CARL VON OSSIETZKY universität OLDENBURG

Fakultät II – Informatik, Wirtschafts- und Rechtswissenschaften
Department für Informatik

Seminarausarbeitung

Softwareentwicklungs Paradigmen/Vorgehensmodelle

vorgelegt von

Mart Köhler

26. Oktober 2007

Inhalt

1 Einleitung

1.1 Motivation

Im Rahmen der Projektgruppe Modulares Enterprise Architecture Management System soll in einer Seminararbeit verschiedene Vorgehensmodelle vorgestellt werden und ein geeignetes Modell für die Projektgruppe vorschlagen. Dieser Vorschlag kann in einer Diskussion im Bezug auf die Eignung für das Projekt Modulares Enterprise Architecture Management System untersucht und angepasst werden.

1.2 Struktur

Im zweiten Kapitel, den Grundlagen, wird die Begrifflichkeit „Vorgehensmodell“ erklärt und einige Vertreter der klassischen und agilen Vorgehensmodelle näher erläutert. Im dritten Kapitel, der Diskussion, werden jeweils ein Vertreter der klassischen und „agilen“ Vorgehensmodelle gegenüber gestellt. Abschließend folgt ein Vorschlag für die Verwendung eines geeigneten Vorgehensmodells im Bezug zur Projektgruppe Modulares Enterprise Architecture Management System.

2 Grundlagen

2.1 Vorgehensmodell

Ein Vorgehensmodell, auch Prozessparadigma [Som01] genannt, ist eine abstrakte Beschreibung des Softwareentwicklungsprozesses aus einer bestimmten Perspektive. Es stellt dabei eine Sammlung von aufeinander abgestimmten Regeln mit zeitlichen und sachlichen Vorgaben dar, um die Durchführung eines Projektes zu unterstützen. Aufgaben, die das Vorgehensmodell definiert, sind beispielsweise Verantwortlichkeiten der qualifizierten Mitarbeiter festlegen, Durchführung der Definitionen der Anforderungen und das System und die durchzuführenden Methoden für Aktivitäten. Durch ein Vorgehensmodell wird der Entwicklungsstand überschaubar und lässt auf den Kosten- und Zeitaufwand, in Bezug auf weitere Planungen, schließen. Es werden dabei drei verschiedene Idealstandards unterschieden[Wir05].
Der sequentiell/phasenorientierte Ansatz wird zum Beispiel vom „Wasserfall“ - Modell praktiziert. Der zweite Standard ist iterativ. Als Beispiel führt der Rational Unified Process den Prozess iterativ durch. Der letzte Standard ist der leichtgewichtige Ansatz mit den „agilen“ Vorgehensmodellen wie das SCRUM dar. Durch Faktoren wie Projektumfang, Genauigkeit der Produktdefinition, Anzahl der Entwickler, Zeit, Budget werden Vorgehensmodelle entsprechend den Bedürfnissen angepasst[Wir05]. Ein Blick zurück in die 60er Jahre erklärt warum ein Vorgehensmodell nötig ist. In der Zeit begann die Komplexität von Softwaresystemen und die Anforderungen daran rapide zu zunehmen. Es gab aber keine geeigneten Programmiersprachen, Methoden und Werkzeuge, um Softwaresysteme klar zu strukturieren und überschaubar zu halten. Neue Mitarbeiter hatten eine lange Einarbeitungszeit und viele Schwierigkeiten die Zusammenhänge im Code zu begreifen. Dadurch stiegen die Gesamtkosten für die Softwareentwicklung so stark an, dass viele Projekte vorzeitig scheiterten. Dieser Mißstand machte ein wohldefiniertes Vorgehen wie in den klassischen Ingenieurswissenschaften notwendig. Vorgehensmodelle schafften es durch den geeigneten Einsatz von Notationen und Werkzeugen den Entwicklungsprozess transparenter zu machen. Planungen wurden effizienter und der Prozess dadurch einfacher. Fehler konnten schneller bereinigt werden und dadurch die Qualität der Software erheblich gesteigert werden, was den ganzen Entwicklungsprozess zudem beschleunigt und Geld gespart hat.[Kla02]

Generell muss festgelegt werden auf welche Art das Projekt realisiert werden soll. Die Entwicklung kann als mögliche Option für die Projektgruppe Prototypen getrieben, Test getrieben, durch objektorientierte Entwicklung oder durch Komponentenentwicklung realisiert werden. Mit einem horizontalen Prototypen wird eine komplette Schicht wie beispielsweise die Benutzerführung realisiert. Bei einem vertikalen Prototypen hingegen werden nur bestimmte Funktionalitäten realisiert. Eine testgetriebene Entwicklung versucht den Qualitätsstandard hoch zu halten. Vorab geschriebene automatisierte Tests garantieren, dass das System nach Spezifikation läuft. Bei der Komponentenentwicklung werden aus verschiedenen Quellen fertiger Code wiederverwendet. Dieser wird auf die eigenen Bedürfnisse angepasst und in das System integriert. Die objektorientierte Entwicklung modelliert das System anhand einer visuellen Methode wie der Unified Modelling Language. Code wird generiert und wird von den Entwicklern mit der nötigen Logik gefüllt. In der inkrementellen Entwicklung wird das System Stück für Stück weiterentwickelt. Diese Vorgehensarten beschreiben zwar teilweise eigenständige Vorgehensmodelle, ihre Funktion wird aber meist als Best Practices in anderen Vorgehensmodellen verwirklicht.

2.2 klassische Vorgehensmodelle

Klassische Vorgehensmodelle zeichnen sich dadurch aus, dass der Prozess im Vordergrund steht. Direkt zu Anfang wird der Prozess komplett und so detailliert wie möglich erfasst. Im späteren Verlauf ist dann eine Änderung des Prozesses mit seinen Anforderungen nicht mehr vorgesehen. Es werden dabei alle Aktivitäten, Planungen, Besprechungen, Probleme und sonstiges Vorgehen genau in Dokumenten festgehalten. Für große Projekte mit strengen Vorgaben sind die klassischen Vorgehensmodelle gut geeignet, weil durch die genaue Planung und den Dokumenten die Kosten und der Ressourcenverbrauch gut abgeschätzt werden kann. [Kla02]

2.2.1 „Wasserfall“ - Modell

Das „Wasserfall“ - Modell ist von Walker Royce 1970 eingeführt worden und stellt eine Weiterentwicklung von „stagewise development“, welches 1956 von Benington kommt [Wir05]. Der Softwareentwicklungsprozess wird in verschiedene Phasen eingeteilt und verläuft sequentiell, d.h. strikt nacheinander. Das bedeutet, dass jede einzelne Phase und Aktivität erst komplett abgeschlossen wird und danach die nächste Phase beginnt. Die Ergebnisse einer Phase werden schriftlich dokumentiert. Diese Dokumente dienen der leichteren Kommunikation zwischen den Entwicklern und werden als Grundlage für die folgenden Phasen verwendet. Der Prozess gliedert sich in Machbarkeitsstudie, Analyse und Definition der Anforderungen, System- und Softwareentwurf, Implementierung und Komponententest, Integration und Systemtests und Betrieb und Wartung [Som01].

