# Software-Projektmanagement braucht klare Strukturen - Kritische Anmerkungen zum "Rational Unified Process"

Wolfgang Hesse

## Zusammenfassung:

Seit 1999 hat die Firma Rational ihrer Modellierungssprache UML den *Rational Unified Process (RUP)* als "Vereinheitlichtes Vorgehensmodell" zur Seite gestellt. Dadurch soll "die Produktivität von Entwickler-Teams verbessert" und den Projektleitern "bessere Kontrolle über Pläne und Projektergebnisse" gegeben werden. Weiter wird der RUP als Modell für "iteratives und inkrementelles, Anwendungsfall-getriebenes und Architektur-zentriertes Vorgehen" angekündigt. In sechs Thesen werden diese Ziele bzw. deren Umsetzung durch den RUP aus der Sicht von Software-Entwicklern und Projektleitern kritisch hinterfragt.

## 1 Der RUP und seine Kernelemente

Die Idee, den Software-Entwicklungsprozeß vereinheitlichen zu wollen, hat durch jüngere Initiativen der Firma Rational neuen Auftrieb gewonnen. Als sich deren ursprüngliches Ziel, verschiedene methodische Ansätze zur objektorientierten (OO) Modellierung zu einer sogenannten *"Unified Method"* zu vereinheitlichen, als zu ehrgeizig herausstellte, beschränkte man sich zunächst auf die Entwickung einer Modellierungssprache - der inzwischen bekannten *Unified Modelling Language UML* [UML 97]. In einem zweiten Schritt folgte kürzlich die Veröffentlichung des RUP, eines gemeinsamen Vorgehensmodells für OO-Software-Projekte ([JBR 99], [Kruc99]).

Für die besonderen Schwierigkeiten, die die Verfolgung dieses zweiten Teilziels mit sich bringt, gibt es verschiedene Gründe:

(1) Beim Vorgehensmodell spielen spezifische Argumente wie Firmenkultur, Art der Anwendungen, bestehende und verankerte Arbeits-, Organisations- und Management-Strukturen, Qualifikation der Entwickler und Projektleiter etc. eine so große Rolle, daß viele Kenner des Gebiets bezweifeln, ob eine Vereinheitlichung des Vorgehens möglich oder überhaupt nur sinnvoll ist.

(2) Mit den Veröffentlichungen von G. Booch [Booc 94], J. Rumbaugh et al. [RBP+ 91] und I. Jacobson [Jaco 93] lagen nicht nur drei sehr unterschiedliche Vorentwürfe der 3 Hauptautoren vor, sondern daneben eine Menge teilweise stark divergierender Ansätze anderer Autoren wie z.B. Coad/Yourdon, Wirfs-Brock oder Martin/Odell - um nur einige aus der Fülle der Vorschläge und Veröffentlichungen zu nennen.

(3) Für das Vorgehen bei der Software-Entwicklung gibt es unterschiedliche Grundansätze, die zu verschiedenen Arbeitsweisen führen und die sich deshalb kaum zu einem gemeinsamen Vorgehensmodell "unifizieren" lassen. So unterscheidet z.B. Denert eine transaktions- oder dokumentenorientierte Arbeitsweise [Dene 93]. Ein transaktionsorientiertes Vorgehensmodell kann die Vorteile einer dokumentenorientierten Arbeitsweise nicht nutzen und unterstützen.

(4) Der Software-Entwicklungsprozeß wird wie alle Prozesse maßgeblich durch die Zielsetzungen bestimmt. So wirkt sich z.B. die Forderung nach Wiederverwendung einer Komponente von Anfang an auf deren Entwicklungsprozeß aus. Ein gemeinsames Vorgehensmodell müßte auch dafür eine Variante anbieten [HeNo 99].

