Seminar Software Engineering WS 2003/2004
Agile vs. klassische Methoden der Software-Entwicklung

Thema:

Inwiefern sind klassische Methoden als agil anzusehen? Diskussion am Beispiel des Rational Unified Process (RUP)

Verfasser:

Peter Egli
Wülflingerstrasse 85
CH-8400 Winterthur
peteregli@datacomm.ch
+41 (0)79 371 44 60

Januar 2004

Inhaltsverzeichnis

zur Seminar-Arbeit *„Inwiefern sind klassische Methoden als agil anzusehen? Diskussion am Beispiel des Rational Unified Processes (RUP)“.*

1. Einleitung

Das vorliegende Papier möchte einen Vergleich ziehen zwischen Methoden der Software Entwicklung, die als „agil“ bezeichnet werden und dem Rational Unified Prozess als ein Vertreter der klassischen oder auch als tayloristisch bezeichneten Vorgehensmethoden. Im Verlaufe dieses Seminars wurden diverse agile Methoden der Software Entwicklung vorgestellt und teilweise bewertet. Wichtige Punkte sollen im zweiten Kapitel zusammengetragen und erläutert werden, um eine gemeinsame Vergleichsbasis zu bieten. Im folgenden dritten Kapitel soll dasselbe für die klassischen Ansätze gemacht werden, um auch eine Bewertung des Rational Unified Process (RUP) gegenüber klassischen Methoden zu erlauben. Dies soll die Basis legen für die Beantwortung von Fragen wie „klassisch“ der RUP ist, beziehungsweise welche Elemente typisch „klassisch“ sind. Folglich muss im vierten Kapitel der RUP behandelt werden, was eine Gegenüberstellung der Methoden im fünften Kapitel erlaubt.

1.1. Einordnung / Übersicht von Prozessmodellen

In Anlehnung an die Einteilung der Object Management Group (OMG) in [OMG_SPEM] gliedern sich die Vorgehensmodelle der Software Entwicklung in folgende Kategorien: Wie aus der Grafik ersichtlich ist, zählen die im Seminar besprochenen und die

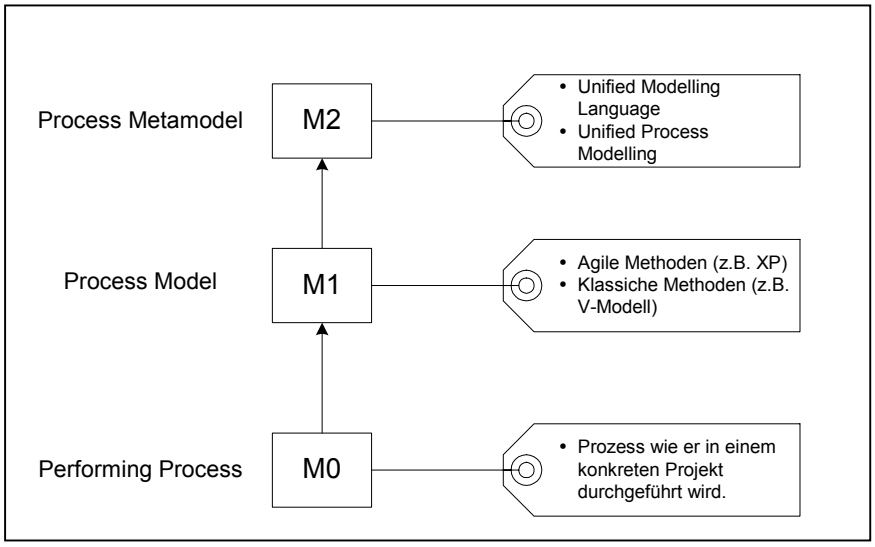

Abbildung 1: Prozessmodelle

klassischen Methoden zu der Kategorie „Process Model“. Von diesem Standpunkt aus sollten diese Methoden dieselben Ansprüche befriedigen. Es scheint mir jedoch die Frage erlaubt, ob nicht innerhalb der Kategorie Process Model eine weitere Aufteilung nötig wäre. So würde ich beispielsweise eine Unterteilung in Prozess Methoden (das Wort „Methode“ kommt im Namen der meisten Agilen Methoden vor) und Modellen oder Frameworks begrüssen. Da hier verschiedene Methoden (Frameworks, wie auch immer) verglichen werden sollen, wollte ich diesen Punkt kurz aufgreifen um ihn aber bis zu den letzten Kapiteln aber gleich wieder zu „vergessen“. Ich werde daher jetzt nicht weiter darauf eingehen und auch nicht diskutieren, ob jede Agile Methode wirklich dieselbe Aufgabe zu lösen bestrebt wie der RUP.
Bezüglich Einordnungen noch eine weitere Anmerkung: Ein weiterer Punkt, der zu Diskussion Anlass geben könnte ist, ob der RUP mit der im Titel implizierten Zuteilung zu den klassischen Methoden richtig kategorisiert ist. Dieser Punkt wird im fünften Kapitel behandelt.

2. Eigenschaften agiler Methoden des SE

Wie wir in den Seminarvorträgen hören konnten, wurden diverseste Methoden der Software Entwicklung „geschaffen“, die sich zu den sogenannten „agilen“ Methoden zählen. Einerseits legen die verschiedenen Modelle ihren Fokus teilweise auf sehr unterschiedliche Punkte und blenden auf der anderen Seite Teilaspekte aus. Sie sind daher schwer unter einen Hut zu kriegen. In diesem Kapitel möchte ich die Kernaussagen und Gemeinsamkeiten dieser Methoden, sowie ihre Widersprüchlichkeiten, Stärken und Schwächen aus den bestehenden Seminararbeiten synthetisieren. Dies wird aufgrund der bestehenden Unterlagen in gestraffter Form geschehen und jeweils kurz kommentiert.

2.1. Kernpunkte und Gemeinsamkeiten

Kernpunkt aller agilen Methoden ist, dass sie sich zum Agile Manifesto der Agile Alliance (http://www.agilealliance.org) bekennen. In diesem Manifest sind Grundprinzipien der Vorgehensmethodik definiert, wie wir es schon im ersten Seminarvortrag gehört haben.

2.1.1. Manifest

1. Mensch und Kommunikation vor Tools und Prozessen	2. Funktionierende Software vor endloser Dokumentation
3. Zusammenarbeit mit dem Kunden statt Verhandlungen mit dem Kunden.	4. Aktiv Änderungen nachvollziehen, nicht einen Plan befolgen.

Tabelle 1: Agile Manifest

2.1.2. Prinzipien

1. Kunden befriedigen mit frühen und kontinuierlichen Releases.	2. Auch späte Änderungen begrüssen. Änderungen zu des Kunden Wettbewerbsvorteil ausnutzen.
3. Regelmässige Auslieferung von Software innerhalb von kurzen Zyklen (Wochen).	4. Tägliche Zusammenarbeit von Entwicklern und Business-Vertretern während des ganzen Projektes.
5. Involviere motivierte Leute und glaube an Sie. Schaffe ihnen ein gutes Umfeld und die nötige Unterstützung.	6. Die effektivste Form der Vermittlung von Informationen in Entwicklerteams ist das direkte Gespräch.
7. Primär wird der Projektfortschritt an funktionierender Software gemessen.	8. Nachhaltige Projekt-Entwicklung: Sponsoren, Entwickler und Benutzer sollten die Arbeitsintensität und -geschwindigkeit immer halten können.
9. Kontinuierliche Überwachung des Designs und der technischen Qualität verbessert die Agilität.	10. Einfachheit zur Vermeidung des sinnlosen Verbrauchs von Arbeitsressourcen.
11. Selbstorganisierende Teams ergeben die beste Architektur, die klarsten Anforderungen und das beste Design.	12. Das Team hat regelmässig die eigene Effektivität zu bewerten und zu verbessern.