Machbarkeitsstudie

In der Machbarkeitsstudie wird die genaue Problemstellung definiert und verschiedene Lösungsansätze erarbeitet. Ebenso werden die voraussichtlichen Kosten und der potentielle Gewinn und somit Nutzen durch die Softwareentwicklung und das resultierende Softwaresystem ermittelt.

Anforderungsdefinition

Die Entwickler erarbeiten gemeinsam mit dem Kunden an einer exakten Spezifikation des zu entwickelnden Softwaresystems. Die resultierende Anforderungsdefinition dient als Vertragsgrundlage.

System- und Softwareentwurf

Es wird festgelegt, wie das Softwaresystem realisiert werden soll. In einem Grobentwurf wird das Gesamtsystem in Teilsysteme zerlegt und so strukturiert, dass ein Grundgerüst entsteht. Noch zu entwickelnde oder bestehende Frameworks, Softwarebibliotheken etc. werden selektiert, sowie die Schnittstellen zwischen den Modulen und Algorithmen werden definiert.

Implementierung und Komponententest

Die Teilsysteme werden vom Entwicklungsteam implementiert und mit geeigneten Verfahren beziehungsweise mit Hilfe von Frameworks wie JUnit für Java getestet.

Integration und Systemtest

Die einzelnen Teilsysteme werden zu einem Gesamtsystem zusammengeführt und als Ganzes getestet. Am Ende dieser Phase ist das Softwaresystem inklusive Handbücher und weiteren Dokumenten fertiggestellt.

Betrieb und Wartung

In der letzten Phase wird das Softwaresystem beim Kunden installiert und in Betrieb genommen. Die Wartung beinhaltet die Bereinigung auftretender Fehler und wenn möglich eventuelle Erweiterugen am System.[Kla02]

In diesem Modell sind Rückschritte in den Phasen nicht vorgesehen. In der Praxis werden aber oft Fehler gefunden, die nur durch einen Sprung in vorangegangenen Phasen behoben werden können. Um diesen Vorgang kontrolliert halten zu können, ist es nur erlaubt immer nur auf die benachbarte Phase zurückzugehen. Zwischen Entwurf und Implementierung herrscht eine Wechselbeziehung, so dass der Entwicklungsprozess zwischen beiden Phasen hin und her springt. Da hohe Entwicklungskosten für die Herstellung und Abnahme der Dokumente entstehen, wird zudem die Anzahl an Wiederholungen der Phasen durch Rückschritte auf nur wenige Wiederholungen beschränkt. Anschließend wird die Arbeit eingefroren und an anderer Stelle fortgeführt. Durch das vorzeitige Einfrieren wird Zeit und Geld gespart, mit dem Risiko schwerwiegende Fehler nicht zu entdecken. Ein großer Vorteil ist, dass das Modell einfach und verständlich anzuwenden ist und aus Managementsicht wenig Aufwand benötigt. Durch die starre Struktur können mit der Zeit veränderte Anforderungen nicht beachtet werden. Je fortgeschritten der Entwicklungsprozess ist, desto teurer und risikoreicher wäre eine Änderung.[Som01] Weiterer Nachteil sind die nach hinten verlagerten Komponenten- und Systemtests. Es wird nicht vorab getestet, sondern erst wenn die Teile fertig sind beziehungsweise am Schluss bei der Integration. Dadurch könnte es zum „Big Bang“ kommen, d.h. das System weist nur Fehler auf und genügt nicht den Anforderungen. Dazu trägt auch das Einfrieren von Teilentwicklungen bei, die beispielsweise am Schluss erst ihre Fehler offenbaren. Abbildung 2.1 zeigt die einzelnen Phasen wie sie von der Machbarkeitsstudie bis zum Betrieb wie ein Wasserfall durchlaufen werden.

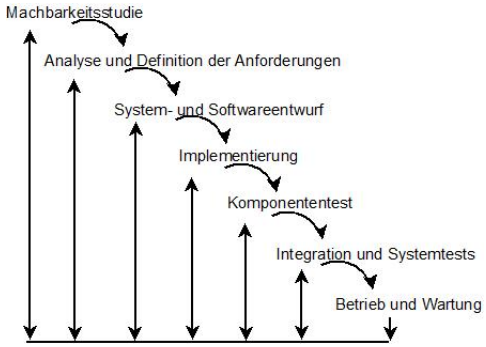

Abbildung 2.1: Das „Wasserfall“ - Modell mit möglichen Rückschritten

2.2.2 V - Modell

Das V - Modell bildet die Grundlage für ein umfangreiches Vorgehensmodell der IT-Infrastruktur der BRD seit 1991, das vom Bundesamt für Wehrtechnik und Beschaffung entwickelt und für Projekte eingesetzt wird. Es wurde von Barry Boehm 1984 eingeführt. Zur Zeit ist die aktuelle Version VM'97 und unterstützt auch schon inkrementelles, objektorientiertes und komponentenbasiertes Vorgehen. Zudem eignet sich das Modell sehr gut für Hardwareentwicklungen, da von Anfang an viel Wert auf die Qualitätssicherung gelegt werden muss. Das V - Modell stellt eine Erweiterung des „Wasserfall" - Modells dar. Es erweitert das bestehende Modell um eine explizite Qualitätssicherung. In der Anforderungsdefinition werden Anwendungsszenarien generiert und mit Akzeptanztests vom Kunden validiert. Bei dem Grobentwurf, Feinentwurf und der Modulimplementation werden Testfälle generiert und mit ihrer Hilfe automatisierte Verifikationstests erstellt und durchgeführt. Abbildung 2.2 zeigt den Zusammenhang im V - Modell. Durch das Qualitätsmanagement in jeder Phase wird sichergestellt, dass die Entwicklung zu jedem Zeitpunkt den Anforderungen genügt. Das V - Modell macht bezüglich der organisatorischen Umsetzung eines Projektes allerdings keine Aussagen.[Wir05]

