Reprint from: *C. Montangero (Ed.): Software Process Technology, 5th European Workshop, EWSPT 96, Springer LNCS 1149, pp. 241-256 (1996)*

Theory and Practice of the Software Process - a Field Study and its Implications for Project Management

Wolfgang Hesse
FB Mathematik/Informatik, Philipps University,
D-35032 Marburg (Germany)

email: hesse@informatik.uni-marburg.de

Abstract:

This report consists of three parts: (a) a summary of an empirical investigation on current software development practice, (b) the presentation of a new software process model based on its results, (c) a discussion of the consequences for managing projects which follow an evolutionary, object oriented approach.

In an interdisciplinary field investigation, software developers, managers and users were asked about their work procedures and conditions. This research aimed at obtaining a status of current software engineering practice and getting insights for future work design and organisation. Among other results, a significant discrepancy was found between the officially prescribed phase-oriented software life cycles and the work processes actually practised. Formal project planning turned out to be less important for project success than a flexible and anticipating task management.

In order to re-harmonize theory and practice and also to cover new requirements resulting from a paradigm change towards object-oriented techniques, we propose a model for *evolutionary, object oriented software development (EOS).* This model is based on a framework of cycles and activities bound to the "objects" of development as, for example, classes and system components. It is shown that management procedures built on this framework allow a multi-level, differentiated planning and control. This implies new challenges on project managers but it enables them to plan and to act in a flexible, dynamic and more adapted way.

1 Introduction: The IPAS project and its general results

In an interdisciplinary field research project (called IPAS[1]) about 300 software developers, managers and users have been questioned about technical, psychological and organisational aspects of their work. The investigation was jointly conducted by computer scientists, work psychologists and sociologists. It was based on the starting

[1] This project was financed by a grant (no. 01 HK 309/3) from the Work and Technology Fund of the Ministry of Research and Technology of the Federal Republic of Germany. The author takes the responsibility for this publication. Other members of this project were: Udo Bittner, Felix C. Brodbeck, Michael Frese, Thorsten Heinbokel, Rolf Ortmann, Johannes Schnath, Sabine Sonnentag, Wolfgang Stolte and Friedrich Weltz.

hypothesis that software quality is essentially influenced by (1) the human factor: i. e. how software engineers are qualified, like their work, feel supported by colleages, managers, and users, (2) the organisational factor: how management practices, work procedures, methods and tools match with the technical, organizational and personal requirements and their dynamic changes. Other, particularly technical factors were also considered important but of secondary role. It was expected that verifying this hypothesis and studying the work procedures and conditions of software engineers in detail would lead to new insights on the software process and result in proposals and guidelines for future work design and organization.

The results of the IPAS project have been published in various articles and books (see for example: [F-H 93, H-W 94, W-O 92, B-F 94, BHS 95]). Basically, the project consisted of two empirical investigations. The first was conducted by a mixed team of work psychologists and computer scientists. About 200 people engaged in 29 projects in 19 enterprises were asked by questionnaires and interviews about technical, organizational and psychological aspects of their work. Among the interviewed persons, 62 % were software developers, 26 % were managers of small or medium-sized projects and 10 % were user representatives.

Key subjects of this investigation were:

- a synopsis of the activities of software developers, their communication and co-operation, psychical stress and burnout,
- studies on the acceptance and use of methods and tools, dealing with instable requirements and change requests, exchange of information and ways of inner-project communication,
- influences of standards, guidelines, and work procedures for process and product quality and other management aspects.

Among other results, this investigation revealed the following facts (cf. also [F-H 93])

- Communication and cooperation play a more important role than usually expected. This is particularly true for the so-called "early" phases, where the developers have intensively to communicate with clients, user representatives, analysts, consultants and with each other. When object oriented techniques are used, there is even an increased force for more cooperation and communcation among the developers.
- The claim for more user participation causes rather heterogeneous reactions. On the one hand side, it is considered very important for the system analysis and model building stages. On the other hand, developers feel "disturbed" by too intense user contacts. These may include lengthy discussions, questioning agreed requirements, unrealistic requests for changes and enhancements, in short: less stability within the project.
- When methods and tools are assessed, their conceptual clarity and uniformity is considered essential. Tools get better scores the less "complicated" they are, the less limited their application area is, the less their users are required to deal with sophisticated interfaces or to (re-) learn complicated input/output procedures and the more they fit into the overall development concept and can be combined with other tools.

