Dinosaur Meets Archaeopteryx?

Seven Theses on Rational's Unified Process (RUP)

Wolfgang Hesse

c/o FB Mathematik und Informatik, Universität Marburg,
Hans Meerwein-Str., D-35032 Marburg
email: hesse@informatik.uni-marburg.de

Abstract

This position paper summarises some critical arguments on the Rational Unified Process (RUP). In particular, claims advertising the RUP e.g. to be *iterative* and *architecture-centric* - are discussed and its core concepts like *phase, iteration, workflow* and *milestone* are investigated. It is argued that their definitions and relationships to each other lack clear structure and are too complex and overloaded for practical use. Major sources of these problems are the adherence of the RUP authors to a phase-oriented process structure, their underestimation of the software architecture and of powerful structuring principles like recursion and orthogonality.

1 Introduction: The "Unified Process" and its core concepts

In the 1990'ties Rational company started its "Unified Method" project aiming to join the various, then existing methodologies for object-oriented analysis and design. To achieve this goal a "merge" of the methods of Rational's chief methodologists G. Booch, J. Rumbaugh and (later) I. Jacobson with contributions from many other authors was attempted. When it became apparent that the original goal was too ambitious the authors decided to proceed in two steps. The first step was to unify the various notations and resulted in the "Unified Modeling Language" (UML) - now documented in its version 1.3 from June, 1999 [UML 99].

In early 1999, the second step was done by the publication of the so-called "Rational Unified Process" (RUP) - now documented by two books [JBR 99], [Kru 99], in the Web [RUP 99] and by further presentations of its authors and other people. With their process description, the authors claim to "enhance team productivity" and to "give project managers control over schedules and deliverables". Further, the RUP is advertised as being "iterative and incremental, use case-driven, and architecture-centric".

In this position paper I attempt a critical review of the RUP. One general argument concerns its over-sophistication. A primary goal of the RUP is to support software engineers working with UML. In a recent paper, K.D. Schewe has called UML a "modern dinosaur" [Sce 00]. Having struggled through the vast jungle of RUP documents, readers might get the impression that the dinosaur UML has got an equally overgrown (and even less viable) companion - the *archaeopteryx* RUP.

According to the RUP, the software life cycle is decomposed into *phases* which may be subject to several *iterations*, are interwoven with *core workflows* and terminated by *milestones*. In the following sections these core concepts of the RUP are discussed and it will be argued that

- *phases* are no longer appropriate means for structuring projects which follow modern development paradigms like object orientation, component-based development, incremental or evolutionary approaches,

- accordingly, *milestones* have to be replaced by a more refined concept, linked to the conclusion of *certain activities* rather than to the termination of "phases",

- *iterations* should indeed be addressed in the process model but rather be linked to architectural units (the software *building blocks*) and *products* than to phases,

- *workflows* are an overloaded concept and in particular, the RUP *core workflows* are misnamed and interwoven with the phases in a dubious and over-sophisticated way.

One major source for these problems is the adherence of the RUP authors to a *phase-oriented process struct*ure and (in contrast to their own claims) their *underestimation* of the *software architecture*. Another source is their *monolithic view* on the software process ignoring its *recursive structure* and its possible decomposition into subprocesses which are associated with the involved *roles* (stakeholders) and which include the *user's process(es)* and their feedback.

In the following main section, these problems are addressed and discussed in more detail. It will then be summarised that, due to the indicated problems, the RUP - at least in its present form - falls short of achieving its own goals, in particular to give more and better support to both software developers and managers.

2 Seven theses on Rational's Unified Process

In order to maintain the style of a "position paper" I shall formulate my critical arguments as *seven theses* - accompanied by explanatory remarks and proposed alternatives. With these theses, a (for the sake of brevity) condensed but thorough and systematic analysis of the RUP is attempted. Since the various aspects of software processes are interrelated in many ways, the corresponding theses cannot be completely independent from one another – but I have tried to focus on one particular aspect with each thesis:

- Thesis 1 deals with the time dimension of software processes – in particular with the role of *phases*,

- Thesis 2 is concentrated on the software *architecture* and its relevance for the RUP,

- Thesis 3 addresses the importance of *iterations* and the question where they should be anchored,

- Thesis 4 investigates one of the most prominent RUP constructs, the so-called *"workflows"*,

- Thesis 5 contains general observations on concepts for *mastering complexity* such as recursion and orthogonality and their use (or ignorance) in the RUP,

- Thesis 6 focuses on the usefulness of the RUP for *project management* and

- Thesis 7 considers the *organisation* of software processes and in particular RUP's attitude towards the *users and their involvement*.