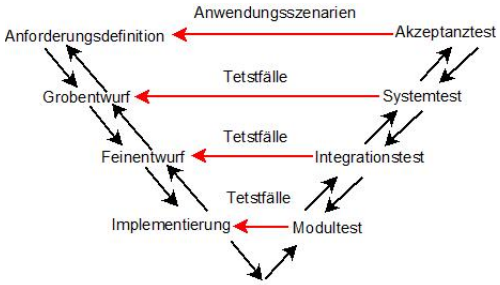

Abbildung 2.2: V -Modell

2.2.3 Rational Unified Process

Der Rational Unified Process mit dem ursprünglichen Namen Objectory wurde erstmals von Philippe Kruchten 1996 [Kru99][PP07] vorgestellt. Der Rational Unified Process ist ein kommerzielles Produkt der Firma Rational Software, die seit 2002 Teil des IBM Konzerns sind. Die Rational Tools decken dabei den Bedarf an Unterstützung auf dem Software Engineering Sektor überwiegend ab. So wird mit der Rational Suite ein Framework geliefert, das den Planungsprozess für ein Projekt unterstützt. Ein Nachteil besteht allerdings in der langen Einarbeitungszeit in die einzelnen Tools. Dabei besteht die Gefahr, den eigentlichen Entwicklungsprozess zu vernachlässigen. Die Vorgehensweise des Rational Unified Process wird mit iterativ, Use Case getrieben und architekturzentriert beschrieben[Spi07]. Die iterative Vorgehensweise wird unter den Best Practices beschrieben. Use Case getrieben beschreibt Szenarien und Anwendungsfälle, die mit Hilfe von Use Case Modellen modelliert werden. Diese Use Case Modelle dienen als Beschreibung des Softwaresystems. Use Cases werden beim Design, der Planung und in der Implementierung verwendet. In jeder Iteration wird eine Teilmenge der Use Cases gewählt und umgesetzt. Nach jeder Iteration entsteht ein Teil des Gesamtsystems, welches bereits lauffähig ist. Die Use Cases werden absteigend nach ihrer Wichtigkeit und Bedeutsamkeit für die Architektur ausgewählt und umgesetzt. Die Architektur des Softwaresystems steht dabei im Vordergrund und stellt das Grundgerüst bereit, in dem Systemelemente und die

Schnittstellen definiert werden.

Folgende sechs Best Practices werden im Rational Unified Process verwendet[Kru99][Spi07]:

Iterative Software Entwicklung

Jede einzelne Phase kann mehrfach wiederholt werden. Dabei werden alle Disziplinen je nach Phase in ihrer vollen Breite durchlaufen. Innerhalb jeder Iteration wird ein zur Gesamtplanung unabhängiger Plan erstellt. Am Ende jeder Iteration wird ein Review durchgeführt. Mit dem Review kann der derzeitige Entwicklungsstand eingeschätzt werden. Das Ergebnis einer Iteration dient als Grundlage für die nächste Iteration. Es steht zudem nach jeder Iteration ein lauffähiges System bereit, welches inkrementell aufgebaut wird. Die iterative Entwicklung stellt das Hauptmerkmal des Rational Unified Process dar und macht das Modell durch die Planung innerhalb der Iterationen flexibel [PP07].

Anforderungsmanagement

Damit das Entwicklungsteam flexibel auf Änderungen der Anforderungen an das Softwaresystem reagieren kann, werden an zentraler Stelle die Anforderungsänderungen dokumentiert und organisiert. So kann jeder zu jedem Zeitpunkt die Änderungen einsehen.

Verwendung komponentenbasierter Architekturen

Durch die Aufteilung des Gesamtsystems in Komponenten kann die Komplexität des Systems verringert werden. Das Ziel ist es die Komponenten unter Beachtung der Schnittstellen wiederzuverwenden und austauschen zu können.

Visuelle Software Modellierung

Die visuelle Modellierung wird mit Hilfe der Unified Modelling Language ermöglicht. Sie wird für Spezifikation, Dokumentation, Visualisierung verwendet. Aus der Visualisierung kann ein Code Grundgerüst generiert werden, welches die Objekte und ihre Zusammenhänge implementiert. Der Entwickler muss nur noch die Kernfunktionalität implementieren. Durch diese Art der Modellierung werden Anforderungen und das System selbst für Entwickler und Kunden verständlicher.

Prüfung der Softwarequalität

Nach Iterationen entsteht inkrementell immer mehr des Gesamtsystems. Das System kann stetig als Ganzes getestet werden, was die Qualität konstant auf hohem Niveau hält. Es wird dadurch auch der „Big Bang“ vermieden, der bei einer Teilentwicklung mit abschließender Integration entstehen kann.

kontrolliertes Änderungsmanagement

An einem großen Projekt sind meist viele Entwickler beteiligt, die viel Code neu schreiben und verändern. Damit bei Fehlern nachvollzogen werden kann von wem, wann und durch welches Feature ein Fehler aufgetreten ist, ist ein zentrales Konfigurationsmanagement nötig. Ebenso hilfreich ist es, wenn man weiß welches Feature durch welchen Entwickler realisiert wurde und man gezielt an diesem Entwickler Fragen richten kann. Durch ein Konfigurationsmanagement wird die Kontrollfunktion über das Prjekt verbessert und ein Integrations- und Buildmanagement ermöglicht.

Die Entwicklung im Rational Unified Process gliedert sich in vier große Phasen[Spi07][PP07]:

Inception

Im Einstieg, Inception, wird die Infrastruktur eingerichtet und die Geschäftsprozesse modelliert. Dabei wird schon grob ein Projektplan erstellt und die Disziplinen, siehe Abbildung 2.3 auf Seite 8, initialisiert. Hier helfen erste Use Cases den Zusammenhang zu erklären. Zudem werden Kosten und Nutzen abgeschätzt und daraus eine Risikoabschätzung erarbeitet und abhängig davon das Projektziel festgelegt. Abgeschlossen wird die Phase mit einem Meilenstein, dem „Lifecycle Objective Milestone.

Elaboration

Während der Ausarbeitung, Elaboration, wird die Planung und Spezifizierung des Projektes und der Anforderungen detailliert vorgenommen. Ein Großteil der Use Cases wird erarbeitet und enger Kundenkontakt gepflegt. In mehreren Iterationen wird die Anforderungsdefinition immer weiter verfeinert und die Architektur des Projektes festgelegt. Den Abschluss bildet der Meilenstein „Lifecycle Architecture Milestone“.

Construction

Diese Phase befasst sich mit der Implementierung und dem Testen, der zuvor festgelegten Architektur und Komponenten. Sich ergebende Änderungen an der Anforderung werden geprüft und nach Beschluss integriert. Am Ende ist das Produkt und die Dokumentation fertig. Die Phase schließt mit dem „Initial Operational Capability Milestone“ ab.

Transition

In der letzten Phase wird, das Produkt bei dem Kunden installiert beziehungsweise in die Infrastruktur integriert und die Anwender geschult. Abschließend werden Fehler, die während der Entwicklung aufgetreten sind analysiert und die gelernten Erfahrungen daraus für folgende Projekte genutzt. Die Transition endet mit dem „Product Release Milestone“.