- The growing variety of concepts, methods, tools, technical platforms and the ever shorter innovation cycles impose increased challenges for developers and project managers. This leads to growing demands for technical standardization and harmonization of methods.

The second investigation was conducted by sociologists and concentrated on aspects of project organization and management. More than 300 members of 46 software projects were interviewed (for a detailed report cf. [W-O 92]).

The following sections focus on the software process and its management aspects. The results being presented are extracted from both investigations and were primarily elaborated by computer scientists and sociologists. *Among other results, they have shown a significant discrepancy between the work procedures officially claimed and those actually practiced.* Starting from the empirical results of the IPAS investigations, a new software process model is introduced and its consequences for future project management are discussed.

The rest of this article is organized as follows: Section 2 contains a report on current life cycle management practice based on a short classification of the so-called life cycle models. In section 3, a model for *evolutionary, object oriented software development* (the *EOS model*) is introduced. Consequences of the EOS model for a dynamic project management are discussed in section 4 and followed by some conclusions in chapter 5.

2 The role of life cycle models in software project practice

Questions concerning the organisation of the software process belong to the standard topics of software engineering and have challenged theorists, methodologists and practitioners for long time. A large number of software life cycle models has been presented, classified, compared and discussed. In a study recently performed by the author, the life cycle models of some known OO methodologies have been compared with respect to selected criteria such as *overall life cycle structure, OO features, evolutionary aspects, completeness, systematics & cohesiveness, dealing with complexity, documentation, diversity* (cf. [Hes 96]).

Here we restrict ourselves to a short summary, based on the slightly extended classifications of Boehm [Boe 88] and Hesse et al. [HMF 92]. This classification is rather rough and not to be used for a detailed comparison of life cycle models. We distinguish two groups of models: *linear* and *non-linear (cyclic)* ones. Among the linear models, there are three subgroups: *"code and fix"*, *phase-* (or so-called *"waterfall-"*) models, and *transformational* models. Among the cyclic models we find: *prototyping*, *spiral, incremental* and *evolutionary* models.

"Code and fix" stands for the unplanned, sometimes chaotic type of work which is normally attributed to the early days of software development but still occurs in recent reports. *Phase* or *"waterfall" models* idealize software development to be a linear process stepping from phase to phase (which sometimes are graphically arranged in a "waterfall"-like fashion). Each phase starts with specifications based on the results of the preceeding phase and has to produce its own results which are to be checked and verified against the specifications for consistency and completeness.

The basic idea of the *transformational* approach is similar but the specifications have to be formal in order to apply automated or semi-automated transformation rules or to verify programs formally based on a rigorous logical calculus.

Prototyping has been advocated as an alternative to the sequential phase models: Requirements which are prematurely fixed and too rigid specifications often turn out to be unrealistic, have to be adapted or even aborted as time passes. This causes a severe danger of expensive errors or mis-developments which in turn may result in the need of major revisions or even re-development. Prototyping tries to avoid such problems by means of a series of prototypes to be produced rapidly in the early project stages. Such prototypes have to be checked (for example, together with user representatives in the case of a user interface design) in order to establish a solid basis for the final system development [BKMZ 84].

A closer analysis shows that protoyping and the waterfall model are rather complementary than alternative approaches. With his *spiral model*, B. Boehm has presented a paradigmatic threefold prototyping cycle followed by a waterfall-like sequence of phases for the final system design and implementation (cf. [Boe 88]).

Incremental system development starts from a relatively small partial or kernel system and proceeds by stepwise adding so called *increments* until the required functionality of the system has been achieved.

Evolutionary software development was originally advocated by M.M. Lehman. This kind of development is primarily dedicated to *socially embedded systems* which are closely interwoven with the organisation, the work procedures and personal requirements of people directly working with or affected by the software. Typically, such systems have to deal with continuous feedback from those organizations and users which leads to ever new requirements for change, adaptation, and enhancement. Are they fulfilled, this is regarded a step of *system evolution* [Leh 80].