Tabelle 2: Agile Principles

2.1.3. Anmerkungen

Die vier Punkte des Manifestes beinhalten jeweils wichtige Kritikpunkte der Agile Alliance gegenüber den klassischen Methoden der Software Entwicklung. Ihr Inhalt definiert den Fokus der agilen Methoden auf folgende vier Punkte:

- Mensch und zwischenmenschliche Interaktion

- Kurze Release-Zyklen ➔ Inkrementell
- Nähe zum Kunden
- Adaptivität (Integration von Änderungen)

Manifest und Prinzipien definieren selbstverständlich noch kein fertiges Vorgehensmodell für die Entwicklung von Software. Konkrete Umsetzungen dieser Grundsätze haben wir während des Seminars in einer ganzen Reihe von Vorträgen kennen gelernt.

2.2. Diskussionspunkte

Innerhalb der Grundprinzipien des Manifestes sind Widersprüche nicht zu erwarten und soweit auch nicht auszumachen. Innerhalb der konkreten Umsetzungen tauchen dann nach meiner Meinung doch einige Punkte auf, die zumindest zur Diskussion Anlass geben. Ohne auf jede einzelne Methode einzugehen, möchte ich an dieser Stelle ein paar Punkte aufzählen, die beim Diskussionsschwerpunkt ob der RUP agil ist oder nicht wieder aufgegriffen werden können.
Die Punkte, welche in der Folge diskutiert werden, sind vor allem im Zusammenhang mit der Methode unter der sie auftreten auffällig. Es ist aber oft auch ein in abgeschwächter Form allgemein auftretendes Problem der agilen Methoden.

2.2.1.1. Test-First Programming

Vertrauen in die Fähigkeiten der Entwickler: Wenn wir davon ausgehen, dass wie oft in agilen Methoden das System anhand von Use Cases spezifiziert und daraus Testfälle abgeleitet werden, die dann als Programmiergrundlage für den Entwickler dienen, steht der Programmierer vor einer klar definierten und eng eingegrenzten Aufgabe, die kaum Bewegungsspielraum frei lässt.
Spezifikation: Wenn das einzige Ziel von Programmierung das Erfüllen von Testfällen ist, dann müssen sehr hohe Anforderungen an die Qualität und Quantität dieser Testfälle gestellt werden. Ansonsten erfüllt scheinbar „korrekte“ Software beispielsweise die Anforderungen eines Akzeptanz –Testes nicht.

2.2.1.2. PSP und TSP

Diese Methoden sind aus meiner Sicht wohl sowieso eher als ein Schulungsprozess eines Entwicklers oder eines Teams zu sehen und lieferten kaum Empfehlungen zur Form der Abwicklung eines konkreten Projektes. Zudem wäre dieser Entwicklungsprozess (falls Zeit und Geld vorhanden) problemlos mit einer klassischen Methoden zu vereinbaren.

2.2.1.3. XP

Projektfortschritt: Die Kontrolle von Projektfortschritt und Qualität obliegt dem Coach. Man mutet ihm zu, ohne grossen Support durch Prozesse und Tools die Aufgabe wahrnehmen zu können. Da dies als schwierig identifiziert wurde, wird auch gesagt, der Coach müsse eine sehr gute und erfahrene Person sein. Erstens ist das für jedes Projekt von Vorteil, wenn der Leiter eine fähige Person ist und zweitens gibt es bestimmte Projekte, die von einer Einzelperson ohne Unterstützung nicht geführt geschweige denn kontrolliert werden können.

2.2.1.4. Dynamic System Development Method

Funktionalität, Zeit, Ressourcen: Ein sehr interessanter Ansatz zur Lösung des allgegenwärtigen Problems der Zielerreichung in Softwareprojekten wurde hier gewählt. In einem hochdynamischen Umfeld erreichen (klassisch geführte) Projekte oft die gewünschte Systemfunktionalität nicht in der vorhandenen Zeit und vorhandenen Ressourcen. Um dieses Problem zu beheben, erklärte man nun die Funktionalität zu einer

Variablen. Aus meiner Sicht macht man dadurch einfach aus der Not eine Tugend und kann danach jedes Projekt als Erfolg bezeichnen mit der Klammerbemerkung, dass es halt nur einen Drittel dessen kann, was vorgesehen war. Ich kann mir kaum vorstellen, dass eine derartige explizit formulierte Projektpolitik von einem Kunden akzeptiert wird.

2.3. Stärken / Schwächen

Es ist relative einfach Stärken und Schwächen von agilen Methoden aufzulisten, wie dies in Tabelle 3 geschieht. Die Problematik liegt aber in deren Bewertung und Gewichtung. Dazu währen fundierte Analysen und Studien von entsprechenden Projekten notwendig, die aufgrund der kurzen Geschichte von agilen Entwicklungsmethoden aber kaum zur Verfügung stehen. So ist es weitgehend ein ziemlich subjektives Abwiegen zwischen den Argumenten der Befürworter und den Gegnern. Wichtig für diese Beurteilung ist auch Kenntnis über die Ausgangsposition der beiden Lager: Klassische Methoden werden seit langer Zeit verwendet, analysiert und weiterentwickelt. Die Erfolgsquote ist aber wie Statistiken belegen eher bescheiden. Dadurch bieten sie Angriffsfläche für Kritik und Raum für neue Ansätze. Diesen füllen zur Zeit die Agilen Methoden. Diese ihrerseits konnten bisher den Erfolg ihres Ansatzes nicht vorbehaltlos glaubhaft machen, geschweige denn beweisen. Naturgemäss besteht seitens der Traditionalisten auch grosse Skepsis, gegen die sich die Agilen Methoden zu behaupten haben.

Stärken	**Schwächen**
Kleine Projekte	Grosse Projekte
Übersichtlichkeit (lean)	Erfassung der Komplexität
Flexibel	Laisser-faire
Eigenverantwortung	Kontrolle
„flexible“ Prozesse	Unreif / Lückenhaft
Angepasste Strukturierung	Methoden-Wirrwarr
Neuer Ansatz / Chance	Empirische Evidenz / Erfolgsausweise

Tabelle 3: Stärken/Schwächen Agile Methoden

Wie die Punkte erkennen lassen, ist die Schwäche oft eine negative Interpretation einer positiven Neuerung oder Forderung der Agilen Methoden. Dies führt wiederum zum Schluss, dass für eine abschliessende Bewertung einerseits die Position des Betrachters und andererseits die konkrete Umsetzung im Projekt entscheidend ist. Erst dann lässt sich schlüssig bestimmen, ob es nun ein flexibel geführtes Projekt war, oder ob einfach im Laisser-faire Stil gewurstelt wurde...

3. Eigenschaften klassischer Methoden des SE

In diesem Kapitel möchte ich kurz auf die klassischen Methoden eingehen. Da diese im Seminar auch schon kurz angesprochen wurden und sie auch Stoff von vorausgesetzten Veranstaltungen sind, wird dieses Kapitel kurz gehalten. Es sollen nur diejenigen Punkte angesprochen werden, die für die folgende Abhandlung des RUP relevant sind.