Die zuvor erwähnten Disziplinen sind Business Modelling, Requirements, Analysis und Design, Implementation, Deployment, Configuration Managment, Project Managment und Environment. Sie haben in den einzelnen Iterationen der einzelnen Phasen unterschiedliche Ausprägungen, werden aber in jeder Phase voll eingebracht. Ausarbeitungen wie Programmteile, Use Cases oder Dokumente werden Artefakte genannt und dem jeweiligen Worker zugeordnet. Ein Worker repräsentiert dabei die Rolle eines Mitarbeiters. Er hat innerhalb des Rational Unified Process festgelegte Aufgaben zu erledigen und die dazugehörigen Artefakte zu erstellen. Nach Vollendung einer Phase wird ein Management Review durchgeführt. Hier wird geprüft, ob die Pläne und Standards eingehalten wurden. Bei Fehlern werden Maßnahmen zur Behebung ergriffen und nach Beendigung kann zur nächsten Phase fortgeschritten werden.

Nachteile des Rational Unified Process

Durch die Pflicht den Prozessablauf ständig zu beobachten, bewerten und anzupassen entsteht ein hoher Managementaufwand. Die Komplexität des Modells ist nicht nur ein Vorteil sondern auch ein

Nachteil. Denn es entsteht dadurch ein hoher Einarbeitungsaufwand. Das korrekte Anpassen an die eigenen Bedürfnisse ist sehr zeitintensiv.[Spi07] Ein weiterer Nachteil ist, dass Rational ein Copyright auf den Prozess hat und die Anwendung des RUP über die Knowledge Base Lizenzgebühren kostet.

Vorteile des Rational Unified Process

Da der Rational Unified Process sehr komplex ist, lässt er sich an die Bedürfnisse des Projektes anpassen, was ein Vorteil sein kann. Der Rational Unified Process ist durch seine Werkzeuge wie der Unified Modelling Language Ideal für objektorientierte/komponentenbasierte Softwareentwicklung. Durch die Einteilung in Disziplinen und Einordnung in die Phasen werden parallele Aktivitäten unterstützt. Das Vorgehensmodell wird immer weiter entwickelt und bekommt ständigen Einfluss durch neue Vorgehensweisen.[Spi07] Ein besonderer Vorteil zu den erwähnten Nachteilen ist, dass es eine Open Source Version angelehnt an den original Prozess gibt. Der Name ist Open Unified Process und ist Bestandteil des Eclipse Process Framework.

Einordnung des Rational Unified Process

An dieser Stelle wird nur kurz erklärt, warum der Rational Unified Process als klassisches Vorgehensmodell eingeordnet wird. Der Rational Unified Process scheint zwar durch seine Praktiken agil zu sein, aber bei genauerer Beobachtung der Struktur, siehe Abbildung 2.3, fällt auf, dass er dem Wasserfall ziemlich ähnlich ist und die Flexibilität vortäuscht. Genau wie das „Wasserfall“ - Modell werden Phaseneinteilung durchgeführt. Zudem sieht man, dass Anforderungssammlung in die Inception Phase fällt, Analyse und Entwurf in die Elaboration Phase fällt und die Implementierung überwiegend in die Construction Phase fällt. Deployment stellt dabei die Installation und Wartung dar und fällt in die Transition Phase. Außerdem gibt es zwar Iterationen, aber diese werden an die Phasen gebunden, als vielmehr an den entsprechenden Entwicklungsprozess oder derzeitigen Baustein. Es soll aber nicht diskutiert werden, ob der Rational Unified Process klassisch oder agil sein soll, dazu gibt es viele verschiedene Meinungen. Es dient als Begründung, warum das Modell unter den klassischen Vorgehensmodellen eingeordnet wird.[Hes00]

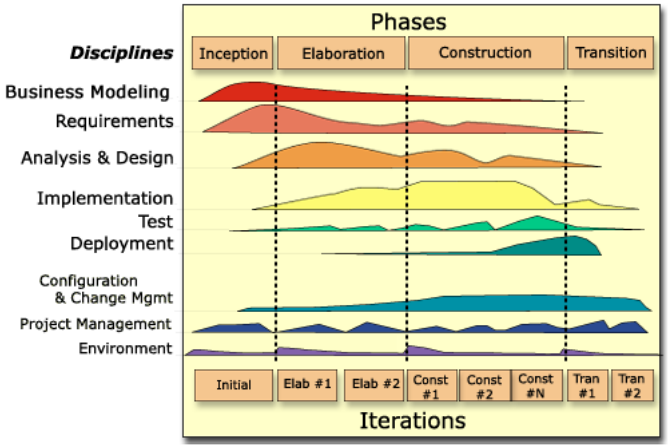

Abbildung 2.3: Überblick des Rational Unified Process, Quelle: www.crispico.com

2.3 „agile“ Vorgehensmodelle

Das Wort „agil“ kommt aus dem lateinischen und bedeutet soviel wie flink, wendig, beweglich. Für die Softwareentwicklung bedeutet „agil“ leichtgewichtig. Agile Vorgehensmodelle bilden den Gegensatz zu den schwergewichtigen „klassischen“ Vorgehensmodellen. Wesentliche Eigenschaften, die die „agilen“ Vorgehensmodelle auszeichnet, sind sehr kurze Entwicklungszyklen. Der Kunde wird stark in den Entwicklungsprozess einbezogen. „Agile“ Vorgehensmodelle sind sehr flexibel und erlauben dem Kunden zu jedem Zeitpunkt seine Ideen einzubringen und dadurch die Anforderungen zu verändern. Der Mensch steht dabei im Vordergrund, nicht wie bei den klassischen Modellen der Prozess. Der ganze Entwicklungsprozess steht regelmäßig zur Diskussion. Es gibt dabei verschiedene Wertvorstellungen, die bei „agilen Vorgehensmodellen“ im Vordergrund stehen. Es wird weniger Wert auf ausführliche Dokumente gelegt, wie es beim Formalismus bei klassischen Vorgehensmodellen der Fall ist. Dafür stehen früh lauffähige Softwaresysteme im Vordergrund. Strikt nach Plan zu arbeiten steht hinter flexibles Einstellen auf Änderungen in den Anforderungen. Eine enge Zusammenarbeit direkt mit dem Kunden ist wichtiger, als feste Verträge zu schließen. Die Konzentration auf die Entwickler mit ihren Fähigkeiten und die Kommunikation zwischen ihnen sind dabei höher zu gewichten als die Konzentration auf strukturierte Prozesse und Werkzeuge.[BF03] Diese Grundsätze wurden 2001 im Manifesto for Agile Software Development der Agile Alliance unter anderem von Kent Beck, Alistair Cockburn, Ward Cunningham und Martin Fowler formuliert. „Agile“ Vorgehensmodelle helfen dabei diese Werte umzusetzen.