Of course, such an analysis can never be complete but I have concentrated on those aspects that seem most important and in particular on those where a constructive alternative is at hand.

Thesis 1: *The RUP is based on a phase oriented software life cycle model which is no longer adequate to support most contemporary development approaches.*

Remarks: According to the RUP descriptions, the software development process is decomposed into *phases* which form the basis for further definitions of iterations, workflows ets. In the RUP, phases have been given new names: *inception, elaboration, construction* and *transition.* Instead, well-known phases of traditional life cycle models like *analysis, design, implementation* and *test* have been converted to so-called *workflows* (cf. discussion below). Their co-existence is a relict from Jacobson's Objectory process [Jac 93] where the management process was distinguished from the development process by different phase identifiers. However, such a terminological distinction is rather confusing and has not proven to be helpful.

Phases are a well-established concept of traditional life cycles (in particular: the waterfall-like ones) but obviously their role becomes less important if software systems are no longer treated as monolithic blocks but are to be built from several sub-products or components which co-evolve in an asynchronous and independent way. This holds true for most modern development approaches like object orientation, component-based development, incremental or evolutionary approaches. Thus phases should no longer play the prominent role they had played in the first decades of Software Engineering.

Alternative: Instead of sticking to waterfall-like phases, a modern process model should be based on a truly "architecture centric" structure. This means: activities and iterations should be linked to architectural units rather than to phases (s. below).

Thesis 2: *In contrast to its authors' claims, RUP is not an "architecture centric" process but it is still dominated by phase structure.*

Remarks: As we have seen, the RUP is decomposed into phases and these in turn into iterations. Neither of them refer to the software architecture. RUP *workflows* overlap with phases (in a rather dubious manner, cf. below) and are associated with "models" - which serve for making the RUP "*architecture centric*" according to its author's claims. However, models do not constitute the architecture but *"… are vehicles for visualizing, specifying, constructing, and documenting architecture*." [JBR 99]. Thus the RUP activities are not *centred* on the architecture but are model-building steps eventually *resulting* in an "architecture".

Alternative: An "architecture centric" approach (which deserves this name) would associate activities, iterations (cycles), revisions, quality checks, management actions etc. rather with the *objects of software development,* i.e. *architectural units* like components, modules, subsystems, prototypes, .. than with phases. This way, well-known management problems with phases (phase overlaps, coordination of iterations, milestone definition etc.) can be avoided. Furthermore, such an approach would support the identification of developers with their (sub-) product and thus facilitate project management [D-L 87] .

Thesis 3: *The RUP does well in introducing iterations in the software development process, but there is much less need for phase iterations than for (sub-) product development cycles.*

Remarks: Iterations and development cycles are indeed an important issue for any practice-oriented process model. But before we just add repetition loops to the phases of a waterfall model we have to analyse the reasons for an iteration: Is it the demand to repeat an unsatisfactory phase or is it rather the defects of a subproduct or the insufficiencies of a component which causes the need for repeating its development cycle? Of course, both cases do occur. But whereas the first case - at least from the project management point of view - can be handled by just prolonging a phase, the second requires much more action and re-planning. Take the case that severe defects require a re-design of a whole component. Before the re-design is started, an analysis of the observed problems and of possible ways for revision is required. Thus the defects lead to a whole *(sub-) product re-development cycle* rather than to just an iteration of a single phase. Such a re-development cycle

requires profound management action and in extreme cases it may even lead to a complete revision of the project plan.

