird mit steigender Anzahl der Teams und Mitglieder problematisch. Die formale Kommunikation wird deswegen in Planungsgetriebenen Projekten auf ein Minimum reduziert. Am Anfang des Projektes werden genügend Informationen aufgenommen, so dass später im Projekt auf die vorhandenen Informationen zurückgegriffen werden kann. Während der Entwicklung sprechen sich Auftraggeber und -nehmer ab, wie und wie oft kommuniziert wird.

Für Projekte, die verschiedene Kulturen, Sprachen und Zeitzonen aufweisen, empfehlen Herbsleb und Grinter [HeGr99a] hauptsächlich wenige Kommunikationskanäle zu benutzen, denn in einer verteilten Umgebung ist es nicht einfach die Kommunikation der Teilnehmer aufrechtzuerhalten. Dadurch minimiert sich die Kommunikation zwischen verteilten Teams durch die physische Distanz und die Zeiten zwischen einer Anfrage und die dazugehörige Antwort erhöhen sich deutlich.

Die direkte Kommunikation ist zwischen verteilten Teams zeit- und ressourcenaufwendig, sie bleibt aber unabdingbar für jedes erfolgreiche Projekt ([HePB05], [HMFG00], [NgBV06], [GrHP99]). Je nach Projektlage kann es vorkommen, dass ein Mitarbeiter der Auftraggeberseite sich für eine gewisse Zeit an jedem Standort aufhält, um potentielle Konflikte lösen zu können. Somit ist in großen und mittelgroßen Teams diese Schnittstellenperson wichtig, denn der Kommunikationsfluss wird von nur einer Person aufgenommen und kann danach filtriert weitergeleitet werden. Durch die Festlegung bestimmter Kommunikationswege kann jedes Teammitglied wissen, wer zu kontaktieren ist, um eine bestimmte Frage- oder Problemstellung lösen zu können. Projektphasen, wie zum Beispiel das Testen oder den Integrationsprozess, erfordern eine große Interaktion zwischen Teammitgliedern. Dieser Bedarf sieht die vor Ort befindenden Teammitglieder als ein wichtiger Bestandteil des Erfolges.

Die Kommunikation ist eine der zentralen Bedingungen von denen der Projekterfolg abhängt. Je mehr Interaktion zwischen den Teams notwendig ist, desto mehr Kommunikationswege werden gebraucht, da zum Beispiel eine wöchentliche Softwarelieferung weitaus mehr Kommunikation erfordert als

monatliche Lieferungen. Meetings sind stets eine Möglichkeit, um Projekte zu beobachten und zu kontrollieren. Dort können die Module und Lösungen, die die durchgeführten Tests bestanden haben, als fertige Produkte deklariert werden. In diesen Meetings dürfen nur so viele Teammitglieder wie unbedingt notwendig anwesend sein, sei es physisch oder virtuell.

Die Einarbeitungszeit für verteilte Teams nimmt eine gewisse Zeit in Anspruch, deswegen werden am Anfang viele Iterationen und viel Training benötigt, bevor die Teams produktiv arbeiten können [HePB05]. Soziale Netze sollen so früh wie möglich aufgebaut werden, um die Zusammenarbeit schon ab dem Projektstart zu unterstützen. Wege müssen gefunden werden, um neue verteilte Gemeinschaften zu bilden und diese aufrechtzuerhalten, damit die Interaktion harmonischer abläuft. Wichtig ist dass die Arbeitskultur der verschiedenen Teams verstanden worden ist und dass genügend Mitarbeiter mit Erfahrung und Offenheit für die Arbeit in verteilten Teams vorhanden sind. In Planungsgetriebenen Projekten ist es notwendig, dass alle Informationen für die Entwicklung der Module frühzeitig alle verteilten Teams erreichen. Wenn Anforderungen, Dokumentationen oder Entwurf nicht rechtzeitig definiert und verteilt werden, ist es unmöglich, dass die Teams über vollständige Informationen verfügen können, um die Module korrekt entwickeln zu können. Die nötigen Kommunikationswege müssen geschaffen werden, bevor die verteilten Teams mit der Aufgabenlösung anfangen. Regelmäßige Workshops mit den betroffenen Teams und den Auftraggebern sind für das Verständnis der Aufgaben sehr wichtig. Inhalte und erstellte Dokumente werden dann in die CDE hochgeladen und bleiben für die Projektlaufzeit immer verfügbar.

### **Kultur**

Die Unterschiede, sowohl kultureller als auch betrieblicher Natur, müssen identifiziert werden. Die Kultur in Asien oder Südamerika ist nicht die gleiche wie in Europa oder in USA. Die beteiligten Teams erarbeiten in den frühen Phasen die einsetzbaren Methodologien, um einen Konsens zwischen den Teams zu erreichen. Dadurch erfolgt die Arbeit harmonischer zwischen Auftraggeber und -nehmer. Die Beteiligten sollen sich in einer ersten Phase gegenseitig kennenlernen, um die Unterschiede erkennen zu können und Kompromisse zu vereinbaren. Kollaborationstools wie Chaträume oder Instant

Messaging sind für die Kommunikation der Teams sehr nützlich. Sie ergänzen die traditionellen E-Mails und unterstützen dabei die synchrone Kommunikation (siehe Kapitel 6.2), behalten aber die asynchrone Kommunikation als wichtigen Bestandteil des Kommunikationsflusses. Es sollen die Protokolle und das Vokabular für den Kommunikationsverkehr für alle Bereiche definiert werden, sei es E-Mail, Chaträume oder Videokonferenz, um Doppeldeutigkeiten vermeiden. Die Zeitzonen werden zusätzlich zu einem komplizierten Faktor in verteilten Projekten, je größer der Zeitunterschied zwischen den Einsatzgebieten der Teams ist. Probleme werden per E-Mail oder Anrufe gelöst, was die Anzahl der E-Mail erhöht, aber den Verkehr in den vorhandenen Foren und Wikis der CDE wiederum reduziert. Tabelle 4 fasst die angepassten Eigenschaften für die Achse „Kultur der verteilten Teams“ zusammen.

	Anpassung	Wenige Teams / Einfache Projekte	Viele Teams / Komplizierte Projekte
<b>Kultur</b>	<b>Kommunikation</b>	Informell Meetings, Paarprogrammierung Wissen ist personenabhängig	Explizit Informationserhebung in der Planungs- und Definitionsphase ressourcen- und zeitaufwendig Schnittstellenperson vorhanden
	<b>Skalierbarkeit</b>	Wenn Agile Methoden angewandt werden, dann ist der Kommunikationsbedarf auch hoch.	Hoher Kommunikationsbedarf: $\frac{n(n-1)}{2}$ Kommunikation wird aber auf ein Minimum reduziert
	<b>Kultur</b>	Workshops	Frühe Erarbeitung von Methodologien Zeitzeitenproblem, erhöhter E- Mail-Verkehr

Tabelle 4. Kultur der verteilten Teams im angepassten Entscheidungsmodell.

### 5.1.4 Dynamik

Die Dynamikachse definiert die Eigenschaft, sich sowohl an stabile als auch an hoch dynamische Umgebungen anzupassen (siehe Abschnitt 4.5.4).

### Synchronisierung

In verteilten Projekten bedarf es mehrerer Abläufe, bis das Gesamtkonzept durch und durch verstanden wird - vor allem bei erstmaliger Zusammenarbeit von Auftraggeber und -nehmer kann nicht im Voraus gewusst werden, wie gut die Anforderungen vom Auftraggeber selbst verstanden werden (siehe auch

Abschnitt 5.1.2 und [Karo98]). Durch eine frühzeitige und ständige Softwareintegration und Tests kann bestätigt werden, dass die Anforderungen richtig verstanden und durchgeführt worden sind.

Die Synchronisierungsarbeiten in der verteilten Softwareentwicklung müssen sorgfältig geplant werden. Die Softwarelieferungen und -Integration sollen so synchronisiert werden, dass so wenige Leerlaufzeiten wie möglich zwischen den Teams entstehen. Lieferungen, die in kurzen Abständen erfolgen, können schneller geprüft und getestet werden. Das erhöht die Kontrolle des Projektes für die Teams. Mit Hilfe von „weekly builds“ ([HePB05]) können zum Beispiel die verschiedenen verteilten Teams Ergebnisse liefern, die einmal in der Woche getestet werden. Die getestete Software fungiert dann als neue Ausgangsposition für die nächsten Iterationen. Mit Hilfe von kleinen Lieferungen und einem gemeinsamen Repository für das Projekt kann zuerst der Quellcode getestet werden, bevor dieser an den Kunden ausgeliefert wird. Sind die Lieferungszeiten eines Projektes schlecht synchronisiert, dann werden Probleme, die von dem einen oder anderen Standort verursacht worden sind erst spät behoben. Die Lieferungshäufigkeit sollte so flexibel gestaltet werden, dass am Anfang des Projektes größere Lieferzeiten liegen, die dann später in kürzeren Abständen synchronisiert werden.

Ein allgemeiner Vorteil der häufigen Lieferungszyklen und Integration ist die Sichtbarkeit des Projektverlaufes [HePB05]. Teammitglieder und Projektleiter können sich beim Betrachten des Projektes in regelmäßigen Zeitabständen ein gutes Bild über etwaige Fortschritte machen. Für den Kunden oder den Auftraggeber ist die Möglichkeit der Fortschrittüberwachung interessant und wichtig. Zudem besteht ein weiterer Vorteil darin, dass eine einzige Integration der Komponenten erspart wird. Lange Perioden der Eigenentwicklungen werden somit überflüssig, was der verteilten Softwareentwicklung zu Gute kommt, da es immer wieder vorkommen kann, dass Module sich nicht integrieren lassen und dies erst recht spät festgestellt wird. Es wird in diesem Fall von einer „Big-bang“ Integration gesprochen [Karo98] und [BoTu03]. Deswegen soll ein Projekt genügend Flexibilität besitzen, um Änderungen in den frühen Projektphasen zu akzeptieren. Wenn die verteilten Teams früh

involviert werden, dann können sie Änderungen durchführen, ohne großen Zeitverlust für Verhandlungen in Kauf nehmen zu müssen.

Probleme für die Agilen Methoden entstehen in diesem Zusammenhang durch die erhöhte Kommunikation innerhalb der kurzen Iterationszyklen. Je kürzer der Zyklus, desto mehr Kommunikation ist erforderlich. Diese Zusatzkosten müssen sorgfältig in der Planungsphase mitberechnet werden. Die Planungsgetriebenen Methoden haben einen entscheidenden Vorteil, wenn die Planung und das Design sorgfältig erstellt wurden. Wenn die Anforderungen stabil sind, kann das Projekt ohne weiteres in kleine, definierte und spezifizierte Teilprojekte unterteilt werden. Ein Teil dieser Module wird von den verteilten Teams übernommen und die Ergebnisse werden nach ein paar Wochen dem Auftraggeber weitergereicht. Sind die Lieferungen korrekt und lassen sie sich in das Gesamtkonzept integrieren, bekommen die verteilten Teams neue zu entwickelnde Module zugewiesen. Dafür müssen diese Lieferungen zwischen den verteilten Teams synchronisiert sein, denn wenn nur ein Teil der verteilten Teams die gefertigten Module liefert, ein anderer Teil des Teams aber nicht, kann dies zu Problemen bei der Durchführbarkeit der Tests führen. Mit Hilfe eines vom Auftraggeber erstellten Aufgabenplans mit Abgabefristen, können die verteilten Teams interne Pläne erstellen, so dass alle Teilnehmer den Aufgabenplan einhalten können.

### **Distanz**

Die geographische Distanz spielt ebenso eine nicht zu vernachlässigende Rolle. Je größer die Distanz zwischen den Teams ist, desto größer scheint auch die Zeit zwischen den Iterationen zu sein. Dieses Verhalten verstärkt sich durch erhöhte Reise- und Kommunikationskosten. Um die Übersicht über das Projekt beizubehalten und den Fortschritt des Projektes beobachten zu können, sollten nicht mehr als zwei bis drei Monate vergehen bezüglich der Iterationen der Softwareintegration [PaLa04]. Es ist durchaus sinnvoll die geplanten Arbeiten in Module zu unterteilen, so dass der Kommunikationsaufwand reduziert werden kann. Das kann allerdings nur mit einer vom Auftragnehmer gut verstandenen Architektur und Planung erreicht werden (siehe oben). Einen speziellen Fall stellen Projekte dar, die sich nicht inkrementell entwickeln lassen, da die Software sich nicht in kleine Module aufspalten lässt. Bei den Iterationen

werden dann nur inkomplette Teilprodukte abgegeben, was die Testbarkeit der Ergebnisse deutlich erschwert.

### **Management**

Die Technik wird dann zu einem Risiko, wenn zum Beispiel sowohl der Auftraggeber als auch der Auftragnehmer nicht dieselben Prozesse für die Softwareprojekte anwenden. Der Auftragnehmer kann die Prozesse, Modelle, Vorlagen, usw. vom Auftragnehmer übernehmen, damit die gewünschte Dienstleistung erbracht wird. Das Problem entsteht aber dann, wenn Dokumente oder Prozessstrukturen zwischen verschiedenen Sprachen übersetzt werden müssen. Wenn hier nicht auf die Genauigkeit der Sprache beachtet wird, läuft das Projekt Gefahr, dass verteilte Teams verschiedene Prozesse und Terminologien für dasselbe Projekt einsetzen.

Wenn mehr als zwei Entwickler am selben Quellcode arbeiten, sind Konflikte fast vorprogrammiert. In verteilten Teams befindet sich der Quellcode zusätzlich an verschiedenen Standorten verteilt. Es können mehrere Versionen desselben Quellcodes parallel existieren oder verschiedene Standorte wollen Updates durchführen. Wenn der Quellcode verteilt entwickelt wird, sollte nicht auf ein Konfigurationsmanagementtool verzichtet werden. Es soll versucht werden, nur eine Version der Software zu entwickeln, ein so genanntes „single branch“ (Einfachzweig).

Mit Hilfe des ausgewählten Versionierungssystems und des Konfigurationsmanagementtool, das in der CDE enthalten ist, wird das Speichern, die Weiterleiten im Fall von Änderungen und die Integration der gelieferten Ergebnisse unterstützt. Die Lieferungen werden so synchronisiert, dass beide Seiten immer auf dem aktuellsten Stand bleiben und sich schnell einen Überblick über den Stand der Software schaffen können, um mögliche Probleme und Fragen schnell zu identifizieren und zu lösen. Falls möglich, sollte an allen Standorten dieselbe oder sehr ähnliche Hardwarekonfiguration vorhanden sein, um Effekte oder Fehler reproduzieren zu können. Der Quellcode, das Design, die Anforderungen, die Testpläne, die Testergebnisse und alle wichtigen Dokumente bezüglich des Projektes sollen allgemein zugänglich sein. Die Rechtevergabe für Ein- und Auschecken von Dateien

ermöglicht, dass nicht jeder im Team Dokumente ändern kann. So kann an allen verteilten Standorten die Produktivität und Kompatibilität der Prozesse am besten unterstützt werden. Die Internetanbindung ist heutzutage ein nicht mehr allzu schwerwiegendes Problem. Projekte müssen aber immer wieder mit genügender Bandbreite ausgestattet werden. Mehr dazu im Kapitel 7.

Tabelle 5 fasst die hier aufgeführten Anpassungen zusammen:

	Anpassung	Wenige Teams / Einfache Projekte	Viele Teams / Komplizierte Projekte
<b>Dynamik</b>	<b>Synchronisierung</b>	Kurze Abstände Daily oder Weekly Builds Erhöhter Kommunikationsbedarf Bessere Projektsichtbarkeit	Weekly Builds Große Iterationszyklen Iterationsintervalle nicht größer als 2-3 Monate Gefahr einer „Big-bang“ Integration
	<b>Distanz</b>	Kürzere Distanzen, wenige Standorte	Große Distanzen mit vielen Standorte möglich
	<b>Management</b>	Multiversionentwicklung denkbar	Single branch (Einfachabzweig) notwendig Ähnliche Hardwarekonfiguration

**Tabelle 5. Eigenschaften der Achse Dynamik im angepassten Entscheidungsmodell.**

### 5.1.5 Personal

Die von Boehm und Turner definierten Personalebene[n] [BoTu03] und die dazu passenden Personalkombinationen (siehe Abschnitt 4.5.5) sind nicht als allgemeingültiges Rezept für die erfolgreiche Gestaltung von Projekten zu sehen. Diese Kombinationen sollen an die Einzigartigkeit eines jeden Projektes mit seinen Methoden und Definitionen angepasst werden. Dafür ist das Management beider Seiten zuständig, das das Personalverhalten, die Werkzeuge und die Arbeitsprozesse an die verteilten Teams anpasst. Wichtig ist dabei, wie die Kunden in das Projekt einbezogen werden können.

### Kundenrolle

Jedes Entwicklerteam verfügt über die nötigen Werkzeuge für die Softwareentwicklung aber sobald mehrere verteilte Teams verschiedene Werkzeuge anwenden, kann die Zusammenarbeit problematisch und komplex werden. Für die Entwickler ist es dann nicht nur wichtig zu wissen was entwickelt werden soll, sondern auch:

1. wie die Arbeiten durchzuführen sind,
2. wann Ergebnisse geliefert werden sollen und
3. wer für bestimmte Arbeiten zuständig ist.

Wird ein Agiler Ansatz konsequent verfolgt, wirkt sich das auf die Einflussnahme des Managements auf den Kunden aus. Im Fall, dass die Software vom Auftragnehmer verteilt entwickelt wird, wird der Auftraggeber (Kunde) stärker in das Projekt integriert. Der Auftraggeber stellt genügend Arbeitskräfte zur Verfügung, die dann dediziert für das Projekt und im Einklang mit dem verteilten Team arbeiten. Der Kunde steht also auf der Entwicklerseite als eine Art menschliche Schnittstelle zwischen Benutzer und Entwickler.

Verfolgt das Management einen Planungsgetriebenen Ansatz, dann wird in der Planungsphase ein Vertrag zwischen Auftragnehmer und -geber aufgesetzt, der eine Liste von Problemen und Lösungswegen enthält und die Durchführbarkeit des Projektes und die Verpflichtungen von Arbeitgeber und -nehmer regelt. Haben sich die Seiten geeinigt, wird der Vertrag unterzeichnet. Bis zu diesem Zeitpunkt sollten die Entwickler wissen, welche Module und Programme sie zu entwickeln haben und die Kunden wissen, was sie bekommen werden. Der Vertrag ist manchmal Auslöser für komplizierte Situationen, da er wenig flexibel ist und alle Kleinigkeiten enthalten muss. Das Vertrauen zwischen Auftraggeber und -nehmer basiert, im Gegensatz zu den Agilen Methoden, auf einer sachlichen Ebene mit Dokumentationen, Verträgen und Prozessen.

Die Benutzung von Werkzeugen reicht allein nicht aus, um die Projektkommunikation verschiedener Kulturgruppen zu verbessern, deswegen werden so genannte Kommunikationsprotokolle benötigt, um Probleme der Verständigung untereinander lösen zu können ([HuOc06], [BCKS01], [HeMo01]). Diese Protokolle werden von Referenzteilnehmern erarbeitet, die jeden Kulturkreis repräsentieren. Diese definieren zusammen wie die Kommunikation untereinander während des Projektes erfolgen soll. Als erster Punkt ist festzulegen, welche Sprache zum Einsatz kommt. Heutzutage ist Englisch die meistbenutzte Sprache für verteilte Projekte ([CaAb06], [HMFG00], [Matl05]).

Wenn der Quellcode gut dokumentiert und formatiert ist, verstehen andere Entwickler, Tester und Projektbeteiligte diesen besser und können Änderungen und Verbesserungen leichter durchführen ([GrHP99], [Balz01], [Karo98]). Es existiert eine Reihe von Autoren die Maßnahmen definieren, wie ein Entwicklerteam mit dem Quellcode konsistenter umgehen kann ([Balz01], [Bind00], [Karo98], [GrHP99], [HeGr99a]) und geben eine erste Idee, in welcher Form die Programmierrichtlinien sich harmonisieren lassen.

### „CRACKS“

Boehm und Turner definieren fünf wichtige Fähigkeiten, die Entwickler besitzen sollten [BoTu03]. Sie müssen:

1. kollaborativ sein (collaborative),
2. die Auftraggeberseite stellvertretend repräsentieren (representative),
3. Entscheidungsvermögen haben (authorized),
4. Engagement besitzen (committed) und
5. klug sein (knowledgeable).

Im Englischen wird das Wort „CRACKS“ benutzt, das sich aus den Buchstaben dieser Wörter zusammensetzt: „Collaborative, Representative, Authorized, Committed und Knowledgeable“. Besitzen die Mitarbeiter diese Eigenschaften nicht, läuft das Projekt Gefahr, dass die Arbeit zwischen den verteilten Teams und dem Auftraggeber nicht funktioniert. Ein typischer Fehler bei einem verteilten Softwareprojekt entsteht bei der Organisation der Personalplanung im Projekt. Im Idealfall bekommt die Firma, die über mehr menschliche Ressourcen verfügt, die wichtigsten Stellen zugeteilt. Dadurch entstehen kürzere Kommunikationswege, um wichtige Entscheidungen treffen zu können.

Die falsche Annahme ist es jedoch, dass die Firma mit mehr Ressourcen gleichzeitig über das bessere Expertenwissen verfügt. Im besten Fall werden die Kommunikationswege tatsächlich verkürzt und den Projektlauf schneller durchgeführt, im schlimmsten Fall verlängert sich dann die Lernkurve der verteilten Teams durch die fehlende Anwesenheit von Experten [Karo98].

Wenn Auftraggeber und -nehmer nicht zusammen arbeiten oder sich nicht einigen können, dann wird das verteilte Team frustriert sein und die Moral stark beeinflusst. Sind die „CRACKS“ keine stellvertretenden Personen der Auftraggeberseite beziehungsweise stehen sie nicht stellvertretend für die Kundenwünsche, kann es zu falschen Lieferungen der Produkte kommen. Nicht autorisierte „CRACKS“ benötigen mehr Zeit bei der Autorisierungssuche, was Verspätungen im Projekt verursacht oder Personen vor dem Problem stellt, dass Entscheidungen ohne Erlaubnis getroffen werden müssen.

Wenn zu guter letzt das Engagement und die erforderlichen Kenntnisse dieser „CRACKS“ nicht vorhanden sind und sie dennoch gebraucht werden, werden verspätete oder unabnehmbare Ergebnisse die Folge sein. Die Notwendigkeit eines sich vor Ort befindenden Mitarbeiters der Auftraggeberseite wird nicht immer gern angesehen, da die Arbeitskraft für diese Zeit dem Auftraggeber selbst nicht zur Verfügung steht. Im Falle eines Planungsgetriebenen Projektes begrenzt sich die Arbeit zwischen Auftraggeber und -nehmer auf die Planung und die Spezifikation des Projektes. In dieser Anfangsphase soll offen und beständig gearbeitet werden, auch mit Hilfe so genannter „CRACKS“. Später wenn das Produkt bereits in Entwicklung ist, werden diese nicht mehr so häufig gebraucht. Nur im Streit- oder Zweifelsfall werden sie herangezogen.

### **Kontrolle der Arbeit**

Der Auftragnehmer legt vor dem Projektstart einen Ablaufplan vor, wie er sich den Projektablauf weiter vorstellt. Die Prozesse sollen so definiert sein, dass die Anzahl der Iterationen und Lieferungen so genau wie möglich ausgeführt werden können. Ein Vertrag oder eine Vereinbarung sollte in zwei Teile gegliedert sein: ein festgelegter und ein veränderlicher Teil. Im ersten Teil wird das Rahmenwerk der Aktivitäten für das Projekt definiert, der Verlauf bei Änderungen und die wichtigsten Anforderungen. Im zweiten Teil werden die Themen aufgezeichnet, die bezüglich des festgelegten Teils variieren können.

Die Motivation des Personals spielt eine entscheidende Rolle, die über Erfolg oder Misserfolg entscheidet. Dabei ist die Rollenverteilung der Verantwortung und Zuständigkeiten zu beachten. Das Management muss Verantwortungen definieren und Zuständigkeiten spezifizieren, damit eine Lösung im Streitfall

schnell gefunden werden kann. Wer Verantwortung übernimmt soll fähig sein, spätere Fragen bezüglich der Handlungen zu beantworten. Karolak [Karo98] definiert den Begriff Verantwortung als:

*„Die Tat der Aufgabendurchführung und der resultierenden Tätigkeiten, wie zum Beispiel das Entwerfens eines Softwareteils, um einen geplanten Meilenstein zu erfüllen“.*

Zuständigkeiten dienen der Koordination und dem reibungslosen Zusammenwirken der verschiedenen Stellen innerhalb des Projektes. Karolak definiert zusätzlich den Begriff Zuständigkeit als [Karo98]:

*„Der anerkennende Besitz der Tätigkeit, unabhängig davon wer die Aufgaben durchgeführt hat, wie zum Beispiel das Liefern eines qualitativen Stückes verteilter Software“.*

Die Moral der Teams, insbesondere der sich vor Ort befindenden Teams, muss unterstützt werden. Die verteilte Entwicklung soll als Chance und nicht als Gefahr gesehen werden. Die Zusammenarbeit zwischen verteilten Teams soll als gesunder Wettbewerb oder Konkurrenz gesehen werden und die Teams sollen wissen, dass die Kooperation mit anderen Teams gut für das Projekt und Gesamtkonzept ist, damit die Teammitglieder die verteilte Softwareentwicklung nicht als mögliche Personalreduzierung sehen.

Im Idealfall soll der Projektmanager beziehungsweise der Auftraggeber alle Teammitglieder - seien diese vor Ort oder verteilt - so leiten als ob sie vor Ort arbeiten würden. Es soll immer beachtet werden, was sie machen und wie sie es machen, also die tägliche Arbeit immer überwachen und dementsprechend steuern. Es soll versucht werden eine Harmonisierung von Prozessen und Infrastrukturen zu schaffen. Tabelle 6 fasst die zwei hier aufgeführten Unterpunkte zusammen.

Personal	Anpassung	Wenige Teams / Einfache Projekte	Viele Teams / Komplizierte Projekte
	Kundenrolle und Kontrolle der Teams	Kunden stark in das Projekt integriert Teammitglieder werden geleitet, als ob diese vor Ort wären	Vertrag zwischen Auftragnehmer und -geber vorhanden. Kontrolle über die Arbeiten der Teammitglieder nicht so einfach
	„CRACKS“ vor Ort	Löst Probleme, kann aber zu Engpässe führen.	Es kann zu Komplikationen führen, denn die Kapazitäten stehen dem Auftraggeber nicht mehr zur Verfügung.

Tabelle 6. Angepasste Personaleigenschaften für das Entscheidungsmodell

Wichtig für das Projekt sind eine gute Kooperation und eine offene Kommunikation zwischen Teammitglieder und Management, um Missverständnisse zu reduzieren [HePB05], denn die Softwareindustrie wird lernen, mit der Erfahrung verteilter Projekte, einen Mehrwert mit geringeren Kosten zu generieren [Kent95]. Tabelle 7 fasst die fünf angepassten Bereiche des Entscheidungsmodells von Boehm und Turner zusammen und zeigt die dazu passenden Eigenschaften von Agilen Methoden und Planungsgetriebenen Ansätzen.

Faktor	Anpassung	Agile Methoden	Planungsgetriebene Entwicklung
Größe	<ul style="list-style-type: none"> <li>▪ Kontrolle und Sichtbarkeit</li> <li>▪ Partnerwahl</li> <li>▪ Zentrale Autorität</li> </ul>	<ul style="list-style-type: none"> <li>▪ Unautoritärer Ansatz</li> <li>▪ Schnelle Produktentwicklung</li> <li>▪ Kurze Iterationen</li> <li>▪ Anpassung an spontane Projektänderungen</li> </ul>	<ul style="list-style-type: none"> <li>▪ Vorhersagbarkeit und Stabilität werden erwünscht</li> <li>▪ Erhöhte Sicherheit</li> </ul>
Bedingungen für die Qualitäts- sicherung	<ul style="list-style-type: none"> <li>▪ Testverfahren</li> <li>▪ Know-how Verlust</li> </ul>	<ul style="list-style-type: none"> <li>▪ Inkrementelles Kodieren</li> <li>▪ Paarprogrammierung</li> <li>▪ Review-Techniken</li> <li>▪ Automatisierte Unit Tests</li> </ul>	<ul style="list-style-type: none"> <li>▪ Spätes Testen</li> <li>▪ Auskopplung von Testdefinitionen möglich</li> </ul>
Kultur der verteilten Teams	<ul style="list-style-type: none"> <li>▪ Kommunikation und Skalierbarkeit</li> <li>▪ Kultur</li> </ul>	<ul style="list-style-type: none"> <li>▪ Informeller Wissensaustausch</li> <li>▪ Bidirektionale und synchrone Kommunikation</li> <li>▪ Wissen personenabhängig</li> <li>▪ Schwierige Aufrechterhaltung aller Kommunikationswege</li> </ul>	<ul style="list-style-type: none"> <li>▪ Explizite, dokumentierte Kommunikation</li> <li>▪ Wissen eher personenunabhängig</li> </ul>
Dynamik	<ul style="list-style-type: none"> <li>▪ Synchronisierung</li> <li>▪ Distanz</li> <li>▪ Management</li> </ul>	<ul style="list-style-type: none"> <li>▪ Kurze Iterationszyklen</li> <li>▪ Mehr Kommunikation erforderlich</li> </ul>	<ul style="list-style-type: none"> <li>▪ Anforderung stabil</li> <li>▪ Projekte in Teilprojekte teilbar</li> <li>▪ Lieferungen von Ergebnisse synchronisierbar</li> </ul>
Personal	<ul style="list-style-type: none"> <li>▪ Kundenrolle</li> <li>▪ „CRACKS“</li> <li>▪ Kontrolle der Arbeit</li> </ul>	<ul style="list-style-type: none"> <li>▪ Mehr Einfluss vom Kunden</li> </ul>	<ul style="list-style-type: none"> <li>▪ Vertrag mit Verpflichtungen und wenig flexibel.</li> </ul>

Tabelle 7. Angepasste Faktoren Eigenschaften der Softwareentwicklung für verteilte Projekte

## 5.2 Organisationsstruktur für Entwicklerteams

Zusätzlich zu dem angepassten Entscheidungsmodell für verteilte Projekte, ist es notwendig zu klären, welches der Mitglieder der verteilten Teams welche Aufgabe zu erledigen hat, um die verteilten Teams besser zu organisieren und zu gestalten. Ob ein Projekt entweder in-house oder ausgelagert wird, wird vom Management vor Projektbeginn entschieden. Dabei spielen unter anderem das vorhandene und nötige Know-how, die Projektkosten und der zeitliche Rahmen eine entscheidende Rolle und müssen deshalb sorgfältig untersucht werden, um zu einer vorteilhaften Entscheidung zu gelangen.