2.3.1 SCRUM

Als leichtgewichtiger Managementprozess ist SCRUM in den neunziger Jahren entstanden. 2001 haben Ken Schwaber und Mike Beedle dieses Vorgehensmodell in ihrem Buch „Agile Software Development with Scrum“ erstmals vorgestellt. [Bög03] Abbildung 2.4 auf Seite 10 zeigt wie der Ablauf von Scrum aussieht. Der Product Backlog ist wie ein Katalog, in dem alle Anforderungen an das zu entwickelnde Softwaresystem nach ihrer Priorität aufgelistet sind. Alle Anforderungen werden vom Scrum Team, dem Scrum Master und dem Product Owner erarbeitet und im Rahmen der Release Planung im Product Backlog erfasst und nach ihrer Priorität sortiert. Der Product Owner ist dabei der alleinige Verwalter des Product Backlogs. Er vertritt dabei das Kundeninteresse. Jeder Beteiligte hat Einblick in das Product Backlog, welches aber nur durch den Product Owner verändert werden darf. Er schätzt zusammen mit dem SCRUM Team den Aufwand für jede Anforderung ab und priorisiert sie nach Wichtigkeit. Der Scrum Master koordiniert die Aufgaben der einzelnen Teammitglieder und überwacht den Arbeitsablauf. Er fungiert als Projektleiter und ist für eine optimale Umsetzung des Projektes verantwortlich. Iterationen werden Sprints genannt und dauern 30 Tage. Dazu wird eine Teilmenge aus dem Product Backlog selektiert und innerhalb eines Sprints abgearbeitet. Diese Teilmenge wird Sprint Backlog genannt. Die Aufgaben im Sprint Backlog werden dann je nach Funktionalität in Backlog Items aufgeteilt und vom SCRUM Master an das SCRUM Team verteilt. Anschließend wird im Sprint Review Meeting das Ergebnis des Sprints präsentiert, analysiert und diskutiert. Das Sprint Review Meeting dient ebenfalls dazu dem Kunden fertige Teile zu präsentieren und mit ihm zusammen das Ergebnis zu diskutieren. Innerhalb eines Sprints bleiben die Anforderungen fix.
Man hat für 30 Tage eine Stabilität der Anforderungen und dadurch einen fixen Plan wie während des gesamten Ablaufs im „Wasserfall“ - Modell. Die Möglichkeit der Anforderungsänderung ist zwischen den Sprints gegeben. Dabei kann das Ergebnis eines Sprints ebenfalls dem Product Backlog

zugeführt werden. So wird nach jedem Sprint ein Teil der Anforderungen in Software umgewandelt. Täglich wird ein kleines 15 minütiges SCRUM Meeting abgehalten, um die anderen Etwickler auf dem laufenden zu halten, was bereits fertig ist und was noch gemacht werden muss.[Sch07]

Aufgrund des Schwerpunktes auf der Management Ebene, kann SCRUM gut mit Vorgehensmodellen kombiniert werden, die sehr praktikenorientiert sind wie eXtreme Programming. SCRUM stellt praktisch nur ein Grundgerüst bereit, das keine konkreten Praktiken bereit stellt.

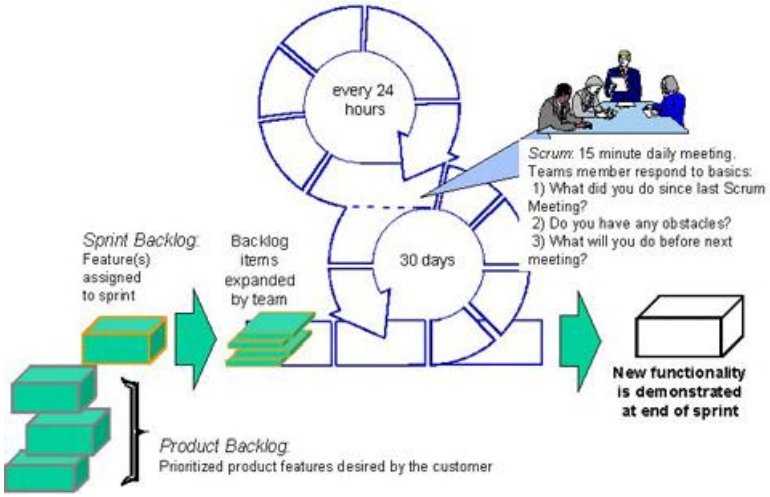

Abbildung 2.4: Verlauf von SCRUM, Quelle: http://www.controlchaos.com

2.3.2 Crystal Methodenfamilie

Crystal ist eine Methodenfamilie, die je nach Teamgröße und Risikoausmaß unterschiedliche Methoden für den Entwicklungsprozess bereit stellt. Sie wurde von Alistair Cockburn, eines der Mitglieder der Agile Alliance, entwickelt. Die anzuwendende Vorgehensweise ist von der Größe des Entwicklungsteams und dem Risiko des zu entwickelnden Systems abhängig. Bei steigender Mitarbeiterzahl und steigendem Risiko verschärfen sich die Regeln, die Anzahl der Pflichtdokumente wird erhöht und mehr Rollen weden definiert. Die Vorgehensmodelle sind Crystal Clear für bis zu 6 Mitarbeiter, Crystal Yellow bis zu 20 Mitarbeiter, Crystal Orange bis zu 40 Mitarbeiter. Die Entwicklung findet in räumlicher Nähe statt. Das Vorgehen ist inkrementell und es wird nach jedem Release, welches alle 1-4 Monate raus gebracht wird, in mehreren Reviews von den Nutzern abgenommen. Festgelegte Meilensteine beinhalten die lauffähige Software ohne Dokumentation. Das Modell kann mit den Best Practices von anderen Vorgehensmodellen kombiniert werden. Automatisierte Tests für die Qualitätssicherung sind fester Bestandteil. Das Vorgehen des Modells wird in Verbesserungs-Workshops am Anfang und Ende eines Inkrements angepasst. Da keine speziellen Angaben über die Arbeitsschritte gemacht werden, müssen die Entwickler bereits sehr gut ausgebildet und erfahren sein, um sich selbst organisieren zu können. Dieses Vorgehen eignet sich daher nicht für Entwicklungsteams, bei denen die Erfahrungsstufen zu unterschiedlich sind. Crystal Clear fordert zudem kurze Anforderungs- und

Designdokumentation, welche bei eXtreme Programming beispielsweise nicht gefordert sind.[Hol03]

2.3.3 eXtreme Programming

eXtreme Programming entstand bei Daimler-Chrysler im Rahmen des Projektes „Chrysler Consolidated Compensation“ in den 90er Jahren. Es sollte ein Abrechnungssystem entwickelt werden, für das Cunningham, Beck und Jeffries Methoden entwickelt haben, um das Projekt optimal und erfolgreich abzuschließen. Sie werden als eXtreme Programming Methodology bezeichnet.[Spi07][Bög03]

eXtreme Programming ist eine Zusammensetzung aus Prinzipien und Praktiken, die auf Werte und Variablen basieren. Der Entwicklungsprozess erfolgt dabei in Teams mit bis zu 15 Mitgliedern und wird iterativ durchgeführt. Die Iterationen dauern 1-3 Wochen[Wel99]. So entsteht das Produkt inkrementell und kann durch ständigen zeitnahen Feedback durch den Kunden angepasst werden.