Alternative: Iterations should be foreseen in any software process model but they should be bound to architectural units like components, subsystems or modules rather than to phases. Such an approach reflects the demand for re-use of components in a natural way: In order to adapt a class or component for re-use one has to start a re-development cycle on that product. Normally, such a cycle comprises all activities ranging from analysis over (re-) design, implementation and test and use of that product.

Thesis 4: *The RUP concept of workflows adds unnecessary complexity to the process. The so-called "core workflows" are misnamed and just activities of the same or a similar kind. They overlap with phases in a confusing way and do not contribute to a clear, transparent process structure.*

Remarks: The RUP authors have decided to rename their former "process components" into "workflows". According to Kruchten's glossary, a workflow is a *"sequence of activities performed in a business that produces a result of observable value to an individual actor of the business"* [Kru 99, p. 239 ff]. But there are at least four possible interpretations of this rather vague definition [Hes 01].

In particular, the RUP documents list five so-called *core workflows*: *Requirements capture* (now corrected from former misnamed "requirements") - *analysis* - *design* - *implementation* - *test*. Obviously, the RUP authors have observed that they represent overlapping process components and thus are no longer to be treated as "phases". But the new term "*core workflows*" suggests that there is e.g. a "flow" of analysis activities (crossing the phase boundaries) eventually resulting in the analysis model (and correspondingly for design, implementation and test "flows").

If we look at the central graphical illustration of the RUP (the "RUP panorama", cf. fig. 1) we observe that the core workflows have their peak intensity in corresponding phases: Requirements caption in the inception and elaboration phase, analysis & design in the elaboration phase etc.

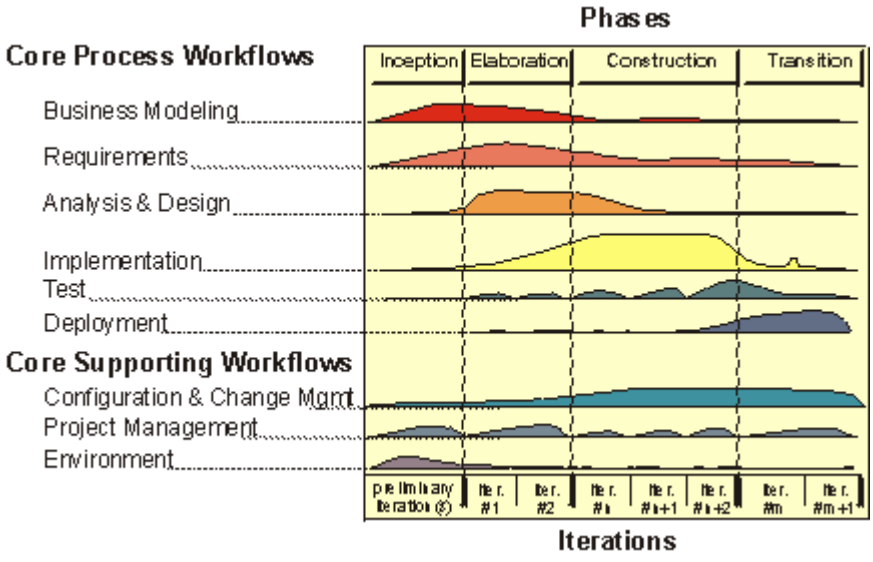

Fig. 1: RUP phases and core workflows

Not surprisingly, we find an almost 1:1 mapping between workflows and phases - but with the workflows somewhat frayed at their ends. What justifies the doubling of terms for the phases and (almost) corresponding workflows? And furthermore, we have to ask again for the reasons which give rise to group various activities to "workflows": Is it really a "flow" of work going from the analysis of one component to that of another? No! In most cases it is just the *same kind of activity* which is applied to different components.

Alternative: Like workflows in other engineering disciplines, activity sequences (analysis, design, implementation, test, etc.) should be organised around *products* and their *components*. Again, the architectural units of a software system are the natural anchor points for such activity sequences. Since they can be repeated, we call them "development cycles" - they are exactly the architecture-based iterations considered above. Such cycles have a clear starting and termination point and consist of determined activities with defined, verifiable results like "analysis of component A", "design of module B" etc. All together they constitute the *(recursive) structure* of the overall software development process (cf. below).