Findet die Entwicklung innerhalb derselben Firma (in-house) statt, dann wird in diesem Fall mindestens vorausgesetzt, dass die Teams an verschiedenen Standorten verteilt sind, um das Projekt als verteilt einstufen zu können. Die Standorte können sich im selben Land oder in verschiedenen Ländern befinden. Dabei ist aber wichtig, dass die Teams zur selben Organisationsstruktur und -kultur gehören und dadurch dieselben Firmenstandards, Methoden und Prozesse benutzen. Das ändert sich bei der Zusammenarbeit mit einem externen Dienstleister. In diesem Bereich sind die Komplexität der Zusammenarbeit und die Risiken höher einzustufen als im Vergleich zu einer internen Abwicklung.

Larson und LaFasto [LaLa89] definieren drei Gebiete, denen jeweils ein Team zugeordnet wird. Je nach Zielvorstellung passt die eine oder andere Organisationsstruktur besser, um die Möglichkeiten eines Teams vollständig ausschöpfen zu können:

- Problemlösung
- Kreativität
- Taktische Durchführung

Ist das Projekt komplex, dann sollte der Fokus auf das Problemlösungsteam gerichtet sein. Hier wird das Lösen eines komplexen oder schlecht definierten Problems unterstützt. In diesem Team befinden sich vertrauenswürdige, intelligente und pragmatische Entwickler. Teammitgliedern analysieren

Probleme, um diese so schnell wie möglich lösen zu können. Dafür ist das Vertrauen an die zu leistende Arbeit sehr wichtig. Eine effektive und rasche Kommunikation ist notwendig, um Informationen schnellstmöglich für alle verfügbar machen zu können. Ein Beispiel eines Problemlösungsteams ist ein Team, dessen Aufgabe darin besteht, Softwarefehler schnell und zuverlässig zu beseitigen.

Beim Kreativitätsteam soll die individuelle und kollektive Selbständigkeit der einzelnen Teammitglieder unterstützt werden. Diese Art von Teams sucht nach Möglichkeiten und Alternativen für ein bestimmtes Problem. Ein solches Team benötigt selbstmotivierte, unabhängige und kreative Entwickler, die über viel Ausdauer verfügen, um zum Beispiel ein neues Feature für ein vorhandenes System zu entwickeln. Durch die Überwachung der individuellen Ergebnisse und die Möglichkeit ständiger Rückmeldungen wird gewährleistet, dass die Entwickler das Richtige tun und im Laufe des Projektes vorhandenes Verbesserungspotential für das Team herausfinden können.

Die Klarheit über die Verteilung der Aufgaben ist eine zu unterstützende Eigenschaft bei taktischen Teams, die die Durchführung eines klar definierten Plans und dessen Ziel erreichen wollen. Die Aufgaben, die die Teammitglieder lösen, sind sehr fokussiert und die Rollen müssen daher gut definiert sein. Ein Projekt, das die Weiterentwicklung oder die Anpassung einer Software vorsieht, ist ein Beispiel für ein taktisches Team. Hier sollen klare Rollen und Verantwortlichkeiten definiert sein, damit jeder weiß welche Aufgaben zu erledigen sind.

Je nach Zielsetzung und Organisationsstruktur lassen sich die verschiedenen Vorgehensmodelle und Methoden anpassen. Zum Beispiel kann die Planungsgetriebene Entwicklung mit dem Wasserfallmodell gut mit einem taktischen Team zusammen funktionieren, da die Rollen und Verantwortlichkeiten klar definiert sind. Das Spiralmodell passt sich gut an alle Strukturen an, sei es bei der Lösung von Fehlern im System, der Erstellung neuer Funktionalitäten oder der Aktualisierung eines bestehenden Systems. Agile Methoden finden in taktischen Teams deutlich stärkere Reibungspunkte als bei kreativen Teams mit vielen Freiheiten. Nur die gesammelte Erfahrung in

verschiedenen Projekten kann die Auswahl der unterschiedlichen Möglichkeiten erleichtern. Tabelle 8 zeigt die Organisationsstrukturen in zusammengefasster Form.

	<b>Problemlösung</b>	<b>Kreativität</b>	<b>Taktische Durchführung</b>
<b>Aufgabe</b>	Lösen eines komplexen oder schlecht definierten Problems.	Sucht nach Möglichkeiten und Alternativen für ein bestimmtes Problem.	Durchführung eines klar definierten Plans und dessen Ziel.
<b>Personal</b>	Vertrauenswürdige, intelligente und insbesondere pragmatische Entwickler.	Selbstmotivierte, unabhängige und kreative Entwickler, die über viel Ausdauer verfügen.	Strukturierte Teams, die an klare Rollen und Verantwortlichkeiten gewöhnt sind.
<b>Bedarf</b>	Vertrauen. Effektive und rasche Kommunikation.	Individuelle und kollektive Selbständigkeit der einzelnen Teammitglieder. Überwachung der individuellen Ergebnisse und die Möglichkeit ständiger Rückmeldungen	Klarheit über die Verteilung der Aufgaben.
<b>Beispiel</b>	Softwarefehler schnell und zuverlässig zu beseitigen.	Ein neues Feature für ein vorhandenes System zu entwickeln.	Weiterentwicklung oder die Anpassung einer Software.
<b>Entwicklungsmodell</b>	Planungsgetriebene Entwicklung. Agile Methoden können problematisch werden.	Agile Methoden.	Planungsgetriebene Entwicklung (Wasserfallmodell, Spiralmodell).

**Tabelle 8. Organisationsstrukturen für Entwicklerteams**

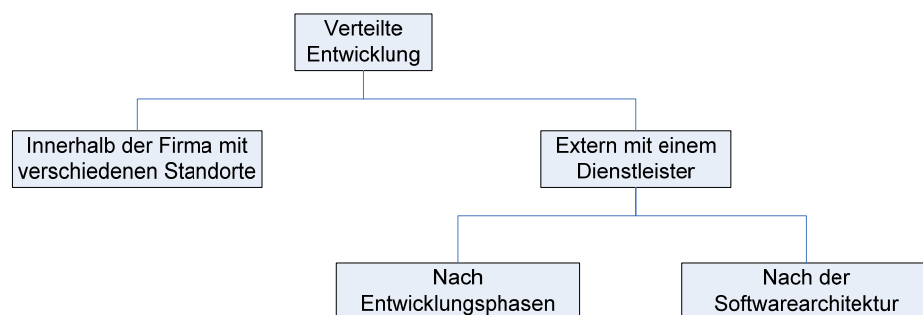
### 5.3 Teamaufteilung

Bei einem verteilten Projekt kann es sich um ein neues Produkt handeln, oder um ein vorhandenes Produkt, das angepasst oder verbessert werden soll oder zum Beispiel, um ein System, dass in eine neue Plattform portiert werden soll. Ein wichtiges Vorhaben bei der Verteilung der Softwareprojekte besteht darin die Fristen so zu setzen, dass alle Rahmenbedingungen wie Abgabefristen oder Produktionskosten erfüllt werden können. Die Idee ist es, in kürzester Zeit die besten Ergebnisse zu liefern und dabei die Produktionskosten so niedrig wie möglich zu halten. Die Projektzielsetzung ist mit der Organisationsstruktur der Entwicklerteams eng gekoppelt. Nach der Einteilung der Teams ist der letzte

Schritt eine Entscheidung zu treffen welches Team welche Aufgaben durchführen soll [Karo98]:

- welche Abteilung gegenüber der Firmenleitung für das Projekt verantwortlich ist,
- wie die Projektteilnehmer aufgeteilt werden sollen und
- ob das Management mit den getroffenen Entscheidungen einverstanden ist.

Abbildung 24 zeigt eine mögliche Aufwandverteilung des Softwareprojektes:



**Abbildung 24. Organisation der verteilten Entwicklung [Karo98]**

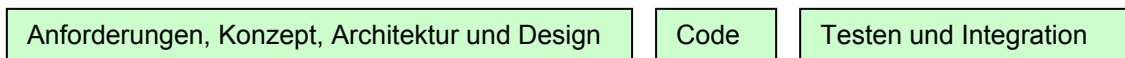
Die Aufgaben sollen so verteilt werden, dass die verteilten Teams ihr maximales Potential ausschöpfen können, um ihr Ziel zu erreichen. Obwohl in [GrHP99] berichtet wird, dass in verteilten Projektumgebungen keine Faustformel für die ideale Teamaufteilung existiert, wird anhand die von Karolak [Karo98] definierten Bereiche eine mögliche Teamaufteilung durchgeführt. Für externe Dienstleister definiert Karolak zwei Bereiche, in die sich ein verteiltes Softwareprojekt unterteilen lässt. Entweder werden die Projektaufgaben nach den Softwareentwicklungsphasen oder nach der Softwarearchitektur eingeteilt.

### 5.3.1 Teamaufteilung nach den Entwicklungsphasen

In diesem Falle können sich verteilte Teams in einem bestimmten Bereich der Softwareentwicklungskette spezialisieren. Dieser Ansatz ist interessant für mittelgroße und große Projekte, bei denen die Tendenz dahin geht, die anstehende Arbeit in verschiedene Phasen zu unterteilen. So können verteilte

Teams ausgesucht werden, die sich in verschiedenen Bereichen der Softwareentwicklung spezialisiert haben. Das Testen der Module oder die Kundenbetreuung sind hier beliebte Beispiele.

Der Vorteil dieser Unterteilung ist, dass jedes Team sich in einem ganz speziellen Bereich spezialisieren kann, was die Qualität der Ergebnisse und die Produktivität des Teams deutlich erhöht. Der große Nachteil ist dabei, dass die Teams auf die Ergebnisse der anderen verteilten Teams angewiesen sind. Arbeiten, die von einem anderen Team durchgeführt wurden, können missverstanden werden - ganz von der benutzten Sprache abgesehen. Die Kommunikationswege und Arbeitsmethoden sind deshalb sorgfältig zu beachten. Die typische Verteilung wird in Abbildung 25 gezeigt. In der Regel stellt ein Team A auf der Auftraggeberseite die Anforderungen, die Konzepte, die Softwarearchitektur und das Design fest. Ein zweites Team B entwickelt die Software mit Hilfe der erstellten Dokumente und Konzepte. Ein drittes Team C führt die Tests und die Integration durch.



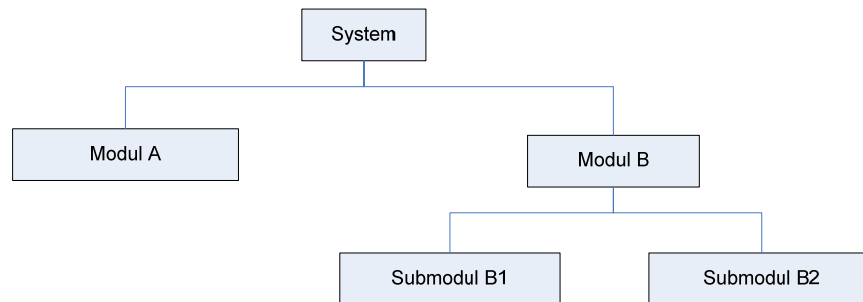
**Abbildung 25. Teamaufteilung nach Entwicklungsphasen [Karo98]**

In manchen Fällen kann ein Team mehrere Aufgaben durchführen, zum Beispiel kann das Team A Managementaufgaben durchführen und zusätzlich, nach der Quellcodeabgabe durch Team B, die Software testen und die Integration der Module durchführen. Es kann auch vorkommen, dass das verteilte Team der Auftragnehmerseite das nötige Training vom Auftraggeber bekommt, um ein vorhandenes System zu überarbeiten oder zu portieren. Dazu werden Quellcode, Dokumentationen und die nötige Expertise des alten Systems zur Verfügung gestellt.

### **5.3.2 Teamaufteilung nach der Softwarearchitektur**

Ein Team nach der Softwarearchitektur aufzuteilen stellt die andere Form der Teamaufteilung dar. Der Ansatz verfolgt das Prinzip von Teile-und-Herrsche, das heißt ein System wird in Subsysteme unterteilt. Die neu entstandenen

Subsysteme werden an verteilte Teams zur Abarbeitung weiter gegeben, die Expertenwissen und Erfahrung besitzen, um das Subsystem zu entwickeln. Dieser beliebte Ansatz hat den großen Vorteil, dass die Arbeit durch die Modulunterteilung erleichtert wird. Für verteilte Projekte ist das gut, denn die Subsysteme erfüllen spezifische Funktionalitäten, die sich isolieren lassen, was die Projektüberwachung erleichtert. Dieses Verhalten kann aber sehr schnell zu einem Problem werden, insbesondere wenn viele verteilte Teams an der Entwicklung beteiligt sind. Ein Modul wird abhängig von der zu implementierten Funktionalität in kleinere Module unterteilt (siehe Abbildung 26). Es wird vorausgesetzt, dass die zu implementierenden Module untereinander definierte Schnittstellen besitzen. Wenn verteilte Teams die Schnittstellen nicht sorgfältig implementiert haben, wird den Aufwand für die Schnittstellenintegration erhöht.



**Abbildung 26. Teamaufteilung nach der Softwarearchitektur [Karo98]**

Die Aufteilung verteilter Teams kann sich ergeben, in dem zum Beispiel ein Team A die Funktionalitäten nah an der Hardwareebene und ein verteiltes Team B die Funktionalitäten der Benutzeroberfläche entwickelt. Ein interessanter Fall ist wenn 2 Firmen, die zusammen an einem verteilten Projekt arbeiten, dasselbe Produkt in verschiedenen Märkten betreiben und dadurch bei der Produktentwicklung zusammenarbeiten. Zum Beispiel werden für den nordamerikanischen Markt einige Funktionen verlangt, die nicht für den europäischen Markt konzipiert sind. In diesem Fall wird die Applikation für einen speziellen Markt vom Team A, das in den USA sitzt, entwickelt und an das sich in Europa befindenden Team B für die Anpassung an den europäischen Markt weitergegeben.

## 5.4 Zwischen Agilität und Disziplin

In den Kapiteln 4.3 und 4.4 wurden Vorgehensmodelle und Methoden der Softwareentwicklung vorgestellt. Die Planungsgetriebene Softwareentwicklung ist für große, komplexe, sicherheitsbedingte und zuverlässige Projekte gedacht. Die Anforderungen und die Umgebung der Projekte sind im besten Fall stabil und vorhersagbar. Die zwei Vertreter, die ausgewählt wurden, sind das Wasserfall- beziehungsweise das Spiralmodell. Das Wasserfallmodell war eine der ersten Softwareentwicklungsmodelle, die 1970 ausgearbeitet und formell definiert wurden [Royc70]. Seine geringe Flexibilität und prozesslastigen Verfahren waren der Grund das Spiralmodell zu entwickeln [McCo01]. Dieses Risiko gesteuertem Modell fügt die nötige Flexibilität hinzu, die in den kleinen und mittelgroßen Projekten notwendig ist. Parallel zu der Planungsgetriebenen Softwareentwicklung entstanden die Agilen Methoden, die in den letzten Jahre durch das Agile Manifest<sup>(30)</sup> und die XP-Gemeinde an Interesse gewonnen haben [LaBa03]. Diese Methoden eignen sich für kleine Teams und Projekte, in denen alle Beteiligten involviert sind. Kunden, Benutzer, Auftraggeber und -nehmer sollen zusammen für den Projekterfolg arbeiten. Im Gegensatz zu dem Planungsgetriebenen Ansatz können diese Methoden mit schwankenden Anforderungen und Arbeitsumgebungen besser umgehen [BoTu03].

Die Globalisierung und die Entwicklung neuer Technologien erfordert, dass heutige Projekte sehr dynamisch entwickelt werden müssen ([HeMo01], [LHKP03], [Scha06a]). Es ist heutzutage fast zu einer Regel geworden, dass Projekte sich viel schneller an eine änderbare Umgebung anpassen müssen als früher. Die eingesetzten Technologien und die Kundenwünsche ändern sich schnell. Die Softwareentwicklung, insbesondere die verteilte Softwareentwicklung muss schneller auf Änderungen und Agilität reagieren, da diese Eigenschaften immer wichtiger in der Softwareentwicklung werden [BoTu04b]. Nichts desto trotz werden heutzutage immer noch Planungsgetriebene Entwicklungsmodelle für die Softwareentwicklung benötigt, da die Agilen Methoden in komplexen Umgebungen nicht reibungslos skalieren und Hilfe einiger Methoden der Planungsgetriebenen Entwicklung brauchen.

---

<sup>30</sup> Agile Manifesto (2001): Manifesto for Agile Software Development.  
Quelle: <http://www.agilemanifesto.org> (Abruf am: 27.03.07)

Um diese Ansätze herum haben Boehm und Turner ein Entscheidungsmodell entwickelt ([BoTu03], [BoTu04a], [BoTu04b]). Kein Projekt unterstützt nur die Disziplin der einen oder nur die Agilität des anderen Ansatzes. Ganz im Gegenteil, Softwareprojekte sollen die Stärken beider Ansätze vereinigen um bessere Ergebnisse zu liefern. Boehm und Turner definierten dieses Entscheidungsmodell, um das Management bei der Suche des besseren Methoden-Mix zu unterstützen. Mithilfe von vier vorher definierten Projektbereichen und fünf kritischen Faktoren, kann das Management den Spielraum für die nötigen Verbesserungen mit Hilfe eines Polardiagramms (siehe Abbildung 16, Kapitel 4.5) finden. Es wurden fünf Faktoren definiert, die in jedem Projekt zu beachten sind:

1. die Projektgröße,
2. die Sicherheitsentscheidende Bedingungen,
3. die Softwareentwicklungskultur,
4. die Dynamik des Projektes und der Umgebung und
5. die Personaleigenschaften.

Die vorgestellten Vorgehensmodelle und Methoden besitzen Eigenschaften, die Vor- und Nachteile aufweisen. Jedes Projekt, in Abhängigkeit seiner Eigenschaften, lässt sich dadurch an die Vorteile jedes Ansatzes anpassen. Karolak definiert drei Risikobereiche, die in jeder virtuellen Organisation vorkommen können [Karo98]. Risiken können organisatorischer, technischer oder kommunikationsbedingter Art sein. In den letzten Jahren ist die verteilte Softwareentwicklung empirisch stark untersucht worden, um neue Erkenntnisse zu gewinnen und Strategien zu entwickeln. Ein Beispiel hierfür zeigen Herbsleb, Paulish und Bass in einer Studie über neun verteilte Projekten [HePB05]. Diese gewonnenen Erkenntnisse sind hier in das Entscheidungsmodell von Boehm und Turner integriert und in diesem Kapitel angepasst worden.

Ein Entscheidungsmodell versucht dem Management zu zeigen, welche Maßnahmen für den Erfolg des Projektes hilfreich sein können, sorgt aber nicht dafür, dass ein Projekt erfolgreich durchgeführt wird. Eine gute Kommunikationsstruktur und eine korrekte Teamaufteilung sind auch hierfür entscheidend. Es existieren drei mögliche Organisationsstrukturen, in denen ein

verteiltes Team eingestuft werden kann. Es kann sich um ein Kreativitäts-, ein Problemlösungs- oder ein Taktik-Team handeln. Je nach Zielsetzung und Organisationsstruktur lassen sich die verschiedenen Ansätze der Softwareentwicklung anpassen [LaLa89]. Wird die entsprechende Organisationsstruktur gefunden und Maßnahmen für das Projekt erkannt, müssen die verteilten Teams schließlich korrekt aufgeteilt werden. Karolak [Karo98] formuliert hierfür eine Aufgabenverteilung entweder nach den Entwicklungsphasen oder nach der Softwarearchitektur. Diese Aufteilung hilft der Projektleitung und den verteilten Teams Verantwortlichkeiten und Hierarchien im Projekt zu identifizieren und zu vergeben.

## 6 Kollaborative Entwicklungsplattform (CDE)

In diesem Kapitel wird die Kollaborative Entwicklungsplattform (Collaborative Development Environment CDE) behandelt, die für jedes verteilte Softwareentwicklungsprojekt angewendet werden kann. Es werden die auf dem Markt verfügbaren Lösungen gezeigt und Überlegungen angestellt, ob die Eigenschaften einer „idealen“ CDE sind, die für eine geographisch verteilte Arbeitsumgebung geeignet sind.

Heutzutage existiert eine große Anzahl an Werkzeugen, die die Zusammenarbeit und Kollaboration im Team unterstützen, zum Beispiel helfen unter anderen Desktop- und Webanwendungen verteilten Teams bei der Verteilung und Zusammenarbeit von Dokumenten. Eine CDE besteht aber aus mehr als nur diesen Werkzeugen [BoBr02]. Sie soll einen virtuellen Arbeitsraum zur Verfügung stellen, um verteilten Teams bei der Lösung gemeinsamer Aufgaben der Softwareentwicklung besser zu unterstützen.

Das Auffinden und die Benutzung einer guten CDE ist für verteilte Teams eine der wichtigsten Grundvoraussetzungen für eine erfolgreiche Zusammenarbeit. Mit Hilfe solcher Umgebungen können Teams tausende von Kilometern entfernt sein und die Teilnehmer müssten sich gar nicht persönlich kennen, um gemeinsam am selben Softwareentwicklungsprojekt zur Erstellung einer Software zusammen zu arbeiten. Dabei gilt es Aufgabenstellungen anzunehmen, Änderungswünsche durchzuführen und Probleme und unerwartete Vorkommnisse zu lösen. Versuche in dieser Richtung hat die Open Source Community erfolgreich durchgeführt. Diese Community hat bewiesen, dass (Hobby-) Entwickler erfolgreich in einem verteilten, großen Softwareentwicklungsprojekt zusammenarbeiten können [Matl05].

## 6.1 Teams

Für eine effiziente Aufgabenlösung muss die Interaktion zwischen den Teammitgliedern reibungslos funktionieren. In [BoBr02] wird aufgezeigt wie unterschiedlich die Entwicklerteams sein können. Die Mitgliederanzahl eines Entwicklerteams, die sich am besten bewährt hat, beträgt zwischen vier und acht Mitgliedern. An zweiter Stelle stehen Teams mit einem oder zwei Entwicklern. Den Teamgrößen sind nach oben keine Grenze gesetzt, es gibt auch Teams mit einer Größe von ein- bis zweihundert Mitgliedern. Wenn alle Projektbeteiligten zu einem Entwicklerteam hinzugezählt werden, sind am Ende zwischen ein paar Dutzend und ein paar Hundert Mitgliedern im selben Team beteiligt. Es gibt somit von vorneherein keine grundsätzliche Anzahl an Teammitgliedern.

Wie in [Stru94] berichtet wird, leisten Entwickler neben dem Kodieren noch andere Aufgaben:

- sie organisieren ihre Arbeitsumgebung,
- gestalten Prozesse,
- vertreten Ideen,
- reden über ihre Entscheidungen bezüglich des Designs des Softwareproduktes und
- arbeiten mit anderen Teammitgliedern.

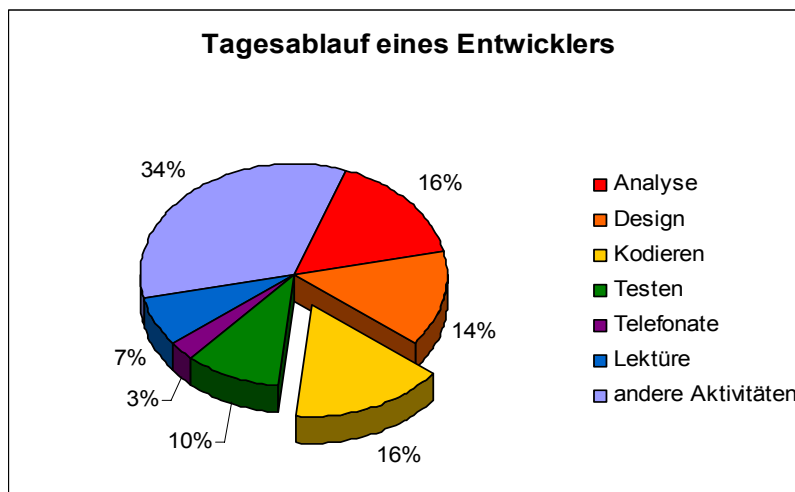


Abbildung 27. Aufgabenverteilung eines Entwicklers [BoBr02]

Eine andere Studie, von [BoBr02], die im März 2001 mit 50 Entwicklern durchgeführt wurde, zeigt die Aufgabenverteilung an einem gewöhnlichen Arbeitstag. Allein für die Analyse benötigt ein Entwickler 16% seiner Arbeitszeit, 14% für das Design, 16% für die Kodierung, 10% für das Testen, 3% für Telefonate und 7% für die Lektüre. Abbildung 27 macht deutlich, dass der Entwickler viel Zeit in Tätigkeiten investiert, die nicht direkt mit dem Kodieren zu tun haben, ihn aber bei der Problemlösung unterstützen.

Herkömmliche Integrierte Entwicklungsplattformen (Integrated Development Environment IDE) wie zum Beispiel VisualStudio<sup>(31)</sup> von Microsoft®, die Open Source Plattformen NetBeans<sup>(32)</sup> und Eclipse<sup>(33)</sup>, WebSphere Studio<sup>(34)</sup> von IBM oder JBuilder<sup>(35)</sup> von Borland sind ursprünglich nicht für eine geographisch verteilte Arbeitsumgebung gedacht gewesen, da sie für die Kodierungsaktivitäten eines einzelnen Entwicklers konzipiert worden sind. Das heißt sie sind auf ein Individuum abgestimmt und deswegen entwicklerorientiert. Aktivitäten wie Interaktion, Kommunikation und Koordination wurden bei der IDE Konzipierung nicht in Betracht gezogen und daher dem Anteil der sozialen Dynamik eines Entwicklerteams keine Beachtung geschenkt. Eine CDE versucht das Team als Zentrum des Entwicklungsprozesses zu betrachten und ist deswegen teamorientiert [SeCS06].

Aktuelle Werkzeuge wie Intranet, E-Mail, Instant Messaging, Chaträume, Wikis oder Diskussionsforen versuchen die Kommunikationslücke von herkömmlichen IDEs zu füllen, jedoch existieren diese Lösungen unabhängig voneinander. Diese Lösungen für die Teaminteraktion müssen in einem virtuellen Raum gesammelt und für alle Teammitglieder zur Verfügung gestellt werden. Dies ist das wichtigste Ziel einer CDE, denn verteilte Teams brauchen eine Plattform, die auch andere Aspekte berücksichtigt als die einer IDE [BoBr02].

---

<sup>31</sup> <http://msdn2.microsoft.com/en-us/vstudio/default.aspx> (Abruf am 27.03.07)

<sup>32</sup> <http://www.netbeans.org/> (Abruf am 27.03.07)

<sup>33</sup> <http://www.eclipse.org/> (Abruf am 27.03.07)

<sup>34</sup> <http://www.ibm.com/hotmedia/> (Abruf am 27.03.07)

<sup>35</sup> <http://www.borland.com/de/products/jbuilder/> (Abruf am 27.03.07)

## 6.2 Fachwissen und Kommunikation

Das formale und das informelle Wissen sind die zwei Wissensquellen, die in einem Softwareentwicklungsprojekt auffindbar sind [SeCS06] (siehe Abbildung 28). Das formale Wissen beinhaltet zum Beispiel Anforderungen, Architekturdiagramme, Schnittstellenspezifikationen, Quellcode oder Testfälle. Das formelle Wissen läuft Gefahr, dass aufgrund der kulturellen Unterschiede eine Anforderung oder ein Kundenwunsch anders verstanden wird und dadurch von den verschiedenen Entwicklern getrennte Wege verfolgt werden.