Die vier Variablen sind Qualität, Zeit, Umfang und Kosten[Eic05]. Der Kunde hat die Freiheit drei dieser vier Variablen festzulegen. Das Entwicklungsteam kann entsprechend die vierte Variable in Abhängigkeit der anderen festlegen. Diese Methode ist nötig, da zu eingeschränkte Variablen den Handlungsspielraum einengen und somit der Kunde dann bei vier von ihm festgelegten Variablen nicht zufrieden gestellt werden kann. Als Beispiel braucht ein Projekt eines gewissen Ausmaßes entsprechende Gelder. Daher darf das Budget nicht zu knapp bemessen werden, weil es sich auf die Qualität und Umfang des Projektes negativ auswirken kann. Durch mehr Zeit kann der Umfang erweitert werden und mehr Zeit in Tests gesteckt werden und somit die Qualität gesteigert werden. Diese Variable wird oft vom Kunden in Form einer Deadline bereits vorgegeben. Durch einen kleinen Umfang des Projektes kann zwar die Qualität gesteigert werden, aber höchst wahrscheinlich die Anforderungen des Kunden nicht ausreichend abdecken. Ein großer Umfang bedeutet höhere Kosten und durch hohe Qualitätsanforderugen zusätzliche Kosten. Sparmaßnahmen an der Qualität können den Zeitdruck und den Umfang kurzfristig klein halten. Auf lange Sicht können Fehler dadurch auftauchen, die aufwändig bereinigt werden müssen. Die vier Werte sind als Kriterien anzusehen, um Lösungen für Probleme finden zu können und eine Richtung bestimmen zu können. Die vier Werte sind[Eic05]:

Kommunikation

Ohne Kommunikation kann ein Softwareprojekt nicht erfolgreich durchgeführt werden. Die Entwickler müssen untereinander stetigen Informationsaustausch pflegen, damit der ganze Prozess organisiert zufrieden stellend organisiert werden kann. Kommunikation mit dem Kunden steht an erster Stelle, um Anforderungen und Anforderungsänderungen direkt umsetzen zu können. Die Kommunikation sollte beispielsweise in kleinen Meetings zeitnah und direkt geschehen, um eine Verfälschung des Informationsflusses zu vermeiden.

Einfachheit

Einfachheit bedeutet, dass nur das Nötigste zur Erfüllung der Anforderung implementiert werden soll. Dadurch reduziert sich die Codelänge und bleibt dadurch einfach zu warten und überschaubar. Planungen, die die Zukunft betreffen, werden zum aktuellen Zeitpunkt nicht berücksichtigt, sondern erst, wenn es benötigt wird. Die ständige Kommunikation trägt zur Einfachheit bei.

Feedback

Feedback ist sehr wichtig, weil sie über den derzeitigen Stand der Entwicklung Auskunft gibt. So erhält der Entwickler durch Unit Tests ein Feedback über die Fehler eines Systems. Der Austausch mit dem Kunden gibt sowohl dem Entwickler durch die Akzeptanz des Kunden, als auch dem Kunden ein Feedback über den Entwicklungsfortschritt. Durch das Feedback bekommt der Kunde mit der Zeit außerdem ein Gefühl dafür, wie er den größten Nutzen aus der Software ziehen kann. Dazu helfen kurze regelmäßige Feedback Zyklen.

Mut

Der Entwickler soll den Mut haben bereits geschriebenen Code verwerfen zu können und von Neuem zu beginnen. Dies kann geschehen, wenn der Code zu komplex geworden ist und es eine einfachere Lösung gibt ein Feature zu realisieren. Die automatisierten Unit Tests geben dann das nötige Feedback, über die Einhaltung der Spezifikation.

Aus den Werten und den Variablen resultieren die Prinzipien, die Handlungsweisen empfehlen und die Werte und Variablen reflektieren. Prinzipien sind keine umsetzbaren Verfahren, sondern Grundsätze, die auf die Praktiken angewendet werden. Zu den wichtigsten Prinzipien zählen das unmittelbare Feedback, welches über verschiedene Zeitabstände zwischen Entwickler und Partner, automatischen Tests, Integration und Kunden stattfindet. Die Annahme der Einfachheit zielt darauf ab minimalen Code zu schreiben, der für die jetzigen Anforderungen ausreicht. Durch die inkrementelle Entwicklung soll das Produkt in kleinen Teilen stetig zu einem ganzen System anwachsen. Änderungen sollen begrüßt werden, weil dadurch mehr Optionen offen gehalten werden. Qualitative Arbeit ist wichtig für die Motivation, weil die Qualität den Erfolg beeinflußt. Weitere Prinzipien werden an dieser Stelle nicht angeführt, da sie zwar nicht unwichtig sind, aber eine weniger Zentrale Rolle spielen.[Mey03]

Als letztes werden die Praktikten angeführt, die auf bewährte Standards und Vorgehensmodellen basieren und die Prinzipien ausführen. Diese Praktiken, die sich zu einem Prozess zusammensetzen, werden Best Practices genannt und folgend erläutert [Wel99][Mey03]:

The Planning Game

Der Kunde offeriert seine Vorstellungen und die Entwickler schätzen den Aufwand dafür ein.Dann entscheiden der Kunde und die Entwickler gemeinsam, welche Aspekte in den nächsten Release kommen und wie groß der Umfang sein soll.

Small Releases

Jeder Release soll so klein und schnell wie möglich sein. Da der Nutzen im Vordergrund steht, können die kurzen Feedbackzyklen realisiert werden und frühzeitig bei Fehlplanungen Maßnahmen ergriffen werden.

Metapher

Damit Kunden und Entwickler eine gemeinsame Kommuniktaionsbasis haben, wird die Systemarchitektur von einem einfachen abstrakten Modell dargestellt. Die Systemarchitektur aus der klassischen Softwareentwicklung wird von der Metapher ersetzt.

