Software and Systems Modeling (SoSyM) Vol. 2. No. 4, pp. 240-247 (2003)

Expert's Voice

Dinosaur Meets Archaeopteryx?

or: Is there an Alternative for Rational's Unified Process?

Wolfgang Hesse

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

Abstract

Since 1999, *Rational's Unified Process (RUP)* is being offered as a guideline for software projects using the Unified Modeling Language (UML). RUP has been advertised to be *iterative*and *incremental*, *use case-driven* and *architecture-centric*. These claims are discussed while RUP core concepts like *phase, iteration, discipline (formerly: workflow)* and *milestone* are reviewed in more detail. It turns out that the RUP constitutes a considerable step towards a broad dissemination of software process modelling ideas but some of the RUP definitions and structures lack clear structure and are too complex and overloaded for practical use.

Among others, I see the following particular problems: (1) phases do still dominate the process and iteration structure, (2) the term "software architecture" is not clearly defined and its role is still underestimated, (3) RUP "disciplines" are a partly redundant concept complicating the process more than supporting it, (4) powerful and transparent structuring principles like recursion and orthogonality do not get the attention they deserve. As an alternative, our model for *Ev*olutionary, *O*bject-oriented *S*oftware development (EOS) is contrasted with the RUP.

1 Introduction: The RUP and its core concepts

In the 1990'ties Rational company aimed to unify the various, then existing methodologies for object-oriented analysis and design into a "Unified Method". This project was realised in two steps: (1) designing and publishing the "Unified Modeling Language" (UML) as a notation for any kind of software modelling results [UML 99], (2) complementing UML by a paradigmatic, idealised process description - the RUP - which is well documented by two books ([JBR 99], [Kru 99], in the Web [RUP 99] and by further presentations of its authors and other people.

By the RUP approach, its authors claim to "enhance team productivity" and to "give project managers control over schedules and deliverables". Furthermore, the RUP is advertised as being "iterative and incremental, use case-driven and architecture-centric". In the following section, these claims shall be discussed - some of them in more detail. In general, it is argued that RUP suffers from its oversize and its over-sophistication. A primary goal of the RUP is to support software engineers working with UML. In a former paper, K.D. Schewe has called UML a "modern dinosaur" [Sce 00]. Having struggled through the vast jungle of RUP documents, guidelines and tools, readers might get the impression that the dinosaur UML has got an equally oversized (and even less viable) companion - the *archaeopteryx* RUP.

The RUP still does (as did the famous *waterfall models*) decompose the software life cycle into *phases* which may be subject to several *iterations*, consisting of *activities* which are interwoven with so-called *disciplines* and which are terminated by *milestones*. In the following section these RUP core concepts are discussed and it will be argued that

- *phases* should no longer be the primary concept for structuring projects - at least not for those which follow modern development paradigms like object oriented, component-based, incremental or evolutionary development,

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

- *iterations* should indeed play an important role in the process model but rather be linked to architectural units (the software *building blocks* or *artefacts*) than to phases,

- *disciplines* (which replaced the former *workflows*) are an overloaded, partly redundant concept and overlap the phases in a fuzzy and over-sophisticated way.

Major reasons for these problems seem to be that the RUP authors still stick to *phases* as the dominating process structure and that - despite of their own claims - they underestimate the *software architecture*. Another source is their *monolithic view* on the software process ignoring its *recursive (fractal)* and *orthogonal structure*. Furthermore, certain *roles* (stakeholders) and their associated (sub-) processes such as quality assurance, user evaluation and feedback are not dealt with in an appropriate manner. Summing up, 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.

Instead of designing always bigger, more sophisticated and hypertrophic process models I recommend to adopt a different view and not to neglect more than 30 years of Software Engineering experience. This experience has taught us to build large and complex software systems as compositions of recursively defined, self-similar and (to a certain degree) self-contained units - like classes, modules or components. Why do we not structure software processes in an analogous way and decompose them into smaller, self-similar subprocesses resulting in a *fractal process structure* (cf. also [Stö 01])?