3.1. Kernpunkte und Gemeinsamkeiten

An dieser Stelle sei die Frage erlaubt, ob es eigentlich DIE Eigenschaft(en) von klassischen Methoden gibt. Es sind hauptsächlich 5 Modelle bekannt, nämlich:

- Wasserfall-Modell
- Ergebnisorientiertes Phasenmodell
- Wachstums-Modell
- Spiralmodell
- V-Modell

Einige darunter sind sich sehr ähnlich (Wasserfall & Ergebnisorientiertes Phasenmodell) oder können als Weiterentwicklung / Verfeinerung eines anderen Modells angesehen werden. Andere sind unter dem hier betrachteten Gesichtspunkt weniger interessant, da keine Aussagen über organisatorische oder prozessuale Umsetzungen gemacht werden (V-Modell).

Als Gemeinsamkeit werden für die klassischen Modelle folgende Punkte erwähnt:

- Grosser Zeithorizont
- Hoher Detaillierungsgrad der Planung
- Prozess im Zentrum

Wie die einzelnen Modelle dies angehen ist jedoch sehr unterschiedlich. Die oben genannten Punkte sind auch im Zusammenhang mit der Abgrenzung von den agilen Methoden zu verstehen.

3.2. Stärken / Schwächen

Die Schwächen und Stärken dieser Modelle sind viel weniger allgemein zu teilbar als bei den agilen Methoden. Daher werden Sie im folgenden kurz für jedes Modell spezifisch aufgeführt [SE_SEM].

Stärken	Schwächen
Wasserfall-Modell	
Einfachheit	Fehler nicht nur aus gegenwärtiger und vorangegangener Phase - Iterationen über mehrere Phasen - Erschwerte Projektführung
Modellierung des natürlichen Lebenslaufes	
Ergebnisorientiertes Phasenmodell	
Leicht verständlich	Für grosse Systeme ungeeignet, da diese evolutionär wachsen
Modellierung des natürlichen Lebenslaufes	Erst spät lauffähige Systeme
Förderung eines planvollen Vorgehens	Komplette Inbetriebnahme in einem Schritt
Wachstums-Modell	
Entspricht natürlichem Verhalten der meisten grossen Systeme	Gefahr, dass viele nicht ganz zusammen-passende Teilsysteme entstehen
Lauffähige Teile entstehen sehr schnell	Konzepte und Strukturen können durch

	Ergänzungen und Änderungen bus zur Unkenntlichkeit entstellt werden
Sanfte Einführung des Systems möglich	
Spiral-Modell	
Hat in einem einzelnen Zyklus grundsätzlich die gleichen Schwächen und Stärken wie das Wasserfall-Modell. Reviews und Risikoanalyse erhöhen jedoch die Anpassungsfähigkeit an die Dynamik. Es ist durch die zyklische Umsetzung auch besser geeignet für grosse Systeme im Vergleich zum Wasserfall-Modell.	

Tabelle 4: Stärken/Schwächen klassische Methoden

4. Einführung in den Rational Unified Process (RUP)

In diesem Kapitel wird der Rational Unified Process erläutert [UNIFIED][RATIONAL]. Dies soll eine anschliessende Diskussion über seine Agilität erlauben.

4.1. Geschichte

4.1.1. The three amigos of Software Engineering

RUP hat diverse Väter und seine Entstehung ist durch die Erfinder eng an die Geschichte von UML geknüpft. Entstanden ist der Rational Unified Process bei der Firma Rational (USA, im Februar 2003 von IBM aufgekauft). Die wichtigsten Personen hinter seiner Entstehung sind Grady Booch, Ivar Jacobson und James Rumbaugh. Sie entwickelten schon bevor sie gemeinsam bei Rational tätig waren diverse Ansätze im Bereich des Software Engineering. Die Entstehung des Rational Unified Process war eine Art Synthetisierung ihrer voneinander unabhängig entwickelten Methoden und Modellen im Bereich Software Engineering. Diese drei Männer zeichnen sich auch zu einem grossen Teil verantwortlich für die Entstehung von UML. Seither sind sie auch bekannt unter dem Namen „The Three Amigos of Software Engineering“. Alle drei leiteten schon lange vorher grosse Projekte und hatten verantwortungsvolle Jobs bei Grossfirmen inne (Ericsson, IBM).

4.1.2. Entwicklung

Die Firma Rational (IBM) entwickelt ungefähr seit 1998 den RUP in seiner heutigen Form und bietet diverse Tools zur Verwendung und Unterstützung des RUP’s an. Heute hat sich Rational den Slogan „The software development platform for an on demand world.“ auf die Fahnen (Website) geschrieben. Die Software-Palette deckt sehr viele Bedürfnisse des Software Entwicklung ab. In diese Palette sind Weiterentwicklungen von Tools integriert, die von den drei Amigos schon in der Zeit davor entworfen wurden (wie z.B. Rational ROSE von Booch).

4.2. Charakteristiken

4.2.1. RUP Prinzipien

1. Frühes und kontinuierliches Angehen von Risiken: Die Hauptrisiken sollen in frühen Iterationen erkannt und angegangen werden. Wichtig dafür ist eine Loslösung vom linearen Projektverlauf.
2. Änderungen von Anfang an integrieren und managen: Die Erkenntnis, dass Software zu komplex ist um Anforderungen, Design und Implementation von Anfang an korrekt zu erhalten macht Change Management nötig. Dies wird unterstützt durch die Aufteilung in Iterationen innerhalb der Phasen.
3. Fokus auf ausführbare Software: Anstelle von endloser Analyse und Design (Analysis - Paralysis) soll Wert auf lauffähige Software gelegt werden.
4. Erstellung von früh ausführbarer und testbarer Architektur: Schlüsselfunktionen sollen schon in der Entwurfs-Phase in Form eines Skeletons implementiert und getestet werden.
5. Komponentenbasierte Systemarchitektur: Sie unterstützt das Change Management durch die Kapselung von Problemdomänen. Dadurch wird das System flexibler und Anpassungen können kontinuierlich und effizient vollzogen werden.
6. Mehrwert für den Kunden sicherstellen: Anhand von Use Cases soll ständig zusammen mit dem Kunden sichergestellt werden, dass das Design und die Implementation auch im Sinne des Endbenutzers sind.

7. Qualität als Lebensphilosophie: Prozesse und Tools sollen zu qualitativ guter Arbeit zwingen. Die Qualität muss kontinuierlich mess- und testbar sein.
8. Arbeite eng zusammen im Team: Direkte aber auch Tool unterstützte Kommunikation innerhalb des Projektteams ist äusserst wichtig für den erfolgreichen Verlauf eines Projektes.

[RATIONAL]

4.2.2. Projekt und Prozesse

Der Rational Unified Process wird gesehen als Vereinigung von sogenannten Best Practice Ansätzen des Bereichs Softwareentwicklung. Dies bedeutet, dass kommerziell erprobte und für gut befundene Methoden verwendet werden und somit gleichzeitig aus Fehlern gescheiterter Projekte gelernt werden kann. Das Vorgehen in Softwareentwicklungsprojekten nach RUP ist grundsätzlich **iterativ**, der Projektfortschritt wird dabei anhand von Meilensteinen gemessen. Eine Kernaussage der RUP Propaganden ist, dass gute Planung des Projektes von eminenter Wichtigkeit für den Erfolg ist. Weitere Erfolgsfaktoren sind Zielorientierung, Messbarkeit, Flexibilität, Anbindung an aktuelle Projektprozesse (Tracking), Regelung von Zuständigkeiten und Erkennung von Abhängigkeiten unter den Aufgaben und Teams [RATIONALWP1]. Angewendete Prinzipien sind **visuelle Modellierung**, Change/Konfigurations-Management, saubere **Anforderungsverwaltung** und **komponentenbasierte Architektur**.

4.2.2.1. Phasen

Das Projekt wird aus Sicht des Management in vier Phasen gegliedert. Am Ende einer Phase stehen die wichtigsten Meilensteine. Diese Perspektive ist für das Business die relevante Sicht.