Thesis 5: *The RUP does not offer appropriate support for structuring complex software processes. It ignores most powerful mechanisms of computer science for mastering complexity: hierarchy, recursion and orthogonality.*

Remarks: Many traditional life cycle models worked satisfactorily with normal-size projects (up to, say, 10 person-years effort) but did not offer much help for very large and complex projects. One of their major shortcomings was their *monolithic view* on the software process: "Phases" like analysis, design, implementation, test always concerned *the one, unique product* - the software system as a whole. This is just the same for the "new" RUP phases inception, elaboration etc. and leads to very complex activity and "workflow" structures.

However, in order to master the complexity of very large systems these have to be decomposed into smaller, manageable units - and so on. This way, *component hierarchies* may be formed using the *recursion* principle for their definition. Why should processes not be structured according to the products they belong to? Doing this we arrive at our next alternative (for more details cf. the EOS cycles described in [Hes 96]and [Hes 97]).

Alternative: Complex software development processes are decomposed into sub-processes in a *recursive way* according to the (sub-) products they belong to: As soon as a component is identified and conceived to encapsulate some piece of system functionality, its own development processes is initiated starting with requirements capture (for that component), followed by analysis, design etc. This schema applies to any further component, possible sub-components, modules etc in an *orthogonal* way.

Thesis 6: *Due to its lack of transparency and structural flexibility, the RUP does not appropriately support management -, in particular that of large projects. The RUP concept of milestones is too weak for complex coordination tasks.*

Remarks: A practicable software process model has to offer clear evaluation criteria and control instruments to software *project managers.* As shown in the above theses, the RUP falls short of offering useful support to the managers for their tasks. Criteria like *"use case model 10%-20% complete"* or *"use case model at least 80% complete"* (cf. [Kru 99], pp. 65/68) are not helpful. In the era of object orientation and component-based development, particular support is required - e.g. for decomposing processes and co-ordinating the resulting sub-processes. *Milestones* are a well-known management instrument adopted by the RUP from the traditional life cycle models. Milestones are necessary and important - but should no longer be linked to the completion of *phases*, instead, more accurate criteria and mechanisms for defining milestones are required.

Alternative: A hierarchical, recursively defined process structure based on the software architecture can be mirrored in a refined concept of milestones - which I prefer to call *revision points* [Hes 96]). A *revision point* defines an (anticipated, planned) state for each (sub-) product (= component) under development. These states need not be the same for all involved components. E.g. revision point R1 may imply component A to be in the state "design completed", component B to be in the state "analysis completed" etc. To define and maintain a series of complex milestones requires considerable management effort but it helps clarifying the inherently complex project structure and is thus much more helpful than the over-simplifying phase-based traditional milestones.

Thesis 7: *The RUP does not satisfactorily address the roles and interactions of various groups concerned with the software process, in particular the role of the users and their feedback on the process is neglected.*

Remarks: The overall software process is focused on software development but it is much more than that: It encompasses the roles and activities of other people concerned like project managers, quality professionals, supporters, tool builders and, last not least, *users*. An evolving software process is always paralleled by a *use process.* This process implies testing (sub-) products, prototypes or other intermediate results and giving the necessary feedback to the main process stakeholders [FRS 89]. The RUP phase of "transition" does not satisfactorily address this important point, neither does the use case analysis that is focused on the very early RUP stages.

Alternative: An overall software process model covering all its important aspects has to foresee concurrent processes for *development, management, quality assurance, support* and, in particular, for the *use and evaluation* of (partial) products - including the users' *feedback* to the other processes (cf. fig. 2). The importance of the users' role and their activities should be reflected by including a "use and revision" activity in the main development cycle and/or by extending the transition phase with this respect.