This is the basic idea behind our model for *Ev*olutionary, *O*bject-oriented *S*oftware development (short: EOS, cf. [Hes 96], [Hes 97]) which will be summarised in section 3. In this model, the software development process is viewed as a composition of cyclic, (mostly) concurrent, fractal subprocesses which all follow a unique pattern and - together with accompanying processes like project management or quality assurance - form a unique, generic, scalable and widely applicable scheme for any kind of Software Engineering projects.

2 Seven theses on Rational's Unified Process

In this section, I present my critical arguments on the RUP as *seven theses* - accompanied by explanatory remarks and proposed alternatives. As a whole, these theses are to give a condensed but thorough and systematic analysis of the RUP from a pragmatic point of view. Since the various aspects of software processes are interrelated in many ways, the corresponding theses cannot be completely independent from one another – but I try to focus on one particular aspect by 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 *"disciplines"* (former *"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 as a whole and in particular RUP's attitude towards the *users and their involvement*.

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

Thesis 1: *The RUP still maintains a phase-like software life cycle model which is no longer adequate to support most contemporary (in particular: component-based) 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, activities etc. 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* (later renamed into *disciplines*). 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 paradigms like the object oriented, component-based, incremental or evolutionary approaches. Thus phases should no longer play the prominent role they have 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 (see below, thesis 3).

Thesis 2: *In contrast to its authors' claims, RUP is not an "architecture centric" process but it relies on a diffuse view on software architecture and 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 *disciplines* overlap with phases 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]. On the other hand, the UML authors define *architecture* as "the organisational structure of a system. An architecture can be recursively decomposed into parts that interact through interfaces, relationships that connect parts, and constraints for assembling parts." [UML 99]. In this sense 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. Such an approach is taken by the EOS model sketched in the following section. It helps to avoid well-known management problems with phases (phase overlaps, coordination of iterations, milestone definition etc.). Furthermore, it is meant to encourage the developers to *identify* themselves with their (sub-) product(s) and thus supports their creativity and self-organisation [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 development cycles centred on (sub-) products.*

Remarks: Iterations and development cycles are indeed an important issue for any process model aiming for practical use. But before just adding repetition loops to the phases of a waterfall model we have to analyse *why* iterations are used at all: Is it the demand for repeating an unsatisfactory *phase* or rather the defects or insufficiencies of a (sub-) *product* (like a component or module) that cause a need for repetition? Of course, both cases do occur. But while 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. Assume 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 cycle requires profound management action and in extreme cases it may even lead to a complete revision of the project plan.

Alternative: Iterations should clearly be foreseen in a modern software process model but they should be bound to architectural units like components, subsystems or modules rather than to phases. This key feature of the EOS model (cf. section 3) reflects the demand for self-contained development and re-use of components in a natural way. For example, 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 disciplines adds unnecessary complexity to the process. The former so-called "core workflows" were misnamed and are 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 repeatedly decided to rename their former "process components" - first into *"workflows"*, then into *"disciplines"*. This terminological confusion reflects their difficulties to motivate and explain this concept. In former papers, I have shown that the term "workflow" was inadequate - at least for the 5 former "core workflows" [Hes 01] - a view which eventually seems to be shared by the authors as the renaming shows.

In particular, the RUP documents list five *core disciplines*: *Requirements capture* (now unfortunately re-renamed to "requirements" - which is neither a phase nor a discipline from a linguistic point of view!) - *analysis & design* - *implementation* - *test* - *deployment*. Obviously, the RUP authors have observed that they represent overlapping process components and thus are no longer to be treated as "phases". But the term "*core workflows*" suggested that there is e.g. a "flow" of analysis activities somehow crossing the phase boundaries and eventually resulting in the analysis model (and corresponding for design, implementation etc. "flows").