The IPAS investigations have confirmed that the phase or waterfall models are still dominating - at least *as far as the official project management standards* are concerned. The *actual practice does*, however, *significantly deviate* from these standards.

An IPAS investigation on the use of phase models has shown that the waterfall paradigm is dominating the standards and guidelines but, in practice, it is often fiction. In particular, among the 29 investigated projects which all followed a sequential phase model, there were 39 reports of explicit loops back to earlier stages. This number is based on a generic "waterfall model" consisting of only four main "phases". In 11 cases, the loops spanned even two or more stages. In some projects, the developers kept themselves an eye on possible future change requests on their programs and spent some effort on designing modular programs with particular provisions for future changes and increments. This way, they anticipated future modifications and system evolution to a cerrtain degree. However, in project planning and management, this aspect was almost neglected [HBS 92, BHS 95].

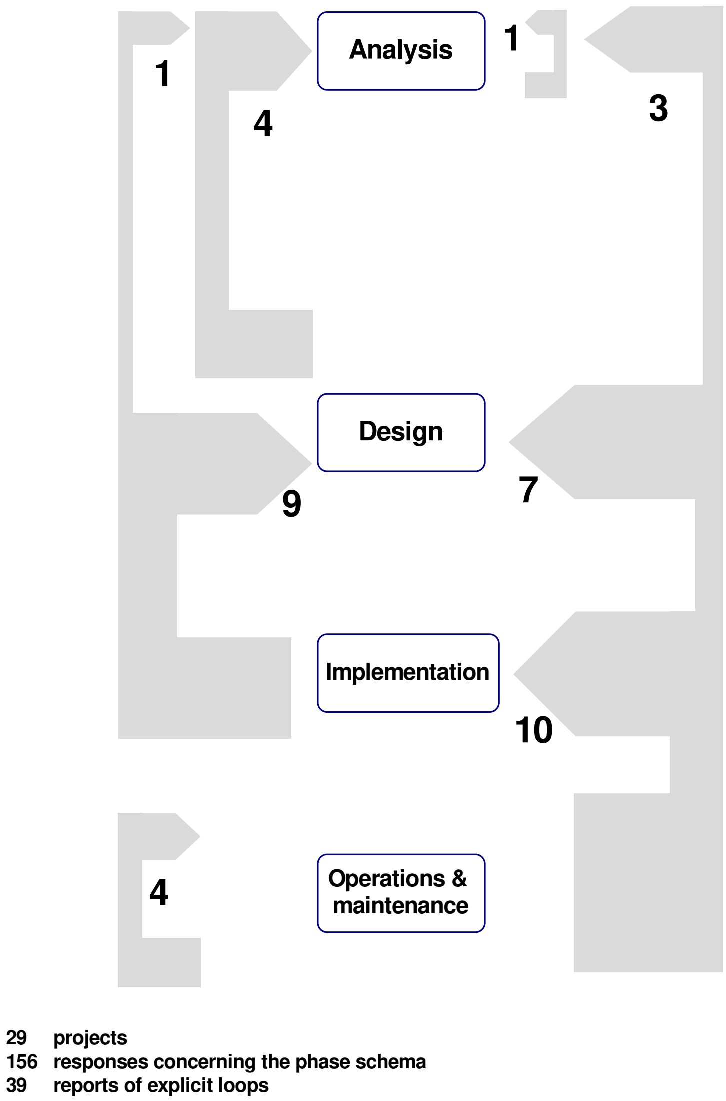

Fig. 1: The use of phase models in software development practice

The sociological investigation showed the following results concerning phase models and project managment (cf. also [W-O 92]):

- 60 % of the investigated projects "officially" followed a linear phase model with defined project phases. About one third of the projects reported an incremental work procedure, another third a prototyping style (multiple answers were admitted).
- In many cases, the actual progress of the project significantly deviated from the planned one. Only in 25 % of the investigated projects the original phase plan was actually followed. 41 % of the projects involved long phase overlaps, 13 % of the projects jumped forth and back between the phases, 13 % were characterized as "anarchical".

- There is almost no correlation between formal planning and project success. Much more important than (pseudo-) exact project plans and detailed standards is the ability of project managers to adapt quickly to changing or new requirements. Most successful were those projects whose managers were able to get direct feedback and flexibly react to it, to adapt the goals quickly to new requirements and to understand software development as a continuous learning process.