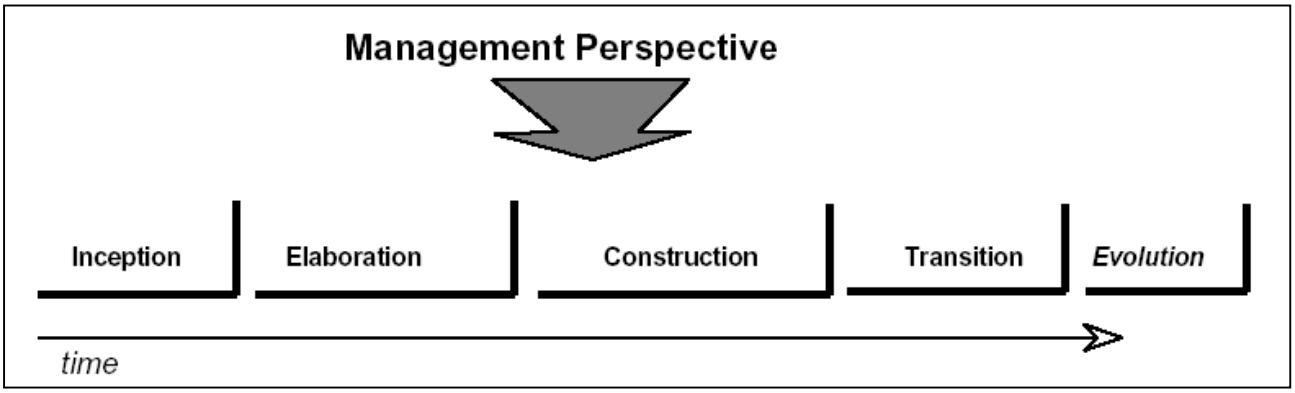

Abbildung 2: RUP Management Perspektive

4.2.2.1.1. Konzeption (Inception)

In dieser Phase geht es darum den Rahmen (Scope) des Projektes zu erfassen. In dieser Phase wird das gemacht, was allgemein auch bekannt ist als eine Grobanalyse. Ziel dieser Phase ist, aus einer Vision eine konkrete Projektidee, einen Business Case zu generieren (Entscheid: “go“ bzw. im negativen Fall ein „no-go“).

4.2.2.1.2. Entwurf (Elaboration)

Diese Phase beinhaltet die Weiterführung der Grobanalyse. Am Ende dieser Phase müssen die high-level Architektur und sowie die Anforderungen an das zu bauende Produkt bekannt sein. Zudem sollen hier auch die wichtigsten Projektrisiken identifiziert werden.

4.2.2.1.3. Realisierung (Construction)

In dieser Phase wird das Produkt erstellt. Nebenbei entstehen hier auch wichtige Deliverables wie interne und externe Dokumentation, Testtools und Testbeschreibungen, Installationshilfen etc.

4.2.2.1.4. Einführung im Betrieb (Transition)

Diese Phase dient der Ablieferung oder besser der geführten Übergabe der Software an den Kunden. Wichtige Aufgaben dabei sind Installation, Konfiguration, Support der Benutzer, Korrekturen und formale Akzeptanz der Benutzer.

4.2.2.1.5. Evolution

Je nach Projekt- und Auftragsart kann nach der Transition eine Evolutionsphase folgen, die ihrerseits wieder aus allen vier vorhergehenden Phasen besteht.

4.2.2.2. Zeitliche Gliederung

Erfahrungsgemäss präsentiert sich die zeitliche Gliederung eines typischen RUP-Projektes bezüglich der Phasen folgendermassen [RATIONALWP3**]**:

Abbildung 3: RUP zeitliche Gliederung der Phasen

	Inception	**Elaboration**	**Construction**	**Transition**
Effort	5%	20%	65%	10%
Schedule	10%	30%	50%	10%

Tabelle 5: RUP zeitliche Gliederung der Phasen

Folgende Tabelle gibt Richtgrössen für die Dauer von Iterationen in Abhängigkeit von der Anzahl der involvierten Personen [SYSTOR]:

Anzahl Personen	**Dauer einer Iteration**
5	1-2 Wochen
15	1 Monat
45	6 Monate
100	12 Monate

Tabelle 6: RUP Projektgrössen und Projektdauer

4.2.2.3. Iterationen

Aus einer technischen Perspektive unterteilt sich das Gesamtprojekt in Iterationen. Jede Iteration sollte ausführbare Software liefern zu Demonstrations- und Testzwecken auf der Grundlage der Projektanforderungen bzw. den daraus abgeleiteten Use Cases. Für jede Iteration wird ein Plan erstellt, der folgende Punkte definiert:

- Detaillierte Beschreibung der zu erledigenden Aufgabe in dieser Iteration
- Definieren der involvierten Rollen, der konkreten Aktivitäten und der Deliverables
- Definieren von klaren Kriterien zur Messung des Fortschritts und Erfolgsbestimmung am Ende der Iteration
- Spezifizierung des Zeitplanes für diese Iteration

Wie die obigen Punkte im Detail definiert werden, ist abhängig von der Phase in der man sich gerade befindet. Das heisst, dass jede Tätigkeit innerhalb von Iterationen abhängig von der Projektphase ihren Fokus auf den Meilenstein der aktuellen Phase zu legen hat.

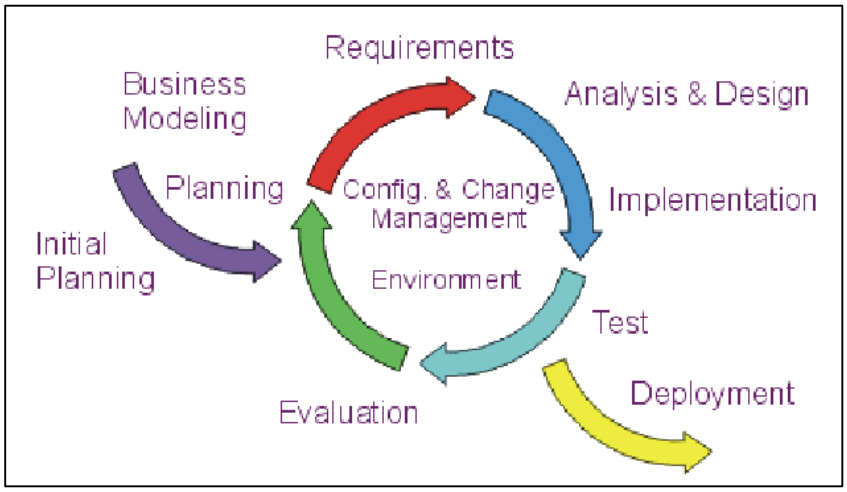

Abbildung 4: RUP Iterationen

Jede Iteration selbst besteht aus den Schritten Planung, Anforderungsspezifikation, Analyse und Design, Implementation, Test und Evaluation. Die Initiale Planung mit der Business Modellierung ist der Einstiegspunkt zu Beginn des Projektes vor der ersten Iteration. Deployment ist der Ausstiegspunkt nach der Testphase der letzen Iteration am Ende des Projektes. Querschnittsaufgaben während allen Iterationen durch alle Phasen sind Projektmanagement, Konfigurationsmanagement und Management des Projektumfeldes.

4.2.2.4. Meilensteine

Aus Sicht der RUP-Väter erfüllen Meilensteine zwei wichtige Funktionen in einem Software-Projekt. Die im folgenden erläuterten Punkte gelten für Top-Level Meilensteine. Nicht zwingend gelten sie für innerhalb eines Teams im kleineren Rahmen definierte Meilensteine.