Das informelle Wissen besteht dagegen aus der menschlichen Ebene mit Teammitgliedern und Benutzern der Software, sowie der sachlichen Ebene mit ad-hoc Dokumenten bezüglich des Softwareentwicklungsprojektes und allen gesammelten Notizen während vergangener Projektsitzungen. Hierbei bildet der menschliche Faktor eine Gefahr für die Wissensverteilung, da das Wissen auf der Beziehungsebene normalerweise nicht dokumentiert werden kann und daher personenabhängig ist, da jedes Teammitglied die Prozesse und Vorgehensweisen anders beschreibt [NgBV06].

Ziel der Fachwissensverteilung ist es einen Schnittpunkt dieser zwei Wissensquellen zu finden und sie dementsprechend zu unterstützen. Das Fachwissen soll dabei automatisch weitergeleitet werden und so effektiv sein wie die direkte, persönliche Kommunikation zwischen Teammitgliedern [Karo98].

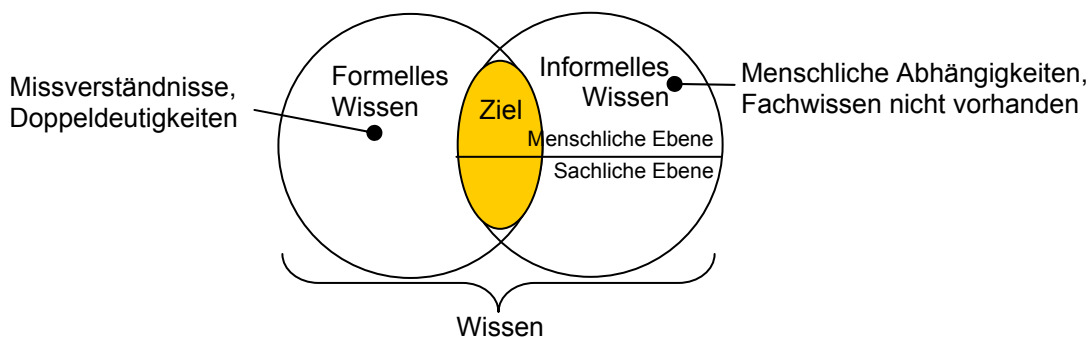
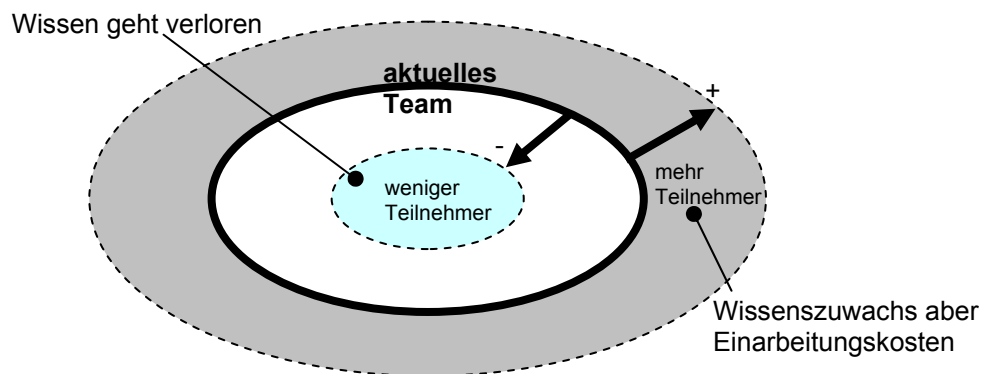


Abbildung 28. Informelles und formelles Wissen [SeCS06]

Die Informations- und Fachwissensuche in verteilten Umgebungen ist sehr zeitaufwendig und daher eine zusätzliche Belastung für die einzelnen Teammitglieder, da diese sich nicht auf ihre eigentlichen Kernaufgaben konzentrieren können [BoBr02]. Wenn neue Mitglieder für das Team gewonnen werden sollen, so genannte Projekteinsteiger, müssen diese zuerst in das Softwareentwicklungsprojekt eingearbeitet werden. Diese Anfangskosten für das Kennenlernen des Teams und des Softwareentwicklungsprojektes zahlt sich allerdings durch den erhofften Wissenszuwachs aus, den die neuen Teilnehmer mit sich bringen. Verringert sich aber die Anzahl der Teilnehmer, verringert sich auch automatisch das vorhandene Fachwissen, da mit dem Verlassen des Softwareentwicklungsprojektes das informelle Wissen des jeweiligen Projektteilnehmers verloren geht. Abbildung 29 stellt dieses Verhalten in graphischer Form dar.

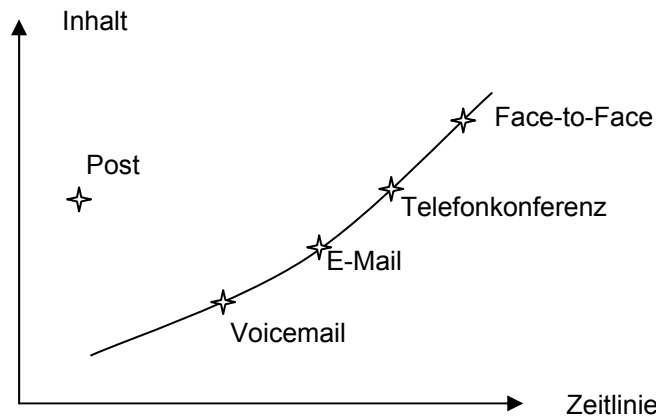


**Abbildung 29. Fachwissen ist von den Teilnehmern abhängig [SeCS06]**

Da es sich keineswegs um ein triviales Problem handelt, das auf einfache Weise gelöst werden kann, sondern um Probleme der Wissensverteilung, die über Erfolg und Misserfolg eines Softwareentwicklungsprojektes entscheiden, ist ein entsprechendes Werkzeug für die Kommunikation im Team und für die Nachhaltigkeit des Fachwissens im Softwareentwicklungsprojekt unabdingbar [SeCS06].

Abbildung 30 zeigt die gängigen Kommunikationsmethoden auf, die in verteilten Softwareentwicklungsprojekten ihre Anwendung finden [Karo98]. Diese Methoden sind von den zwei Dimensionen Zeit und Kommunikationsinhalt

abhängig. Die Zeitlinie definiert sich durch die Geschwindigkeit, mit der eine vom Sender verschickte Information empfangen wird. Der Kommunikationsinhalt ist dann die Menge an verbaler und schriftlicher Kommunikation, die der Sender am Stück verschicken kann.



**Abbildung 30. Effektivität der Kommunikationsmethoden [Karo98]**

Die Post ist dabei die langsamste Methode der Kommunikation, kann aber einen höheren Informationsanteil enthalten. Eine E-Mail-Benachrichtigung versucht sich an die Qualität eines Briefes anzunähern, ist lange nicht so persönlich und anfassbar wie ein Schreiben, wird aber schneller verbreitet und kann mehrere Empfänger gleichzeitig erreichen. In einer Telefonkonferenz wiederum können mehrere Teilnehmer, die sich an verschiedenen Standorten befinden, die Aktualität des Softwareentwicklungsprojektes, die vorhandenen Probleme und etwaige mögliche Lösungen direkt diskutieren. Dieses Medium scheint viel besser für die Kommunikation geeignet zu sein als E-Mail- oder Voicemailkontakt, kann aber über einen genauso hohen Anteil an Doppeldeutigkeiten verfügen, da der physische Kontakt und die körpersprachlichen Signale nicht vorhanden sind.

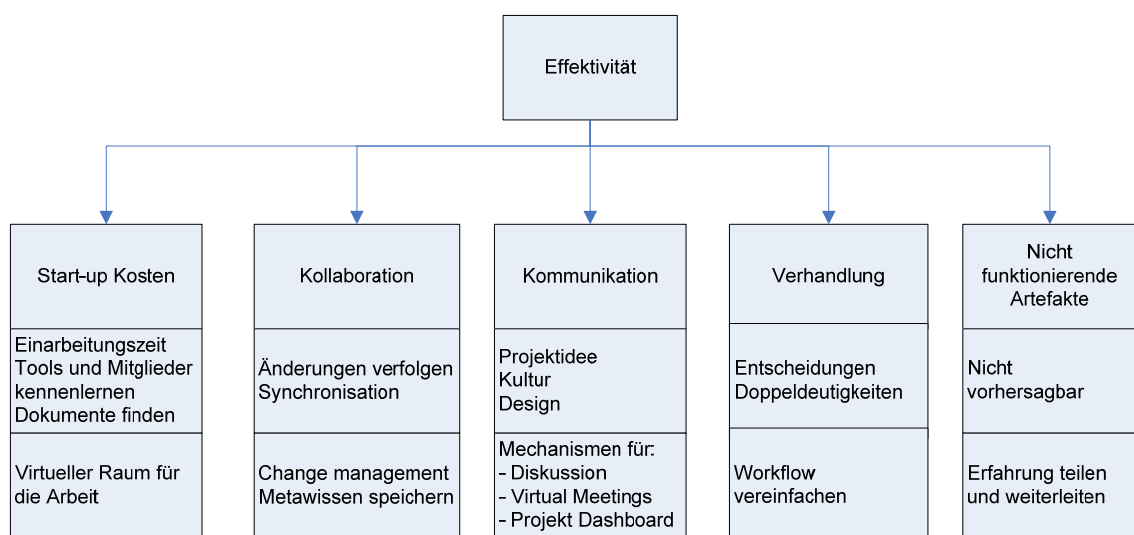
Tatsache ist, dass keines der Kommunikationsmedien die direkte Kommunikation zwischen zwei Menschen ersetzen kann, daher ist es wichtig eine synchrone, persönliche und direkte Art der Kommunikation im Softwareentwicklungsprojekt zu unterstützen oder, wie in diesem Fall, mit nötigen Werkzeugen für die Fachwissensvermittlung zu simulieren.

### 6.3 Reibungspunkte

Ziel einer CDE ist es eine reibungslose Arbeitsumgebung für die Softwareentwicklung zu schaffen. In [BoBr02] werden sechs Reibungspunkte, so genannte „points of friction“, definiert, die ihre Auswirkung auf die Effektivität im Team haben:

- Start-Up Kosten
- Kollaboration
- Kommunikation
- Zeitverlust
- Verhandlung zwischen Teilnehmer
- Nicht funktionierende Artefakte

Abbildung 31 zeigt eine Variante des Ansatzes von Booch und Brown in einer graphischen Darstellung. Ein verteiltes Team kann nur dann effektiv sein, wenn es diesen Punkten entgegen steuert.



**Abbildung 31. Reibungspunkte [BoBr02]**

Der erste Punkt behandelt die Start-Up Kosten. Sie beschreiben die Kosten, die durch die Einarbeitung neuer Mitglieder, das Kennen lernen neuer Werkzeuge, die Suche nach neuen und alten Dokumenten und durch weit entfernte Teammitglieder verursacht werden. Diese Kosten können durch eine

gemeinsame CDE, die immer den aktuellen Stand des Softwareentwicklungsprojektes beinhaltet, für alle Mitglieder reduziert werden.

Eine gute CDE soll die Zusammenarbeit geographisch verteilter Teams unterstützen. Die Variablen Raum und Zeit stellen eine Hürde für die Kollaboration zwischen den Teams dar. Je länger ein Softwareentwicklungsprojekt dauert und je weiter verstreut die Teams sind, desto mehr Mitarbeiter sind in den verschiedenen Projektphasen involviert, wodurch die Zuständigkeiten mit der Zeit nicht mehr eindeutig klar zu definieren sind. Eine gute CDE soll Projektveränderungen und Zuständigkeiten automatisch verfolgen und das Wissen über das Softwareentwicklungsprojekt speichern.

Kommunikation ist der zentrale Faktor einer CDE, jedoch auch der am schwersten Umzusetzende. In verteilten Umgebungen steigt der Kommunikationsbedarf drastisch an [OHRG04] (siehe Kapitel 5.1). Es müssen Mechanismen zur Verfügung gestellt werden, die die Qualität der Kommunikation zwischen den Teams fördern.

Um Doppeldeutigkeiten zu vermeiden, müssen sämtliche Projektmitglieder miteinander in Verbindung stehen, um sich so mehr Klarheit über den aktuellen Zustand des Softwareentwicklungsprojektes verschaffen zu können. In verschiedenen Teams können die Mitglieder zwar verschiedene Weltanschauungen besitzen, diese müssen aber auf einen gemeinsamen Nenner gebracht werden. Diese Doppeldeutigkeiten können dadurch vermieden werden, dass der Arbeits- und Informationsfluss automatisiert und vereinfacht wird.

Der letzte Reibungspunkt behandelt die Technologie und die Werkzeuge der verschiedenen Teammitglieder, die nicht immer zusammen passen. Verschiedene Konfigurationen der Rechner im Einsatz können Kompatibilitätsprobleme erzeugen. Diese Probleme sind in den meisten Fällen nicht vorhersehbar und können mit Hilfe der Zusammenarbeit der Teams ausgeschlossen werden. Die CDE soll das vorhandene Wissen und gewonnene Erfahrungen weiterleiten, damit alle Beteiligten auf dem aktuellsten Stand sind.

## 6.4 Aktuelle CDEs

Aktuell existiert eine stetig wachsende Anzahl an CDEs auf dem Markt. Alle gängigen Lösungen versuchen die verschiedenen Eigenschaften einer idealen CDE widerzuspiegeln (siehe Abschnitt 6.6) und sich an verteilten Teams zu orientieren. Es werden in diesem Abschnitt einige CDEs mit ihren Eigenschaften, ihren Stärken und ihren Schwächen vorgestellt. Einerseits gibt es die kostenfreien CDEs wie MILOS, GotDotNet und SourceForge.net, zum anderen kostenpflichtige Werkzeuge wie SourceForge® Enterprise Edition, CollabNet und GForge.

### 6.4.1 MILOS

Der Name MILOS kommt aus dem Englischen und steht für „Minimally Invasive Long-term Organizational Support“. Die Software wurde in Zusammenarbeit der Universitäten Calgary und Kaiserslautern Ende der Neunziger Jahre entwickelt. MILOS ist eine CDE, die in Java programmiert wurde und als Open Source Softwareentwicklungsprojekt betrieben wird. Diese webbasierte Applikation unterstützt hauptsächlich Koordinationsarbeiten und die Automatisierung der Softwareentwicklungsprozesse innerhalb des Teams über das Internet [MSHK99]. Die Zielgruppe dieser CDE soll hoch dynamisch aufgebaut sein, das heißt, dass die genauen Projektaufgaben am Anfang oftmals noch nicht konkret feststehen und sich im Laufe der Zeit ändern werden.

MILOS baut auf einem dreischichtigen Ansatz auf: Client, Applikation und Daten [MSHK99]. Auf der Client-Seite wird beim Benutzer der Arbeitsbereich angezeigt. Der MILOS Server wird untergliedert in: der Ressourcenpool, die Prozessmodellierung, das Projektmanagement und das Workflowmanagement (siehe Abbildung 32).

Der Ressourcenpool ist zuständig für das Management von Rollen und Agenteneigenschaften des Entwicklerteams, das heißt für die Eigenschaften der Projektteilnehmer. Diese Funktion zeigt die Strukturen der Teams und der Firmen auf und soll der Unterstützung der Aufgabenplanung dienen. Mit Hilfe

des MILOS-Servers kann der Projektmanager nun entsprechende Teammitglieder für eine bestimmte Aufgabe finden. Mit der Komponente der Prozessmodellierung werden formale Prozessdefinitionen wie Flusskontrolle, Prozessverbesserung, Datenflussspezifikationen oder zusätzliche Qualifikationen abgebildet. Die dritte Komponente ist die des Projektplanmanagements. Diese unterstützt den Projektmanager beim Planen und Anpassen des Softwareentwicklungsprojektes, da er mit dieser Komponente den Start und das Ende des Softwareentwicklungsprojektes oder eine Teilaufgabe desselben besser verwalten und Teammitgliedern Aufgaben zuweisen kann. Die letzte Komponente, die des Workflowmanagements, verwaltet so genannte „to-do“-Listen für involvierte Agenten und den aktuellen Projektstatus.

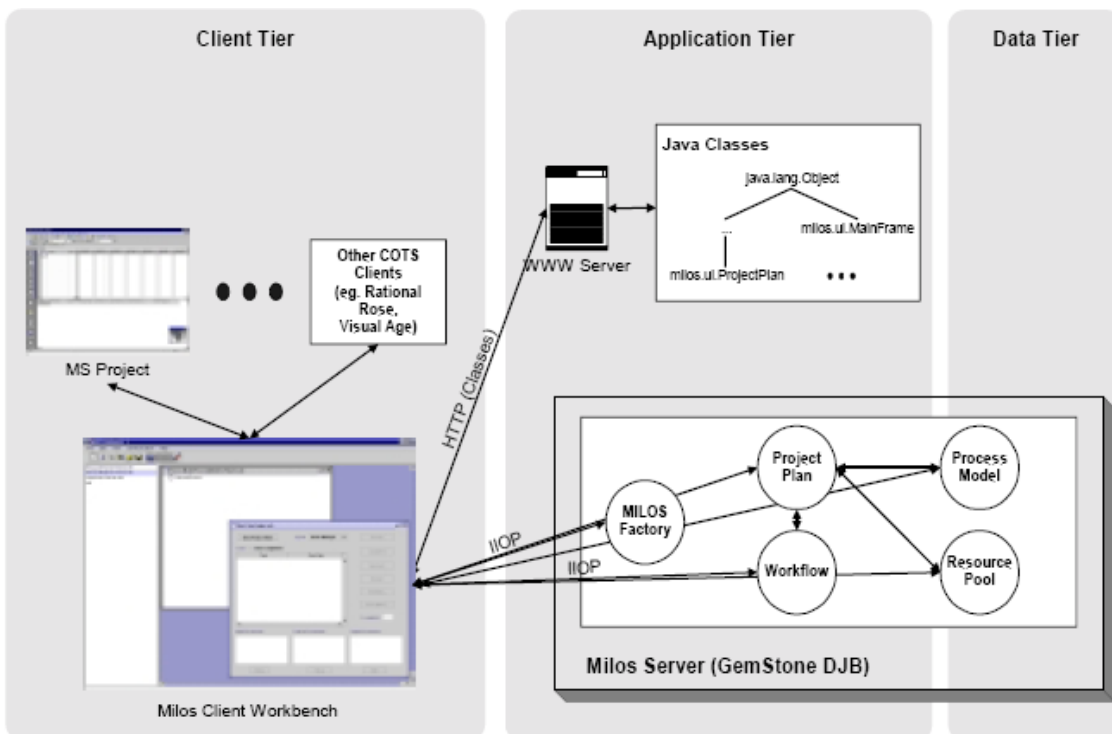


Abbildung 32. Milos Applikationsmodell [RiMa03]

Damit die Unterstützung des Softwareentwicklungsprozesses mit MILOS erfolgreich in Gang kommt, muss der Projektplaner zuerst einen vorläufigen Ablaufplan mit Zuständigkeiten in einem externen Programm erstellen, zum Beispiel in MS-Project von Microsoft®. Dieser Ablaufplan wird dann nach MILOS exportiert. Teammitglieder und Entwickler können sich dann in MILOS via Internet anmelden, wobei jedes Teammitglied eine eigene angepasste

Ansicht bekommt mit einer Liste von offenen Punkten („to-do“ Liste) und Beschreibungen der Aufgaben. Für die zu lösende Aufgabe kann der Einzelne die relevanten Informationen und Dokumente nachschlagen, die auch in der Benutzeransicht des Arbeitsbereiches in Kurzform angezeigt werden. Die betroffenen Teammitglieder werden dann sofort automatisch benachrichtigt, wenn nötige Dokumente freigegeben wurden oder andere Teammitglieder Änderungen im Projektplan durchgeführt haben. Jede Aufgabe wird mit einem Start- und einem Endtermin vorgesehen. Wenn das zuständige Teammitglied die Termine nicht einhalten kann, werden alle Betroffenen, insbesondere der Projektleiter, automatisch informiert. Zwischenstände können abgespeichert und exportiert werden, so dass jedes Team immer den aktuellen Status abfragen kann.

Der gesamte Prozess ist dynamisch anpassbar, das heißt wenn sich Teammitglieder mit Aufgaben verspäten, kann der Ablaufplan korrigiert werden. Wenn Zuständigkeiten zu Beginn des Softwareentwicklungsprojektes nicht klar definiert sind, können diese zu einem späteren Zeitpunkt zugewiesen werden. Auch Aufgaben und Prozesse können geändert, hinzugefügt oder neu definiert und leicht ergänzt werden.

Das Projekt MILOS wurde im Laufe der Zeit in zwei Projekte aufgeteilt. In Kaiserslautern wurde MILOS-KL weiter entwickelt mit dem Fokus auf die Softwareentwicklung in verteilten Unternehmen, der Metaplanung und prozessorientiertem Wissensmanagement. In Calgary entwickelte sich MASE (MILOS-ASE) weiter. Der Fokus lag hier auf Agilen Methoden, der webbasierten Systementwicklung und dem Wissensmanagement [ChMa04]. Somit spricht für MILOS [RiMa03], dass die Verfolgung von Entscheidungen verbessert wird, Ablaufpläne während der Entwicklung dynamisch angepasst oder verbessert werden können und dass alle betroffenen Teammitgliedern automatisch benachrichtigt werden, wenn Änderungen stattgefunden haben.

MILOS ist ein Zusatzwerkzeug, das die Zusammenarbeit im Team verbessert. Das Team kommt sich näher, da alle Teilnehmer über Projektänderungen und -informationen automatisch vom System informiert werden. Die Entwicklung des Quellcodes wird weiterhin mit einer vorhandenen IDE weiter durchgeführt und

der Benutzer arbeitet weiterhin mit den ihm bekannten Werkzeugen für die Softwareentwicklung, wobei er jedoch MILOS als „Tool über allen anderen Tools“ anwendet, was den Lernprozess für das neue Werkzeug verkürzt.

Alle offenen Fragen bezüglich Zugangssicherheit, Firewalls, und so weiter müssen im Voraus geklärt werden. Der MILOS-Server kann mit seinem zentralen Ansatz zu einem möglichen „single point of failure“ werden und muss dementsprechend geschützt sein. Wo MILOS ausbaufähig bleibt, ist der Bereich der Fähigkeitsfindung sämtlichen Teilnehmer, wofür dieses Werkzeug keine Funktionalitäten präsentierte.

### 6.4.2 GotDotNet

Microsoft® hat für die .NET Community eine Webseite für Open Source Projekte entwickelt, die Arbeitsbereiche für Softwareentwicklungsprojekte zur Verfügung stellt. Diese Webseite ist unter der URL <http://www.gotdotnet.com><sup>(36)</sup> zu finden. Die wichtigen Funktionen dieser Lösung sind:

- Die Verwaltung des Quellcodes eines verteilten Entwicklerteams. Die verfügbaren Schnittstellen für die Zusammenarbeit sind dabei das Web, der Windows Forms Client und Visual Studio.
- Konfigurationsmanagement mit Versionskontrolle.
- Aufzeichnung von Defekten, Aufgaben und Vorschläge. Definition von Meilensteinen für zukünftige Projektphasen.
- Kommunikation zwischen Teams, Diskussion und Ermittlung von Nachrichten mit Hilfe von RSS Feeds und E-Mail.
- Speicherraum mit 30MB und Downloadbereich mit 10MB für Releases. Der Speicherraum kann mit einer speziellen Genehmigung bis auf 60MB erweitert werden.

Für Teammitglieder stehen drei mögliche Benutzergruppen mit unterschiedlichen Richtlinien zur Verfügung: Besitzer, Administrator und Teilnehmer. Der Besitzer ist der Projektmanager; er erstellt den Arbeitsbereich,

---

<sup>36</sup> Abruf am: 27.03.07

lädt andere Teammitglieder ein und lädt die nötigen Dateien und Dokumenten für den Projektstart hoch. Die Administratorengruppe hat die gleichen Rechte wie der Besitzer mit dem einzigen Unterschied, dass diese nur vom Projektbesitzer hochgestuft werden kann. Benutzer der Teilnehmergruppe haben nur eine beschränkte Erlaubnis. Sie dürfen keine Teilnehmerlisten verwalten, verfügen aber über die komplette Funktionalität des Arbeitsbereiches. Alle Teilnehmer werden immer automatisch per RSS Feeds und E-Mail benachrichtigt, wenn Releases oder Änderungen stattgefunden haben.

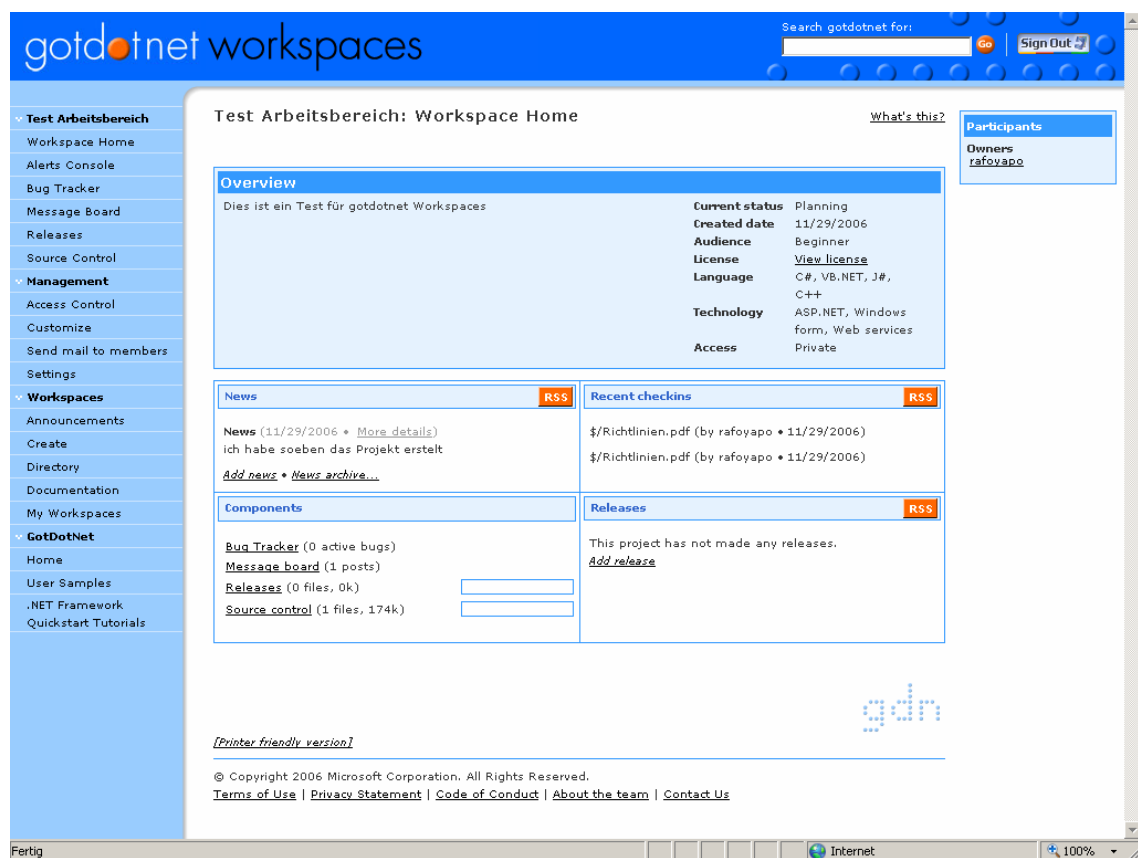


Abbildung 33. Beispiel eines Arbeitsbereiches in GotDotNet<sup>(37)</sup>

Die Arbeitsbereiche (siehe Abbildung 33) können öffentlich oder privat verwaltet werden. Ein öffentlicher Arbeitsbereich kann von jedem Besucher der Webseite mit Lesezugriff angesehen werden, wohingegen ein privater Arbeitsbereich nur von berechtigten Teilnehmern angesehen werden kann.

<sup>37</sup> Quelle: <http://www.gotdotnet.com> (Abruf am 27.03.2007)

GotDotNet ist eine schlanke Lösung für kleine Softwareentwicklungsprojekte, die für die Windows-Umgebung gedacht sind. Diese Lösung unterstützt die minimalen Anforderungen einer CDE, erreicht aber schnell ihre Grenzen. Es gibt jedoch negative Eigenschaften von GotDotNet, die die Benutzung dieser CDE bei bestimmten Problemlösungen ausschließt. Gelöschte Arbeitsbereiche können zum Beispiel nicht wieder hergestellt werden. Die Sicherung der Dateien liegt in der Verantwortung des Projektbesitzers, da GotDotNet keinerlei extra Sicherung der Daten anbietet. GotDotNet ist nicht komplett Plattformunabhängig, von Fall zu Fall kann es notwendig sein proprietäre Windowslösungen anzuwenden, wie zum Beispiel das .NET Framework, den Internet Explorer oder Visual Studio.NET. Das Konfigurationsmanagement erlaubt eine durchschaubare aber sehr einfache Versionskontrolle mit E-Mail und RSS Feeds Benachrichtigungen, das parallele Auschecken der Dateien wird allerdings nicht unterstützt, es dürfen keine zwei Entwickler gleichzeitig an einer selben Datei arbeiten.