- There is a remarkable discrepancy between official planning and rigorous regulations on the one hand side and flexible reactions and self organizing processes of the developers on the other. In many cases, this discrepancy gave reason for misunderstandings, hidden conflicts, psychical tension, and loss of efficiency. Visible deficiencies were: there was no or insufficient cost control, projects were not evaluated *ex post* and the gained knowledge was not or only unsatisfyingly transferred from project to project.

- Success or failure of software projects is at least as much dependent on informal self organizing processes and on "tacid services" of the developers as on enforced, formalized management procedures, rigid planning and control instruments.

- An overall resume on the investigated projects resulted in the impression that most problems were less due to software-technical gaps and deficiencies, but rather to unsatisfactory project management. That means, well-understood project management is much more the key for project success than new programming languages, methods, software architectures or CASE tools. This way, the so-called "software crisis" turns out (at least to a considerable part) as a *crisis of project management*.

3 EOS: A model for evolutionary, object oriented software development

In order to harmonize theory and practice of software development and to combine principles of evolutionary system development with the concepts of object orientation, a model for *evolutionary object oriented software development* (EOS) has been proposed [H-W 94]. Earlier approaches tended to embed (small-scale) object development cycles into a (large-scale) sequential phase schema - as, for example, G. Booch does with his micro and macro processes. The EOS model goes the other way around: Every piece of software is considered an "object" which is developed in a cycle of uniform structure. It is based on a methodology which can be characterized in short terms by the following principles:

- *Object orientation as overall development paradigm*

Software development is based on uniform, evolving object structures. Single objects (called *building blocks*) emerge as bunches of requirements (a) from (use case) analysis steps conducted in the application area or (b) from (design case) analysis steps performed when higher, encompassing building blocks are to be refined. A building block forms a data encapsulation and is designed and implemented following the principles of data abstraction, inheritance and polymorphism (cf. [Mey 88]).

Building blocks are treated as self-contained units and can thus independently be tested, maintained, changed, re-developed or replaced. This way, a uniform system structure is maintained through the whole development process and structural or terminological clashes between different phases of development are avoided.

- *Hierarchical system structure*

A static decomposition structure and a dynamic process and interaction structure are distinguished. From a static perspective, a system is composed of *classes*, which represent objects of the same kind. Several classes which form a logical unit or have close relationships to each other are grouped to *components*. Components are disjoint to each other and form a hierarchical (tree) structure. This way, a three-level hierarchy of software building blocks is formed:

(S.) System

(X.) Components (or class complexes)

(C.) Classes

Moreover, classes (and objects) are connected by inheritance and aggregation relationships. Dynamically, building blocks can pass messages to each other and delegate tasks by activating operations of other building blocks. Collections of classes which are linked together in order to obtain a partial functionality of the whole system (e.g. for testing, demonstration, or incremental integration) purposes, are called

(SS.) Subsystems.

Subsystems can be (and normally are) non-disjoint, i.e. they overlap each other.

- *Cyclic development*

Each software building block (irrespective of its position in the system hierarchy) has its own life cycle which is characterized by the following four essential *activities* (cf. also fig. 2):

(.A) Analysis

(.D) Design

(.I) Implementation

(.O) Operational use (installation, usage and revision)

Of course, this has several implications, mainly on project organisation and synchronisation of activities (cf. the following section).

- *Software use and revision as integral part of each cycle*

An outstanding difference to conventional (waterfall-like) models is the fact that software development is no longer seen as an isolated construction project starting from scratch and ending in a rather vaguely defined "maintenance phase", but as a continuous cyclic process which encompasses analysis and construction steps as well as usage and revision steps. This does not only apply to the system as a whole but also to any kind of building blocks on all development levels, subsystems, prototypes etc..