1. Meilensteine dienen der Synchronisierung zwischen den Phasen und Iterationen.

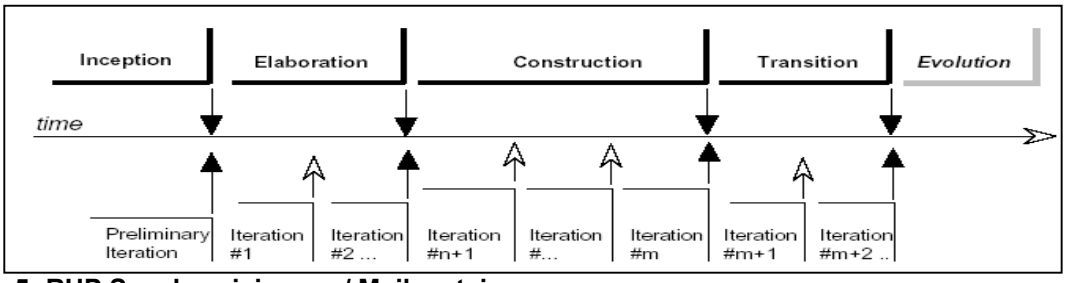

Abbildung 5: RUP Synchronisierung / Meilensteine

2. Die Meilensteine geben an worauf in den verschiedenen Iterationen der Fokus zu legen ist im Sinne eines übergeordneten Ziels. Inhaltlich haben alle erstellten Dokumente und Produkte einer Iteration den nächsten Meilenstein als Zielvorgabe zu befriedigen. Daraus folgt auch, dass die Aufgaben innerhalb der Iterationen je nach Phase in welcher sich das Projekt befindet sich in ihrer Art ändern oder zumindest der zeitlichen Belastung variiert.

4.2.3. Dokumente und Produkte (Artifacts)

Wie unschwer zu vermuten ist, wird wo immer angebracht (in technische Dokumentationen) UML als Darstellungsform verwendet. Diese durchgängig verwendete

Form der Darstellung erlaubt eine verständliche Kommunikationsform zwischen den verschiedenen Aufgabenbereichen und Teams. Für viele Dokumente bestehen auch Vorlagen (Templates), die der Vereinheitlichung und damit zur besseren allgemeinen Verständlichkeit beitragen. Die Tabellen 7 und 8 listen die wichtigsten Dokumente oder Gruppen von Dokumenten auf, die im RUP definiert werden. Sie sind jedoch nicht vollständig. Der RUP definiert über 100 verschiedene Dokumente, die jedoch nicht alle zwingend zu erstellen sind.

4.2.3.1. Technische Dokumente

Bezeichnung	**Beschreibung**
Benutzerhandbuch	Möglichst FRÜH im Verlaufe des Projektes zu erstellen.
Software Dokumentation	Anhand von Use Cases, Klassendiagrammen, Prozessdiagrammen und selbst-dokumentierendem Code. Diese Dokumente sollen mit Unterstützung von CASE Tools erstellt/generiert werden.
Software Architektur	Im Idealfall auch durch Tools extrahierte Abstraktion aus der vorhanden Softwaredokumentation. Darstellung von Klassenkategorien, Komponenten, kritische Schnittstellen und Prozessen.

Tabelle 7: RUP Technische Dokumente

4.2.3.2. Management Dokumente

Bezeichnung	**Beschreibung**
Projektorganisation	Dokumentiert die konkrete Umsetzung des RUP-Frameworks in diesem Projekt.
Vision	Beschreibt die Systemanforderungen, dessen Qualität und Prioritäten.
Business Case	Finanzielles Angelegenheiten, vertragliches, erwartete ROI etc.
Entwicklungsplan	Gesamtprojektplan mit der Einteilung in Phasen und Iterationen und einem Detailplan zur aktuellen Iteration.
Evaluationskriterien	Anforderungskatalog, Akzeptanzvorgaben, Wird in jeder Iteration erweitert um die entsprechenden Ziele und Detailvorgaben.
Release Beschrieb	Beschreibung für jeden Release.
Deployment	Beschriebe für die Übergabe, das Training, die Installation, Verkaufsunterstützung etc.
Status-Assessment	Punktuelles Festhalten von Projektfortschritt mit Angaben über Ausgaben, erreichte Resultate, Personelles, nächste Schritte etc.

Tabelle 8: RUP Management Dokumente

Folgende Grafik illustriert einen üblichen Verlauf des Fortschritts von Dokumenten.

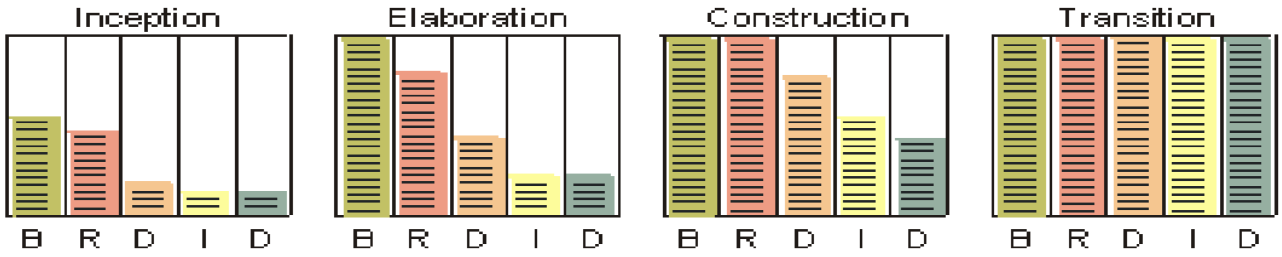

Abbildung 6: RUP Dokumenten Entwicklung

B=Business set, R=Requirements set, D=Design set, Implementation set, Deployment set [SYSTOR]

4.2.4. Organisation / Rollen

Im RUP werden äusserst detailliert Rollen definiert und Aufgabenbereiche zugeteilt. Über 30 Rollen sind definiert und Aufgaben- und Kompetenzbereiche zugeteilt. Die vier wichtigsten Rollen sind:

Rolle	Fachwissen
Projektleiter	Organisation
Qualitätsmanager	Projektziele
Domänenexperte	Anwendungsbereiche
Architekt	Technologie

Tabelle 9: RUP Rollen

Der Besetzung dieser und auch allen weiteren Rollen wird äusserste Wichtigkeit beigemessen. Eine Person kann gleichzeitig mehrere Rollen bekleiden. Welche Rollen in welchem Umfang besetzt werden müssen ist von Projekt zu Projekt zu definieren und Aufgabe der Planung. Gute gegenüber Fehlbesetzungen können die Aufwände bis zu einem Faktor 10(!) beeinflussen.

Bezüglich der Organisation der Teams konnte ich keine konkreten Angaben finden. Es wird nicht gesagt wie diese im RUP besetzt zu sein haben. Es existieren gewisse Normen oder Empfehlungen, die als Referenzwerte aus „typischen“ RUP-Projekten hergenommen werden, festlegen tut man sich aber nicht.