Der RUP geht in seinen Wurzeln auf Jacobson's *Objectory Process* [Jaco 93] und das Mitte der 90er Jahre bei Rational Software praktizierte Vorgehen zurück (vgl. [Kruc 99], S. 32). Er unterstellt ein inkrementelles Entwicklungs-Paradigma und unterteilt den Software-Entwicklungszyklus in *Phasen (phases).* Diese können in mehreren *Iterationen (iterations)* durchlaufen werden und durch *Meilensteine (milestones)* abgeschlossen werden. Diese Struktur ist überlagert von sogenannten *Kern-Arbeitsprozessen (core workflows).*

Im folgenden sollen diese Grundkonzepte des RUP anhand von Thesen diskutiert werden und es wird u.a. ausgeführt, daß

- *Phasen* sich zwar für herkömmliche Entwicklungsprojekte bewährt haben, aber für neuere Ansätze wie Objekt- oder Komponenten-orientierte Entwicklung nicht mehr oder nur noch bedingt geeignet sind,

- *Iterationen* vorgesehen werden sollten - allerdings nicht als Wiederholungen von *Phasen,* sondern von Prozessen, die zu bestimmten (noch nicht befriedigenden) *Arbeitsergebnissen* geführt haben,

- *Meilensteine* durch ein differenzierteres, nicht mehr an Phasen gebundenes Konzept ersetzt werden müßten und

- *Kern-Arbeitsprozesse* ein mit der Phasenstruktur konkurrierendes, für Management-Zwecke wenig hilfreiches Konzept sind, das sich in einfacher Weise durch *an Bausteine geknüpfte Aktivitätstypen* ersetzen ließe.

# 2 Sechs Thesen zum RUP

In den folgenden sechs Thesen werden wesentliche Kritikpunkte an der RUP-Struktur formuliert, näher erläutert und - soweit in der geforderten Kürze möglich - mit konkreten Gegenvorschlägen konfrontiert. Dabei werden vorrangig strukturelle Aspekte des Vorgehensmodells angesprochen, die detaillierte Behandlung inhaltlicher Aspekte würde dagegen den Rahmen dieses Positionspapiers sprengen.

**These 1:** Der RUP steht mit seinem Phasen- und Meilenstein-Konzept in der Tradition der herkömmlichen Vorgehensmodelle. Damit wird er den Anforderungen objekt- oder komponenten-orientierter Entwicklungsmethoden nicht gerecht.

**Diskussion:** Phasen-strukturierte (sog. Wasserfall-) Vorgehensmodelle haben in der Softwaretechnik eine lange Geschichte und sind im Praxiseinsatz vielfach bewährt. Dafür sind nicht zuletzt ihre (scheinbar) einfache Handhabung für das Management und ihre gute Eignung für herkömmliche Entwicklungsvorhaben verantwortlich. Für den Einsatz neuerer Methoden wie Prototyping, Objektorientierung, inkrementelle oder evolutionäre Entwicklung hat sich der Wasserfall-Ansatz jedoch als weniger geeignet erwiesen und es wurden hierfür verschiedene Alternativen vorgeschlagen und erprobt (vgl. z.B. [Booc 94], [FRS 89], [Hess 96]).

Der RUP steht mit seinen Phasen in der Wasserfall-Tradition - daran können auch neue Namen wie *Inception, Elaboration, Construction* und *Transition* mit entsprechenden Meilenstein-Bezeichnungen und eingebaute *Iterationen* (vgl. unten) nichts ändern. Die vom RUP suggerierte Vorstellung, *projektweit phasensynchron* arbeiten zu können, erweist sich jedoch spätestens dann als Fiktion, wenn verschiedene Komponenten gleichzeitig neu entwickelt, wiederverwendet oder angepaßt werden müssen, d.h. mehrere Prozesse in unterschiedlichen Entwicklungsstadien gleichzeitig ablaufen und koordiniert werden sollen. Phasen und Meilensteine sind dafür offensichtlich zu grob und müssen durch differenziertere, den geänderten Anforderungen besser angepaßte Strukturen ersetzt werden. So wurden z.B. als Alternativen zu der herkömmlichen Meilensteinen Referenzlinien [FRS 89] oder Revisionspunkte [Hess 96] vorgeschlagen.

**These 2:** Der RUP wird zwar als "architektur-zentriertes" Vorgehen angekündigt, nimmt aber dennoch auf die Architektur, im besonderen auf die Baustein- und Ergebnisstruktur, zu wenig Bezug.

**Diskussion:** Für die "Architektur-Zentrierung" werden von den Autoren die verschiedenen, u.a. in der UML definierten Modelle (Analyse-Modell, Entwurfs-Modell, Implementierungs-Modell etc.) ins Feld geführt, die verschiedene Sichten auf die schrittweise entstehende Architektur spezifizieren und dokumentieren sollen ([JBR 99], [Jaco 99]). Diese Modelle kommen allerdings in der RUP-Phasenstruktur (vgl. Abb. 2) nicht vor, sondern nur in den zugehörigen Erläuterungen und als Gegenstand der Kern-*Arbeitsprozesse (core workflows*), die zu den Phasen und Iterationen nur eine sehr unscharfe Beziehung haben (vgl. unten).