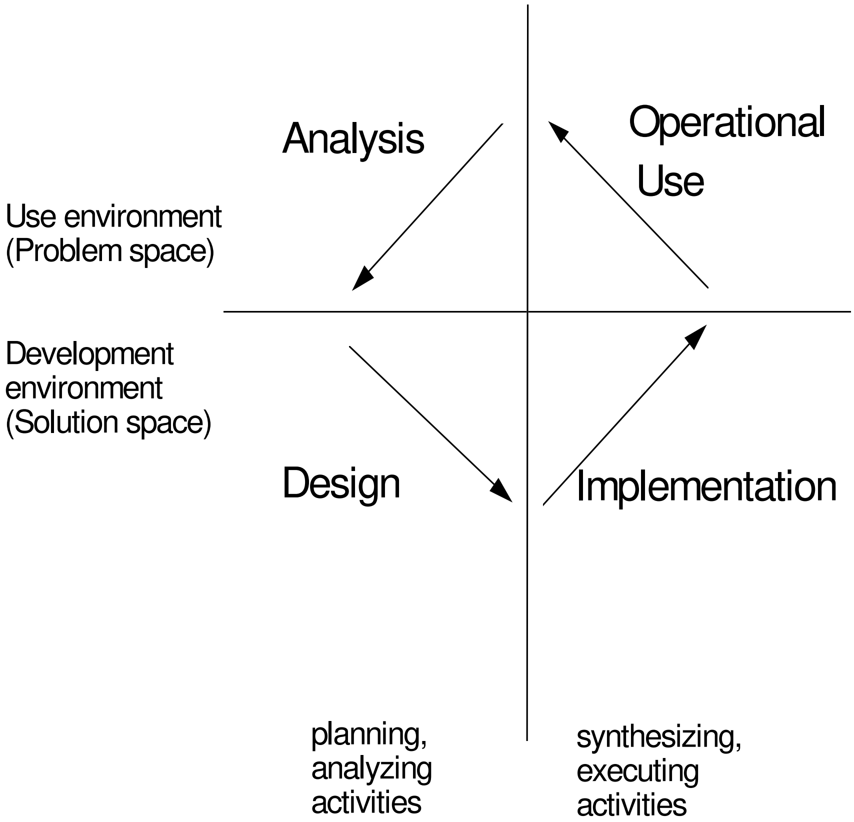

Fig. 2: The structure of an EOS development cycle

- *Provisions for further development and reuse*

One of the leading goals of all object oriented techniques is to improve the *reusability* of software building blocks. This requires to establish a *building block library (BBL)* which contains the building blocks from previous work and those defined so far. These can be used to derive new or modified building blocks, for example by specialization, generalization, combination, extension. To work with the BBL is particularly important in the early and late stages of a cycle: in order to search for reusable blocks during the analysis stage and to complement the library by including new blocks during the usage and revision stage.

- *Systematic and orthogonal composition of activities and development cycles*

As all development cycles have a uniform structure they can be combined in a most flexible way. A *system development cycle* consists of the development cycles of all of its constituents. The figures 3 and 4 show two (paradigmatic) ways of such combinations.

In figure 3, the development cycles on the three hierarchy levels have been linked together. Multiple occurrences of components and classes are symbolized by multiple arrows. Every activity becomes identifiable in such a network: The identifiers for building blocks introduced above (together with an index or identifier specifying the particular building block) and those for activities are combined in a straightforward manner to identify each particular activity: For example, SA_1 means analysis of system 1, XD_n means design of component n, CI_m means implementation of class m, etc..