4.2.5. Aufgaben

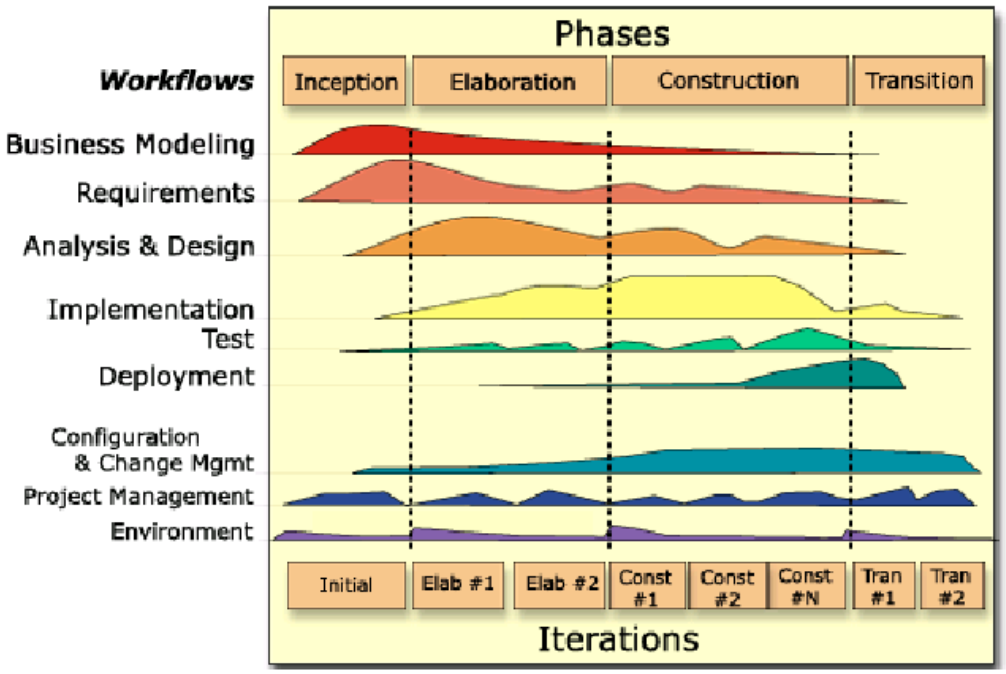

Abbildung 7: RUP Aufgaben

Die obige Graphik illustriert wie die verschiedenen Aufgaben in einem typischen RUP-Projekt anfallen können. Ihre Intensität ist abhängig vom allgemeinen Intensitätsniveau und vom aktuellen Status des Projektes bzw. in welcher Phase es sich befindet. Je nach Quelle (und Aktualität) werden die Aktivitäten / Workflows leicht abweichend benannt und anders strukturiert. Aufgaben und Rollen von Projektmitgliedern sind naturgemäss eng verknüpft. Die oben dargestellten Schwankungen des Arbeitsanfalls pro Aufgabengruppe haben daher ziemlich direkten Einfluss auf die Zahl der zu einem bestimmten Zeitpunkt im Projekt beschäftigten Mitglieder pro Aufgabenbereich. Die Aufgaben sind weitgehend

selbsterklärend, ich werde daher darauf verzichten sie hier weiter zu erläutern. In der Dokumentation sind sie sehr detailliert beschrieben und die einzelnen Schritte auch mit Ablaufdiagrammen dargestellt.

4.2.6. Anforderungen

Der Rational Unified Process wird nicht primär von Anforderungen gelenkt. Dies ist die logische Konsequenz aus der Erkenntnis, dass sich Anforderungen im Laufe eines Projektes ändern oder sogar erst entstehen. Anforderungen werden innerhalb einer Iteration definiert und umgesetzt. Folgende drei Quellen existieren für Anforderungen:

1. Auf höchstem Level definiert der Business Case die Anforderungen an das Projekt. Dies geschieht oft in Form von Ressourcen Restriktionen.
2. Funktionale Hauptanforderungen sollten der Vision (siehe Artifacts) entnommen werden können. Es wird erwartet, dass sich diese Anforderungen normalerweise nur wenig ändern im Laufe des Projektes.
3. Detailanforderungen werden hauptsächlich in der Entwurfsphase anhand von Use Cases erarbeitet. Später werden sie innerhalb der Umsetzungsiterationen verfeinert. Festgehalten werden diese Anforderungen im Dokument Evaluationskriterien.

4.2.7. Kundenrolle

Über die Interaktion mit dem Kunden selbst gibt RUP wenig konkrete Empfehlungen ab wie das z.B. XP macht (Kunde vor Ort). Anhand von Abnahmen der diversen Dokumente und der Anforderungsdefinierung mit Use Cases soll ständig sichergestellt werden, dass im Sinne und zur Zufriedenheit des Kunden gearbeitet wird. Nur schon durch die Zahl der abzusegnenden Dokumente scheint mir eine enge Einbindung von Kundenvertretern gegeben. Über das Wesen der Vertreter selbst (Bildung, Position, Funktion) konnte ich auch keine konkreten Hinweise finden.

4.2.8. Risk Management

Der RUP wird als Risiko gesteuerter Prozess bezeichnet. Dies bedeutet, dass die Risiken, die einer Aufgabe zugeordnet werden, definieren, wann und mit welchen Ressourcen sie angegangen wird. Die grössten Risiken sollen früh identifiziert und angegangen werden.

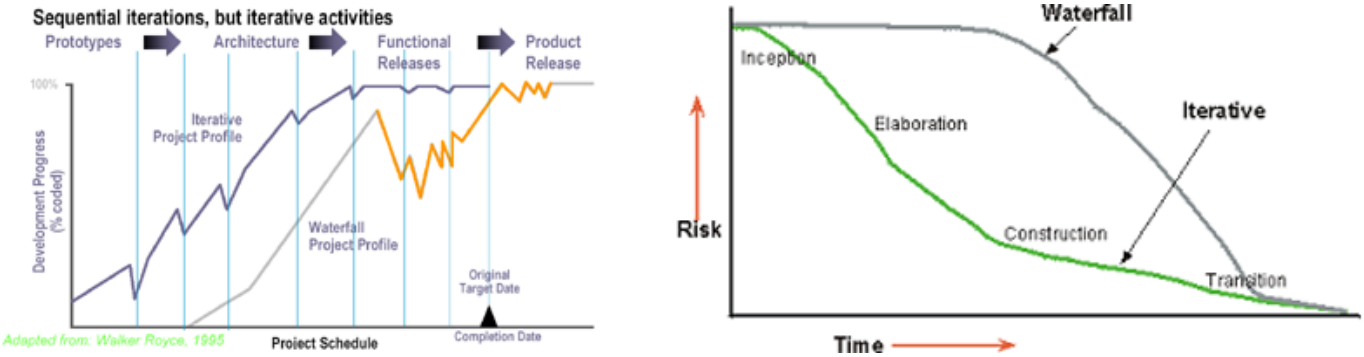

Abbildung 8: Projektfortschritt und Risiko

Die rechte Grafik zeigt das Risikoniveau im Verlaufe der Zeit in einem traditionellen Wasserfall-Projekt und in einem RUP Projekt. Die linke Grafik zeigt den damit verbunden Projektfortschritt über die Zeit.

4.3. Software-Tools

Die Rational Tools decken einen Grossteil des Bedarfs an Toolunterstützung für das Software Engineering ab. Sie können weitgehend unabhängig voneinander eingesetzt werden. Sie sind daher uneingeschränkt an die spezifischen Bedürfnisse eines Projektes anpassbar.