Mit dieser Sicht erweist sich der RUP als *transaktionsorientiertes* Vorgehensmodell im Sinne von [Dene 93]. "Architektur" wird dort als ein Bündel von ko-existierenden, durch Transaktionen ständig verfeinerten *Sichten* verstanden, die zusammengenommen den jeweiligen Entwicklungszustand des Gesamtsystems dokumentieren [Kruc 95]. Ein "architektur-zentriertes" Vorgehen, das diesen Namen verdient, müßte die Systemstruktur mit ihren Bausteinen sichtbar machen und Aktivitäten und Arbeitsergebnisse an deren jeweilige Entwicklungszustände koppeln. (vgl. Abb. 1). Es würde zu einer stärker *dokumenten-* oder *ergebnis-orientierten* Arbeitsweise (im Sinne von [Dene 93]) führen und eine Reihe von Vorteilen mit sich bringen, die z.T. in den weiteren Thesen noch angesprochen werden.

**These 3:** Der RUP sieht zwar Iterationen im Vorgehen vor, bindet diese allerdings an Phasen - statt wie es aus Gründen der Flexibilität notwendig wäre - an Bausteine, Arbeitsergebnisse und die zugehörigen Entwicklungsprozesse.

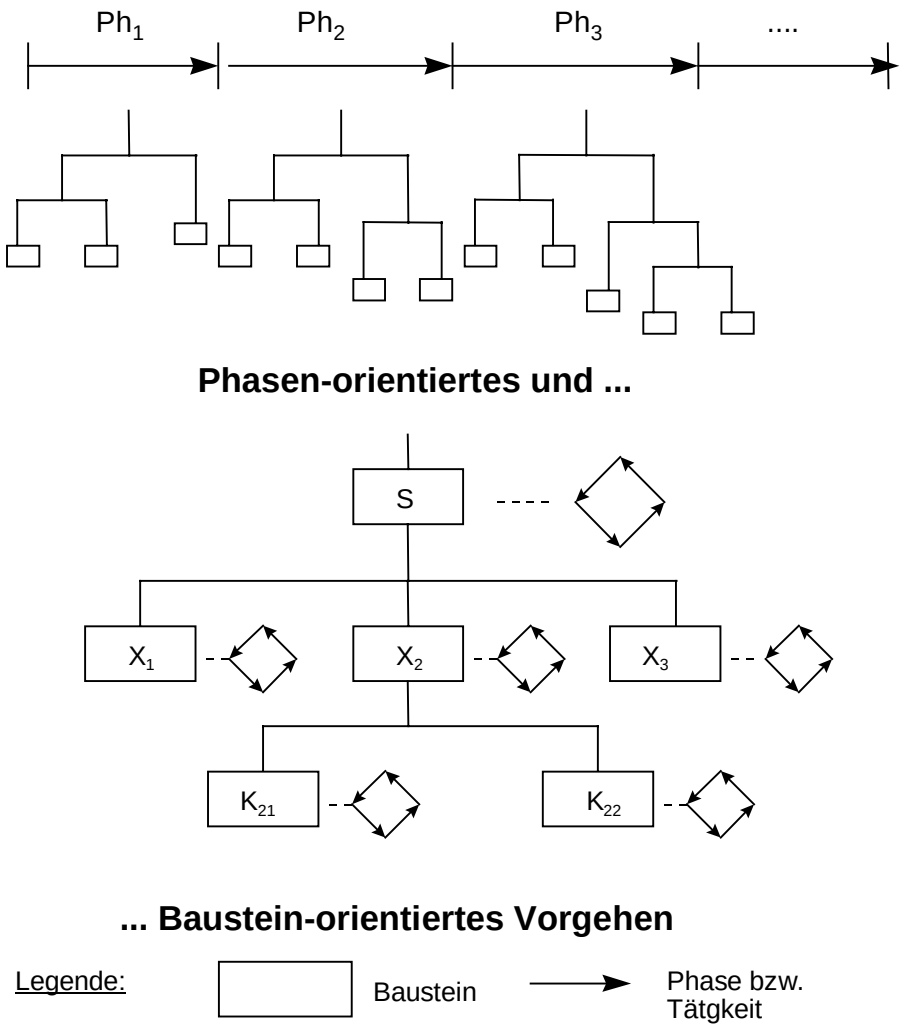


Abb. 1: Phasen- vs. Baustein-orientiertes Vorgehen

**Diskussion:** Das Fehlen von Wiederholungs-Möglichkeiten ist zu Recht als Mangel vieler Wasserfall-Modelle erkannt worden. Der RUP sieht zwar ausdrücklich Iterationen vor, bezieht diese allerdings auf die mögliche Wiederholung einzelner Phasen (oder des gesamten Entwicklungszyklus). So sehr die *generelle* Notwendigkeit von Iterationen betont wird, so wenig findet man eine plausible Begründung für *phasenweise* Wiederholungen. Dem Manager bieten solche Wiederholungen jedenfalls relativ wenig Hilfestellung, da sie in der Regel zu inflexibel sind. Für seine Planung wird die Wiederholung einer Phase in voller Breite lediglich auf eine Verlängerung der Phase hinauslaufen. In einem so arbeitsteiligen Prozeß wie der hier betrachteten Entwicklung großer Software-Systeme ist es jedoch höchst unwahrscheinlich, daß alle in der betreffenden Phase entstandenen Ergebnisse und Teilprodukte gleichzeitig revisionsbedürftig sind.