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

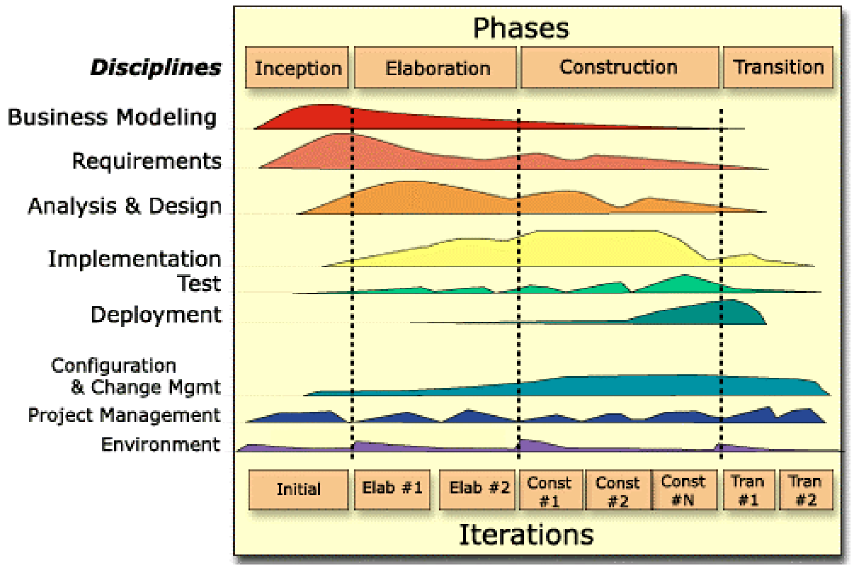

Fig. 1: RUP phases, disciplines an iterations

Not surprisingly, we find an almost 1:1 mapping between phases and disciplines - but with the latter somewhat frayed at their ends. What justifies the doubling of terms for the phases and (almost) corresponding disciplines? And what is it that analysis & design activities have in common in the elaboration and construction phase? Is it really a "workflow" 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. Thus the "core workflows" or "disciplines" turn out to be simple *activity types* - not to be confused with certain *activity sequences* which deserve an extra process construct (cf. the EOS development cycles below).

Note that the last arguments do not apply for the rest of the disciplines (business modelling, conf. & change management, project management, environment) where I (more or less) share the RUP point of view. However, at least two important "disciplines" - *quality assurance* and *the use and evaluation* of all kinds of project results - have been forgotten (cf. thesis 7).

Alternative: Like workflows in other engineering disciplines, special activity sequences (for example, consisting of analysis, design, implementation, test, and deployment) 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 normally are 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 *fractal structure* of the overall EOS 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* - i.e. the software system as a whole. This still holds true for the "new" RUP phases inception, elaboration etc. and it leads to very complex activity and "discipline" structures.

However, in order to master the complexity of very large software projects, their processes have to be decomposed into smaller, manageable units - in analogy to forming *component hierarchies* using the *recursion* principle for their definition. Why shouldn't we structure processes according to the products they belong to? This is exactly the way proposed by the EOS model to master complex software processes.

Alternative: Complex software development processes are decomposed into sub-processes in a *hierarchical, 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 analysis and requirements capture (for that component), followed by design, implementation etc. This schema applies to any further component, possible sub-components, modules etc in an *orthogonal* way. For more details cf. the EOS cycles described in [Hes 96] and [Hes 97].

Thesis 6: *Project management is not given adequate support by the RUP - due to its lack of transparency, scalability and structural flexibility. The RUP does not offer clear criteria for the termination of tasks and phases and its milestone concept is too weak for complex coordination tasks.*

Remarks: A process model meant to be applicable for projects of any size must be scalable. Due to its one-dimensional phase structure RUP does not satisfactorily support this requirement. Further, a practicable software process model has to offer clear evaluation criteria and control instruments to software *project managers.* However, 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 for project managers in real-life projects. As every experienced manager knows only *100% complete* is a reliable status information. While this normally cannot be stated (and verified) for one large, complex process (for obvious reasons) it can be a very useful criterion for its many small component subprocesses.