Bezeichnung	**Funktion**
Die Rational Suite • Configuration Management • Process Delivery Tools • Process Authoring Tools • Community / Marketplace	RUP Kernfunktionalitäten zur Planung des Projektes: • Definieren von Parametern • Definieren von Prozessen • Publizieren von Prozessen (web-basiert) • Interaktionstools
Rational ClearCase	Configuration Management Environment
Rational ClearQuest	Change Management
Rational PurifyPlus	Debugging und Diagnose
Rational Requisite Pro	Verarbeitung von Anforderungen
Rational Robot	Automatisiertes Testen
Rational Rose	Modellierung mit UML
Rational SoDa	Dokumentationsgenerierung in verschiedenen Formaten
Rational TestManager / Test Realtime / TestFactory	Support für Testing: Testgenerierung, Durchführung, Auswertung
Rational Process Workbench	Unterstützungstool für die Projektkonfiguration
RUP Builder	Setup von Projekten, Management von Plugins
Rational sponsored Plug-Ins RUP Plug-In for Sun iAS RUP Plug-In for SunOne RUP Plug-In for Extreme Programming (XP) RUP Plug-In for System Engineering RUP Plug-In for User-Experience Modeling RUP Plug-In for J2EE RUP Plug-In for RealTime RUP Plug-In for Business Modeling RUP Plug-In for Asset-Based Development RUP Plug-In for Creative Web Design	**Partner Plug-Ins** RUP Plug-In for Existing Software Reuse by Klocwork RUP Plug-In for Business Rules RUP Plug-In for Achieving Straight-Through Processing ICONIX QuickStart Plug-In for RUP RUP Plug-In for CAST Application Mining Suite RUP Plug-In for Operations and Application Surveillance Management

Tabelle 10: RUP Software Tools

Grundsätzlich sind die Tools unabhängig einsetzbar von einer im Projekt zu verwendenden Technologie oder anderen projektspezifischen Parametern wie z.B. Grösse. Diese Parameter sind individuell durch den Anwender einstellbar, was die Tools vielseitig aber auch komplex macht. Darin liegt sicher auch Kritik begründet, dass sich Projektteilnehmer lange und intensiv in die Tools einarbeiten müssen und somit zuwenig Zeit für die wahre, produktive Tätigkeit finden.

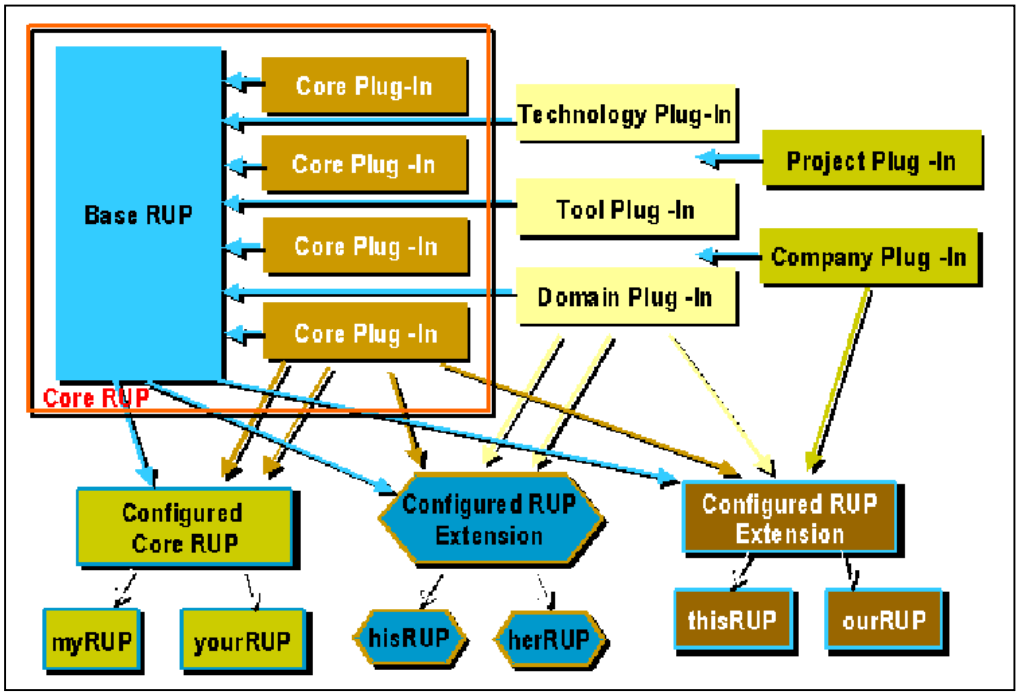

Abbildung 9: Componentized RUP

Um dieser Kritik zu begegnen und aus der Erkenntnis, dass es die Lösung „one-size-fits-all“ wohl nicht gibt bei Softwareprojekten, ist es möglich verschiedenste vorkonfigurierte Systeme zu beziehen, die auf gewisse Gegebenheiten eines Projektes optimiert sind (Componentized RUP). Die Mächtigkeit des Frameworks wird erhöht, indem man es mit Plug-Ins ergänzt, welche die Integration von Technologien (z.B. J2EE) oder Sprachen (C, C++, Java etc.) erlauben. Plug-Ins werden auch von Drittfirmen angeboten. Die Absicht dahinter ist meist die Integration eines eigenen Produktes in RUP zu ermöglichen.

5. Einordnung des RUP

Hier soll diskutiert werden, ob es korrekt ist, den RUP als Vertreter der klassischen Methoden zu einem Vergleich mit den agilen Methoden herbeizuziehen. Es muss vorweggenommen werden, dass jegliche Qualifizierung bezüglich Agilität des RUP nicht einfach auf alle anderen klassischen Methoden übertragen werden kann. Damit würden einige der klassischen Methoden in ein zu gutes aber auch falsches Licht gestellt werden.

5.1. Klassische Aspekte

5.1.1. Taylorismus

Zur Qualifizierung der Software Engineering Methoden taucht im Zusammenhang mit den hier als „klassisch“ bezeichneten Methoden oft der Begriff „tayloristisch“ auf. Dieser Begriff stammt ursprünglich aus der Betriebswirtschaftslehre und wurde von Frederick Winslow Taylor (1856-1915) zu Anfang des 20 Jahrhunderts geprägt. Eine Definition des Taylorismus von Gunther Dueck lautet:
„Der Tayorismus oder das Taylorsystem studiert und plant die genauen Zeit- und Arbeitsverläufe. Es wird für Arbeiten eine "allein richtige" Bewegungsfolge gefunden. Die Einhaltung dieser allein richtigen Bewegungsfolge wurde von sogenannten Funktionsmeistern ständig kontrolliert.“
Dieser Begriff ist heute teilweise negativ behaftet, weil damit ein menschenverachtendes und materialistisches Menschenbild in Verbindung gebracht wird. Nichtsdestotrotz sind seine Ideen der Unternehmungsführung bis heute weit verbreitet und in vielen Firmen praktisch umgesetzt.

Als tayloristischer Aspekt in klassischen Methoden der Software Entwicklung wird die minutiöse Planung des Projektes in Teilaufgaben, Teilschritte und Teilziele bezeichnet. Diese Einteilung soll möglichst früh im Analysestadium des Projektes geschehen und wenn immer möglich auch so durchgesetzt werden. Der RUP selbst weichte diesen Ansatz insofern auf, als die Detailplanung erst zu Beginn bzw. innerhalb einer Iteration vorgenommen werden soll. Dies resultierte aus der Erkenntnis, dass vollständige und korrekte Spezifikation und damit verbundene Planung bei der grossen Dynamik von Software Projekten illusorisch ist. Es existiert aber immerhin schon zu Beginn des Projektes ein Plan, der zeitlich den gesamten Projektrahmen abdeckt.

5.1.2. Dokumente / Wissensmanagement

Auch dieser Punkt ist durch die Lehre von Taylor beeinflusst. Nach der tayloristischen Auffassung sollen Prozesse detailliert und vollständig dokumentiert werden. Dies soll einerseits kürzere Einarbeitungszeiten durch Vorgehen nach „Checkliste“ ermöglichen. Andererseits dient es der Wissenserhaltung und Wissensvermittlung innerhalb der Prozessketten.