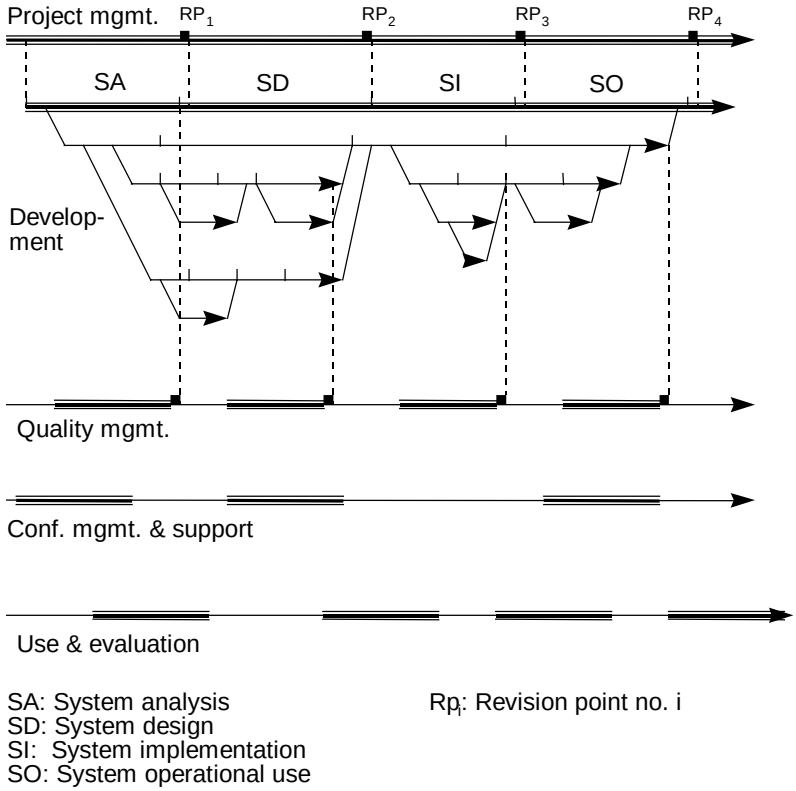

Fig. 2: The overall structure of the software process

3 Conclusions

In the previous theses, I have critically reviewed the overall structure of Rational's Unified Process and I have illuminated some points which I consider problematic for software projects following a modern development paradigm. In particular, the RUP concepts of phase, iteration, core workflow, milestone and the question of user involvement have been addressed. The alternatives I have presented can be found in a coherent form in the EOS model [Hes 96], [Hes 97].

In this respect, the question arises whether a process model should prescribe a certain process at all or better offer the software engineers some sort of toolbox for designing their own, individually tailored processes. This has lead us to our multi-variant approach (cf. [H-N 99]) which might give some impulse for a more flexible RUP. Anyway, I am sure that there will be many deviations from the "pure" RUP and much more discussion on its particular concepts and decisions. Such a discussion should help to improve the RUP and make it a real, valuable tool to support software developers, managers and other people involved.

References:

[D-L 87] T. DeMarco, T. Lister: Peopleware - Productive projects and teams; Dorset House Publ. Co. 1987.

[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

[Hes 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)

[Hes 97] W. Hesse: Improving the software process guided by the EOS model. In: *Proc. SPI '97 European Conference on Software Process Improvement*. Barcelona 1997

[H-N 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)

[Hes 01] W. Hesse: RUP - A process model for working with UML? Critical Comments on the Rational Unified Process - Book chapter in: *K. Siau et al. (eds): Unified Modeling Language*. Idea Group Publ. 2001

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

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

[Kru 99] Ph. Kruchten: The Rational Unified Process (An Introduction). Addison Wesley 1999

[Roy 98] W. Royce: Software Project Management - A Unified Framework, Addison Wesley 1998

[RUP 99] Rational Unified Process- Product Overview. http://www.rational.com/products/rup as of 15th Sept. 1999

[Sce 00] K.D. Schewe: UML: A Modern Dinosaur? – A Critical Analysis of the Unified Modelling Language. Proc. 10th European-Japanese Conf. on Information Modelling and Knowledge Bases. Saariselkä/Finland

[UML 99] Unified Modeling Language (UML) 1.3 Documentation. OMG document ad/99/06-09. Rational Software Corp., Santa Clara, CA 1999. http://www.rational.com/uml/resources/documentation as of 24th May 2000

Author's address:

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

Tel.: +49-6421-2821515, Fax: +49-6421-2825419,
email: hesse@informatik.uni-marburg.de

Length of paper: 3373 words