Dagegen müßte eine gezielte und detaillierte Ursachenanalyse (z.B. im Rahmen der im RUP geforderten *Assessments*) bei den *Bausteinen* der Entwicklung ansetzen und Wiederholungen gezielt dort anstoßen, wo sie benötigt werden - nämlich bei solchen Bausteinen oder Ergebnissen, die die gestellten Anforderungen noch nicht (ausreichend) erfüllen oder bei denen sich diese geändert haben. In der Regel wird also nicht die Wiederholung einer (projekt-weiten) Phase notwendig sein, sondern lediglich die

nochmalige Analyse und ggf. ein Neuentwurf und eine Neu-Implementierung des betroffenen Bausteins, d.h. die Wiederholung eines *auf den Baustein beschränkten* Entwicklungszyklus. Für das Management ergeben sich daraus möglicherweise Konsequenzen, die zu einer *strukturellen Planrevision* führen: abhängige Prozesse sind betroffen, neue Abhängigkeiten entstehen und müssen berücksichtigt werden.

Aus diesen Gründen sollten Iterationen nicht an Phasen, sondern an *Bausteine* und deren *Entwicklungszyklen* gebunden werden (vgl. [Hess 96]). Solche Bausteine können die bei der System-Modellierung definierten Pakete, Komponenten oder Subsysteme sein, aber z.B. auch Prototypen und wiederverwendete (und anzupassende) Komponenten: Diese durchlaufen zwar spezielle, auf den besonderen Fall zugeschnittene Zyklen, die aber alle eine analoge Grundstruktur aufweisen. Damit sind auch in einen solchen Modell Iterationen zu allen Zeitpunkten und auf allen Ebenen möglich.

**These 4:** Die *Kern-Arbeitsprozesse* des RUP verwischen die Prozeßstruktur und tragen nicht zu deren besseren Verständnis bei. Es handelt es sich bei diesen nicht um logisch zusammengehörige Prozesse, sondern lediglich um *Aktivitäten gleichen Typs*, die an verschiedenen Bausteinen durchgeführt werden und folgerichtig diesen zugeordnet sein sollten.

**Diskussion:** Die vertrauten Phasennamen früherer Modelle - Analyse, Entwurf (*design*), Implementierung, Test, Installation - tauchen beim RUP als sog. *core workflows* auf, die zu den o.g. Phasen in einer unscharfen Beziehung stehen. Diese wird durch die folgende Grafik (Abb. 2, hier kurz "RUP-Panorama" genannt) symbolisiert.