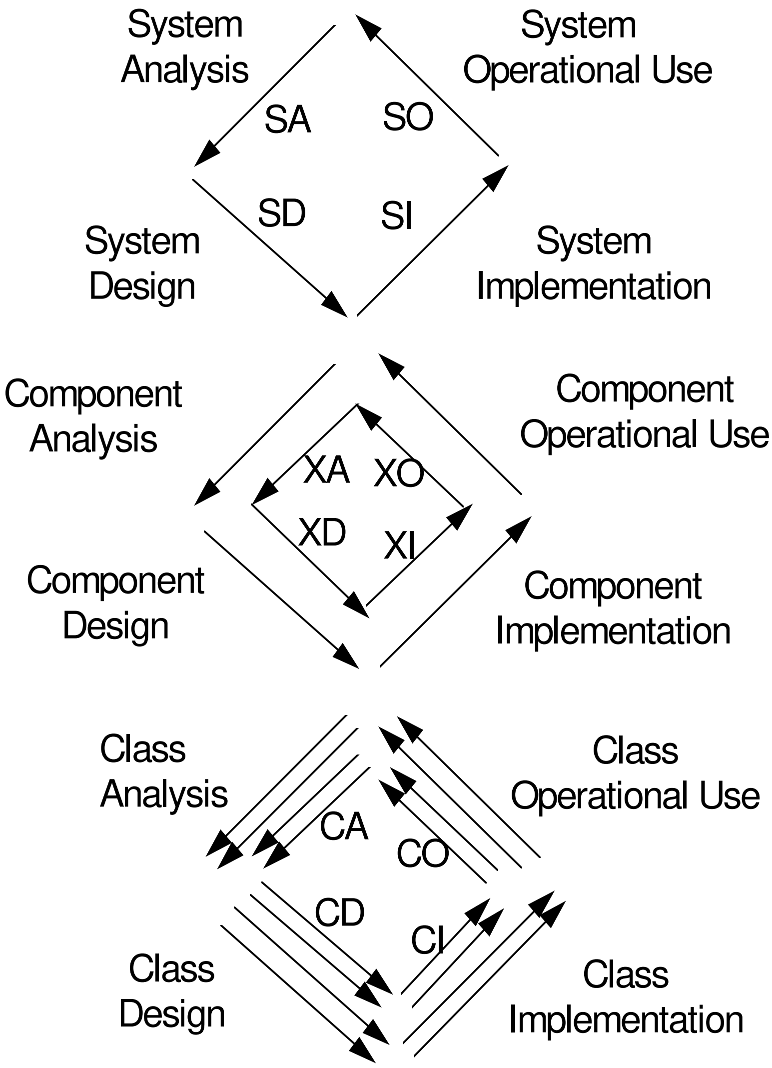

Fig. 3: Combining development cycles in the traditional way

Looking for a management procedure for coordinating these activities, we first apply the traditional development strategy of *top down analysis & design and bottom up implementation and usage*. With respect to figure 2, this would mean: first to go down along the arrows on the left hand side of the diagram and then return to its top following the arrows on the right hand side. Basically, this procedure corresponds to the phase/activity structure of advanced phase models as, for example, described in [Hes 84] and [HMF 92].

Another combination of activities, which much better reflects the requirements of object-oriented development and system evolution, is depicted in figure 4. With a slight temporal interpretation of the starting points of arrows, this diagram shows that new development cycles of building blocks can start at many places and that an over-simplified synchronisation as suggested in figure 3 is neither necessary nor desirable.

For project management this means: A certain developmemt cycle has to be started at the moment *most appropriate* to reach the intended result when *it is to be used*.