Therefore, particular management support is required 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 associated with the completion of *phases*. Instead, more elaborated criteria and mechanisms are needed for project control

Alternative: As a counterpart to the milestones of waterfall-like models EOS offers the *revision point* concept which corresponds to its overall hierarchical, recursively defined process structure. A revision point defines an (anticipated, planned) state for each (sub-) product (= component) under development [Hes 96]. These states need not be synchronous for all components in work at a certain point of time. E.g. revision point R1 may state *component A* to be in the state *"design completed"*, while *component B* still is in the state *"analysis completed"* etc. To define and maintain a series of complex revision points requires considerable management effort but it helps clarifying the inherently complex project structure and thus it is much more helpful than the over-simplifying 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 almost neglected.*

Remarks: Software processes are always focused on software *development* but they are much more than that: They encompass the roles and activities of other people involved like project managers, quality professionals, supporters, tool builders and, last not least, *users*. Every software process which has to do with application should be paralleled by a *use process* [FRS 89]. This use process implies testing (sub-) products, prototypes or other intermediate results and giving the necessary feedback to the main process stakeholders. In the RUP, neither the use case analysis which is limited to the very early project stages nor the "transition" phase focused on the very last stages do satisfactorily address this important point.

Alternative: In the EOS model, the importance of the users' role and their activities is not only reflected by including a "use and revision" activity in each development cycle but also by an extra subprocess of the overall software process which covers - among other roles and their subprocesses - the *use and evaluation* of all products resulting from the development activities (cf. also the following section and fig. 3 below).

3 EOS - a practicable alternative for the RUP

In this section, I try to summarise the above arguments and simultaneously give a brief outline of the main features of the EOS model. From my point of view, a generic (i.e. widely applicable) process model has to follow a certain *vision* and has to offer some indispensable *key features*:

- Software development is a complex process which can best be managed by decomposing it into smaller, self-similar subprocesses resulting in a *fractal process structure*. Instead of associating iterations, activities, artefacts and milestones with phases as do the conventional life cycle models (including the RUP), processes (and their phases) should be linked to the system structure and its main architectural units (cf. fig. 2)