### **6.4.3 SourceForge.net und SourceForge® Enterprise Edition**

Das Werkzeug SourceForge.net ist eine online CDE und wurde von der Firma VA Software<sup>(38)</sup> entwickelt, die auch die CDE SourceForge® Enterprise Edition (SFEE) als gebührenpflichtige Lösung anbietet. Die Internetplattform SourceForge.net besitzt nach eigenen Angaben gegenwärtig mehr als eine Million Mitglieder und mehr als 100,000 registrierte Softwareentwicklungsprojekte und ist damit momentan vermutlich die meistbenutzte CDE, die weltweit existiert. Viele der dort verwalteten Softwareentwicklungsprojekte sind jedoch Ein-Mann-Projekte oder so klein, dass sie für eine geographisch verteilte Softwareentwicklung nicht von Bedeutung sind. Als Beispiel für ein Open Source Softwareentwicklungsprojekt zeigt Abbildung 34 das Open Source Projekt Gaim, das auch verteilt entwickelt wird.

Die Registrierung eines Softwareentwicklungsprojektes erfolgt durch den Projektmanager oder Besitzer des Projektes. Der Registrierungswunsch wird durch einen Mitarbeiter von SourceForge.net überprüft, was bis zu zwei

---

<sup>38</sup> URL: <http://www.vasoftware.com> (Abruf am 27.03.07)

Arbeitstage dauern kann. Im Softwareentwicklungsprojekt muss Englisch gesprochen werden, weitere Sprachen werden nicht unterstützt. Wenn es mehrere Softwareentwicklungsprojekte gleichzeitig zu verwalten gilt, können alle gemeinsam unter einem so genannten „umbrella project“ administriert werden.

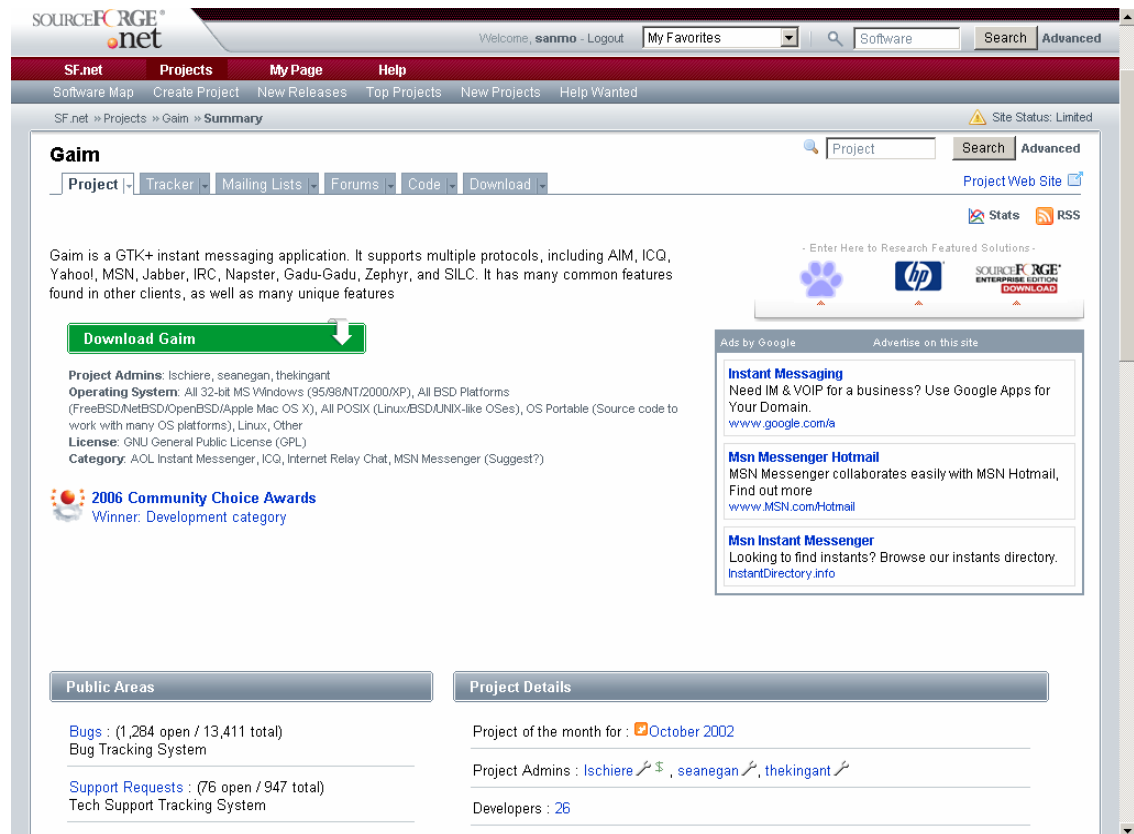


Abbildung 34. Die SourceForge.net Entwicklungsplattform<sup>(39)</sup>

SourceForge.net bietet drei Projekttypen an: Dokumentationsprojekte für Open Source Software, Softwareentwicklungsprojekte und Gruppen für den Support von Open Source Software. Der Quellcode und die Dokumente für das Softwareentwicklungsprojekt werden jeweils vom Besitzer hochgeladen. Um neue Teammitglieder zu finden und zu rekrutieren, wird entweder auf der „Project Help Wanted“ Seite eine Ausschreibung getätigt oder über das Diskussionsforum die Mitglieder direkt angesprochen. Beide Möglichkeiten helfen bei der Suche nach neuen Teammitgliedern für bestimmte Softwareentwicklungsprojekte. CVS und Subversion, die unterstützte Konfigurationsmanagementsysteme, sind in SourceForge.net integriert, was die

<sup>39</sup> Quelle: <http://www.vasoftware.com> (Abruf am 27.03.07)

Versionskontrolle flexibler macht, jedoch wie GotDotNet keinerlei extra Backup-Möglichkeiten bietet. Eine zusätzliche Projektsicherung muss deshalb durchgeführt werden.

SourceForge.net bietet einen Dokumentenmanager, Statistiken, Mailinglisten sowie Diskussionsforen, die Möglichkeit Screenshots anzuzeigen, ein Fehlerverwaltungsprogramm, Konfigurationsmanagement via CVS oder Subversion, Speicherplatz für die Freigabe neuer Versionen und die Möglichkeit einer Projektspende, um die Open Source Version zu unterstützen.

SourceForge® Enterprise Edition (SFEE) ist das Pendant zu SourceForge.net und baut auf den Erfolg der Open Source Variante auf. Dieses Werkzeug wird mittelgroßen und großen Unternehmen angeboten. Als eine zentral verwaltete Entwicklungsplattform basiert es auf J2EE und bietet gegenüber SourceForge.net mehr Performancemöglichkeiten. Oberstes Ziel ist die Vereinigung von heterogenen Werkzeugen und Prozessen, eine verbesserte Sicherheit und Projektdokumentation sowie die Kontrolle der einzelnen Entwicklungsprozesse und die Darstellung des Projektstatus (siehe Abbildung 35).

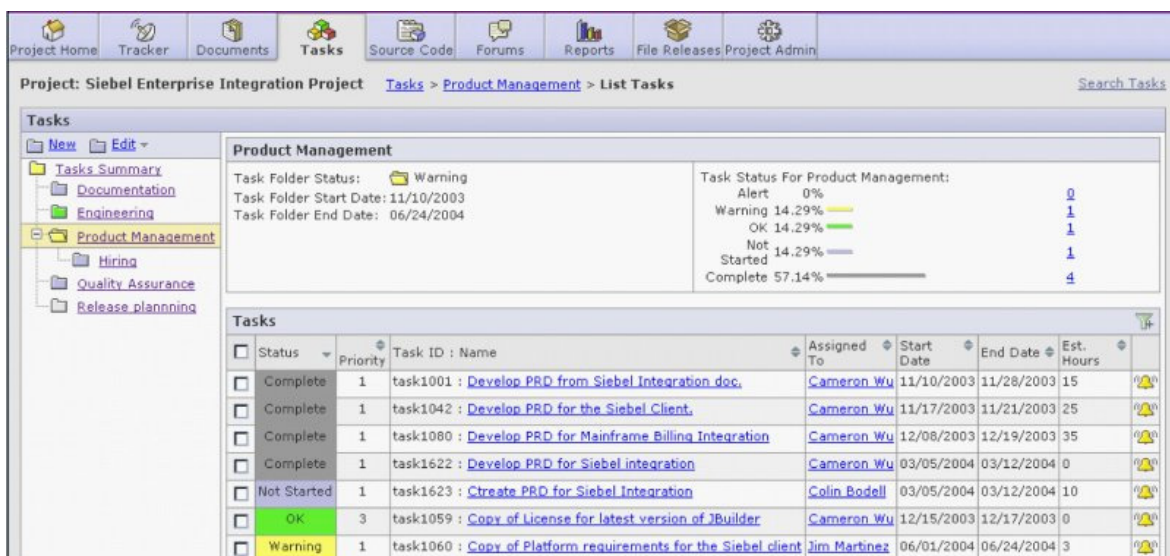


Abbildung 35. Ausschnitt eines Softwareentwicklungsprojektes in SFEE<sup>(40)</sup>

<sup>40</sup> Quelle: <http://www.vasoftware.com> (Abruf am 27.03.07)

Die Projektressourcen werden in einem zentralisierten Repository abgespeichert, der Zugriff hierfür erfolgt per Webbrowser. Dieser zentralisierte Ansatz verspricht eine schlanke Projektstruktur und bessere Zusammenarbeit.

Eigenschaften einer guten CDE, die SFEE unterstützt, sind:

- Die Quellcodeverwaltung, die durch das zentralisierte Repository sichergestellt wird. Ein Dokumentenmanager und ein Quellcodemanager ergänzen die Projektverwaltung.
- Die Rückverfolgung von Änderungen und Anforderungen wird durch das Fehlerverwaltungsprogramm (Tracker) und durch die automatische Überwachung und Benachrichtigung gewährleistet.
- Mit Hilfe des „Task Manager“ werden Ablaufpläne oder „to-do“ Listen organisiert. Das „File Release System“ garantiert den Zugang zu Dokumenten und Projektergebnissen.
- Diskussion und Kommunikation werden über Foren und Mailinglisten realisiert. Eine Wiki-Applikation steht in der aktuellen Version von SFEE zur Verfügung um dynamische Inhalte zu verwalten.
- Das „Global Development Dashboard“ garantiert die Aktualität der Aufgaben und des Softwareentwicklungsprojektes.
- Die Zugangskontrolle steht dem Projektmanager zur Verfügung um die Erlaubnis eines Zutritts der rollenbasierten Teilnehmer zu kontrollieren.

Für eine Installation des Werkzeugs wird ein Red Hat-basiertes Linux Betriebssystem empfohlen<sup>(41)</sup>. Die Clients benötigen nur einen Webbrowser, zum Beispiel den Microsoft® Internet Explorer oder den Mozilla Browser. VA Software bietet auch die komplette Administration der SourceForge Umgebung als eine Dienstleistung des internen Geschäftsmodells.

Zwischen den beiden CDEs, die von VA Software angeboten werden, existieren klare Zielgruppenunterschiede. Tabelle 9 zeigt einen Vergleich, die diese Unterschiede aufzeigt.

---

<sup>41</sup> Quelle: <http://www.vasoftware.com> (Abruf am 27.03.07)

	<b>SourceForge.net</b>	<b>SourceForge® Enterprise Edition</b>
<b>Idee</b>	Öffentliche Webseite für Open Source Entwicklung	Unternehmenslösung
<b>Ziel</b>	Maximale Offenheit, Transparenz und Geschwindigkeit	Sicherheit, Erweiterbarkeit, Benutzbarkeit, robustes Arbeiten hinter einer Firewall
<b>Architektur</b>	Skriptsprachen (PHP, Python, Perl) und Komponenten der Open Source Community	Plattformunabhängige J2EE-Architektur
<b>Sicherheit</b>	Offen und zugänglich für alle	128bit SSL Verschlüsselung
<b>Benutzer-Schnittstelle</b>	Einfache webbasierte Benutzerschnittstelle	Moderne, intuitive und konsistente Benutzerschnittstelle
<b>Vor- und Nachteile</b>	Kein modulares Design, keine Verschlüsselung für sicheren Zugang, kostenfrei	Integration mit anderen proprietären Werkzeugen, kostenpflichtig

Tabelle 9. Vergleich zwischen SourceForge.net und SFEE <sup>(42)</sup>

Für Open Source Softwareentwicklungsprojekte scheint SourceForge.net eine gute CDE-Alternative zu sein, für private Unternehmen hingegen kann eine Benutzung des SourceForge.net problematisch werden, da alle registrierten Softwareentwicklungsprojekte die Open Source Definition, die durch die Open Source Initiative (OSI)<sup>43</sup> spezifiziert wurde, erfüllen müssen:

*“The license shall not restrict any party from selling or giving away the software as a component of an aggregate software distribution containing programs from several different sources. The license shall not require a royalty or other fee for such sale.”*

Diese Definition für Open Source Software macht eine Nutzung dieses Werkzeugs für private Unternehmen schwierig. Für Unternehmen, die lizenzierte Software anbieten wollen, ist SFEE die besser geeignete Lösung. Die Sicherheit bei SourceForge.net liegt auf der Seite des Entwicklerteams, jeder ist für die Sicherheit und Sicherung seiner Arbeitsumgebung selbst zuständig.

<sup>42</sup> Quelle: <http://www.vasoftware.com> (Abruf am 27.03.07)

<sup>43</sup> URL: <http://www.opensource.org/> (Abruf am: 27.03.07)

#### 6.4.4 CollabNet Enterprise Edition

Die Firma CollabNet<sup>(44)</sup> ist das Pendant der Firma VA Software und verfügt auch über zwei Produkte: das kommerzielle Produkt CollabNet Enterprise Edition (CEE) und das Open Source Softwareentwicklungsprojekt Tigris.org<sup>(45)</sup>. Beide Produkte stehen in Konkurrenz mit den Produkten von VA Software und bieten ähnliche Funktionalitäten an.

CEE stellt eine webbasierte Umgebung für die Zusammenarbeit im Team bereit, welcher bis zu 15 Benutzer kostenfrei zur Verfügung steht, wobei der Softwaresupport immer kostenpflichtig ist. Dieses Werkzeug existiert bereits in vier verschiedenen Sprachen: Chinesisch, Japanisch, Koreanisch und Englisch. Das zeigt die Wichtigkeit des asiatischen Marktes für die Zielsetzung von CollabNet, die in der Unterstützung von Outsourcing- und Offshoringprojekte in Asien liegt.

CEE stützt sich auf vier in Abbildung 36 beschriebenen Eigenschaften einer CDE: Kollaboration, Konfigurationsmanagement, Projektmanagement und Applikationsmanagement. Die Kollaboration wird mit Hilfe von Mailinglisten und Diskussionsforen unterstützt. Mailinglisten unterstützen die Koordinierung der Arbeiten und die Projektteilnehmer bekommen dabei einen Überblick über die Entwicklung des Softwareentwicklungsprojektes. Diskussionsforen unterstützen den Austausch von Wissen und Fragen. Dieses Projektwissen wird im Server gespeichert, so dass jedes Teammitglied jederzeit Zugang auf die ausgetauschte Information hat. Die nötigen Dokumente und Dateien werden automatisch verwaltet und indiziert. Konzepte, Spezifikationen, Screenshots und Quellcodedokumente werden alle zentral gespeichert. Bei Änderungen werden automatische Benachrichtigungen an die einzelnen Teilnehmern verschickt.

Das Konfigurationsmanagement wird mit Hilfe des Versionierungssystems Subversion erleichtert. CollabNet ist der Hauptsponsor von Subversion und entwickelt dieses System seit 2000 weiter. Subversion ist der Nachfolger von

---

<sup>44</sup> URL: <http://www.collab.net> (Abruf am: 27.03.07)

<sup>45</sup> URL: <http://tigris.org> (Abruf am: 27.03.07)

CVS [Buds07], das auch in CEE unterstützt wird. CEE besitzt auch Funktionen, die Aufgaben und Anforderungen verwalten.

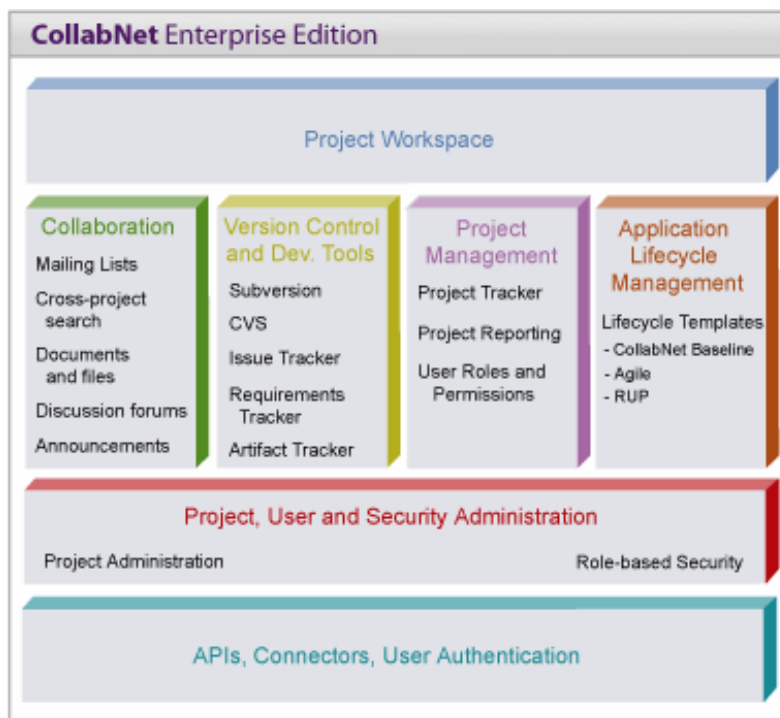


Abbildung 36. Architektur von CEE <sup>(46)</sup>

Der Projektmanager kann, wie in den anderen CDEs, Rollen und Agenteneigenschaften verwalten oder Teammitglieder für neue Projektaufgaben finden. Der aktuelle Status des Softwareentwicklungsprojektes wird über das Reportingtool abgefragt und eine graphische Oberfläche zeigt Berichte um die entsprechenden Maßnahmen vorzunehmen. Diese Eigenschaften unterstützen die Idee der Teamorientierung. Alle entwickelten Module werden mit Hilfe des „Project Tracker“ verwaltet. Somit ersparen sich die Teammitglieder eine mögliche doppelte Entwicklung desselben Moduls. Wenn Duplikate entstehen, kann ein Entwickler seine geleistete Arbeit vergleichen und entscheiden welche Version die bessere für die Problemlösung ist. Teammitglieder können Module suchen und für die entsprechende Aufgabe modifizieren. Benachrichtigungen per E-Mail mit Links zu den geänderten Dokumenten werden daraufhin automatisch erstellt.

<sup>46</sup> Quelle: <http://www.collabnet.net> 2006

Letztlich unterstützt CEE Entwicklungsmethoden mit sämtlichen Vorlagen für den Softwarelebenszyklus. Teammitglieder werden durch die einzelnen Lebenszyklusphasen begleitet und unterstützt. Es können Vorlagen definiert werden, die später für andere Softwareentwicklungsprojekte weiter benutzt werden. Aktuelle Standards wie RUP [Vers00] oder die Agilen Methoden (siehe Kapitel 4.4) werden damit ebenfalls unterstützt. Der große Vorteil dieser Eigenschaft ist, dass der aktuelle Status der Entwicklungsphasen, und dadurch das Gesamtprojekt, immer abgefragt werden kann. Der Zugang zum Werkzeug ist über die vorhandene Sicherheitsschicht gewährleistet, Authentifizierung der Benutzer und rollenbasierte Sicherheit durch Rechtevergabe helfen die nötige Sicherheit zu schaffen.

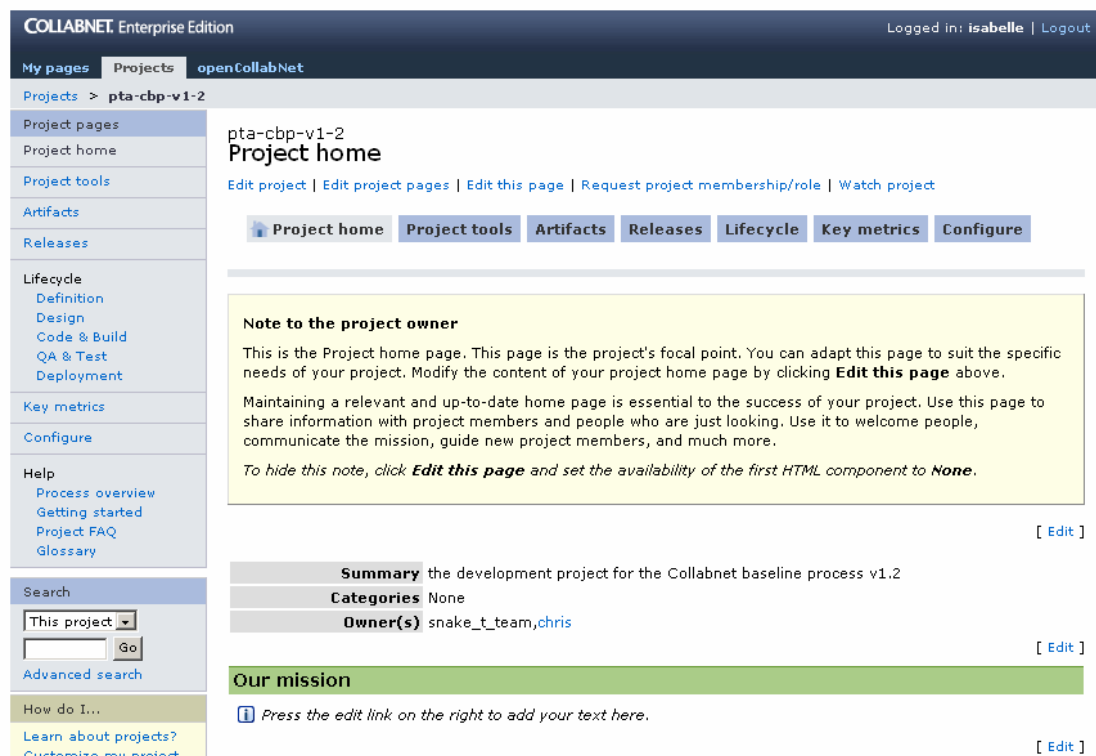


Abbildung 37. Projektarbeitsbereich im CollabNet Enterprise Edition (CNEE)<sup>(47)</sup>

CEE benötigt einen dedizierten Red Hat Enterprise Linux-Server für die Installation. Der Projektarbeitsbereich (Abbildung 37) ist der Mittelpunkt der CDE. Mehrere Softwareentwicklungsprojekte können, genauso wie mit SFEE, ebenfalls gleichzeitig verwaltet werden. Softwareentwicklungsprojekte sind privat oder öffentlich zugänglich und Mitglieder können selbst den Wunsch

<sup>47</sup> Quelle: <http://www.collabnet.net> 2006 (Abruf am: 27.03.07)

äußern, Teil eines Softwareentwicklungsprojektes zu sein oder sie werden vom Projektmanager eingeladen. Angemeldete Projektmitglieder haben Zugang zu den nötigen Dokumenten und integrierten Entwicklungswerkzeugen um in einem Softwareentwicklungsprojekt arbeiten zu können. Der Wiki-ähnliche Editor steht den Teammitgliedern zur Verfügung um Informationen zwischen Teams auszutauschen. Der Projektmanager kann das Design für den Projektarbeitsbereich erstellen, dennoch existieren Vorlagen, die die Designerstellung vereinfachen.

CollabNet bietet seine Softwarelösung als „on demand“ oder „on site“ Dienstleistung. Im ersten Fall wird die Software von CollabNet gehostet, gewartet und bereitgestellt. Im zweiten Fall wird sie auf der unternehmenseigenen Hardware hinter einer Firewall installiert. Je nach Projektart und Firmenprofil kann die eine oder die andere Art von Nutzen sein.

### 6.4.5 CodeBeamer

CodeBeamer ist eine CDE, die von Intland Software<sup>(48)</sup> weiterentwickelt wird. Diese J2EE-basierte CDE (siehe Abbildung 38) steht in direkter Konkurrenz mit SFEE. Für die Installation der CDE ist ein Windows- oder ein Linux-basierter Server notwendig.

Die Projektstartseite ist klarer, angenehmer und besser strukturiert als bei der Konkurrenz. Der Benutzer bekommt eine einfach zu verstehende Ansicht mit offenen Aufgaben und aktuellen Softwareentwicklungsprojekten, die mit seinen Berechtigungen zugänglich sind. Wenn das gewünschte Softwareentwicklungsprojekt ausgewählt wurde, öffnet sich die Übersichtsseite des Softwareentwicklungsprojektes (siehe Abbildung 39). Hier wird der aktuelle Status des Softwareentwicklungsprojektes angezeigt. Der Seiteninhalt kann vom Projektmanager dynamisch angepasst werden. Standardmäßig werden eine kurze Projektbeschreibung, die Anzahl der involvierten Teammitglieder, die letzten Neuigkeiten (Forum) bezüglich des Softwareentwicklungsprojektes und die Zusammenfassung der noch zu lösenden Aufgaben und Fehler angezeigt.

---

<sup>48</sup> Quelle: <http://www.intland.com> (Abruf am: 27.03.07)

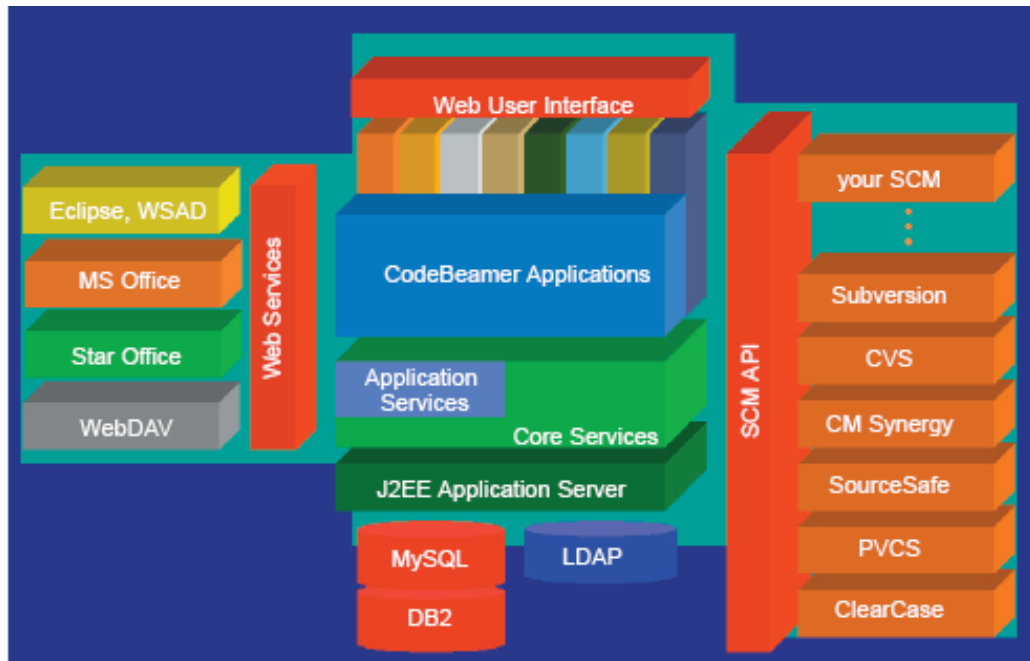


Abbildung 38. CodeBeamer Architektur<sup>(49)</sup>

Die Funktionalität des Änderungsmanagements unterstützt erfolgreich die Eigenschaften einer guten CDE. CodeBeamer unterstützt die Projektverfolgung, die Aufgabenverteilung, die Erstellung von „to-do“ Listen, die automatischen Nachrichtenfunktionen und die automatische E-Mail-Benachrichtigung. Der aktuelle Stand des Softwareentwicklungsprojektes kann durch diese Eigenschaften abgefragt werden und das Entwicklungsteam bleibt immer über den aktuellen Projektstatus informiert. Mit Hilfe von Grafiken wird zusätzlich eine klare Übersicht über das Softwareentwicklungsprojekt gewonnen.

CodeBeamer unterstützt das Konfigurationsmanagement CVS, Subversion und auch andere externen Werkzeuge wie SourceSafe oder PVCS. Der Zugang und die Kontrolle des Softwareentwicklungsprojektes erfolgen standardmäßig über den Internetbrowser mittels Subversionen. Die nötigen Dokumentationen, die freigegebenen Dateien und die dazugehörigen Start und Endtermine werden dokumentiert und automatisch verwaltet. Die Erlaubniskontrolle ist dem Projektmanager untergeordnet und die Qualität der Dateien kann somit gesichert werden. Entwickler, Kunden und beteiligte Personen bekommen eine passende Benutzeransicht mit vorher definierten Berechtigungen.