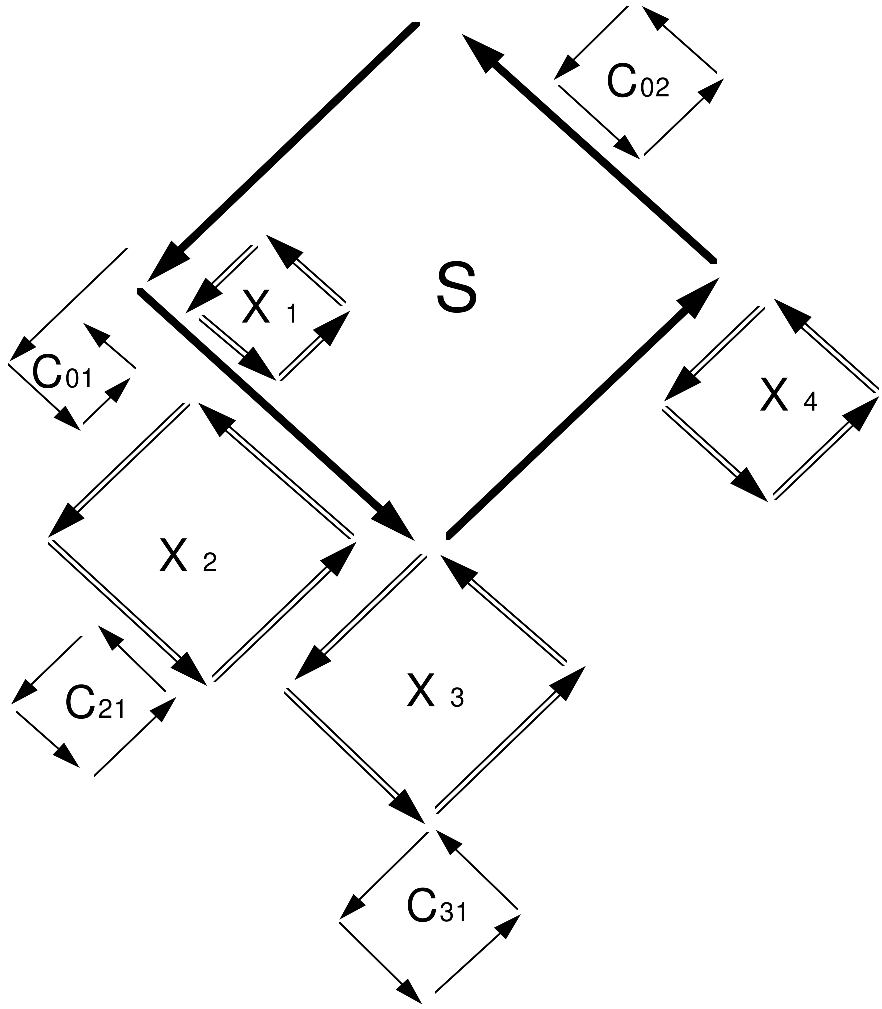

Fig. 4: Combining development cycles in an evolutionary way

- Management procedures based on the EOS structures

Procedures for project management can only be effective, if they conform to the technical development procedures. This requires management guidelines which are particularly cast to the EOS process model. Planning and project control are no longer based on conventional phases but on the EOS structures: overlapping development cycles and activities for particular building blocks. This principle results in particular requirements and consequences for project management which are considered in the following section.

4 Consequences for project management

Before we consider the consequences for project managers arising from new development paradigms such as object orientation or system evolution, we will shortly recall the present situation: Why are conventional (waterfall-like) phase models still so popular - in spite of their apparent (and often discussed) shortcomings? Possible answers are:

- They structure the development process in a simple and transparant way.
- Planning of the overall project seems to be rather easy, as the phase structure is very transparent, coarse and linear.
- The phase schema offers a simple pattern for defining *milestones* and using them to terminate activities, check and release the results.

In the early project stages, the phase model looks attractive for managers, as it facilitates planning of time, costs and resources for software development in a seemingly exact way. In later stages, however, these plans and calculations based on them out to be unreliable, are to be modified or become even unsubstantial.

Project management for *evolutionary system development* looks much more complex from the very beginning, as it does not start with a complete plan but only with a preliminary one which needs to be refined throughout the whole project and re-adapted according to the actual project status and the (remaining) requirements. This means: goals and requirements have continuously to be checked, re-adjusted or re-defined during the project life cycle. Software producer and user organizations do not only sign one single contract but they have regularly to cooperate in order to keep their plans adjusted to the current situation.

This implies new challenges for project managers and their capabilities to achieve consensus among their teams, to revise decisions, and to encourage project members to more self-organization and autonomy.

A *dynamic project management* cast to the needs of evolutionary system development has to deviate from traditional management in the following points:

- The strong sequence (and separation) of planning and executing has to be abandoned.
- Instead of (re-) acting to fulfill predetermined plan data as required by a phase model, managers have to combine planning and execution of processes and activities according to an *integral management strategy*.
- Planning must be more flexible. This implies to re-adjust technical goals and requirements as well as to change dates, to adapt personal and resource allocations to the current situation. All plan data have to be re-examined (and possibly to be modified) throughout the whole software development process.

Of course, this kind of project management requires a change of thoughts from all people concerned - particularly from clients who want to "buy" a piece of software for a predetermined price. They have to be convinced that this is only possible for standard applications "sold off the peg" but not for novel solutions tailored to their specific needs.

The rest of this section will be devoted to the particular implications of the EOS model for project management. What distinguishes planning and control of EOS-like project from traditional ones? How can project managers deal with the specific requirements arising from the object-oriented and evolutionary development style?

- The EOS hierarchy of development levels is the starting point for planning steps of any granularity. Correspondingly, there are three planning cycles: The project-wide planning of the S-cycle(s) is complemented by more detailed plans for teams dealing with work packages on the component level (X-cycles) and for single project members working on the class level (C-cycles).