Im RUP wird durch weitgehende Toolunterstützung viel wert auf die Dokumentation gelegt. Dies ist alleine an der Liste der sogenannten „Artifacts“ von Kapitel 4.2.3 zu erkennen. Die Kritik der Agilitäts-Verfechter sieht hier diverse Nachteile für einen effizienten Software Entwicklungsprozess.

5.2. Agile Aspekte

Das wohl wichtigste Prinzip ist die **inkrementelle** Erstellung der Gesamtfunktionalität in Zyklen. RUP bekennt sich dazu, dass ein System nicht in einer einzigen vorgelagerten Analysephase vollständig spezifiziert und danach in linearem Verlauf entworfen und erstellt werden kann. Damit ist einer der grössten Kritikpunkte an den klassischen Methoden ausgemerzt. Doch dieser Punkt alleine macht aus RUP noch keine agile Methode. Das Wachstums-Modell beispielsweise propagierte auch schon inkrementelles Vorgehen.

Eine weitere Forderung der agilen Methoden erfüllt RUP, indem von jeder Iteration **lauffähige Software** gefordert wird. Dies erlaubt kontinuierliches Testen, Verifizieren der Anforderungskompatibilität (auch durch den Kunden) und Minimieren der technischen Risiken. In diesem Sinne hat sich auch der RUP den Kampf gegen die „Analysis-Paralysis“ auf die Fahnen geschrieben und propagiert das schnelle Erstellen von Software. (Auch wenn dies vielleicht ein wenig im Widerspruch zur ausführlichen Dokumentierung steht...)

RUP macht die Länge der **Iterationen** abhängig von der Zahl der Projektmitglieder. Bei kleinen Projekten führt dies zu sehr kurzen Iterationszyklen im Bereiche von Wochen, was mit vielen agilen Methoden zu vergleichen ist.

Ein viel verwendetes Prinzip der Gewinnung von **Anforderungen** und **Einbindung des Kunden** in agilen Projekten wird auch von RUP verwendet: **Use-Cases.** Genauer gesagt wurde das heute bekannte Konzept der Use Cases von Ivar Jacobson lange vor der Zeit von RUP entworfen als er noch bei der Firma Ericsson tätig war.

Zum Schluss zählt auch im RUP enge Teamzusammenarbeit und Kommunikation zu den wichtigen Erfolgsfaktoren.

Ein interessanter Punkt an der ganzen Sache ist ja, dass zwar viele traditionelle Software Entwickler die neuen agilen Ansätze kritisieren. Trotzdem ist zu beobachten, dass zumindest beim RUP viel Wert darauf gelegt wird, auch als agil angesehen zu werden. Im RUP wird offensichtlich versucht, die Vorzüge beider Philosophien unter einen Hut zu bringen. Dies hat konsequenterweise Kompromisse zur Folge. [RATIONALWP2]

6. Fazit

„…the big system people see RUP as the answer to their problems; the small system community sees XP as the solution to their problems. Our experience indicates that most software projects are somewhere in between - trying to achieve the right level of process for their situation. Neither end of the spectrum is sufficient for them…“ (Gary Pollice, Rational; Expanding Upon eXtreme Programming)

Wie dem obigen Zitat zu entnehmen ist, scheint die Industrie den RUP als mächtiges Tool und passende Lösung für grosse Projekte zu betrachten. Gerne würde Rational den RUP jedoch auch als agile Methode für kleinere Projekte eingesetzt sehen. Anstrengungen in diese Richtung haben sie mit dem Ansatz „Componentized RUP“ und vordefinierten Konfigurationen für beispielsweise XP unternommen. „One-Size-Fits-all” scheint wirklich nicht zu funktionieren beziehungsweise die “Silver-bullet” nicht zu existieren.

Die Erkenntnis, dass der RUP für kleine Projekte zu schwergewichtig ist und damit der Forderung nach Agilität nicht genügend nach kommt, scheint zumindest für seine herkömmliche Form akzeptiert zu sein. Daher wird nun seitens Rational und diversen RUP Anwendern versucht, diesen so anzupassen und zu konfigurieren, dass er für kleine Projekte effizient eingesetzt werden kann.

Ob dann am Ende noch ein „echter“ RUP da steht, führt eigentlich wieder zur Frage des ersten Kapitels. Ist der RUP und z.B. XP wirklich auf derselben Stufe in der Hierarchie der Prozessmodellierung anzusiedeln? Oder können bestimmte Agile Modelle als Untermengen und Instanzen von RUP angesehen werden? Obwohl diese Aussage sicher gewichtige Gegenargumente findet, sprechen einige Punkte dafür:

- Die Beschreibung des RUP-Modelles ist in seiner Art abstrakter und allgemeiner gehalten als diejenige von agilen Methoden.
- RUP definiert sich selbst als Modell für alle mögliche Projektvolumen.
- RUP und seine Tools sind flexibel konfigurierbar und die Projektparameter anpassbar.
- RUP-Projekte der Praxis mit reduziertem Aufgaben-, Rollen- und Dokument-Definitionen konnten erfolgreich agil gemanagt werden [ZUEHLKE].

Unabhängig davon ob nun eine oder mehrere agile Methoden als Instanzen von RUP angesehen werden können oder nicht, scheint mir klar, dass ein solcher Vergleich immer insofern hinkt, dass diese Methoden oder eben Frameworks nicht ganz auf derselben Ebene anzusiedeln sind. Die Tatsache, dass diverse Punkte eines Projektes wie z.B. Zahl und Art der Teambesetzung, Form der Kommunikation oder Form der Kundenintegration nicht klar vorgegeben sind in RUP, sind klare Indizien dafür, dass wir uns auf einem Level befinden der weniger nah am konkret durchgeführten Prozess ist als die Beschreibung mancher agiler Methoden. Zudem bezeichnet sich RUP selbst als ein Framework...

7. Abbildungsverzeichnis

8. Tabellenverzeichnis

9. Quellenverzeichnis

- [OMG_SPEM] Object Management Group, Software Process Engineering Metamodel Specification, Oktober 2002, http://www.omg.org/technology/documents/formal/spem.htm
- [SE_SEM] Vorträge Seminar Software Engineering Wintersemester 03/04 http://www.ifi.unizh.ch/req/courses/seminar_ws03/
- [AGILE] http://www.agilemanifesto.org und http://www.agilemanifesto.org/principles.html, 11.01.2004
- [RATIONAL] RUP Trial-Software Version 2003.06.01.04 https://www7b.software.ibm.com/dl/RATLe-RUPWIN-EVAL/RATLe-RUPWIN-EVAL-i
- [RATIONALWP1] David West, Rational/IBM, Planning a Project with the IBM Rational Unified Process, 2003 http://www3.software.ibm.com/ibmdl/pub/software/rational/web/whitepapers/
- [RATIONALWP2] Gary Pollice, Rational/IBM, Using the IBM Rational Unified Process for Small Projects:Expanding Upon eXtreme Programming, 2003 http://www3.software.ibm.com/ibmdl/pub/software/rational/web/whitepapers/
- [RATIONALWP3] Philippe Kruchten, A rational development process, 1996 http://www3.software.ibm.com/ibmdl/pub/software/rational/web/whitepapers/
- [UNIFIED] Grady Booch, Ivar Jacobson, James Rumbaugh, Unified software development process, Addison-Wesley
- [SYSTOR] RUP Einführung, 2001 http://www.systor.com/dl_know_campus_vorl_ecommerce_rup1.pdf
- [ZUEHLKE] Michael Hirsch, Making RUP agile, Schlieren, 2002 http://oopsla.acm.org/extra/pracreports/MakingRUPAgile-Report.pdf