<sup>49</sup> Quelle: <http://www.codebeamer.com> 2006 (Abruf am: 27.03.07)

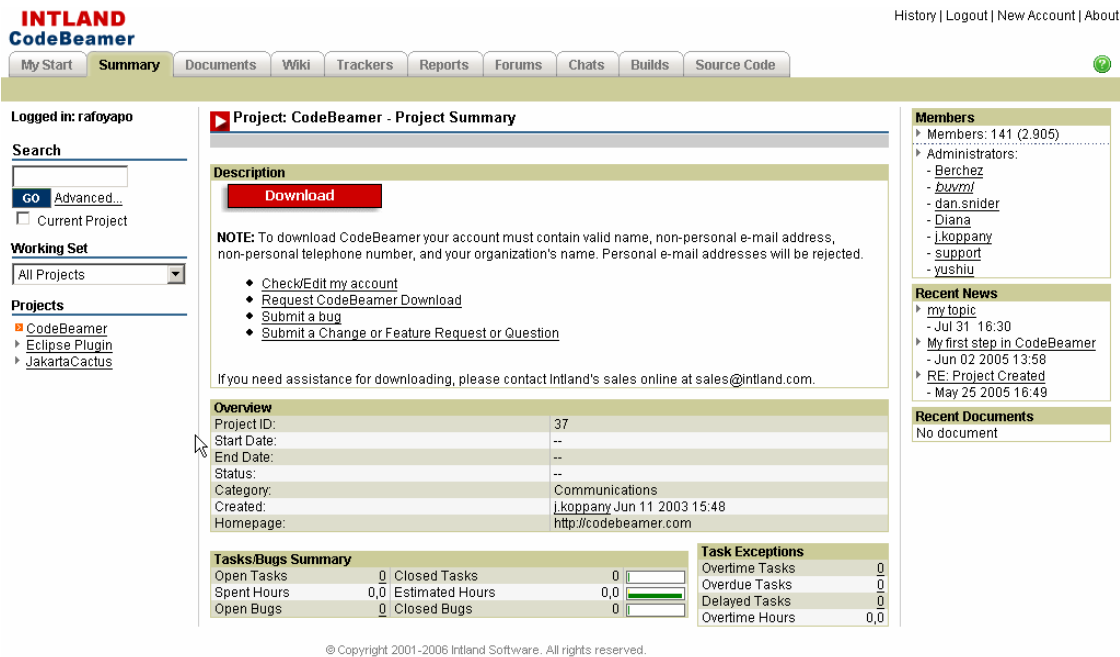


Abbildung 39. Übersicht CodeBeamer<sup>(50)</sup>

Die einfache Explorer-Ansicht der Dokumentenverwaltung schafft einen Raum, in dem Dateien verteilt, hochgeladen, heruntergeladen, verfolgt oder durchsucht werden können. Die Rechtevergabe erlaubt Lese- oder Schreibzugriffe. Das Verhalten der Teammitglieder wird verwaltet, indem das Lesen, Speichern oder Löschen nachverfolgt und aufgezeichnet wird. Das Diskussionsforum wird von CodeBeamer, wie bei den anderen CDEs, gut unterstützt. Der Wiki stellt auch einen weiteren Schritt Richtung Wissensverteilung und der Verwaltung von Informationen dar. Die Wiki-Funktionalität ist eine Anreicherung der Kommunikation im Team, somit können Informationen dynamischer als mit einem Forum ausgetauscht werden.

## 6.5 CDE-Vergleich

Die verteilte Softwareentwicklung, die von vielen kommerziellen Softwareunternehmen betrieben wird, hat sich dank der gesammelten Erfahrungen der Open Source Community weiter entwickeln können [Matl05]. Die Open Source Community hat vor einigen Jahren die ersten Schritte in

<sup>50</sup> Quelle: www.codebeamer.com (2006)

Richtung verteilter Softwareentwicklung erfolgreich genommen und im Laufe der Jahre bewiesen, dass Barrieren wie Geographie und Kultur überwunden werden können, um gute Software herstellen zu können. Nichts desto trotz ist die Natur eines proprietären Softwareentwicklungsprojektes anders als die von einem Open Source Softwareentwicklungsprojekt, denn die Faktoren Kosten und Zeit spielen in einem proprietären Softwareentwicklungsprojekt eine wichtigere Rolle. Die Nutzung einer passenden CDE-Lösung ist daher unabdingbar.

Bei einigen der untersuchten CDEs ist ein gutes Stück Weg zurückzulegen, wie beim Suchen, Finden und Speichern des Projektfachwissens deutlich wird. Innerhalb eines verteilten Softwareentwicklungsprojektes existieren einige natürliche Barrieren, die das Austauschen der Informationen zwischen Teilnehmern schwer machen ([CaAb06], [Matl05], [HuOc06], [NaHe05], [OHRG04], [BCKS01]).

In [BoBr02] und [Four01] werden Eigenschaften einer guten CDE aufgelistet. In jeder guten CDE sollen die Kollaboration, die Koordination und die Gemeinschaftsbildung stark unterstützt werden. Einige der zu unterstützenden Eigenschaften sollen sein: Die Verwaltung des Quellcodes und dessen automatische Speicherung in einem Repository, die Rückverfolgung von Wunsch- oder Anforderungsänderungen oder die Versionierung der Dateien, die für jedes verteilte Softwareentwicklungsprojekt eine wichtige Aufgabe bleibt.

Firmen wie Microsoft®, VA Software oder CollabNet haben Produkte auf dem Markt platziert, die die verteilte Entwicklung unterstützen. Aktuelle CDEs unterstützen, im Gegenteil zu herkömmliche IDEs, nicht nur das Individuum als zentrale Spielfigur, sondern sehen viel mehr das Team als Kern des Projekterfolges und ermöglichen die Standardisierung vieler Entwicklungsprozesse, sowie die Kommunikation im Team [BoBr02]. Jede dieser CDEs versucht den Eigenschaften einer guten CDE Herr zu werden und bietet gleichzeitig mit Konfigurations-, Änderungs- und Projektmanagementtools die nötigen unterstützenden Funktionen für alle Teammitglieder und für das Management. Dadurch können sich die Projektteilnehmer auf das Kernproblem ihrer Arbeit konzentrieren: Kodieren, Kommunizieren und Verwalten [Stru94].

Die aktuellen CDEs stellen einen Schritt in die richtige Richtung dar, einige benötigen zusätzliche Unterstützung der Kommunikation und die Verteilung von Wissen, um ein verteiltes Softwareentwicklungsprojekt reibungslos zu unterstützen. Tabelle 10 zeigt zusammenfassend einen Vergleich der hier aufgeführten CDEs.

Alle CDEs implementieren die Funktionalitäten einer guten IDE, dazu wird das Konfigurationsmanagement besser unterstützt und in das Gesamtkonzept integriert; die Versionskontrolle mit Hilfe von CVS oder Subversion ist standardmäßig bei allen CDEs vorkonfiguriert. Die Problem- und die Projektverfolgung so wie die Projektberichte werden von den kostenpflichtigen Lösungen besser abgedeckt als von den Open Source Softwareentwicklungsprojekten. Mit Funktionen wie der Volltextsuche, der automatischen E-Mail-Benachrichtigung bei Änderungen oder der Verfolgung bei Änderungen haben CEE, SEE und CodeBeamer einen Vorteil gegenüber den kostenlosen GotDotNet oder SourceForge.net. CEE kann gleichzeitig zwischen mehreren Softwareentwicklungsprojekten parallel nach Informationen suchen.

Im Bereich des Projektmanagements und der Administration sind in allen CDEs Funktionen wie die Rollenvergabe, die Aufgabenverteilung, die Dokumenten- und Dateiverwaltung vorhanden. Die Versionierung und der rollenbasierte Zugriff sind auch in allen anderen Lösungen zu finden. CodeBeamer ist in diesem Bereich, durch die E-Mail-Integration und eine E-Mail-Benachrichtigung etwas reifer [GHKR06]. CEE ist die einzige mehrsprachige CDE, was sicherlich ein Vorteil in den asiatischen Märkten ist. CodeBeamer schafft sich insbesondere bei der Benutzerunterstützung einen Vorteil gegenüber den Konkurrenten. Diese CDE bietet mehr Interoperabilität als die anderen Lösungen ([GHKR06], [Pets06]), unter anderem ein Plugin für die Entwicklungsumgebung Eclipse. Die Entwicklungsumgebung Eclipse stellt daher ein Werkzeug dar, in dem sich neue Technologien integrieren lassen, um das Team besser unterstützen zu können. Diese Flexibilität gibt dieser IDE einen Vorteil bezüglich anderen Werkzeugen [SiSC05].

Diese Werkzeuge unterstützen hauptsächlich die asynchrone Kommunikation, lassen aber die synchrone Kommunikation meist außer Acht. GotDotNet, zum

Beispiel, besitzt einen Bereich für Nachrichten (Message Board), in dem die Kontaktdaten der Teammitglieder und die versandten Nachrichten erfasst sind, besitzt aber keine Funktionalität, die die Wissensverwaltung und die synchrone Kommunikation unterstützt. Ähnliches geschieht auch bei der Benutzung von SourceForge.net. Obwohl hier die Suche nach Information besser unterstützt wird als bei GotDotNet, verfügt dieses Werkzeug noch immer nicht über ein ganzheitliches Konzept. Mailinglisten und Foren für die asynchrone Kommunikation sind zwar verfügbar, ebenso wie eine Suchfunktion, um nach Informationen in Foren, Mailinglisten, Dateien und Projektphasen zu recherchieren. Die gesuchte Information wird aber nicht untereinander gekoppelt, was die Nützlichkeit der Ergebnisse stark beeinträchtigt, da die gefilterten Daten nur nach Trefferquote dargestellt werden und nicht untereinander verglichen werden können.

CodeBeamer, CollabNet Enterprise Edition und SourceForge Enterprise Edition bieten als proprietäre Produkte mehr Funktionalitäten als die Open Source Varianten. Zu den von Open Source CDE unterstützten Funktionalitäten wird in der Regel eine integrierte E-Mail-Unterstützung zusätzlich angeboten. Hier können direkt E-Mails gesendet und empfangen, Verknüpfungen und Screenshots hergestellt und hinzugefügt werden, die CDEs bieten aber keine synchrone Kommunikation an.

Für eine gute synchrone Kommunikation muss heutzutage leider immer noch auf externe Werkzeuge zurückgegriffen werden. Die Fachwissensuche wird aber von allen proprietären Werkzeugen als Standardfunktionalität wenigstens in einer Grundform unterstützt. Eine Verallgemeinerung und Verbesserung der Möglichkeiten bei der Suche nach Wissensträger bleibt eine Herausforderung in zukünftigen Versionen der CDEs.

Die passende CDE muss für jedes Unternehmen in Abhängigkeit der Projektgröße und des Grads der Verteilung ausgewählt werden. In großen, proprietären Softwareentwicklungsprojekte werden SFEE, CEE oder CodeBeamer bevorzugt. In kleinen, einfachen Softwareentwicklungsprojekte sind GotDotNet, SourceForge.net oder Tigris.org eine gute Lösung.

Mailinglisten, Foren, RSS-Feeds und Wikis werden zunehmend von allen CDEs verbessert und ins gesamte Konzept besser integriert. Die Kommunikation und die Wissensverteilung bleiben aber ausbaufähige Bereiche dieser Werkzeuge. Keine CDE kann die Kommunikation in einem Entwicklerteam vollständig ersetzen, die Kommunikation der Projektteilnehmer untereinander ist unabdingbar. CDEs sind aber ein gutes Komplement um eine reibungslose Kommunikation zwischen den Standorten zu gewährleisten. Die synchrone Kommunikation zwischen Teammitgliedern wird noch nicht überall genügend unterstützt. Diese Thematik wird im nächsten Kapitel tiefer analysiert, in dem die Wissensverteilung und die Verbesserung der Kommunikation untersucht werden.

	Gotdotnet	Source Forge.net	SourceForge Enterprise Edition (SFEE)	CollabNet Enterprise Edition (CEE)	CodeBeamer
<b>Quellcodeverwaltung</b>	+	++	++	++	++
<b>Projektverfolgung</b>	++	++	++	++	++
<b>Konfigurations-Management</b>	+	++	++	++	++
<b>Projektplan-Management</b>	+	+	++	++	++
<b>Rollenmanagement</b>	++	++	++	++	++
<b>Workflow-Management</b>			++	++	++
<b>Mailinglisten, Diskussionsforum</b>	++	++	++	++	++
<b>RSS Feeds, Wiki, E-Mail</b>	++	+	+	+	++
<b>Mehrsprachunterstützung</b>				++	
<b>Kommentare</b>	<ul style="list-style-type: none"> <li>- Einfache Versionskontrolle</li> <li>- Kleiner Speicher-raum</li> <li>- Kein online Backup möglich</li> <li>- Plattform-abhängig</li> <li>- Schlanke Lösung</li> </ul>	<ul style="list-style-type: none"> <li>- Registrierung durch SourceForge.net</li> <li>- Kein online Backup möglich</li> <li>- Alle Projekte sind offen gelegt</li> <li>- Kostenlos</li> <li>- Benutzeranzahl unbegrenzt</li> </ul>	<ul style="list-style-type: none"> <li>- Unternehmenslösung</li> <li>- 128bit SSL Verschlüsselung</li> <li>- Zentrales Repository</li> <li>- Komplette Administration als Dienstleistung</li> <li>- Integration mit anderen Werkzeugen</li> </ul>	<ul style="list-style-type: none"> <li>- Bis zu 15 Benutzer kostenlos</li> <li>- Vorlagen für Softwarelebenszyklen</li> <li>- Software as a Service (SaaS)</li> <li>- Installation auf ein Linux-Server notwendig</li> </ul>	<ul style="list-style-type: none"> <li>- Windows- und Linuxunterstützung</li> <li>- Plattformübergreifendes Konfigurationsmanagement</li> <li>- Gute Interoperabilität mit externen Werkzeugen</li> </ul>

+: Unterstützt diese Eigenschaft ++: Sehr gute Unterstützung dieser Eigenschaft

**Tabelle 10. Vergleich der CDEs**

## 6.6 „Ideale“ CDE

Nachdem die auf dem Markt verfügbaren CDEs aufgezeigt wurden und in Anbetracht der von Booch und Brown definierten Reibungspunkte (Abschnitt 6.3), wird in diesem Abschnitt eine CDE definiert, die als Muster dienen soll, um diese Reibungspunkte zu überwinden. Dafür muss die CDE auf den Eigenschaften einer guten IDE aufbauen und zusätzlich neue Funktionalitäten aufweisen, die in verteilten Umgebungen vorhanden sind. In [BoBr02] und in [Four01] sind wichtige Eigenschaften für eine optimale CDE aufgelistet. Diese werden in drei Bereiche aufgeteilt, die in Abbildung 40 zu sehen sind:

1. Kollaboration,
2. Koordination und
3. Gemeinschaftsbildung.

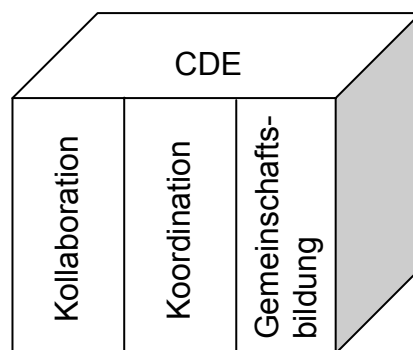
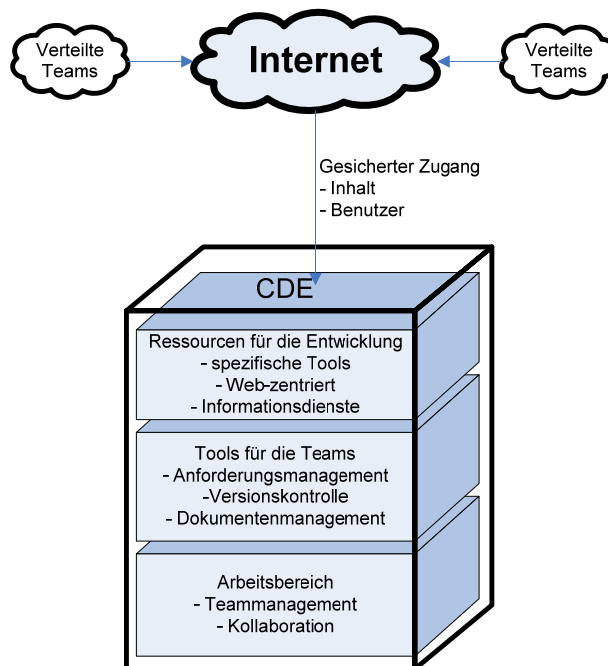


Abbildung 40. Bereiche einer CDE ([BoBr02], [Four01])

Abbildung 41 hingegen zeigt diese drei Bereiche der idealen CDE mit den dazugehörigen Eigenschaften. In dieser Abbildung ist unter anderem der Einfluss der verteilten Teams zu sehen. Diese können sich beispielsweise über das Internet auf der CDE anmelden und den Arbeitsbereich gemeinsam nutzen. Die Benutzung des Internets ist nicht zwingend erforderlich, wichtiger ist es, egal welcher Zugang bevorzugt wird, die Verbindungen sicher zu schaffen, damit die Geheimhaltung des Softwareentwicklungsprojektes sichergestellt ist. Es gibt auch andere Möglichkeiten für die Zusammenarbeit, doch das Internet wird hierfür als natürliches Medium für die Teamarbeit genutzt ([SeCS06], [LHKP03], [Scha06b], [Pets06]). Ziel einer guten CDE ist es die in Abbildung 41

beschriebenen Eigenschaften in einem Ganzen zu verbinden und gegebenenfalls das Internet als gemeinsamen Zugang anzubieten.



**Abbildung 41. Eigenschaften einer CDE [BoBr02]**

Im Bereich der Kollaboration werden die Mechanismen für die Teamunterstützung zur Verfügung gestellt. Die Koordination dient dazu, das Aufgabenmanagement zu unterstützen und zu automatisieren. Mithilfe der Werkzeuge, die den Teams zu Gute kommen, wird die verteilte Zusammenarbeit besser koordiniert. Das Internet dient dabei als unterstützender Faktor der Gemeinschaftsbildung. Zusammenfassend lässt sich sagen, dass eine CDE die Stärken einer IDE unterstützt aber auf die Gebiete der Kommunikation, des Managements und der Kontrolle besser eingeht, um die Reibungspunkte eines jeden Softwareentwicklungsprojektes zu minimieren [SeCS06].

Die wichtigsten Eigenschaften einer CDE sind Folgende [BoBr02]:

1. Quellcodeverwaltung
2. Rückverfolgung sämtlicher Änderungen
3. Konfigurationsmanagement
4. Anforderungsmanagement

5. Projektmanagement
6. Designmanagement

Eine CDE soll nicht nur das kollaborative Kodieren zwischen Teams unterstützen, sondern auch andere Managementaufgaben wie Anforderungs-, Projekt- oder Designmanagement. Zusätzlich zu diesen Eigenschaften sollte die „ideale“ CDE weborientiert sein. Wie aus der Einleitung hervorgegangen ist, stellt sich das Internet als die ideale Plattform für die Arbeit in verteilten Teams heraus, denn dort existieren keine physikalischen Grenzen, die das Softwareentwicklungsprojekt bremsen können. Leider ist das Internet aber nicht immer sicher vor unerwünschten Zugriffen, so soll die CDE zum Beispiel mehr Sicherheit gegen unerwünschte Angriffe innerhalb des Projektteams bieten. Jede Firma muss die zu ihr passende CDE benutzen, die ihre firmeninternen Sicherheitsaspekte erfüllt. Eine weitere wichtige Eigenschaft der CDE ist die Multidimensionalität, die auf die unterschiedliche Funktion der einzelnen Benutzer abgestimmt ist, das heißt: jeder Benutzer bekommt eine spezifische auf ihn angepasste Ansicht nach dem Motto „jedem das Seine“. Im Anschluss werden fünf aktuelle CDEs vorgestellt und analysiert, ob diese alle nötigen Eigenschaften einer guten CDE erfüllen.

## 7 Koordination der Testaktivitäten

Die Entwicklerteams versuchen im Laufe der Entwicklungsaktivitäten vorhandene Defekte<sup>(51)</sup> auf einen ertragbaren Zustand zu reduzieren, denn das Vorhandensein und Entfernen dieser Defekte verursachen extra Kosten für das Softwareentwicklungsprojekt [Kent95]. Je später der Fehler gefunden wird, umso teuer wird seine Beseitigung [Thal00].

In verteilten Softwareentwicklungsprojekten wird die Quellcodeentwicklung dezentral durch verteilte Teams durchgeführt. Ein Team entwickelt, ein anderes Team an einem anderen Ort führt die Testsuiten<sup>(52)</sup> aus und auf der Auftraggeberseite wird geprüft, ob alle entwickelten Module sich integrieren lassen (siehe auch Abschnitt 5.3). Aktuelle Testtechniken und -verfahren für die Qualitätssicherung von in-house Softwareentwicklungsprojekten werden in verteilten Softwareentwicklungsprojekten weiterhin unverändert angewandt ([BCKS01], [SeCS06]). Der dezentrale Ansatz der Softwareentwicklung weckt allerdings neue Fragen bezüglich dem Koordinierungsbedarf des Testens für verteilte Teams und den Auftraggeber. In [SeCS06] wird über drei Teilbereiche berichtet, die in verteilten Softwareentwicklungsprojekten mehr Aufmerksamkeit auf sich ziehen, als bei in-house Softwareentwicklungsprojekte:

1. die Datengeheimhaltung,
2. die Größe der Testdaten und
3. die Integration der entwickelten Module.

---

<sup>51</sup> Ein Softwaredefekt („fault“) wird von [IEEE04] und [Bind00] als fehlender oder falscher Programmtext definiert, ein Fehler („error“) dagegen als die menschliche Aktion, die ein falsches Resultat verursacht.

<sup>52</sup> Eine Testsuite ist eine Sammlung von Testfällen. Diese Testfälle haben normalerweise ein gemeinsames Testziel oder eine gemeinsame Abhängigkeit [Bind00]. Test Suites unterstützen das Management, um zu wissen was und wie es getestet wurde [Karo98].

## 7.1 Geheimhaltung und Datengröße

Wenn ein Softwareentwicklungsprojekt verteilt entwickelt wird, existiert nicht nur die Gefahr, dass die Kernkompetenzen des Auftraggebers verloren gehen (siehe Abschnitt 5.1.2), sondern auch die Gefahr, dass der Auftragnehmer auf sensible Daten des Softwareentwicklungsprojektes Zugriff bekommt. Diese Daten können für andere Zwecke benutzt werden, die nichts mit dem Projekt zu tun haben. Die Patientendaten eines Krankenhauses stellen so ein Beispiel für sensible Daten dar [WiBi98], denn die Daten beinhalten Informationen über Diagnosen, Rezepte, Rechnungen, Geburts- oder Mortalitätsrate.

Der Idealfall findet statt, wenn Vereinbarungen zwischen Auftraggeber und -nehmer im Voraus existieren und Erfahrungen aus früheren Softwareentwicklungsprojekten vorhanden sind. Das erspart Zeit bei der Testdurchführung und das Umgehen mit sensiblen Daten wird einfacher geregelt [Kent95]. Wenn aber das implizite Vertrauen zwischen Auftraggeber und -nehmer nicht vorgegeben ist, sind der Auftraggeber oder der Endkunde öfters nicht bereit sensible Daten des Produktivsystems frei zu geben [SeCS06]. Dieses fehlende, in verteilten Umgebungen verstärkte, Vertrauen verursacht einen hohen Zeitverlust beim Testen der Module. Verteilte Teams führen dann die Tests ohne echte Daten durch und das Verhalten kann unter Umständen nicht vollständige Ergebnisse liefern.

Die Herausforderung für die verteilte Softwareentwicklung ist es, Mock-Testdatenbanken erstellen zu lassen, die die Komplexität des Produktivsystems widerspiegeln können, ohne echte Daten zu gefährden und dabei nicht viel Overhead verursachen [SeCS06]. Die generierten und freigegebenen Daten dürfen keine vertraulichen Informationen offen legen, die der Besitzer nicht freiwillig zur Verfügung stellen will, um den Datenschutz aufzubewahren [WuWZ03].

In [WiBi98] wird ein Sicherheitsfiltersystem vorgestellt. Ein so genannter „Security Mediator“ wird zwischen die Produktivdatenbank und das zu testende System „verankert“. Abbildung 42 zeigt die Komponenten dieses Systems, das

Datenabfragen annimmt und verzerrte Ergebnisse zurückliefert, die die transformierte Information beinhalten. Somit bleibt die Geheimhaltung der Produktivdatenbank gesichert. Abfragen, die vom System nicht automatisch bearbeitet werden, werden vom Sicherheitsbeauftragten manuell abgearbeitet. Diese Person ist für das Regelsystem zuständig und definiert die Richtlinien, Regeln und Strategien des Systems, die ständig aktualisiert werden, so dass die Produktivdatenbank nur über dieses Regelsystem abrufbar ist

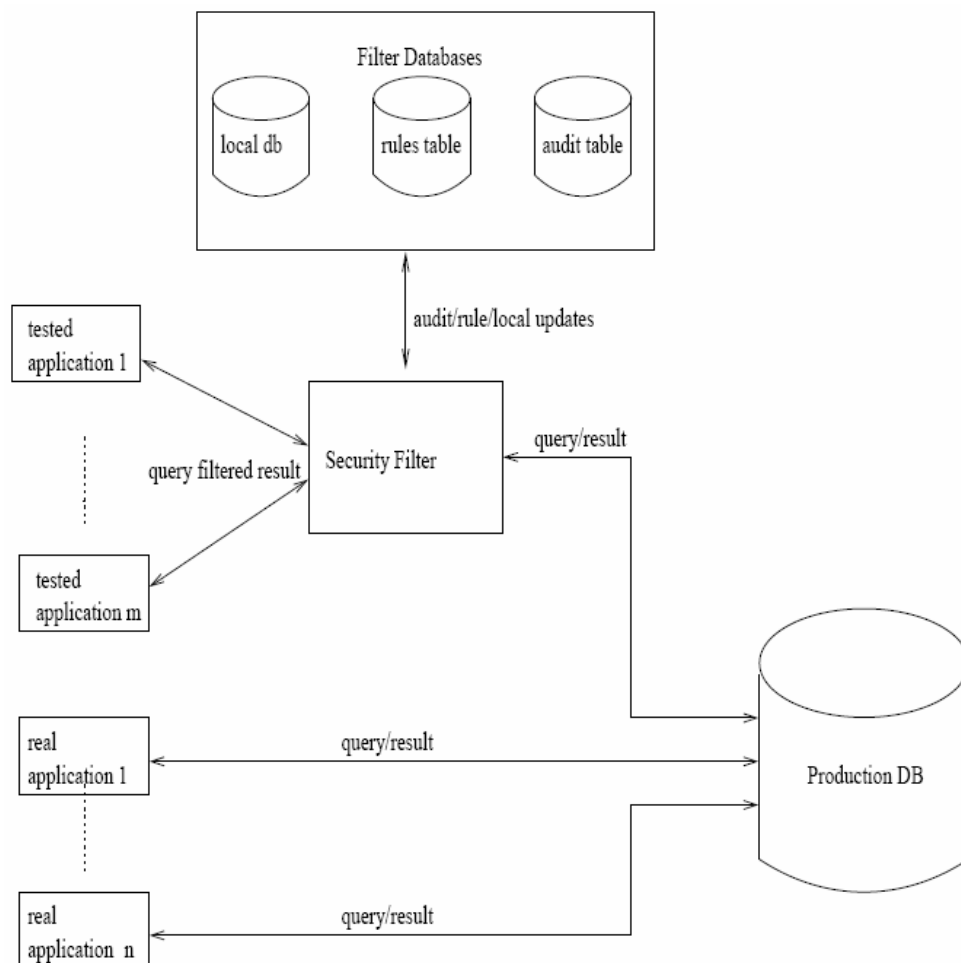


Abbildung 42. Komponenten eines Sicherheitsfiltersystems [WuWZ03]

Die Informationen, die aus einer produktiven Datenbank heraus gefiltert werden, sind abhängig vom aktuellen Status der Datenbank. Diese Abhängigkeit beeinflusst die Stabilität der Datenbank und ist der größte Nachteil dieser Technik. Einen anderen Ansatz wird in [WuWZ03] vorgestellt. Diese Technik extrahiert Regeln und statische Daten aus der Produktivdatenbank und produziert damit eine neue Datenbank mit neuen Daten, die künstlich hergestellt werden und abhängig der definierten Regeln sind. Diese generierten

Daten werden dann für Testzwecke in isolierten Umgebungen benutzt. Um die Daten zu generieren, extrahiert das System drei Informationen der Produktivdatenbank (siehe Abbildung 43):

- der deterministische Regelsatz  $R$  beinhaltet die deterministischen Regeln,
- der nicht deterministische Regelsatz  $NR$  verfügt über nicht deterministische Informationen und
- der statische Satz  $S$  gibt Auskunft über die Statistiken einer Datenbank-Instanz.