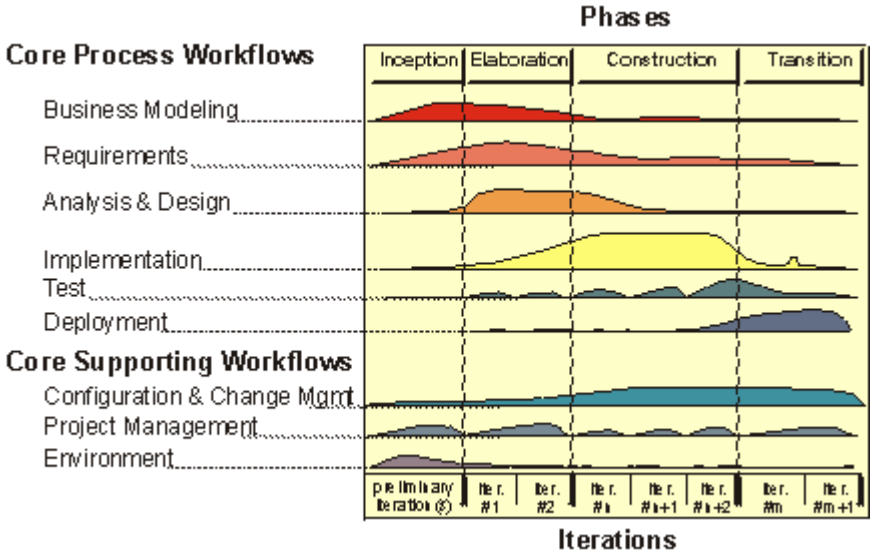


Abb. 2: Das RUP-Panorama

Dieses Diagramm legt den Schluß nahe, daß zwischen Phasen und Kern-Arbeitsprozessen eine "Beinahe-Korrespondenz" besteht: Der *requirements workflow* (= Anforderungssammlung) findet hauptsächlich in der *Inception phase* statt, Analyse und Entwurf in der *Elaboration phase* usw. Wozu braucht es dann diese beiden Konzepte nebeneinander und eine Verdoppelung der Phasen/Workflow-Bezeichnungen?

Weiter suggeriert die Bezeichnung *workflow*, daß sich hier kontinuierliche Analyse-, Entwurfs- und Implementierungsprozesse am Gesamtsystem vollziehen. Abgesehen davon, daß *Anforderungen* kein Prozeß und deshalb (allein sprachlich) hier fehl am Platze sind, verkennt (oder verschleiert) diese Sicht die Ursachen für die scheinbar kontinuierlichen Prozesse. Sie bestehen in der Vielzahl der *Bausteine*, an denen *Tätigkeiten dieses Typs* zu möglicherweise verschiedenen Zeitpunkten durchgeführt werden müssen. Macht man diese Bausteine sichtbar (vgl. oben), so entpuppen sich die "workflows" als simple Aktivitätstypen.

Für das Management bringen diese zwei Sichtweisen ganz unterschiedliche Konsequenzen mit sich: *Workflows* suggerieren eine *tätigkeits-bezogene Arbeitsteilung* und die Delegation der Arbeiten an Analyse-, Entwurfs-, Implementierungs- und Test-Spezialisten, während eine Kopplung von Aktivitäten an Bausteine eine *baustein-bezogene Arbeitsteilung* nahe legt: Entwickler bzw. kleine Teams sind für Bausteine und damit für alle daran auszuführenden Aktivitäten verantwortlich. Damit lassen sich Verantwortlichkeiten im Projekt leichter definieren und verfolgen (vgl. [Dene 91]).

**These 5:** Software-Entwicklungsprozesse sind oft hoch-komplex und verlangen deshalb mächtige Konzepte zur Komplexitätsbeherrschung wie hierarchische Strukturen und Rekursion. Der RUP macht hiervon zu wenig Gebrauch und bietet deshalb dem Management zumindest für große Projekte nicht genügend Unterstützung.

**Diskussion:** Typisch für Wasserfall-artige Vorgehensmodelle ist ihre lineare Grundstruktur mit i.W. sequentiell ablaufenden Phasen. Diese Strukturen waren für kleinere bis mittelgroße Projekte ausgelegt, für große, monolithische Projekte reichten sie gerade für eine Grobplanung aus. Heute stellen jedoch viele Projekte neue, erhöhte Anforderungen an das Management: Projekt-Umfang und -Komplexität sind im Durchschnitt ständig gestiegen (vgl. [IEEE 96]), dazu erfordern neue Anforderungen wie verteilte System-Plattformen, Wiederverwendung oder Komponenten-Technologie heterogene Entwicklungen und Projektstrukturen. Die zweistufige Phasen- und Iterationsstruktur des RUP reicht dafür - wie oben ausgeführt - vielfach nicht aus. Die Überlagerung mit den *Core Workflows* schafft dagegen zusätzliche, überflüssige Komplexität.