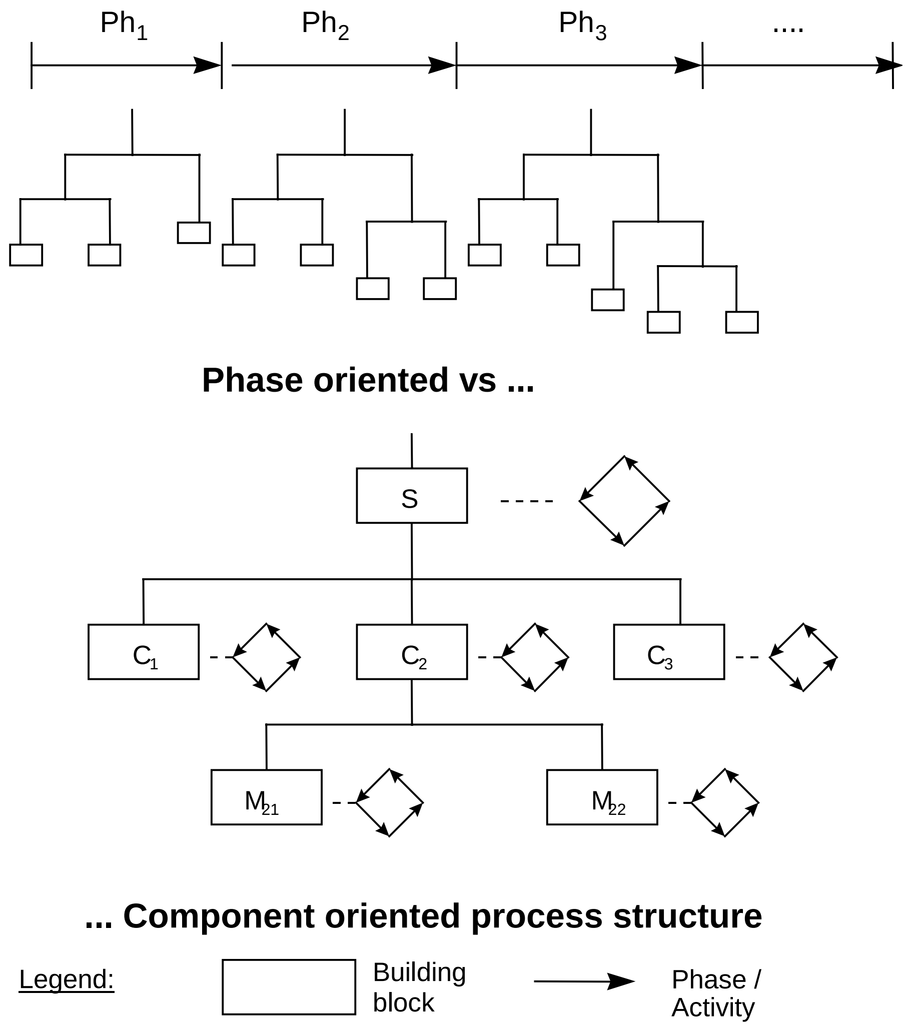

Fig. 2: Two approaches to structure the software process

- All processes are structured in an analogous way by the four main process phases *analysis*, *design*, *implementation* and *operational use* - depicted by the four arrows in the lower part of fig. 2. Treating them as what they are - simple activity types - avoids the definition of dubious concepts like workflows or disciplines. Instead of exposing a complex network of phases, iterations, disciplines and activities as the RUP does, EOS can do with an adapted description of development cycles and their associated process phases and activities on three refinement levels - the system (S.), component (C.) and module (M.) level (cf. fig. 2, lower part). This feature makes the EOS approach a truly *architecture-centric approach* and results in a process structure as *scalable* as our hierarchic system decomposition structures are.

- In general, I prefer to define production processes in harmony with the structure of their resulting products. Since during the last decades *object orientation* has proven to be a viable and efficient product structure we should not hesitate to design the production processes in an analogous way: Any software development process is designed around a piece of software to be developed, i.e. a piece of the (evolving) software architecture. This view leads to a real OO process model - i.e. one which is *oriented* at the *objects* of software development.

- Besides software development, there are several parallel (sub-) processes associated with other important roles in the Software Engineering field (cf. the German V-model [Ver 99]). These are summarised in our overall software process model which is composed of five subprocesses for *development, quality management, configuration management & support, use & evaluation* and - last not least - for *project management* (cf. fig. 3).