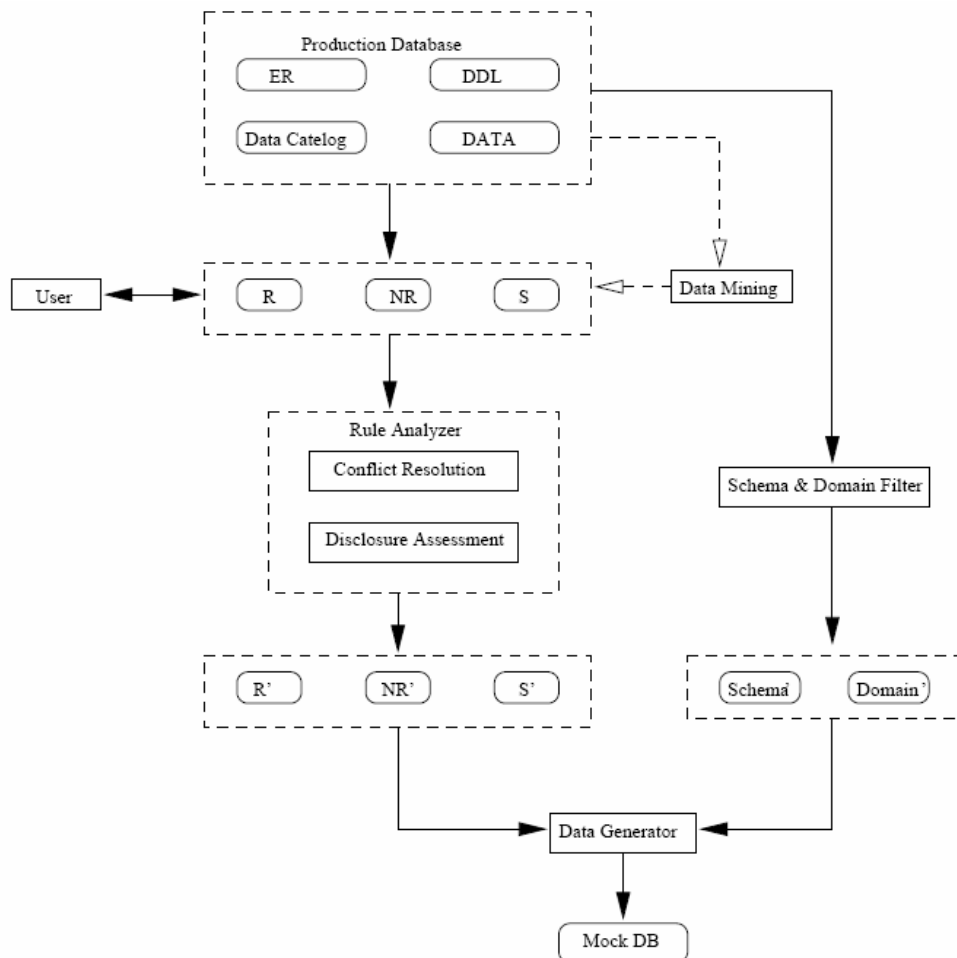


Abbildung 43. Komponenten des Ansatzes von [WuWZ03]

Mit dem extrahierten Tripel  $(R, NR, S)$  wird eine neue Datenbank erstellt, die keine sensiblen Daten beinhaltet und die vom Besitzer definierte Vertraulichkeit der Daten ermöglicht.

Wenn die Probleme der Datengeheimhaltung geklärt sind und wenn verteilte Teams auf die realen oder verformten Daten eines produktiven Systems zugreifen können, stellt sich die Frage nach der Größe dieser zu testenden Daten. Obwohl in den letzten Jahren das Vorhandensein von Bandbreite im Internet sich stark verbessert hat und die Länder, die für verteilte Softwareentwicklungsprojekte in Frage kommen wenige Probleme bei der Aufbau verlässlicher Internetverbindungen aufweisen<sup>(53)</sup>, können trotzdem die nötigen Daten und Datenbanken für die durchzuführenden Tests zu groß für den Transport über die Telefonleitung oder über das Internet sein [SeCS06].

Wenn verteilte Teams nicht über genügend Bandbreite verfügen, wird dieses Problem teilweise umgangen, indem verteilte Teams nur einen Teil der Daten aus dem produktiven System zu Testzwecken nehmen. Diese abgespeckte Datenmenge, auch „data slice“ genannt, kann verursachen, dass die ausgewählten Daten in Tabellen und Datenbanken nicht mehr wahrheitstreu sind, wenn diese Daten abhängig von anderen Daten sind. Wenn alle nötigen Daten nicht korrekt ausgewählt werden, werden wichtige Informationen außer Acht gelassen und die kleinen, abgespeckten Testdaten beeinträchtigen die Sicht realer Testbedingungen.

Abgespeckte Datenbanken oder Daten sollen drei Eigenschaften besitzen [WuWZ03]:

1. sie müssen gültig sein,
2. Ähnlichkeiten zu den realen Daten aufweisen und
3. die Datengeheimhaltung sichern.

Die letzte Eigenschaft wurde oben näher erläutert. Die Gültigkeit der Daten wird erreicht, wenn die abgespeckte Datenbank oder Datenmenge alle Bedingungen und Regeln des produktiven Systems erfüllt. Die zweite Eigenschaft besagt, dass die ausgewählten Daten zusätzlich ähnliche statistische Verteilungen wie die realen Daten besitzen müssen. Die Koordinierung der Testarbeiten zwischen Teams basiert nicht nur auf die ausgewählte Technik korrekte

---

<sup>53</sup> Die ITU (International Telecommunication Union) stellt sämtliche Informationen über dieses Thema zur Verfügung: <http://www.itu.int/osg/spu/ni/promotebroadband/> Abruf am: 19.04.2007

Testdaten zu erstellen, sondern auf das nötige Vertrauen, die in verteilten Softwareentwicklungsprojekten vorhanden sein muss. Eine der Techniken arbeitet mit Abfragen der Daten aus der Produktivdatenbank, eine andere Technik erstellt eine verformte Datenbank für Tests in isolierten Umgebungen. Obwohl der komplette Datenzugriff gewährleistet sein kann, stellt die Bandbreite auch eine Hürde bei der Datenvermittlung dar.

## 7.2 Integration

Viele der Probleme und Missverständnisse in verteilten Softwareentwicklungsprojekten erscheinen erst, wenn die entwickelten Module an einem bestimmten Standort - meistens beim Auftraggeber - zusammengeführt werden. Module werden davor in getrennten Standorten entwickelt und getestet. Der Bereich der Integration der verschiedenen Module und Komponenten wird in [HeGr99a] und [HeGr99b] als die schwerste Aufgabe der verteilten Softwareentwicklung beschrieben und wurde sehr früh als eine der ersten Herausforderung erkannt, mit der verteilten Softwareentwicklungsprojekte zu kämpfen hatten [SeCS06]. Dabei wird der Begriff der Integration definiert als die Arbeit, um das Produkt mit den Komponenten zu assemblieren<sup>(54)</sup> [HeGr99a].

In [Somm01] wird erklärt, wie die inkrementelle Systemintegration funktionieren sollte (siehe Abbildung 44). Seien  $T1$ ,  $T2$  und  $T3$  Testreihen, die auf den Modulen  $A$  und  $B$  einer minimalen Konfiguration ausgeführt werden. Die dabei entdeckten Fehler werden vor Ort korrigiert. Später in der Integrationsphase wird ein drittes Modul  $C$  integriert. Die Testreihen  $T1$ ,  $T2$  und  $T3$  werden wiederholt, um potentielle Fehlerquellen mit den Modulen  $A$  und  $B$  auszuschließen. Wenn Fehler auftreten, existiert eine große Wahrscheinlichkeit, dass dieses Verhalten auf das neue Modul  $C$  zurückzuführen ist. Somit ist die potentielle Problemquelle eingegrenzt. Wenn die Fehler korrigiert wurden, wird eine neue Testreihe  $T4$  ausgeführt, die weitere Funktionalitäten testet. Dieses vereinfachte Modell ist selten so zu finden und spätestens in einer verteilten

---

<sup>54</sup> „All the work that is necessary to assemble the product from its components“ [HeGr99a]

Umgebung kann eine Korrektur, bedingt meistens durch die physische Distanz, nicht mehr so einfach durchgeführt werden, vor allem wenn die informelle, implizite Kommunikation nicht so leicht aufzubauen ist [SeCS06].

Die doppeldeutige Dokumentation oder das Missverständnis der Schnittstellenanforderungen wird bei der Integration in verteilten Teams zu einem Problem:

*„Interface specifications lacked essential details such as message type, return types and assumptions about performance“ [HeGr99a]*

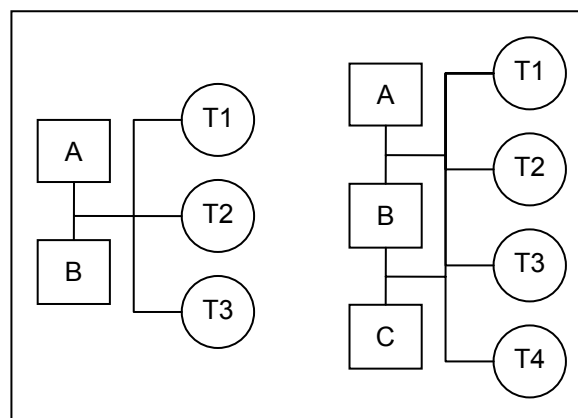


Abbildung 44. Inkrementelle Integration [Somm01]

Die Abhängigkeiten zwischen Modulen und Schnittstellen müssen von allen Entwickler und Teams, die diese Module entwickeln, verstanden werden [Grin98]. Das Entwicklerteam, welches die Integrationsarbeiten durchführt, muss über sämtliche Informationen verfügen, um die Module integrieren zu können. Zum Beispiel ist es wichtig zu wissen, ob und welche Entwurfsänderungen im Laufe der Zeit entstanden sind oder welche Abhängigkeiten zwischen Modulen und Schnittstellen existieren. Mit Hilfe des in Kapitel 4.5 vorgestellten Entscheidungsmodell und insbesondere die Dynamik des Projektverhaltens (Abschnitt 4.5.4 und 5.1.4) wird auf das Problem der Kommunikation bei der Komponentenintegration eingegangen. Zuerst wird die Koordinierung für Liefer- und Synchronisierungszeiten je nach Projektgröße und -komplexität abgestimmt, danach wird die Integration der abgelieferten Module in das Gesamtsystem durch das Integrationsteam erleichtert.

## 7.3 Inspektionen

Während der Integration der Module werden meistens Fehler entdeckt, die nur teuer zu beseitigen sind. Inspektionen und die dadurch verknüpfte Koordinierungsarbeiten der verschiedenen Entwicklerteams helfen Fehler gezielt während des Softwareentwicklungsprozesses aufzudecken und dabei zusätzliche Arbeit in späteren Phasen zu minimieren ([Stan06], [LaMa03]). Ein wichtiges Merkmal dieses Best Practice Ansatz [McCo01] ist es, dass der Mensch als menschlicher Tester gefragt ist und der Einsatz des Computers nach hinten rückt [Thal00].

Wenn die gelieferten Module integriert werden, wird getestet ob das Produkt richtig entwickelt wurde. Es wird in diesem Fall von Softwarevalidierung geredet. Mit Hilfe von Inspektionen des Softwareentwicklungsprozess wird geprüft, ob das richtige Produkt entwickelt wird. Hier ist von Verifikation die Rede [Raki97].

Eine typische Inspektion ist ein mehrstufiges Prozess und besteht aus sechs Phasen ([Whee66], [Faga76]):

1. Planungsphase
2. Einführung zur Inspektion (optional)
3. Vorbereitende Arbeiten
4. Gemeinsame Sitzung (Inspektion)
5. Nacharbeit
6. Überprüfung

In der Planungsphase sorgt der Moderator der Inspektionsteams für die Dokumentationen und Organisation der Termine. Dieser Moderator ist in der Regel der Leiter eines Entwicklerteams, welches für ein bestimmtes Softwaremodul zuständig ist. Eine Einführung zu der Inspektionsmethodik kann nützlich werden, wenn Entwickler nicht mit dieser Technik vertraut sind. Die Inspektoren, meistens Entwickler des Teams, sind zuständig Defekte in den Dokumenten zu finden. Bevor die gemeinsame Sitzung stattfindet, können sich

die Inspektoren mit den vom Moderator ausgeteilten Dokumenten vertraut machen und erste potentielle Fehler entdecken. In der gemeinsamen Sitzung treffen sich Inspektoren, der Moderator und der Autor der zu prüfenden Dokumente. In der Sitzung werden dann potentielle Fehler gesammelt und zusammen diskutiert. In [PSTV97] werden die während der Sitzung gefundenen potentiellen Defekte in drei klassifiziert:

1. „False Positives“ sind potentielle Defekte, für die keine Korrektur erforderlich ist, auch bekannt als falsche Positive.
2. „Soft Maintenance“ sind Defekte, die korrigiert werden, um die Lesbarkeit von Quellcode und die Programmierstandards zu unterstützen.
3. „True defects“ sind echte Defekte, die die Anforderung, der Softwareentwurf oder die Effizienz des Systems beeinträchtigen. Diese Defekte müssen korrigiert werden.

Die zu korrigierenden Defekte werden in der Nacharbeitsphase durchgeführt. Schließlich prüft der Moderator in der Überprüfungsphase, ob die entdeckten und diskutierten Defekte korrigiert worden sind. Prozess- und Produktdaten werden dabei gesammelt, um Vorschläge zur Qualitätsverbesserung des Softwareentwicklungsprozesses auszuarbeiten.

Der typische Inspektionsprozess hat sich im Laufe der Jahre weiter entwickelt. Dabei kann die historische Entwicklung der Inspektionswerkzeuge in vier Generationen unterteilt werden [Hedb04]:

1. Früh-Werkzeuge („Early tools“)
2. Verteilte Werkzeuge
3. Asynchrone Werkzeuge
4. Webbasierte Werkzeuge

Die erste Generation von Werkzeugen unterstützte nur Inspektionsteams, die geographisch am selben Ort waren. Der oben definierte Inspektionsprozess eignete sich für diese Werkzeuggeneration. Die zweite Generation ermöglichte die geographische Verteilung von Inspektionsteams, die sich noch in derselben Zeitzone befanden. Die asynchronen Werkzeuge der dritten Generation

befreiten sich von der Variable Zeit und Raum und erlaubten die Durchführung von Inspektionen unabhängig von Ort und Zeitzone. Die vierte Generation von Werkzeugen sieht das Internet als Bestandteil der Arbeit in verteilten Softwareentwicklungsprojekten. Diese Werkzeuge behalten die Eigenschaft der Asynchronizität, implementieren aber die Unterstützung der Web-Technologien. Beispiele für Werkzeuge der vierten Generation sind das freie System IBIS und die kostenpflichtige Lösung ReviewPro<sup>(55)</sup>.

### 7.3.1 IBIS

IBIS<sup>(56)</sup> (Internet Based Inspection System) gehört zu den Inspektionswerkzeugen der vierten Generation und überwindet die Probleme der Zeit und Raum mit einer webbasierten Lösung. Ziel dieses Werkzeuges ist es, die Kosten der Inspektionen zu senken oder wenigstens linear zu gestalten und dabei eine hohe Effektivität bei der Fehlerfindung zu behalten. Da der Inspektionsprozess sich schwer an die Bedingungen verteilter Softwareentwicklungsprojekte anpassen lässt, wurde bei der Entwicklung von IBIS dieser Prozess neu definiert. Den sechs Phasen einer traditionellen Inspektion fügt IBIS eine neue Phase hinzu [LaMa03].

Abbildung 45 zeigt die neue Unterteilung der Aktivitäten, dabei wird die tatsächliche Inspektion in zwei neue Phasen aufgeteilt, um die asynchrone Kommunikation und die Distanz zu unterstützen. Der Moderator einer Inspektion plant online der Inspektionsprozess und lädt per E-Mail Inspektionsteams ein, fügt Anhänge mit den nötigen Dokumentationen ein und schlägt Termine für die Inspektion vor.

Die optionale Einführungsphase findet mit Hilfe eines Peer-to-Peer-Konferenzwerkzeugs statt und wird synchron gehalten. Die Entdeckungsphase ersetzt die traditionelle Vorbereitungsphase, in dem nicht nur die Dokumente und Quellcode untersucht werden, sondern auch potentielle Defekte entdeckt und diese in das System für die späte Diskussion eingetragen werden. Der

---

<sup>55</sup> URL: <http://www.sdtcorp.com/reviewpro.html> (Abruf am: 03.04.2007)

<sup>56</sup> URL: <http://cdg.di.uniba.it/index.php?n=Research.IBIS> (Abruf am: 03.04.2007)

Moderator wird darüber informiert und kann die gelieferten Ergebnisse analysieren. Alle durch die Inspektoren gefundenen potentiellen Defekte werden in eine Liste gespeichert. Duplikate können vom Moderator mit Hilfe von Sortierfunktionen identifiziert werden und die Entscheidung treffen, welche Defekte, abhängig von der Anzahl der Duplikaten, für die weitere Diskussion auf der Liste bleiben und welche Inspektoren, abhängig von der Anzahl der Befunde, weiterhin teilnehmen dürfen.

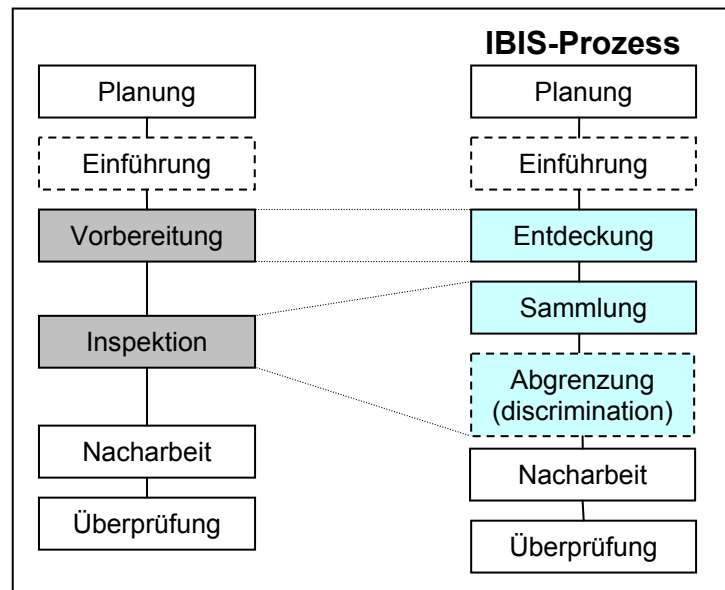


Abbildung 45. Der IBIS-Prozess im Vergleich mit traditionellen Inspektionen [LaMa03]

In der optionalen Abgrenzungsphase werden die ausgewählten Defekte im Forum zur Diskussion freigegeben. Jeder Defekt bekommt einen so genannten Thread zugewiesen. Andere Inspektoren können sich dadurch austauschen und über potentielle Defekte diskutieren. Diese asynchrone Diskussion dient dem Moderator zwischen echten Defekten und falschen Positiven eine Unterscheidung zu machen. Die Nacharbeit und Überprüfung der korrigierten Defekte werden dann wie bei einer traditionellen Inspektion durchgeführt. In der Nacharbeitsphase korrigiert der Autor der Dokumente die Fehler. Die Ergebnisse seiner Module werden in einem Formular gespeichert, in dem zu lesen ist, wie er die Probleme gelöst hat. Der Moderator überprüft die Ergebnisse und kann gegebenenfalls eine Wiederholung der korrektiven Tätigkeiten erfordern.

Webbasierte Inspektionswerkzeuge der vierten Generation haben die Möglichkeit der Benutzung von Web-Technologien für sich entdeckt und damit die Unterstützung geographisch verteilter Softwareentwicklungsprojekte erreicht. Das größte Problem dieser Werkzeuge bleibt aber, dass sie nicht in das Gesamtkonzept integriert worden sind. Das heißt, dass die Integration dieser Werkzeuge nicht in die Kollaborative Entwicklungsplattform (CDE) unterstützt werden [Stan06]. Zukünftige Lösungen sollen den Softwareinspektionsprozess als ein Ganzes betrachten und dabei die Flexibilität dieses Prozesses verbessern, in dem dieser in die CDE integriert wird [Hedb04]. Die zu korrigierende Dokumente sollen zentral in einem Repository für das Inspektionsteam zugänglich sein, sodass nur eine Version der Dokumente existiert und diese immer aktuell bleiben. Die heutzutage fehlende Unterstützung von Grafiken als Inspektionsdokumente soll in der nächsten Generation überwunden werden [Stan06].

### **7.4 Zusammenspiel der Koordinierungsaktivitäten**

Melvin Conway stellte 1968 fest, dass es einen Zusammenhang zwischen der Systementwurf und die Organisation, die diesen Entwurf entwickelt, existiert [Conw68]. Diese Feststellung wurde als „Conway’s Law“ bekannt und viele Jahre später von Herbsleb und Grinter wieder aufgenommen und an die Natur der verteilten Softwareentwicklung angepasst ([HeGr99a], [HeGr99b]). Die verschiedenen Standorte, die Zeitzonen, das vorhandene Wissen und die vorhandene Technologie sind Variablen, die die Koordinierungsarbeiten in verteilten Softwareentwicklungsprojekten komplexer machen.

Die Koordination der Testdaten ist eines der Probleme, die schwer zu überwinden sind [SeCS06]. Die Datengeheimhaltung, die Größe der Testdaten und die Integration der Module gehören zu den Problemen, die in verteilten Softwareentwicklungsprojekten komplexer werden. Die Testdaten oder Testdatenbanken müssen drei Eigenschaften besitzen:

- Datengeheimhaltung,
- Gültigkeit und
- Ähnlichkeit.

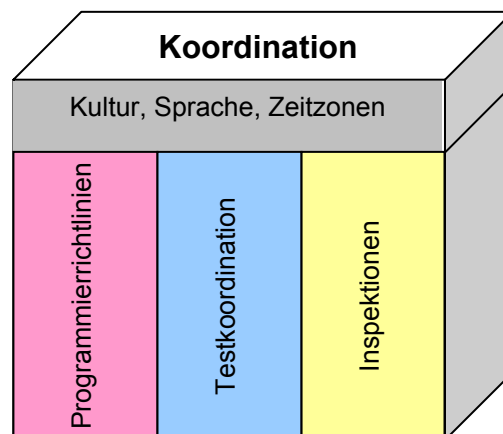
Um die Datengeheimhaltung zu unterstützen, wurden zwei Ansätze vorgestellt. Der erste Ansatz greift auf vorher filtrierte Daten der Produktivdaten zu [WiBi98]. Diese Daten sind vom aktuellen Status der Datenbank abhängig, was die Stabilität der Datenbank beeinflusst. Dies ist der größte Nachteil dieser Technik. Der zweite Ansatz extrahiert Regeln und statische Daten aus der Produktivdatenbank und produziert damit eine neue Datenbank für Tests in isolierten Umgebungen mit neuen Daten, die künstlich hergestellt werden und abhängig der definierten Regeln sind [WuWZ03]. Der Nachteil dieser Technik ist, dass die Datenbank jedes Mal neu hergestellt werden muss. In der vorgestellten Version des Systems findet keine inkrementelle Aktualisierung der Daten statt.

Die Gültigkeit der Daten wird gewährleistet, wenn die erstellte Datenbank oder Datenmenge alle Bedingungen und Regeln der produktiven Daten erfüllt. Die Ähnlichkeit der Daten erfolgt, wenn ausgewählte Daten ähnliche statistische Verteilungen wie die reellen Daten besitzen [WuWZ03]. Obwohl die verfügbare Bandbreite nicht mehr ein großes Problem darstellt, soll beachtet werden, dass alle Standorte gut vernetzt sind, sonst läuft das Projekt Gefahr, dass Testteams nur eine unvollständige Datenmenge für Testzwecke auswählen.

Die letzte Aufgabe der Testkoordination ist die Integration der Module. Das Entwicklerteam, das die Integrationsarbeiten durchführt, muss über sämtliche Informationen verfügen, um die Module integrieren zu können. Die Dynamik des Projektverhaltens (Abschnitt 4.5.4 und 5.1.4), was ein Teil des in Kapitel 4.5 vorgestellten Entscheidungsmodells ist, geht auf die Probleme der Kommunikation bei der Komponentenintegration ein.

Eine Inspektion verifiziert ob das richtige Produkt entwickelt wird und entdeckt gemachte Fehler während der Entwicklung ohne bis zum Integrationsprozess zu warten [Raki97]. Der Inspektionsprozess hat sich in den letzten Jahren entwickelt und sich dabei von der Variablen Zeit und Raum unabhängig

gemacht. Die erste Generation von Inspektionswerkzeugen konnte Teams unterstützen, die geographisch nur am selben Standort waren. Diese frühen Werkzeuge wurden durch die zweite Generation ersetzt, die die geographische Verteilung von Inspektionsteams ermöglichten, die sich aber immer noch in derselben Zeitzone befinden mussten. Die asynchronen Werkzeuge der dritten Generation erlaubten die Durchführung von Inspektionen unabhängig von Ort und Zeitzone. Die vierte Generation von Werkzeugen sehen die Web-Technologien, insbesondere das Internet als Bestandteil der Arbeit in verteilten Softwareentwicklungsprojekten [Hedb04]. IBIS wurde als Werkzeug der vierten Generation vorgestellt. Dieses kostenlose Werkzeug ermöglicht Inspektionen in verteilten Teams mittels einer Internet-Applikation. Dafür wurde der traditionelle Inspektionsprozess neu definiert (siehe Abbildung 45). Werkzeuge der nächsten Generation müssen, zusätzlich zu der neuen Web-Technologien den Softwareinspektionsprozess als ein Ganzes betrachten und dabei die Flexibilität dieses Prozesses verbessern, in dem dieser an die CDE integriert wird [Hedb04].



**Abbildung 46. Zusammenspiel der Koordinierungsaktivitäten**

Abbildung 46 zeigt das Zusammenspiel einiger Koordinierungsaktivitäten in graphischer Form dar. Verschiedene Kulturen, Sprachen und Zeitzonen beeinflussen die Koordinierungsarbeiten in verteilten Softwareentwicklungsprojekten. Entwickler und Management müssen sich an diese Variablen anpassen, um die Effektivität und Produktivität des Projektes sicherzustellen. Die Zusammenarbeit zwischen Entwickler, die Durchführung der Tests und die Suche nach Fehlern während der Entwicklung sind drei Aspekte, bei denen in verteilten Softwareentwicklungsprojekten Anpassungen vorgenommen werden müssen.

## 8 Vorgehensentwurf bei verteilten Projekten

In diesem Kapitel wird das Zusammenspiel der in den letzten Kapiteln aufgeführten Wege zur Qualitätssicherung mit Hilfe von zwei fiktiven Fallbeispielen vorgestellt, die repräsentativ für verteilte Softwareentwicklungsprojekte allgemein stehen. Diese Projektbeispiele sind aus eigener Erfahrung und aus dem Ergebnis einer Expertenbefragung entstanden. Im ersten Beispiel geht es um die Neuentwicklung eines Warenwirtschaftssystem für den Mittelstand. Das zweite Beispiel behandelt die Portierung einer vorhandenen Software für das Krankenhauswesen von einer Programmiersprache in eine Andere. Um auf die verteilten Risiken der Softwareentwicklung besser eingehen zu können, werden für jedes Beispiel Entscheidungen und Vorgehensmodelle simuliert, die in den vorherigen Kapiteln aufgezeigt wurden.

### 8.1 Die Projektbeispiele

Tabelle 11 zeigt die jeweiligen Eigenschaften der beiden Softwareentwicklungsprojekte auf. Das Ziel des Softwarehauses „AG1“ im ersten Beispiel ist die Entwicklung eines neuen Warenwirtschaftssystem „WaWiSys“ für einen Kunden des produzierenden Gewerbes. Die geschätzte Größe der Applikation beträgt ca. 20.000 Quellcodezeilen (LOC). Für dieses Softwareentwicklungsprojekt sind vor Ort 3 Entwickler eingeplant, die im ständigen Kontakt mit dem Kunden stehen. Bedingt durch enge Abgabetermine und fehlende menschliche Ressourcen hat AG1 entschieden einen Teil der Applikation verteilt zu entwickeln.

Im zweiten Beispiel geht es um die Portierung einer Software auf eine neue Programmiersprache. „AG2“ verfügt über jahrelange Erfahrung in der Softwareentwicklung für das Krankenhauswesen. Die Software „CoSo“ ist eine Controlling-Lösung, die sowohl medizinische als auch betriebswirtschaftliche Anforderungen berücksichtigt und wurde in der Programmiersprache Visual

Basic geschrieben. AG2 hat sich entschieden die Software von Visual Basic auf Visual Basic .NET zu portieren, um sich dadurch Marktvorteile zu sichern und weiterhin konkurrenzfähig zu bleiben.

	<b>Warenwirtschaftssystem (WaWiSys)</b>	<b>Controlling-Lösung (CoSo)</b>
<b>Kennung</b>	AG1 - DL1	AG2 - DL2
<b>Größe des Entwicklerteam</b>	5	20
<b>Größe der Applikation</b>	ca. 20000 LOC	ca. 100000
<b>Kunde</b>	Produzierendes Gewerbe	Krankenhauswesen
<b>Dauer</b>	Max. 6 Monate	Ca. 12 Monate
<b>Risiken</b>	<ul style="list-style-type: none"> <li>▪ Finanzielle Verluste durch verspätete Ablieferung der Software</li> <li>▪ Imageverlust für zukünftige Projekte</li> </ul>	<ul style="list-style-type: none"> <li>▪ Sicherheit und Vertraulichkeit der Patientendaten gefährdet</li> <li>▪ Fehlende Erfahrung mit .NET Technologien</li> <li>▪ Verlust des Marktanteils</li> </ul>
<b>Anforderungen</b>	<ul style="list-style-type: none"> <li>▪ Kreative Teams</li> <li>▪ Nicht alle Anforderungen im Voraus spezifiziert, einige aber stabil und bekannt</li> </ul>	<ul style="list-style-type: none"> <li>▪ Taktische Teams</li> <li>▪ Stabile und bekannte Anforderungen</li> </ul>
<b>Ziele</b>	<ul style="list-style-type: none"> <li>▪ Schnelle Ablieferung der Software</li> <li>▪ Anpassbarkeit</li> <li>▪ Skalierbarkeit</li> </ul>	<ul style="list-style-type: none"> <li>▪ benutzerfreundliche Anpassung der Software</li> <li>▪ Sicherheit der Daten</li> </ul>

Tabelle 11. Zusammenfassung der Applikationseigenschaften

## 8.2 Arbeitsweise der Teams

Das kleine Unternehmen AG1 hat fünf Jahre lang Erfahrung in der Softwareentwicklung und Betreuung anderer kleinen und mittelgroßen Unternehmen (KMU) gesammelt. AG1 beschäftigt 3 Entwickler und 2 Personen in Managementpositionen. Die Arbeitsweise der Entwickler ähnelt den Agilen Praktiken (siehe Abschnitt 4.4) und zwei der Entwickler haben erste Erfahrungen in der Paarprogrammierung gesammelt. Der Kunde wird in alle Softwareentwicklungsprojekte mit einbezogen, um die Anforderungen an die Software spezifizieren zu können. Das neue Softwareentwicklungsprojekt WaWiSys und die Auslagerungsentscheidung der Geschäftsführung stellen

dabei eine durchaus interessante Herausforderung dar, sowohl für das Management als auch für die Entwickler, obwohl die Abgabetermine und die menschlichen Ressourcen den Spielraum des Softwareentwicklungsprojektes stark eingrenzen.