In der Informatik sind hierarchische Systemstrukturen und daran gekoppelte rekursive Algorithmen seit langem als geeignete Mittel zur Beherrschung auch hoher Komplexitäts-Anforderungen bekannt. Daher sollte ein modernes Vorgehensmodell derartige Strukturen enthalten. Prozesse werden (auch hier ganz im Sinne eines "objektorientierten Paradigmas) an die "Objekte" (=Bausteine) einer hierarchischen Systemstruktur, also etwa an Subsysteme, Komponenten, Pakete, Module, oder Klassen geknüpft und lassen sich dementsprechend rekursiv (d.h. in beliebiger Tiefe) in Unterprozesse aufgliedern. Für das Management ergeben sich dadurch klare Führungsstrukturen und Verantwortlichkeiten - allerdings verlangt eine Vielzahl parallel laufender Entwicklungsprozesse einen hohen Koordinationsaufwand.

**These 6:** Software-Entwicklung vollzieht sich im Zusammenspiel mehrerer nebenläufiger Teilprozesse, für die verschiedene Personengruppen (repräsentiert durch *Rollen*) verantwortlich sind. Der RUP trägt dem zu wenig Rechnung, vor allem dem Zusammenspiel von Entwicklungs- und Benutzungsprozessen.

**Diskussion:** Software-Entwicklung ist nicht allein ein technischer Prozeß, sondern ein komplexes Gewebe vieler nebenläufiger Aktivitäten, die wir grob den dafür hauptsächlich Verantwortlichen zuordnen können. Das RUP-Panorama deutet neben den Kern-Prozessen drei dieser "unterstützenden" Prozesse (hier ist die Bezeichnung *supporting workflow* berechtigt!) an: Projekt-Management, Konfigurations-Management, Umgebung(sprozesse). Wir sehen zumindest die *Qualitätssicherung* als einen weiteren wichtigen unterstützenden Prozeß an. Für die Behandlung unterstützender Prozesse in der Form von Submodellen könnte das deutsche V-Modell (vgl. [BrDr 95]) als Vorbild angesehen werden.

Mit dem Stichwort "Umgebung" (*environment*) weist der RUP auf das notwendige Wechselspiel zwischen Produkt-Entwicklung und der Vorbereitung und Anpassung der Umgebung an das Produkt hin. Der wichtigste Teil der Umgebung sind die Benutzer. Ch. Floyd sieht in ihrem STEPS-Modell Software-Prozesse als nebenläufige, sich gegenseitige beeinflussende Aktivitäten von Entwicklern und Benutzern [FRS 89]. Alle beteiligten Gruppen arbeiten kontinuierlich und über die gesamte Projekt-Laufzeit hinweg gemeinsam an einer Lösung. Im RUP sorgen zwar die allgegenwärtigen Anwendungsfälle (*use cases*) für die notwendige Berücksichtigung der Benutzer-*Anforderungen*, die o.g. kontinuierliche Kooperation und Evaluation spielt jedoch eine eher untergeordnete Rolle. Das zeigt z.B. ein Blick auf das RUP-Panorama: Dort ist der dafür zuständige Umgebungsprozeß mit der *"Inception"*-Phase fast schon erledigt.

# 3 Fazit, Ausblick

Im vorangegangenen Abschnitt wurde der RUP anhand von sechs Thesen kritisch beleuchtet. Wesentliche Kritikpunkte betreffen das Festhalten an einer nach wie vor dominanten Phasenstruktur, während die Rollen der Software-Architektur und der Rekursion als Mittel zur Beherrschung komplexer Prozesse zu wenig zum Tragen kommen. Dem phasen - und transaktions-orientierten RUP wurde eine baustein- und dokumenten-orientierte Arbeitsweise als Alternative gegenübergestellt, die für das Management Vorteile mit sich bringt.

Die Diskussion und ihre Vorgeschichte haben ferner gezeigt, daß der Versuch einer Vereinheitlichung von Vorgehensmodellen prinzipiell problematisch ist. Eine praktikable Alternative besteht darin, Software-Prozesse statt mit vorgefertigten Modellen lieber mit einem "Werkzeugkasten" zu unterstützen, der z.B. deren Basismaterial in Form von Beschreibungen für Aktivitäten, (Ergebnis-) Dokumenten, Rollen, Techniken etc. bereitstellt und dazu verschiedene Vorgehens-Varianten anbietet [HeNo 99]. Solche Varianten zeigen (etwa im Sinne von *guided tours*), wie sich aus diesem Basismaterial praktikable Prozesse zusammenstellen lassen, die auf die spezifischen Belange des Unternehmens und der verfolgten Projektziele zugeschnitten sind.