Einfaches Design

Es wird nur das implementiert, was auch wirklich benötigt wird. So bleibt der Code schlank und übersichtlich und dadurch leicht anpassbar.

Test-getriebene Entwicklung

Vor dem eigentlichen Code werden Tests geschrieben. Dann wird der zu testende Code geschrieben und solange korrigiert bis der Test bestanden ist. Es werden nach eigenem Empfinden sinnvolle Tests geschrieben und automatisiert aufgerufen. Erst wenn alle Tests bestanden sind, wird ein neuer Release freigegeben. Die Entwickler stellen Komponententests her und die Kunden führen Akzeptanztests durch. Eine genügende Sammlung beider Testsorten und einem anschließenden Refactoring steigert die Chance auf einfachen, effizienten und korrekten Code.

Continual Integration

In klassischen Ansätzen werden einzelne Module entwickelt und in einer späteren Phase zusammengesetzt und als Ganzes getestet. Bei Continual Integration hingegen werden kleine Teile Code schon direkt in das Gesamtgefüge integriert und korrigiert, bis die automatisierten Tests erfolgreich sind. So braucht man am Ende keine Angst vor dem „Big Bang“ haben.

Pair Programming

Pair Programming beschreibt die Entwicklung mit zwei Personen. Es sitzen zwei Entwickler an einem Rechner. Während der eine programmiert, sitzt der andere daneben und kontrolliert den Ablauf. Er kann Fehler entdecken, die ein Einzelner nicht entdecken würde. Außerdem kann er sich viel mehr auf den Gesamtkontext konzentrieren, in der die derzeitige Entwicklung situiert ist. Dabei konzentriert sich derjenige, der programmiert, auf den derzeitigen Code. Die Softwarequalität steigt dadurch, dass eine Qualitätsinstanz hinzugefügt wurde. Gesteigerte Softwarequalität macht sich später durch Einsparungen in der Entwicklung und Motivationsgewinn der Mitarbeiter bemerkbar.

Collective Code Ownership

Collective Code Ownership besagt, dass der Quellcode jedem gleich zugänglich ist und von jedem verändert werden darf. Da jeder eine andere Sicht zu dem Entwicklungsprozess hat, können versteckte Fehler aufgedeckt werden. Zudem lastet dann die Verantwortung spezieller Codeteile nicht auf Einzelne, sondern verteilt sich auf alle. Es verringert sich auch das Risiko des Wissensabgangs, falls ein Mitglieder das Team verlassen müssen.

Refactoring

Beim Refactoring wird der Code ständig verbessert und durch einfacheren Code mit gleicher Funktionalität ersetzt. Spätere Suchen nach Fehlern werden vereinfacht, weil der Code nachvollziehbar wird.

Sustainable Pace

Der Arbeitsaufwand wird konstant auf gleichem Level gehalten. Es soll keine Stoßzeiten geben, an denen dauerhaft Überstunden gemacht werden müssen. Die 40 Stunden Woche dient dabei als Richtwert. Darüber hinaus sollte nicht gearbeitet werden, damit die Mitarbeiter motiviert bleiben das Projekt eifrig voranzutreiben. Bei auftretenden Überstunden werden diese durch Freizeit ausgeglichen.

On Site Customer

Der Kunde stellt einen seiner Mitarbeiter als Kundenvertreter dem Entwicklungsteam bereit. Dieser arbeitet über die gesamte Projektdauer mit dem Entwicklungsteam zusammen und steht dem Team für Fragen bezüglich der Anforderungen zur Verfügung. Mit ihm zusammen werden Testfälle erarbeitet, die dann später implementiert und automatisiert werden. Der Kundenvertreter hat dabei volle Entscheidungsvollmacht und kann das Projekt in die ihm gewünschte Richtung lenken.

Shared Coding Standards

Durch Einsatz von Standards wird effizientes Entwickeln im Team gefördert. Alle Entwickler sollen sich an einmal festgelegte Konventionen halten, damit der Code für die Entwickler selbst verständlich bleibt und neue Entwickler sich leichter einarbeiten können. Bei Verwendung von passender Design Patterns muss zudem keine Lösung für Probleme mehr gefunden werden.

Stand-Up Meeting

In einem kurzen täglichen Meeting, welches nur ein paar Minuten dauert, wird den anderen Teammitgliedern berichtet, was am Vortag erledigt wurde, wo es dabei Schwierigkeiten gab und was als nächstes Ziel in Angriff genommen wird.

Durch die Wahl entsprechender Praktiken, ist es möglich das Modell individuell an die eigenen Bedürfnisse anzupassen. In einem Projekt, das eXtreme Programming verwendet, sollten die Rollen Kunde, Programmierer, XP Coach, Terminmanager, Projektmanager und Tester vergeben sein. Der XP Coach hat dabei die Aufgabe den Prozessablauf im Auge zu behalten. Er dient der Einhaltung der XP Prinzipien und Werte.

3 Diskussion

3.1 „Wasserfall“ - Modell vs eXtreme Programming

Mit der Gegenüberstellung des „Wasserfall“ Modells mit dem eXtreme Programming soll nicht nur zwischen den Modellen verglichen werden, sondern auch der Unterschied zwischen „klassischen“ und agilen Vorgehensmodellen. Klassische Modelle, wie das „Wasserfall“ - Modell richten die komplette Planung nach dem Prozess, der in Phasen unterteilt und sequentiell abgearbeitet wird. Die Anforderungsdefinition wird direkt zu Anfang möglichst komplett und detailliert verfasst. Sie dient als Vertragsgrundlage und ist aus betriebswirtschaftlicher Sicht optimal. Durch mit der Zeit ändernde Anforderungen führt es zu nicht zufrieden stellenden Produkten. EXtreme Programming löst das Problem mit ändernden Anforderungen durch enge Zusammenarbeit mit dem Kunden. Es muss aber

ein anderes Maß für einen Vertrag zwischen Kunden und Entwicklern gefunden werden. Das ´“Wasserfall“ - Modell liefert aufgrund seiner Struktur erst zur letzten Phase hin ein lauffähiges System mit nachgelagerten Tests, was zum „Big Bang“ führen kann. EXtreme Programming wirkt dem mit seiner iterativen Entwicklung entgegen, in der das gesamte System dauerhaft getestet wird. So kann dann durch Feedback vom Kunden nah an den Anforderungen gearbeitet werden. Da kein fester Plan der Struktur besteht, ist die Abschätzung über Ressourcen- und Zeitaufwand schwer und ist daher nur für kleinere Projekte empfehlenswert. Bei großen klar definierten und strukturierten Projekten lässt sich hingegen beim „Wasserfall“ - Modell eine genauere Prognose über Kosten und Zeit aufstellen, solange keine notwendigen Änderungen gemacht werden müssen. Mit fortschreitender Entwicklungsdauer erhöhen sich die Kosten für Änderungen rapide. Bei der iterativen Entwicklungsmethode und der Möglichkeit Änderungen einfließen zu lassen, werden die Kosten für Änderungen und Planunsaufwand auf einem konstanten Niveau gehalten. Dieser ist beim „Wasserfall“ - Modell zu Anfang sehr groß, zu dem der bürokratische Aufwand beiträgt. eXtreme Programming sieht es vor nur da Dokumente anzulegen, die auch als wichtig erachtet werden.