AG2 ist ein mittelgroßes Unternehmen, das sich besonders in der Herstellung von Software für das Krankenhauswesen spezialisiert hat. Die Software CoSo wurde vor ca. 10 Jahren in ihrer ersten Version zum ersten Mal verkauft. Heutzutage ist die 9. Version die Aktuellste und beinhaltet mehr als 100.000 LOC. Die Software ist für alle Krankenhausgrößen entwickelt worden. Das Unternehmen verfügt über mehr als 30 Kunden, hat 20 feste Entwickler und ist CMMI (Ebene 2) zertifiziert (siehe Abschnitt 2.2). AG2 will seine Position auf dem Markt behaupten, in dem die Software auf die .NET Technologie portiert wird, ohne dabei die Weiterentwicklung der alten Version zu vernachlässigen.

### **8.3 Analyse der angepassten Faktoren**

AG1 entscheidet, dass ein Teil der Entwicklung von einem verteilten Team durchgeführt wird. WaWiSys verfügt über mehrere Module, die vom Kunden grob definiert wurden. Das System verwaltet über 100 Stammkundendatensätze, über 10000 Stammartikel und die Erfassung von Warenein- und -ausgängen. Zusätzlich soll die Lösung ein Reporting-Tool besitzen, um offene Positionen oder Lagerbestände abzufragen und zu kontrollieren. Die Endversion der Software soll innerhalb von sechs Monate beim Kunden zum Einsatz kommen. AG1 entscheidet sich dazu selbst die Module für die Kundendatensätze und Stammartikel zu entwickeln. Ein Dienstleister „DL1“ soll die anderen Module entwickeln und liefern. Bedingt durch den Zeitdruck und großen Kommunikationsbedarf vor allem für die Schnittstellendefinition wird ein verteilter Dienstleister ausgesucht, dessen Standort nicht mehr als vier Zeitzonen „entfernt“ sein darf. Es muss nämlich möglich sein, dass täglich zwischen den verteilten Teams synchron kommuniziert werden kann. Die endgültige Integration und die Tests der Module erfolgt beim AG1 vor Ort.

Da die Portierung der Software nicht zu den Kernaufgaben von AG2 gehört, lässt sie diese komplett von einem anderen Dienstleister „DL2“ durchführen. AG2 übergibt die komplette Information der Software an DL2 und übernimmt die Tests und die Kontrolle der Ergebnisse in zeitlichen Abständen. Die Software soll innerhalb eines Jahres portiert werden und im Einsatz sein. Der Standort und die zeitlichen Unterschiede von DL2 spielen bei diesem Softwareentwicklungsprojekt keine große Rolle. Der nächste Schritt in der Projektplanung für AG1 und AG2 erfordert die Analyse der Risiken und der kritischen Faktoren des angepassten Entscheidungsmodells von Boehm und Turner (siehe Kapitel 5).

Abbildung 47 stellt die von AG1 und AG2 durchgeführten Analysen dar. Das rote Fünfeck steht für die Analyse des CoSo-Projektes, das blaue Fünfeck dagegen für die Analyse des WaWiSys-Projektes. Während CoSo plangetriebene Eigenschaften besitzt, wie in der Abbildung 47 deutlich wird, ähnelt WaWiSys den Eigenschaften eines Agilen Projektes. Das Verhalten von CoSo und WaWiSys lässt sich an der jeweiligen Größe der Fläche der Analysen festmachen (BoTu03). Im folgenden Abschnitt werden die einzelnen Achsen des Entscheidungsmodells näher analysiert.

AG1 mit dem Softwareentwicklungsprojekt WaWiSys besitzt eine relativ kleine Teilnehmeranzahl. Während die eigenen Entwickler von AG1 sich mit den Modulen für die Kundendatensätzen und Stammartikeln beschäftigen, entwickeln die Entwickler auf der DL1-Seite die restlichen zwei Module. AG2 verfügt dagegen über eine größere Anzahl menschlicher Ressourcen als AG1, was sich auf der Achse „Größe“ widerspiegelt. Während die lokalen Entwickler bereits an neuen Funktionalitäten der Software CoSo arbeiten können, werden DL2-Entwickler die Software auf die neue Technologie portieren.

Die Achse „Sicherheit“ zeigt, dass sowohl für AG1 als auch für AG2 die Gefahr finanzieller Verluste existiert. Für AG1 ist die verspätete Lieferung der Software mit einer finanziellen Strafe verbunden, potenzielle Softwareentwicklungsprojekte könnten zudem zukünftig schwerer an Land gezogen werden. Hier wird deutlich, dass die Koordinierung und Synchronisierung der verteilten Standorte

zu einem großen Risiko werden kann. AG2 riskiert mit einem verspäteten Markteintritt der neuen Lösung, genauso wie AG1, neue potentielle Kunde zu gewinnen. Hinzu kommt auch die fehlende Erfahrung der lokalen Entwickler in der .NET Technologie.

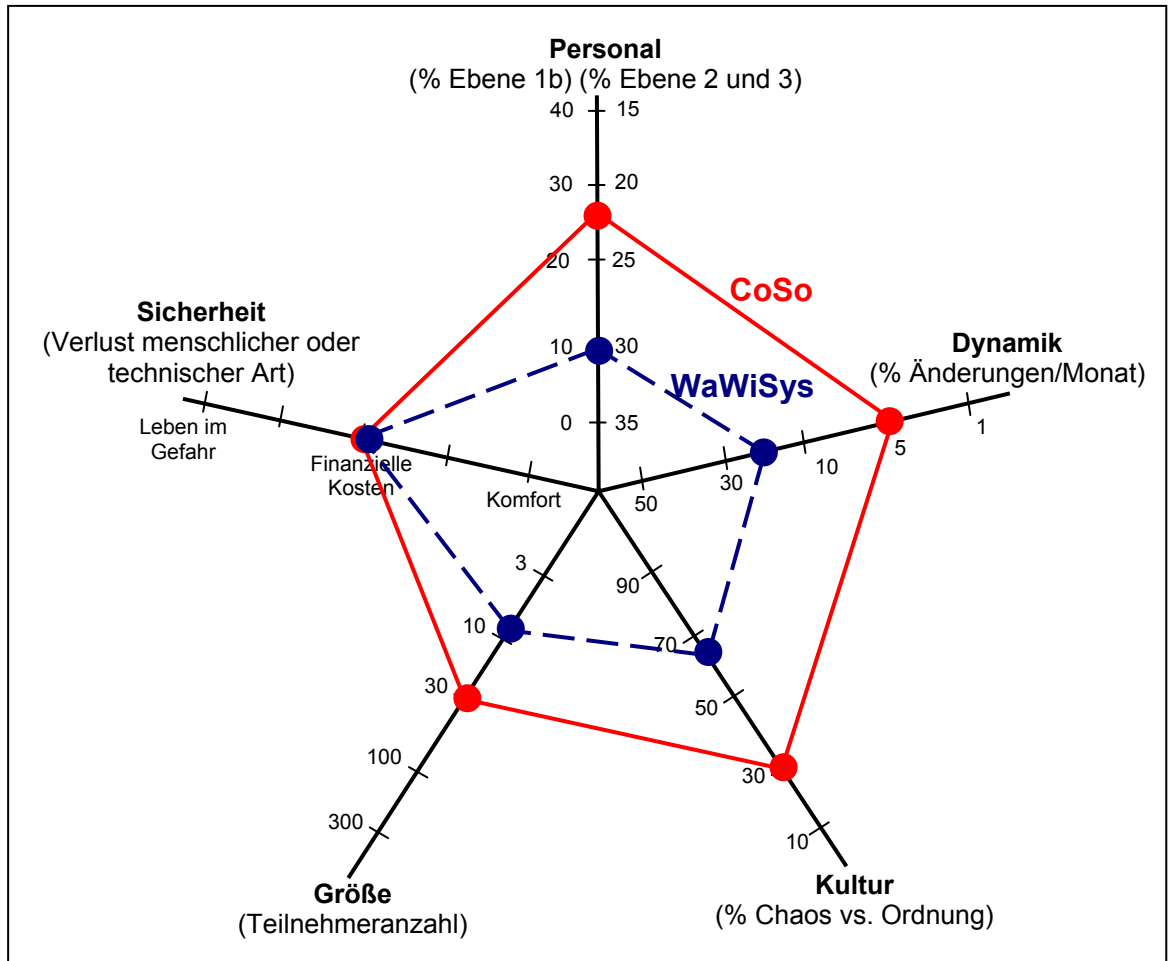


Abbildung 47. Analyse der kritischen Punkte von AG1 und AG2

WaWiSys benötigt kreative Teams (siehe Kapitel 5.2), die in der Lage sind nach Möglichkeiten und Alternativen für die Entwicklung der Module zu suchen. Die Teammitglieder benötigen ständige Rückmeldungen über die Fortschritte anderer Mitglieder. Diese eher chaotische Art der Zusammenarbeit und Kommunikation (siehe Abschnitt 4.4) steht im Gegensatz zu den Anforderungen der Softwareentwicklungskultur für die Software CoSo, was auf der Achse „Kultur“ abzulesen ist. Hier werden taktische Teams vorgezogen, die in der so genannten Ordnung arbeiten. Klar definierte Aufgaben, Rollen und Pläne werden benötigt, um die Ziele zeitnah zu erreichen. Da AG1 CMMI zertifiziert

ist, passt sich diese Aufgabe gut an die institutionalisierten Prozesse an, dennoch existiert die Gefahr unnötiger Bürokratie.

Die Dynamiken beider Projekte (Achse „Dynamik“) unterscheiden sich grundsätzlich voneinander. Während die Firma AG1 über ein klares Produktdesign, Datenbankmodell und einen genauen Arbeitsplan verfügt, werden bei der Entwicklung von WaWiSys viele Anforderungen ad-hoc festgestellt, da der Kunde zu Beginn des Softwareentwicklungsprojektes die Anforderungen nur grob definiert hat.

Bezüglich der Eigenschaften des Personals lässt sich sagen, dass CoSo über durchschnittliche Entwickler der Ebene 1B (siehe Abschnitt 5.1.5) verfügt. Die Entwicklungsaufgaben sind klar definiert und die Umgebung strukturiert. Dagegen benötigt WaWiSys ein größeres Spektrum an Fähigkeiten des Personals, deswegen der große Unterschied auf der Achse „Personal“. Entwickler der Ebene 1A, die gut in Agilen Projekten arbeiten können, werden bevorzugt. Zusätzlich werden aber genügend andere Entwickler der Ebene 2 benötigt, die in der Lage sind sowohl Agile als auch Planungsgetriebene Projekte zu leiten. Die Projektleiter können diese Funktion übernehmen.

## **8.4 Vorgehensmodell**

Nach der Analyse der angepassten Faktoren sind verschiedene Änderungen nötig, um die Prozesse der Projekte an die verteilte Softwareentwicklung anzupassen und so die Qualität der Ergebnisse zu sichern.

### **8.4.1 Prozessrahmenwerk**

Der ausgewählte Dienstleister DL2 für das Softwareentwicklungsprojekt CoSo muss eine CMMI-Zertifizierung („appraisal“) aufweisen. Da AG2 schon CMMI zertifiziert ist, und DL2 auch eine Zertifizierung aufweisen kann, erfordert der nächste Schritt einige der hier vorgestellten angepassten KPAs (siehe Abschnitt

2.2.1) zusätzlich zu institutionalisieren. Für AG2 kommen fünf KPAs in Frage (siehe Abbildung 11 und [RaKK05]):

1. "Individual product, functional ownership, and responsibilities identified and allocated to stakeholders"
2. "Mechanisms for division of labour devised; guidelines for distribution of tasks established"
3. "Policies for common-knowledge transfer framework established and practiced"
4. "Consistency in use of tools and processes for project management tracking and reporting"
5. "Product development, review, test tools, and processes established and practiced"

Diese KPAs unterstützen gemeinsame Projektinteressen, um das vorhandene Wissen über verteilte Standorte weiterzuleiten. Zudem ist weiterhin denkbar, die Entwickler und die Teams mit dem PSP oder TSP zu unterstützen (siehe Abschnitt 2.2).

Für AG1, das keine Zertifizierung besitzt, könnte eine Zwischenlösung wie die Benutzung von PSP und TSP (siehe Abschnitt 2.2.2) oder die Einführung von ITIL-Ansätzen (siehe Abschnitt 2.2.3) die Grundlage für die Verbesserung der Planung und Qualität der Software darstellen, ohne dabei auf die Agilität und Spontaneität seiner Entwickler verzichten zu müssen.

#### **8.4.2 Größe**

WaWiSys und CoSo werden als Softwareentwicklungsprojekte einen bedeutenden Einfluss auf den zukünftigen Erfolg von AG1 und AG2 haben. Während AG1 die Kontrolle über die Entwicklungsphasen des Projektes beibehält, besteht für AG2 die Gefahr, dass die Sichtbarkeit und Kontrolle über die verteilten Teams im Laufe des Projektes verloren geht. Daher ist für AG2 besonders wichtig, seine Kernkompetenzen nicht auszulagern. DL2 muss zum einen ausgereifte Technikenkenntnisse in der .NET Technologie besitzen, zum

anderen darf er aber nicht die komplette Kontrolle des Produktes und der Entwicklung allein für sich behalten. Wichtige Positionen des Softwareentwicklungsprojektes werden von AG2-Personal abgedeckt (siehe Kapitel 5.3).

### **8.4.3 Bedingungen für die Qualitätssicherung**

AG1 muss sich zusammen mit DL1 im Voraus über die Spezifikation der Schnittstellen absprechen, um potentielle Verständnisprobleme zu vermeiden. DL1 muss zum Beispiel wissen, wie und mit welchem Format die Kunden- und Artikeldatensätze angelegt werden, welche Informationen für das Reporting Tool erfasst werden müssen und wie diese dargestellt werden sollen. Änderungen der Spezifikation, bedingt durch neue Kundenwünsche, sollen kollaborativ zwischen AG1 und DL1 in zeitlichen Abständen erfolgen. Um die Qualität der entwickelten Module zu testen, können webbasierte Inspektionen durchgeführt werden (siehe Kapitel 7.3). Zusätzlich werden die risikoreichen Funktionen der Module lokal von AG1-Entwicklern getestet, um die Kommunikation der Schnittstellen zu überprüfen.

Die Software CoSo verfügt über eine formale und präzise Spezifikation. DL2 kann anhand dieser Information und des vorhandenen Datenbankmodells die ersten Entwürfe der neuen Version ausführen. Dabei ist die sorgfältige Aufbewahrung von Informationen in beiden Softwareentwicklungsprojekte sehr wichtig. DL1 könnte an wichtige Informationen über die Marktwirtschaft des gewerblichen Kunden gelangen, während DL2 Zugriff auf persönliche Patientendaten bekommen könnte.

### **8.4.4 Kultur der verteilten Teams**

Eine grundsätzliche Unterscheidung der Softwareentwicklungsprojekte liegt in der Art der benötigten Kommunikation. Während WaWiSys den informellen Weg vorzieht, kommt für CoSo nur eine formelle Kommunikation in Frage (siehe 6.2). Beide Softwareentwicklungsprojekte benötigen ein Werkzeug für

die Unterstützung der Kommunikation, wobei die Teams für das WaWiSys Projekt, bedingt durch die Distanz, eine synchrone Art der Kommunikation bevorzugen werden. Die Koordination der kommunikativen Tätigkeiten im Projekt CoSo muss flexibel gestaltet werden, so muss besonders am Anfang des Projektes ständig kommuniziert werden, um das nötige Fachwissen vom AG2 auf den DL2 zu übertragen, in späteren Phasen kann die Kommunikation dann verringert werden.

Über die Benutzung einer fremden Sprache in verteilten Softwareentwicklungsprojekten wird gesagt, dass die die fließend Englisch sprechen öfters via Telefon(-konferenz) kommunizieren. Diejenigen, die dieser Sprache nicht mächtig sind, ziehen die schriftliche Kommunikation vor, denn dadurch gewinnen sie Zeit, um Mitteilungen besser verfassen zu können [HeGr99b]. Um die Kommunikationswege für beide Softwareentwicklungsprojekte reibungsloser zu gestalten werden Protokolle definiert, die die Form der Kommunikation festlegen. Diese Feststellungen helfen von Anfang an im Softwareentwicklungsprojekt falsche Annahmen zu verhindern und schaffen zusätzlich ein besseres Verständnis für die jeweils anderen Kulturkreise.

Für die schriftliche und mündliche Kommunikation im Projekt WaWiSys werden Vorgehensweisen erarbeitet, um den Kommunikationsfluss zu erleichtern. Für die schriftliche Kommunikation bestimmen die Teammitglieder zuerst welche Prioritäten sie setzen wollen und wie schnell auf eine Nachricht geantwortet werden soll. Statistiken sagen, dass das Beantworten von E-Mails in verteilten Teams länger dauert als bei Teams, die sich vor Ort befinden ([HeGr99a], [HuOc06]). Das liegt vermutlich daran, dass eine Nachricht in einer verteilten Umgebung nicht so schnell wahrgenommen wird, wenn der Absender unbekannt ist. Aber auch wenn der Absender bekannt ist, kann es vorkommen, dass nicht schnell genug reagiert wird. Der Kontext einer schriftlichen Mitteilung wird vergrößert, wenn Anhänge wie Zeichnungen oder wichtige Dateien beigefügt werden, deswegen müssen für das laufende Projekt Standards für Dateiformate verabschiedet werden. Damit auf eine Anfrage nicht zu lange auf eine Antwort gewartet werden muss, werden Vorgehensweisen bei Anwesenheit, Urlaub oder Krankheit in das Kommunikationsprotokoll aufgenommen und das benutzte E-Mail-Programm muss diesen An-

beziehungsweise Abwesenheitsstatus der Mitglieder unterstützen. Ziel ist es, direkt über die Verfügbarkeit anderer verteilter Mitglieder zu erfahren, damit die Projektleitung bei Fragen bezüglich Projektwissens und Verfügbarkeit entlastet wird (siehe Kapitel 6.2).

Bei der Portierung der Software CoSo soll das Vertrauen der Entwickler von AG2 in andere Teams (DL2) aufgebaut werden. Dabei soll die informelle Kommunikation der verteilten Teams gestärkt und am Anfang des Projektes Treffen organisiert werden, bei denen der direkte Kontakt und ein erstes Kennenlernen deutlich im Vordergrund stehen. Die Projektleitung beider Softwareentwicklungsprojekte sollte frühzeitig einen Teil des Budgets für Reisen einplanen um die potentiell im Laufe des Softwareentwicklungsprojektes auftretenden Konflikte bereits am Anfang anzugehen [GrHP99]. Diese Treffen dienen auch dazu, die Protokolle für die Kommunikation und die zu gebrauchende Terminologie zu entwickeln sowie sich mit den kulturellen Unterschieden zu befassen. Aufgabe des Managements ist es, die ständige Kommunikation aller Beteiligten zu fördern, denn immer wieder zeigt sich, dass es besser ist mehr zu kommunizieren, als in späten Phasen der Entwicklung teure Korrekturen durchführen zu müssen.

Nicht nur das Management soll die verteilten Teams während des Projektes besuchen, sondern auch die Entwickler, die durch ein persönliches Kennenlernen der anderen verteilt arbeitenden Entwickler deren Arbeit besser schätzen lernen können. Dabei soll nicht nur im Problem- oder Streitfall gereist werden, sondern viel mehr um neue Synergieeffekte zu schaffen, die die Produktivität des Projektes steigern [LiLL06]. Wenn sich die Reisekosten als zu hoch erweisen, dann werden diese durch Schulungen, Workshops, Video- oder Telefonkonferenzen ersetzt. Eine Instant Messaging Lösung (IM) oder eine virtuelle Videokonferenz, die als Teil der CDE vorhanden sind, können zumindest ein virtuelles Gefühl von Nähe erzeugen ([HeGr99a] und Kapitel 6).

### **8.4.5 Dynamik**

Anders als CoSo verlangt WaWiSys, bedingt durch den Zeitdruck und die Art der Entwicklung einen erhöhten Synchronisierungsbedarf. Die Lieferungen der Ergebnisse seitens des DL1 werden, wenn möglich, im wöchentlichen Rhythmus erwartet, um eine „Big-bang“ Integration zu vermeiden (siehe Abschnitt 5.1.4). Sollte AG1 einen ähnlichen Ansatz wie bei PSP/TSP vorziehen, sind „Launch“, „Relaunch“ und „Post Mortem“ Phasen hilfreich, um den Entwicklungsprozess zu verbessern (siehe Abschnitt 2.2.2). Die Notwendigkeit eines gemeinsamen Repository für die Quellcodedateien des WaWiSys Projekt ist ebenfalls erforderlich.

Die Portierung von CoSo verlangt dagegen keine ständige Synchronisation der Ergebnisse. In den ersten Phasen des Projektes wird mit einem erhöhten Kommunikationsbedarf gerechnet, um die Spezifikation und Planung des Projektes zu organisieren. Später reicht es, wenn das Softwareentwicklungsprojekt monatliche Ergebnisse liefern kann.

### **8.4.6 Personal**

Im Abschnitt 5.1.5 wurden die erwünschten Eigenschaften der Teammitglieder, so genannte „CRACKS“, vorgestellt. Es ist notwendig, dass die Teammitglieder des WaWiSys Projekt sehr kundenorientiert arbeiten, da die Spezifikation, die Änderungen während des Projektes und die endgültige Abnahme immer direkt mit dem Kunden abgesprochen werden sollen. Es ist auch denkbar, dass AG1 einen finanziellen Bonus für alle Teammitglieder vorschlägt, wenn sie bestimmte Qualitätsmerkmale oder Termine einhalten. AG2 benötigt auf lokaler Ebene kollaborative Entwickler, die bereit sind ihr Know-how über CoSo weiterzugeben. Entwickler auf der DL2-Seite sollten ein starkes Interesse und Engagement liegen, die Spezifikationen der Software zu verstehen und die Portierung schnell umzusetzen, um zukünftige Aufträge oder eine Weiterempfehlung vom AG2 zu bekommen.

## 8.5 CDE

Die auszuwählende CDE soll auf jeden Fall die Bereiche der Kollaboration, Koordination und Gemeinschaftsbildung der Projekte abdecken (siehe Abbildung 40). AG1 hat, bedingt durch die Anzahl der Teammitglieder, mehr Auswahlmöglichkeiten als AG2. Kostengünstige Varianten stellen die Open Source Lösung SourceForge.net und die proprietären Lösungen SourceForge® Enterprise Edition oder CodeBeamer dar. Die letzten beiden CDEs sind aber nur bedingt, für bis zu 15 Benutzer, kostenfrei und deswegen besonders interessant für AG1. Im Projekt CoSo wird sich die Projektleitung vermutlich für eine kostenpflichtige Lösung entscheiden müssen. Zu den vorher genannten CDEs wird auch CollabNet gezählt. Der im Kapitel 6.5 durchgeführte CDE-Vergleich kann weitere Hilfestellung für die Entscheidung über eine CDE leisten.

Dateien, die den Quellcode der verteilten Softwareentwicklungsprojekte beinhalten sollen in einem zentralen Repository (in den Büroräumen von AG1 beziehungsweise AG2) verwaltet werden (siehe Kapitel 6). In diesem zentralen Repository werden alle Projektdateien gespeichert, so dass diese jederzeit die aktuellen Projektstrukturen widerspiegeln können. Die Dateien werden je nach Zuständigkeitsbereich in verschiedenen Unterverzeichnissen des Repository gespeichert, also separat und nicht alle in ein einziges Verzeichnis. Das ist eine wichtige Bedingung, um unter anderem das Erkennen von Änderungen im Projekt zu gewährleisten. Ein verteiltes Softwareentwicklungsprojekt, das in Paketen und Komponenten organisiert ist, hilft verteilten Teams die Funktionalitäten unabhängig von anderen Teams zu organisieren. Dadurch werden mehr Stabilität und eine bessere Vorhersagbarkeit des Quellcodes gewonnen, da Änderungen in einer Komponente nicht unbedingt andere Komponenten beeinflussen müssen.

Tabelle 12 zeigt die hier aufgeführten Prozessverbesserungen auf. Diese fiktiven Softwareentwicklungsprojektbeispiele machen deutlich, dass die Zusammenarbeit und Koordination verteilter Teams eine komplexe Aufgabe ist, die es stets weiter zu verbessern gilt.

## Qualitätssicherung im Rahmen verteilter Softwareentwicklung

	WaWiSys	CoSo
Größe	<ul style="list-style-type: none"> <li>▪ DL1 entwickelt nur einen Teil der Software.</li> </ul>	<ul style="list-style-type: none"> <li>▪ Domänenwissen von DL2 sehr wichtig.</li> <li>▪ Kernkompetenzen sollen nicht ausgelagert werden</li> </ul>
Sicherheit	<ul style="list-style-type: none"> <li>▪ Absprache für Spezifikation der Schnittstellen</li> <li>▪ Änderungen kollaborativ in zeitlichen Abständen</li> <li>▪ Möglichkeit für Inspektionen</li> <li>▪ Testen der risikoreichen Funktionen</li> <li>▪ Sorgfältige Aufbewahrung der Information</li> </ul>	<ul style="list-style-type: none"> <li>▪ Formale und präzise Definitionen</li> <li>▪ Testen der Funktionalität bei AG1 im Vergleich mit vorhandener Software</li> <li>▪ Sorgfältige Aufbewahrung der Information</li> </ul>
Kultur	<ul style="list-style-type: none"> <li>▪ Kommunikation informell</li> <li>▪ Werkzeugunterstützung notwendig</li> <li>▪ „CRACKS“ notwendig</li> </ul>	<ul style="list-style-type: none"> <li>▪ Formelle Kommunikation</li> <li>▪ Häufigkeit der Kommunikation soll flexibel gestaltet werden</li> <li>▪ Monatliche Lieferungen</li> </ul>
Dynamik	<ul style="list-style-type: none"> <li>▪ Ständige Lieferungen</li> <li>▪ Erhöhter Synchronisationsbedarf</li> <li>▪ Häufige Integration und Tests</li> <li>▪ Repository notwendig</li> </ul>	<ul style="list-style-type: none"> <li>▪ Große Distanz</li> <li>▪ Synchronisation nicht dynamisch</li> </ul>
Personal	<ul style="list-style-type: none"> <li>▪ Kundenbezogen</li> <li>▪ Motivation</li> </ul>	<ul style="list-style-type: none"> <li>▪ Kollaborative Entwickler</li> </ul>
CDE	<ul style="list-style-type: none"> <li>▪ Open Source Lösung denkbar</li> </ul>	<ul style="list-style-type: none"> <li>▪ Kostenpflichtige Lösung</li> </ul>

**Tabelle 12. Prozessverbesserungen der Softwareentwicklungsprojekte**

## 9 Zusammenfassung und Ausblick

Über einen Zeitraum von mehreren Jahren wurde die Softwareentwicklung als eine in-house Aktivität verstanden [LiLL06], was sich allerdings in den letzten Jahren stark verändert hat. Die Softwareentwicklung hat die Möglichkeit der Spezialisierung und Aufsplitterung von Prozessen und Produkten für sich entdeckt, die neue Ansätze bot die Softwarequalität weiter zu steigern und die Entwicklung zu beschleunigen [Kola04].

Die verteilte Softwareentwicklung ist eine Form der Softwareentwicklung, die an zwei oder mehreren Standorten geographisch verteilt stattfindet und dadurch verschiedene Kulturen mit einbezieht. Dadurch entstehen neue Möglichkeiten für die Softwareentwicklung. Diese neue Möglichkeiten zeigen sich in vielerlei Gestalt, so kann zum Beispiel eine weltweit billige und wettbewerbsfähige Software produziert werden, welche potentielle Geschäftsvorteile durch die Bindung an neue Märkte möglich macht, eine schnelle Bildung von virtuellen Teams wird erleichtert, verteilte Entwickler machen eine Arbeit „rund-um-die-Uhr“ möglich und es besteht die Möglichkeit sich mit anderen Organisationen zusammenzuschließen um so potentielle Marktchancen optimal zu nutzen.