# Literaturhinweise

[Booc 94] G. Booch: *Object-Oriented Analysis and Design with Applications;* 2nd Edition, Benjamin/Cummings Publ. Comp. 1994

[BrDr 95] A.-P. Bröhl, W. Dröschel (Hrsg.): *Das V-Modell.* Der Standard für die Softwareentwicklung mit Praxisleitfaden. 2. Auflage. Oldenbourg 1995

[Dene 91] E. Denert*: Software Engineering - Methodische Projektabwicklung*; Springer 1991

[Dene 93] E. Denert: Dokumentenorientierte Software-Entwicklung, in: *Informatik-Spektrum 16/3,* pp. 159-164 (1993)

[FRS 89] Ch. Floyd, F.-M. Reisin, G. Schmidt: STEPS to software development with users. In: *C. Ghezzi, J. McDermid (eds.): ESEC '89, Second European Software Eng. Conference,* LNCS 387, pp. 48-64. Springer 1989

[Hess 96] W. Hesse: Theory and practice of the software process - a field study and its implications for project management; in: *C. Montangero (Ed.): Software Process Technology, 5th European Workshop, EWSPT 96*. Springer LNCS 1149, pp. 241-256 (1996)

[HeNo 99] W. Hesse, J. Noack : A Multi-Variant Approach to Software Process Modelling. In: *M. Jarke, A. Oberweis (Eds.): CAiSE'99,* LNCS 1666, pp. 210-224 (1999)

[IEEE 96] T. DeMarco, A. Miller (Guest Editors): Managing Megaprojects; Special Issue, *IEEE Software 13(3),* July 1996

[Jaco 93] I. Jacobson: *Object-Oriented Software Engineering - A Use Case Driven Approach;* Revised Printing, Addison- Wesley 1993

[Jaco 99] I. Jacobson: *Applying UML in The Unified Process.* Presentation material. Rational Software Corp., Santa Clara, CA 1999

[JBR 99] I. Jacobson, G. Booch, J. Rumbaugh: *The Unified Software Development Process.* Addison-Wesley 1999

[Kruc 95] P.B. Kruchten: The 4+1 view model of architecture. *IEEE Software 12 (6),* pp. 42-50 (1995)

[Kruc 99] P.B. Kruchten: *The Rational Unified Process (An Introduction).* Addison Wesley 1999

[RBP+ 91] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, W. Lorensen*: Object-Oriented Modelling and Design;* Prentice Hall 1991

[UML 97] G. Booch, J. Rumbaugh, I. Jacobson: *Unified Modeling Language (UML) for Object-Oriented Development (Documentation Set - Version 1.1).* Rational Software Corp., Santa Clara, CA 1997

# Software-Projektmanagement braucht klare Strukturen -

# Kritische Anmerkungen zum "Rational Unified Process"

Positionspapier

Wolfgang Hesse, Universität Marburg

## Zusammenfassung:

Seit 1999 hat die Firma Rational ihrer Modellierungssprache UML den *Rational Unified Process (RUP)* als "Vereinheitlichtes Vorgehensmodell" zur Seite gestellt. Dadurch soll "die Produktivität von Entwickler-Teams verbessert" und den Projektleitern "bessere Kontrolle über Pläne und Projektergebnisse" gegeben werden. Weiter wird der RUP als Modell für "iteratives und inkrementelles, Anwendungsfall-getriebenes und Architektur-zentriertes Vorgehen" angekündigt. In sechs Thesen werden diese Ziele bzw. deren Verwirklichung durch den RUP aus der Sicht von Software-Entwicklern und Projektleitern kritisch hinterfragt.

**Adresse des Autors:**

Prof. Dr. Wolfgang Hesse
FB Mathematik und Informatik, Universität Marburg
Hans Meerwein-Str.
D-35032 Marburg
Germany

Tel.: +49-6421-282 1515, Fax: +49-6421-282 5419, email: hesse@informatik.uni-marburg.de