3.2 Anwendungsvorschlag

Als passendes Vorgehensmodell würde sich der Rational Unified Process in der Open Source Fassung aus dem Eclipse Process Framework anbieten. Es ist ein klassisches Modell, was durch seine Methoden und durch seine Anpassbarkeit versucht eine Brücke zwischen klassischen und agilen Vorgehensmodellen zu schlagen. Das Vorgehen sollte ein inkrementell aufgebauter Prototyp sein, da durch die Vorgabe von Iterationen früh lauffähige Software vorhanden ist. Dieses Vorgehen kann mit der Test getriebenen Entwicklung, Komponentenentwicklung und Objektorientierung kombiniert werden. So würde das System stückweise wenn möglich durch fertige Komponenten und eigenen Entwicklungen erweitert werden können und modular aufgebaut werden können. Um die korrekte Lauffähigkeit zu garantieren, werden automatisierte Testroutinen entwickelt. Die objektorientierte Entwicklung soll dabei das Verständnis für das System schulen. Der Aufwand des Rational Unified Process lässt sich gut planen und bietet zudem die nötige Flexibilität. Da eine genaue Definition über den Funktionsumfang noch nicht fest steht, könnte sie sich je nach Aufwand im Laufe der Zeit ändern. Das Modell kann mit Methoden aus eXtreme Programming, welches in der Eclipse Process Framework neben SCRUM ebenfalls unterstützt wird, erweitert werden. Dazu bieten sich viele Best Practices wie das Pair Programming an. Die sofortige Integration wird durch das Vorgehensmodell und der Vorgehensweise direkt umgesetzt. Die Metapher wird durch die Unified Modelling Language unterstützt und ist ebenfalls Bestandteil des Modells. Shared Coding Standards gehören zudem zu den gestellten Anforderungen der Projektgruppe und sind fester Bestandteil. Zusätzlich wäre auch der Einsatz von SCRUM in der Construction Phase denkbar, um den Ablauf der Iterationen genauer zu Spezifizieren. Andere klassische Vorgehensmodelle sind ungeeignet, da sie zu unflexibel sind. Die Nutzung von agilen Methoden bringen im Rational Unified Process mehr Flexibilität in das Vorgehen, in dem Best Practices aus dem eXtreme Programming und ein flexibles Grundgerüst für Iterationen aus SCRUM genutzt werden. Es soll hierbei aber nur ein Vorschlag sein und sollte in der Projektgruppe diskutiert werden.

Literatur

[BF03] BLEEK, Wolf-Gideon ; FLOYD, Christiane: *Softwaretechnik und Software-Ergonomie*. Universität Hamburg, 2003 http://www.informatik.uni-hamburg.de/SWT/attachments/LVTermine/STE-VL16-WS-03-04.pdf. – zuletzt besucht am 07.10.2007

[Bög03] BÖGLI, Alex: *Die Rolle der Anforderungen in agilen Methoden*. Universität Zürich, 2003 http://www.ifi.unizh.ch/groups/req/courses/seminar_ws03/12_Boegli_Anforderungen_Ausarbeitung.pdf. – zuletzt besucht am 07.10.2007

[Eic05] EICKER, Prof. Dr. S.: *Paradigmen und Konzepte der Softwareentwicklung 1*. Universität Duisburg, 2005 http://www.softec.wiwi.uni-due.de/studium-lehre/lehrveranstaltungen/wintersemester-05-06/pks-1-93/. – zuletzt besucht am 12.10.2007

[Hes00] HESSE, Prof. Dr. W.: *Software-Projektmanagement braucht klare Strukturen - Kritische Anmerkungen zum Rational Unified Process*. Universität Marburg, 2000 http://www.mathematik.uni-marburg.de/~hesse/papers/Hes_00a.pdf. – zuletzt besucht am 15.10.2007

[Hol03] HOLLENSTEIN, Silvan: *The Crystal Family*. Universität Zürich, 2003 http://www.ifi.unizh.ch/groups/req/courses/seminar_ws03/04_Hollenstein_Crystal_Folien.pdf. – zuletzt besucht am 24.10.2007

[Kla02] KLARNER, Martin: *Software Engineering*. Universität Erlangen, 2002 http://www8.informatik.uni-erlangen.de/IMMD8/Lectures/WEB/vorlesung/sources/27112002/Vorlesung7.pdf. – zuletzt besucht am 07.10.2007

[Kru99] KRUCHTEN, Philippe: *The Rational Unified Process*. Addison-Wesley, 1999

[Mey03] MEYER, Manuel: *Extreme Programming*. Universität Zürich, 2003 http://www.ifi.unizh.ch/groups/req/courses/seminar_ws03/02_Meyer_XP_Ausarbeitung.pdf. – zuletzt besucht am 15.10.2007

[PP07] PANKRATH, Andrea ; PAKOZDI, Pascal: *Das Eclipse Process Framework (EPF) zur Modellierung von Prozessmodellen*. Universität Marburg, 2007 http://www.informatik.uni-hamburg.de/SWT/attachments/LVTermine/epf_Final.pdf. – zuletzt besucht am 24.10.2007

[Sch07] SCHERER, Josef: *Der Scrum Prozess im Überblick*. 2007 http://scrum-fibel.de/. – zuletzt besucht am 07.10.2007

[Som01] SOMMERVILLE, Ian: *Software Engineering*. Pearson Studium, 2001 (6. Auflage)

[Spi07] SPIEGL, Klaus: *Projektmanagement life - Best Practices und Significant Events im Software Projektmanagement*. Technische Universität Wien, 2007 http://www.pri.univie.ac.at/Publications/2007/SPIEGL_BPSWPM.pdf. – zuletzt besucht am 17.10.2007

[Wel99] WELLS, Don: *Extreme Programming: A gentle introduction*. 1999 http://www.extremeprogramming.org. – zuletzt besucht am 18.10.2007

[Wir05] WIRSING, Martin: *Methoden des Software Engineering*. Universität München, 2005 http://www.pst.informatik.uni-muenchen.de/lehre/WS0405/mse/folien/E1-Vorgehensmodelle6p.pdf. – zuletzt besucht am 07.10.2007