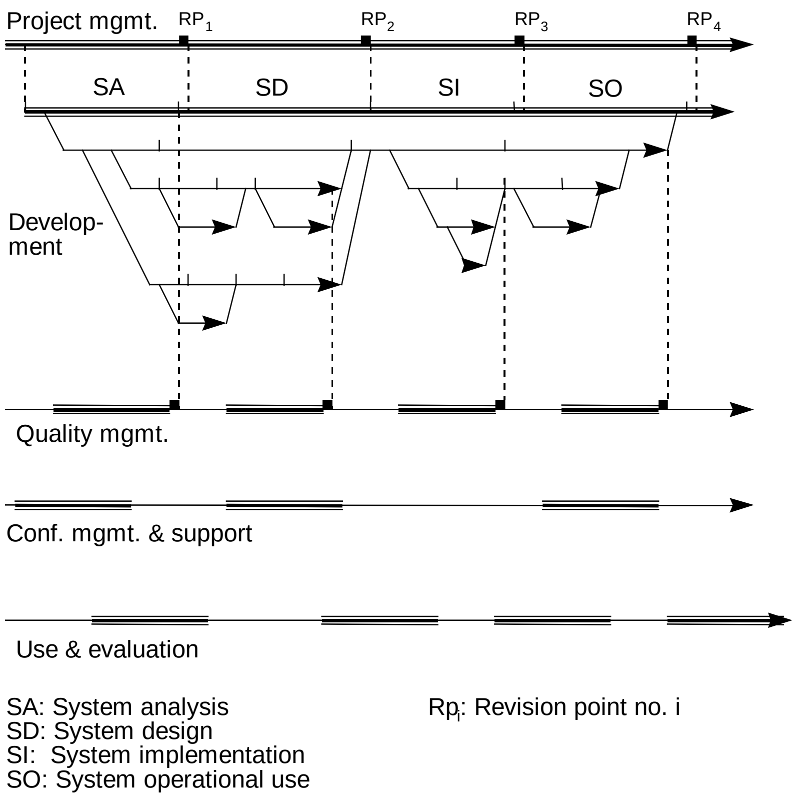

Fig. 3: The overall structure of the software process

- Software project management is much more than to place milestones at the ends of project phases in a one-dimensional way. It has to deal with the ambitious and difficult task of bringing order into the fractal diversity, i.e. *synchronising* a multitude of concurrent processes and subprocesses. In the EOS model, this is achieved by defining, controlling and maintaining so-called *revision points*. Instead of dealing with dubious statements like *"product x to y% complete"* revision points offer the facilities to define *complex projects states* like *"system analysis complete", "component A analysis & design complete", "component B analysis complete", "module C analysis, design & implementation complete"*, etc. ([Hes 96], [Hes 01]).

4 Summary and outlook

In the theses of section 2, I have critically reviewed the overall structure of Rational's Unified Process and some of its key features. In particular, the RUP concepts of phase, architecture, iteration, discipline, milestone and the necessity of user involvement have been addressed. I have summarised my alternatives in the outline of the EOS model in the previous section.

In this respect, the question arises to which degree a process model can or should be "unifying" at all. One might argue that instead of prescribing a certain idealised process a process model should rather offer the software engineers some sort of toolbox for designing their own, individually tailored processes. This has led us to our multi-variant approach (cf. [H-N 99]) which might give some more impulses for a more flexible RUP. The so-far reception of the RUP and reactions of its users show that such flexibility is much needed. Recent RUP modifications and the trend to derive "lightweight" processes confirm this demand.

In a current students project, we attempt to "implement" our EOS model using Rational's Process Workbench and some basic structures and artifacts of the RUP. This project gives us much more insight into the involved approaches, their common features and their diversities. We believe that much more work is needed but there is a good chance to eventually arrive at a common view of software process modelling - maybe not in the form of a *unified process* but rather in the form of a *unified box of instruments* enabling us to build processes adapted to everybody's specific needs and aims.

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 03] Rational Unified Process- Product Overview. http://www.rational.com/products/rup as of 18th Aug. 2003

[Sce 00] K.D. Schewe: UML: A Modern Dinosaur? A Critical Analysis of the Unified Modelling Language. In: H. Jakkola et al. (eds.) Information Modelling and Knowledge Bases XII. Proc. 10th European-Japanese Conference , pp. 185-202, Vol 67, IOS Press 2001

[Stö 01] H.Störrle: "Describing Fractal Processes with UML". Proc PROFES - 3rd European Workshop on Product Focused Software Process, Springer LNCS 2188 (2001)

[UML 03] Unified Modeling Language (UML) 1.5 Documentation. OMG documentformal/03-03-01. Rational Software Corp., Santa Clara, CA 2003. http://www.rational.com/uml/resources/documentation as of 18th Aug. 2003

[Ver 99] G. Versteegen: Das V-Modell '97 in der Praxis - Grundlagen, Erfahrungen, Werkzeuge. dpunkt-Verlag 1999