Dabei existieren drei große Bereiche, die bei jedem verteilten Softwareentwicklungsprojekt zu Problemen führen können ([SeCS06], [Scha06a], [Scha06b], [CaAb06], [CaAb06], [HeMo01], [BCKS01]), nämlich die kulturellen Unterschiede der Beteiligten, die verschiedenen Zeitzonen und die potentiellen Kommunikationsprobleme. Weitere Probleme der verteilten Softwareentwicklung sind ([Karo98], [BoTu03], [PaLa04]):

- die Abhängigkeit der verteilten Entwicklerteams,
- die Koordinationsschwierigkeiten zwischen diesen,
- die Schwierigkeiten bei der Verteilung der Arbeit und Aufgaben an die verschiedenen Teams, oder
- die von verteilten Entwicklerteams falsch implizierten Annahmen.

Der ausgewählte Partner kann einem anderen Kulturkreis angehören und sich auf einem anderen Kontinent befinden, was bedeutet, dass sich zum Beispiel die Zeitunterschiede als problematisch herausstellen können. Eine gute Koordination der genannten Bedingungen entscheidet über Erfolg oder Misserfolg eines verteilten Softwareentwicklungsprojektes. Die Koordination beschäftigt sich mit dem Zusammenspiel zwischen verteilten Teammitgliedern, Prozessen, Informationen und Technologien [Wire06].

Diese Diplomarbeit hat eine Reihe von Maßnahmen für drei aktuelle Forschungsbereiche der verteilten Softwareentwicklung [SeCS06] vorgestellt:

- die Vorgehensmodelle und Methoden der verteilten Entwicklung,
- die Koordination der Testaktivitäten und
- die Kollaborative Entwicklungsplattform und Wissensverteilung,

Standards und Metriken (Kapitel 2) haben im Laufe der Jahre ihre Wichtigkeit für die Qualitätsverbesserung eines Projektes bewiesen. Diese Diplomarbeit verfolgt den Ansatz, dass die Produktqualität verteilter Softwareentwicklungsprojekte eng an die Prozessverbesserung gekoppelt ist. Aktuelle Ansätze wie CMMI, PSP, TSP, ITIL, SLA und der ISO-9000 wurden im Kapitel 2 vorgestellt. Es wurde herausgearbeitet, dass diese teilweise an die Eigenschaften verteilter Softwareentwicklungsprojekte angepasst werden müssen, um ihren großen Nutzwert weiterhin ausschöpfen zu können. Für CMMI und ITIL wurden Vorschläge gemacht, wie diese an die verteilten Umgebungen angepasst werden können. Während für CMMI 24 neue KPAs vorgestellt wurden [RaKK05], lassen sich die Definitionen von ITIL und PSP/TSP an die notwendigen Vorgaben für die verteilten Teams anpassen. ITIL unterstützt sowohl die Arbeit mit internen als auch externen Dienstleistern [Kirk05]. Außerdem kann PSP/TSP, unabhängig von der Verteilung der Teams, immer für die kontinuierliche Verbesserung der verschiedenen Teams genutzt werden [SeMo04]. Um die strategischen Ziele auch erreichen zu können, ist es wichtig die Produktqualität und die Produktivität der Softwareentwicklung stetig zu verbessern. Abbildung 12 (Kapitel 3) zeigt, wie die Interaktion dieser Prozessrahmenwerke aussehen kann.

Das Entscheidungsmodell von Boehm und Turner (BoTu03) versucht mit Hilfe von fünf Faktoren eine bessere Mischung zwischen Agilität und Disziplin im Rahmen des Vorgehensmodells zu erreichen. Diese sind (Abbildung 16, Kapitel 4):

- Größe des Softwareentwicklungsprojektes
- Bedingungen für die Qualitätssicherung
- Softwareentwicklungskultur
- Dynamik des Projektes und der Teams
- Personalbedingungen

Diese Faktoren wurden in Kapitel 5 mit zusätzlichen Erfahrungen verteilter Softwareentwicklungsprojekte [HePB05] an die verteilte Softwareentwicklung angepasst. Die Erfahrungen, die mit der Zeit gesammelt worden sind, ergänzen das Entscheidungsmodell für verteilte Softwareentwicklungsprojekte. Zusätzlich zu den angepassten Faktoren wurden, abhängig von Projektart und Aufgabenstellung, Organisationsstrukturen und Teamaufteilungen vorgestellt. Zum Schluss, in Kapitel 6, wurden die Unterstützung der Entwickler und der Kontrollaktivitäten behandelt. Die Kollaborative Entwicklungsplattform (CDE) hilft den Entwicklern so genannte Reibungspunkte zu verringern. Diese sechs Reibungspunkte sind [BoBr02]:

- Start-Up Kosten
- Kollaboration
- Kommunikation
- Zeitverlust
- Verhandlung zwischen Teilnehmer
- Nicht funktionierende Artefakte

Die aufgeführten aktuellen Lösungen wie MILOS, GotDotNet, SourceForge.net, SourceForge® EE, CollabNet Enterprise Edition oder CodeBeamer können alle als ein Schritt in die richtige Richtung gedeutet werden, um die verteilte Zusammenarbeit der Teammitglieder und die Projektkontrolle zu gewährleisten. Tabelle 5 (Kapitel 6) zeigt einen Vergleich der Vorteile der CDEs. Der dezentrale Ansatz der Softwareentwicklung weckt neue Fragen bezüglich des

Koordinierungsbedarfs, des Testens für verteilte Teams und den Auftraggeber ([BCKS01], [SeCS06]). Diese wurden in Kapitel 7 behandelt und zeigen drei Teilbereiche auf, die in verteilten Softwareentwicklungsprojekten mehr Aufmerksamkeit auf sich ziehen, als in in-house Projekten:

- die Datengeheimhaltung,
- die Größe der Testdaten und
- die Integration der entwickelten Module.

Im Kapitel 8 wurden, in exemplarischer Form, zwei Projektbeispiele vorgestellt, die Entscheidungen und Vorgehensmodelle simulieren, die in den vorherigen Kapiteln aufgezeigt wurden. Der Nachteil dieser Beispiele liegt sicherlich in ihrer fiktiven Herkunft begründet. Eine Validierung dieser Beispiele bleibt offen. Nichts desto trotz sollen die vorgestellten Projektbeispiele den Entscheidungsträger dabei helfen, frühzeitig potentielle Risiken zu identifizieren, um die Qualitätssicherung im Rahmen der verteilten Softwareentwicklung zu erleichtern.

Heutzutage müssen die Firmen vorsichtiger mit der Auslagerung von Softwareentwicklungsprojekten umgehen, denn die Kostenvorteile werden durch die Auslagerung von Teilen der Produktion, der IT oder Dienstleistungen zunehmend geringer:

*„Laut dem Jahresbericht 2006 für globale Dienstleistungen der Wirtschaftsberatung A.T. Kearney wird sich das Lohngefälle zwischen etablierten und sich entwickelnden Volkswirtschaften zwar noch über die nächsten 20 Jahre halten, aber zunehmend kleiner werden. Denn durch zunehmende Qualifizierung der Mitarbeiter und steigende Löhne in den beliebtesten Outsourcing-Regionen sowie unter Druck geratene Löhne in den Industrieländern sowie Währungseffekte wird der Kostenvorteil kleiner.“<sup>(57)</sup>*

---

<sup>57</sup> <http://www.heise.de/newsticker/meldung/87305>. (Abruf am 23.03.2007)

Die Entscheidungsträger sollten nicht nur an kurzfristige Kostenvorteile denken, sondern vielmehr langfristige Faktoren wie Qualifikation und örtliche Bedingungen berücksichtigen. Die Auslagerung der Softwareentwicklung ist ein weiter steigender Trend. In naher Zukunft werden Entscheidungskompetenzen an Länder abgegeben, in denen die Softwareentwicklung stattfindet<sup>(58)</sup>.

Diese Diplomarbeit untersucht die Möglichkeiten zur Verbesserung der Qualität der Ergebnisse verteilter Softwareentwicklungsprojekte. Sie zeigt die Probleme der verteilten Entwicklung und die adäquaten Lösungsvorschläge auf. Alle in dieser Diplomarbeit aufgeführten Vorschläge laufen auf das Ziel hinaus eine Verbesserung der Prozessqualität der Softwareentwicklung zu erreichen. Diese sind sowohl für komplett neue Softwareentwicklungsprojekte als auch für Projekte, bei denen der Quellcode schon vorhanden ist hilfreich. Es wäre wünschenswert gewesen, wenn sich die hier vorgestellten Lösungsansätze in einer realen Umgebung validieren ließen. Zukünftige Studien können diese Vorschläge in realen Projekten umsetzen und dadurch neue Erkenntnisse gewinnen.

---

<sup>58</sup> [http://www.handelsblatt.com/news/Default.aspx?\\_p=201197&\\_t=ft&\\_b=1242778](http://www.handelsblatt.com/news/Default.aspx?_p=201197&_t=ft&_b=1242778) (Abruf am 19.03.2007)

## 10 Literaturverzeichnis

- [AICK05] Al-Kilidar, H., Cox, K., Kitchenham, B.: **The Use and Usefulness of the ISO/IEC 9126 Quality Standard**. International Symposium on Empirical Software Engineering, 2005.
- [BaBM96] Basili, V., Briand, L., Melo, W.: **A validation of object-oriented design metrics as quality indicators**. IEEE Transactions on Software Engineering, 22(10), Seiten: 751-761, 1996.
- [Balz01] Balzert, H.: **Lehrbuch der Softwaretechnik**. Spektrum Akademischer Verlag, 2001.
- [BCKS01] Battin, R. D., Crocker, R., Kreidler, J., and Subramanian, K.: **Leveraging Resources in Global Software Development**. Software, IEEE. Volumen: 18, Issue: 2, Seiten: 70-77. 2001.
- [Beck05] Beck, Kent.: **Extreme programming explained embrace change**. Boston Addison-Wesley, 2005.
- [Bind00] Binder, R.: **Testing Object-Oriented Systems**. Addison-Wesley. 2000.
- [BMC06] **ITIL für Kleine und Mittelständische Unternehmen**. bmcsoftware. 2006.  
URL: [www.bmc.com/de\\_DE/doc\\_depot/White\\_Paper\\_ITIL\\_Kleine\\_Mittelstaendische\\_Unternehmen.pdf](http://www.bmc.com/de_DE/doc_depot/White_Paper_ITIL_Kleine_Mittelstaendische_Unternehmen.pdf)  
Abruf am 28.03.2007.
- [BoBr02] Booch, G.; Brown, A.: **Collaborative Development Environments**. Rational Software Corporation. 2002.
- [Boeh88] Boehm, B. W.: **A Spiral Model of Software Development and Enhancement**. Computer 21, 5, Seiten: 61-72. 1988.
- [BoTu03] Boehm, B. and Turner, R.: **Using Risk to Balance Agile and Plan-Driven Methods**. Computer 36, 6, Seiten: 57-66. 2003.
- [BoTu04a] Boehm, B. and Turner, R.: **Balancing Agility and Discipline: Evaluating and Integrating Agile and Plan-Driven Methods**. International Conference on Software Engineering. IEEE Computer Society, Washington, DC, Seiten: 718-719. 2004.
- [BoTu04b] Boehm, B. W.; Turner, R.: **Balancing agility and discipline : a guide for the perplexed**. Boston; Munich [u.a.] : Addison-Wesley, 2004.
- [Buds07] Budzuhn, F.: **Subversion**. Galileo Press, Bonn. 2007.

- [Buds05] Budsuhn, F.: **CVS - Windows- und Open Source-Projekte managen**. Galileo Press, 2005.
- [CaAb06] Carmel, E., Abbott, P.: **Configurations of global software development: offshore versus nearshore**. International Workshop on Global Software Development For the Practitioner. GSD '06. ACM Press, New York, NY, Seiten: 3-7, 2006.
- [ChKe94] Chidamber, S., Kemmerer, C.: **A metrics suite for object oriented design**. IEEE Transactions on Software Engineering. 20(6): Seiten: 476-492, 1994.
- [ChMa04] Chau, T., Maurer, F.: **Tool Support for Inter-Team Learning in Agile Software Organizations**. 6th International Workshop on Learning Software Organizations 2004, Springer-Verlag, 2004.
- [Conw68] Conway, M.E.: **How Do Committees Invent?** Datamation, Vol. 14, Nr. 4, Seiten: 28-31. 1968.
- [CuKO92] Curtis, B., Kellner, M.I., und Over, J.: **Process Modelling**. Commun. ACM 35, 9, Seiten: 75-90. 1992.
- [CuMu03] Čubranić, D., Murphy, G. C.: **Hipikat: recommending pertinent software development artifacts**. International Conference on Software Engineering. IEEE Computer Society, Washington, DC, Seiten: 408-418. 2003.
- [DeBa04] Deutsche Bank Research: **IT-Outsourcing: Zwischen Hungerkur und Nouvelle Cuisine**, e-economics Nr. 43, 6. April 2004.
- [DHRY05] Dodson, K., Hofmann, H., Ramani, G., Yedlin, D.: **Adapting CMMI for Acquisition Organizations: A Preliminary Report**. 2005.  
URL:<http://www.sei.cmu.edu/programs/acquisition-support/cmmi-acq.html>  
Abruf am 18.04.2007
- [Faga76] Fagan, M.: **Design and Code inspections to Reduce Errors in Program Development**. IBM Systems Journal, Seiten: 182-211, 1976.
- [Fent94] Fenton, N.: **Software Measurement: A Necessary Scientific Basis**. IEEE Transactions of Software Engineering, vol. 20 no. 3, 1994.
- [Four01] Fournier, R., Infoworld, March 5, 2001
- [Garv84] Garvin, D.: **What Does "Product Quality" Really Mean?** Sloan Management Review, vol. 24, 1984.

- [GHKR06] Geisser, M., Hildenbrand, T., Klimpke, L., Rashid, A.: **Werkzeuge zur kollaborativen Softwareerstellung – Stand der Technik**. Working Paper Universität Mannheim, 2/2006.
- [Gies05] Giese, H.: Softwaretechnikpraktikum. Universität Paderborn Fachgebiet Softwaretechnik, 2005.
- [Glob05] Globke Wolfgang.: **Software-Metriken**. Seminar "Moderne Softwareentwicklung" SS 2005. Universität Karlsruhe. Juni 2005.
- [GrHP99] Grinter, R. E., Herbsleb, J. D., und Perry, D. E.: **The geography of coordination: dealing with distance in R&D work**. Conference on Supporting Group Work. GROUP'99. ACM Press, New York. 1999.
- [Grin98] Grinter, R. E.: **Recomposition: putting it all back together again**. Conference on Computer Supported Cooperative Work. CSCW '98. ACM Press, New York, NY, Seiten: 393-402. 1998.
- [Hedb04] Hedberg, H.: **Introducing the Next Generation of Software Inspection Tools**. Department of Information Processing Science, University of Oulu, Finland. 2004.
- [HeGr99a] Herbsleb, J., Grinter, R.: **Splitting the Organization and Integrating the Code: Conway's Law Revisited**. International Conference of Software Engineering, ACM Press, New York. Seiten: 85-95. 1999.
- [HeGr99b] Herbsleb, J., Grinter, R.: **Architectures, Coordination, and Distance: Conway's Law and Beyond**. IEEE Software, Seiten: 63-70, Sept/Okt 1999.
- [HeMo01] Herbsleb, J. D.; Moitra, D.: **Global software development Software**, IEEE Volume 18, Issue 2, March-April 2001 Seiten:16-20, 2001.
- [HePB05] Herbsleb, J. D., Paulish, D. J., und Bass, M. **Global software development at siemens: experience from nine projects**. International Conference on Software Engineering. ICSE '05, Seiten: 524-533. 2005.
- [HHXJ06] Hongxun, J., Honglu, D., Xiang, Y., und Jun, S.: **Research on IT outsourcing based on IT systems management**. ICEC 2006, vol. 156. ACM Press, New York, NY, Seiten: 533-537. 2006.
- [HMFG00] Herbsleb, J. D., Mockus, A., Finholt, T. A., und Grinter, R. E.: **Distance, dependencies, and delay in a global collaboration**. Conference on Computer Supported Cooperative Work, CSCW 2000. ACM Press, New York, 2000.

- [HoLS03] Holzmüller, H., Lammerts, A., Stolper, M.: **Studie: ITIL-Status und Trends in Deutschland**. 2003.  
URL: <http://www.it-surveys.de/itsurvey/its-b.html>,  
Abruf: 28.03.2007
- [HoZB05a] Hochstein, A., Zarnekow, R., Brenner, W.: **ITIL as Common Practice Reference Model for IT Service Management: Formal Assessment and Implications for Practice**. 2005
- [HoZB05b] Hochstein, A., Zarnekow, R., Brenner, W.: **Evaluation of Service-Oriented IT Management in Practice**. IEEE. 2005.
- [HuOc06] Huang, H., Ocker, R.: **Preliminary insights into the in-group/out-group effect in partially distributed teams: an analysis of participant reflections**. Conference on Computer Personnel Research: Forty Four Years of Computer Personnel Research: Achievements, Challenges and the Future. SIGMIS CPR '06. ACM Press, New York, NY, Seiten: 264-272. 2006.
- [Hump97] Humphrey, W.: **A discipline for software engineering**. Addison-Wesley. 1997.
- [Hump99] Humphrey, W.: **Pathways to Process Maturity: The Personal Software Process and Team Software Process**. SEI Interactive. 06/1999.
- [Hump00] Humphrey, W.: **Introduction to the Team Software Process**. Addison Wesley Longman, 2000.
- [HuTh03] Hunt, A., Thomas., D.: **The Pragmatic Programmers. Pragmatic Unit Testing in Java with JUnit**. Raleigh, North Carolina Dallas, Texas, 2003
- [IEEE04] IEEE Std 1061-1998 (R2004).: **IEEE Standard for a Software Quality Metrics Methodology**. 2004
- [ISO9126] ISO/IEC.: **Informationstechnik - Beurteilen von Softwareprodukten, Qualitätsmerkmale und Leitfaden zu deren Verwendung**. 1991
- [ISO9126a] ISO/IEC.: **ISO/IEC 9126-1 Software engineering- Product quality- Part 1: Quality model**. 2001.
- [ISO9126b] ISO/IEC.: **ISO/IEC 9126-2 Software engineering -Product quality- part2: External metrics**. 2002.
- [ISO9126c] ISO/IEC.: **ISO/IEC 9126-3 Software engineering -Product quality- part3: Internal metrics**. 2002.
- [ISO9126d] ISO/IEC.: **ISO/IEC 9126-4 Software engineering -Product quality- part4: Quality In Use metrics**. 2002.

- [Ju06] Ju, D.: **A concerted effort towards flourishing global software development.** International Workshop on Global Software Development for the Practitioner. GSD '06. ACM Press, New York, NY, Seiten: 62-65. 2006.
- [Kala03] Kalayci, O.: **CMMI versus XP.** Impact of software process on quality (IMPROQ 2003) Workshop. Bilkent University, Ankara. 2003.
- [Karo98] Karolak, Dale Walter.: **Global software development managing virtual teams and environments.** Los Alamitos, Calif. [u.a.] IEEE Computer Society, 1998.
- [Kent95] Kent, B.: **Extreme programming explained: embrace change.** Addison-Wesley, Boston. 1995.
- [Kirk05] Kirkwood, S.: **Intelligent Outsourcing. Applying ITIL Process Governance and Architecture to Outsourced IT Operations.** INS Whitepaper. 2005.
- [Kola04] Kolawa, A.: **Outsourcing: devising a game plan.** Queue 2, November. Seiten: 56-62. 2004.
- [Kneu95] Kneuper, R., Sollmann, R.: **Normen zum Qualitätsmanagement bei der Softwareentwicklung.** Informatik Spektrum, Band 18, Seiten: 314-323. 1995.
- [Kneu03] Kneuper, R.: **CMMI.** dpunkt-Verlag. 2003.
- [LaBa03] Larman, C., Basili, V.: **Iterative and Incremental Development: A Brief History.** IEEE Computer Society. Seiten: 47-56. 2003
- [LaLa89] Larson, C., LaFasto, F.: **Teamwork: What Must Go Right; What Can Go Wrong.** Newbury Park, Calif.: Sage, 1989.
- [LHKP03] Lee, J., Huynh, M. Q., Kwok, R. C., and Pi, S.: **IT outsourcing evolution---: past, present, and future.** Commun. ACM 46, 5 Seiten: 84-89. 2003.
- [LiLL06] Lindqvist, E., Lundell, B., und Lings, B.: **Distributed development in an intra-national, intra-organisational context: an experience report.** International Workshop on Global Software Development For the Practitioner. GSD '06. ACM Press, New York, NY, Seiten: 80-86. 2006.
- [LaMa03] Lanubile, F.; Mallardo, T.: **An empirical study of Web-based inspection meetings.** International Symposium on Empirical Software Engineering, ISESE 2003. Seiten: 244- 251. 2003.
- [Matl05] Matloff, N.: **Offshoring: What Can Go Wrong?** IT Professional 7, 4, Seiten: 39-45. 2005.

- [McCo01] McConnell, S.: **Rapid development taming wild software schedules**. Redmond, Washington Microsoft Press, 2001.
- [MoHe02] Mockus, A., Herbsleb, J. D.: **Expertise browser: a quantitative approach to identifying expertise**. International Conference on Software Engineering. ICSE '02. ACM Press, New York, NY, Seiten: 503-512. 2002.
- [MSHK99] Maurer, F., Succi, G., Holz, H., Kötting, B., Goldmann, S., und Dellen, B.: **Software process support over the Internet**. International Conference on Software Engineering. International Conference on Software Engineering. IEEE Computer Society Press, Los Alamitos, CA, 642-645. 1999.
- [Müll04] Müller, M.: **Should we use programmer pairs or single developers for the next project?** Technischer Bericht 2004-8, Fakultät für Informatik, Universität Karlsruhe, 2004
- [Müll06] Müller, M.: **Do Programmer Pairs make different Mistakes than Solo Programmers?** In Conference on Empirical Assessment In Software Engineering (EASE), Keele, UK, April 2006.
- [MüPa03] Müller, M., Padberg, F.: **On the Economic Evaluation of XP Projects**. In Joint European Software Engineering Conference (ESEC) and SIGSOFT Symposium on the Foundations on Software Engineering (FSE), Seiten: 168–177, Helsinki, Finland, September 2003.
- [MüPa04] Müller, M., Padberg, F.: **An Empirical Study about the Feel good Factor in Pair Programming**. In International Symposium on Software Metrics (Metrics), Chicago, Illinois, USA, September 2004.
- [NaHe05] Narayanaswamy, R., Henry, R. M.: **Effects of culture on control mechanisms in offshore outsourced IT projects**. Conference on Computer Personnel Research. SIGMIS CPR '05. ACM Press, New York, NY, Seiten: 139-145. 2005.
- [NaWo01] Nawrocki, J., Wojciechowski, A.: **Experimental Evaluation of Pair Programming**. In European Software Control and Metrics (Escom), London, UK, 2001.
- [NgBV06] Nguyen, P. T., Babar, M. A., und Verner, J. M.: **Critical factors in establishing and maintaining trust in software outsourcing relationships**. International Conference on Software Engineering. ICSE '06. ACM Press, New York, NY, Seiten: 624-627, 2006.
- [Nose98] J. Nosek.: **The Case for Collaborative Programming**. Communications of the ACM, 41(3):105–108, März 1998.
- [OfGC00] Office of Government Commerce: **IT Infrastructure Library**. London: The Stationary Office, 2000

- [OHRG04] Oza, N., Hall, T., Rainer, A., and Grey, S. **Critical factors in software outsourcing: a pilot study.** Workshop on interdisciplinary Software Engineering Research. WISER '04. ACM Press, New York, NY, Seiten: 67-71.,2004.
- [PaLa04] Paasivaara, M., Lassenius, C.: **Using iterative and incremental processes in Global Software Development.** Third International Workshop on Global Software Development. ICSE 2004. Edinburgh, Scotland, 2004.
- [PiAP06] Pilatti, L., Audy, J. L., and Prikladnicki, R.: **Software configuration management over a global software development environment: lessons learned from a case study.** GSD 2006. ACM Press, New York, NY, Seiten: 45-50, 2006
- [Pets06] Petschik, R.: **Optimierung von verteilten Software Projekten – Ein Erfahrungsbericht.** Software & Systems Quality Conferences, 10.- 12. Mai 2006, Congress Center Düsseldorf. 2006.
- [PSTV97] Porter, A., Siy, H., Toman, C., Votta, L.: **An Experiment to Assess the Cost-Benefits of Code Inspections in Large Scale Software Development.** IEEE Transactions On Software Engineering, VOL. 23, NO. 6. Seiten: 329-346, 1997.
- [Raki97] Rakitin, S.: **Software Verification and Validation – A practitioner's Guide.** Artech House, 1997.
- [RaKK05] Ramasubbu, N., Krishnan, M. S. und Kompalli, P.: **Leveraging Global Resources: A Process Maturity Framework for Managing Distributed Development.** IEEE Software. 22, 3 (Mai 2005), Seiten: 80-86, 2005.
- [RiMa03] M. Richter, F. Maurer.: **MILOS and MASE: Past & Present,** 2003.
- [Royc70] Royce, Dr. W.: **Managing the development of large software systems.** IEEE WESCON, Seiten: 1-9, August 1970.
- [Sanc05] Sánchez-Moreno, R.: **Ein Vergleich der Fehler von Programmierpaaren und Einzelentwickler.** Studienarbeit, Fakultät für Informatik, Universität Karlsruhe, 2005.
- [Scha06a] Schaffer, E.: **Offshore usability: helping meet the global demand?** interactions 13, 2, Seiten: 12-13. 2006.
- [Scha06b] Schaffer, E.: **A decision table: offshore or not? (when not to use offshore resources).** interactions 13, 2 (Mar. 2006), Seiten: 32-33, 2006.

- [SeCS06] Sengupta, B., Chandra, S., and Sinha, V. **A research agenda for distributed software development**. International Conference on Software Engineering. ICSE '06. ACM Press, New York, NY, Seiten: 731-740, 2006.
- [SeMo04] Serrano, M., Montes de Oca, C.: **Using the Team Software Process in an Outsourcing Environment**. Crosstalk: The Journal of Defense Software Engineering, März 2004.
- [SiSC05] Sinha, V., Sengupta, B., and Chandra, S.: **EGRET: A Collaborative Tool for Distributed Requirements Management**. IBM Research Technical Report, RI06001, 2005.
- [SiSM06] Simon, F., Seng, O., Mohaupt, T.: **Code Quality Management**. 1. Auflage. Dpunkt Verlag, 2006.
- [Somm01] Sommerville, I.: **Software Engineering**. 6. Auflage, Pearson Studium, 2001.
- [Stan06] Stani, D.: **Werkzeuggestützte Reviews**. Bachelorarbeit im Studiengang Informatik. Universität Hannover. Hannover, Mai 2006.
- [Stru94] Strubing, J.: **Designing the Working Process: What Programmers Do Besides Programming**. User-Centered Requirements for Software Engineering Environments, Springer, Berlin, Germany, 1994.
- [TGSC06] Taylor, P., Greer, D., Sage, P., Coleman, G., McDaid, K., Keenan, F.: **Do Agile GSD Experience Reports Help the Practitioner?** GSD 2006, Shanghai, China. ACM Press, New York, 2006.
- [Thal00] Thaller, G.: **Software-Test: Verifikation und Validation**. Verlag Heinz Heise, Hannover, 2000.
- [TuFR02] Turk, D., France, R., Rumpe, B. **Limitations of Agile Software Processes**. International Conference on eXtreme Programming and Agile Processes in Software Engineering. XP 2002. May 2002.
- [VaVP06] van Bon, J., van der Veen, A., Pieper, M.: **Foundations in IT Service Management basierend auf ITIL®**. Van Haren Publishing, 2006.
- [Vers00] Versteegen, G.: **Projektmanagement mit dem Rational Unified Process**. Springer Verlag, Berlin, 2000.
- [ViGü05] Victor, F., Günther, H.: **Optimiertes IT-Management mit ITIL**. Friedr. Vieweg & Sohn Verlag, 2. Auflage, Wiesbaden, 2005.
- [Wein80] Weinberg, Gerald M.: **Adaptive Programming: The New Religion**. Australasian Computerworld. 1980.

- [Whee66] Wheeler, D.: **Software Inspection: An industry best practice.** Los Alamitos, 1966.
- [WiBi98] Wiederhold, G., Bilello, M.: **Protecting Inappropriate Release of Data from Realistic Databases.** International Workshop on Database and Expert Systems Applications. DEXA. IEEE Computer Society, 1998.
- [WiHN93] J. Wilson, N. Hoskin und J. Nosek.: **The benefits of collaboration for student programmers.** In SIGCSE technical symposium on Computer science education, Seiten: 160–164, 1993.
- [Wire06] Wiredu, G.: **A Framework for the Analysis of Coordination in Global Software Development.** GSD 2006, Shanghai, China. ACM, 2006.
- [WKCJ00] L. Williams, R. Kessler, W. Cunningham und R. Jeffries. **Strengthening the Case for Pair-Programming.** IEEE Software, Seiten: 19–25, Juli/August 2000.
- [Wong02] B. Wong.: **An Investigation of the Cognitive Structures used in Software Quality Evaluation.** PhD Thesis in Information Systems, Technology and Management. Sydney: University of New South Wales, 2002.
- [WuWZ03] Wu, X., Wang, Y., und Zheng, Y.: **Privacy preserving database application testing.** ACM Workshop on Privacy in the Electronic Society. WPES '03. ACM Press, New York, NY, Seiten: 118-128. 2003.