  The generic activity structure of the EOS model can be taken as a basis for detailed planning of activities and control steps of a particular project. This initial plan is flexibly to be adapted to the specific project situation at a time, to the current requirements of customers, users, developers and project leaders.

- In the EOS model, the classical separation of non-technical (requirements) analysis and technical building phases has been settled. It is replaced by development cycles which comprise both aspects, use uniform models and structures and which open the way for a close cooperation of users and developers and thus for a continuous balancing of user requirements with the technical design capabilties.

- Work distribution and responsiblities are based on products, i.e. EOS building blocks. Several activities to be performed on one building block (for example: to analyse, design, implement and test one particular class or component) are associated to one single person or to one cooperating team - they are not assigned to different persons or institutions. For larger components and for the whole application system, long-term *service teams* have to be defined. They are responsible for (further) development work as well as for operational support.

  A clear determination of responsibilities, as given by these rules, provides developers and users the individual freedom they need. Their self-organizing capabilities are not restricted by rigid top-down planning directives but stimulated by enhanced personal responsibilities.

- Project management needs appropriate instruments for setting milestones, surveying the project status and reviewing (intermediate) project results. In the EOS model, traditional milestones are replaced by differenciated *revision points*. An example of a revision point definition is: "Components A, B, C finished, component D and classes E, F implemented, components G und H designed, component J analysed".

  Revision points are not completely defined in all detail at project start but are roughly planned in the beginning and reviewed several times as the project goes on. This way, they are frequently adapted to the current facts and requirements. The concept of revision points resembles the reference lines of the STEPS methodology [FRS 89] but the latter are based on a different, much coarser and less orthogonal process structure.

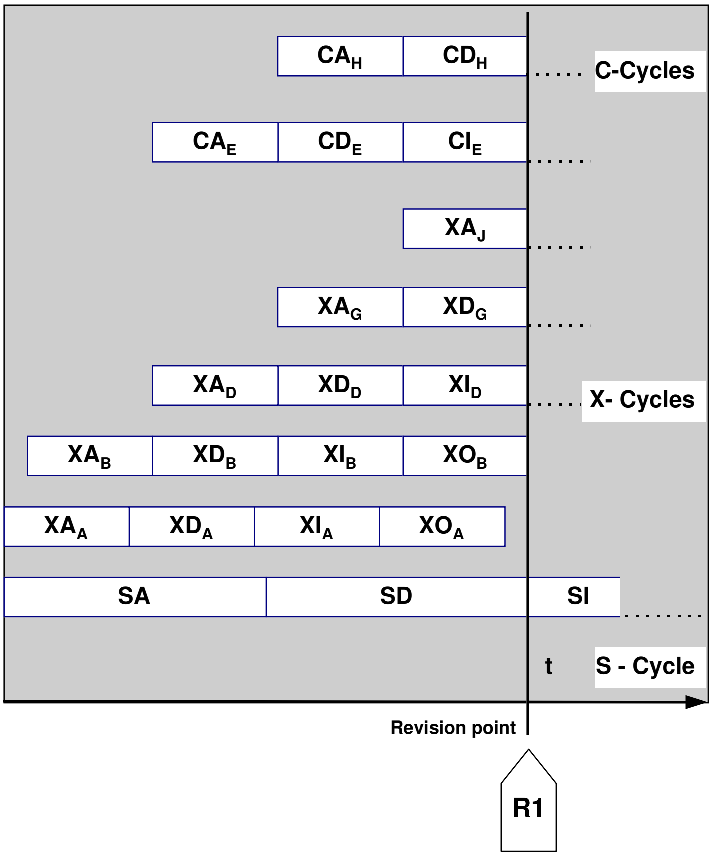

Fig. 5: An example of a revision point

Revision points requires flexible and refined instruments and tools for project planning and control. Their primary focus is to provide management always the optimal information on the project status - not to enforce rigid planning of future steps. Approved traditional management tools for status reports, plan/achievement comparison, and cost calculation can also well be used for dynamic project management

- Software development is viewed as a looped system of decision processes. These replace the traditional project plan which had to be "signed" in the very beginning, which was often poorly checked at a few milestones and (in the worst cases) had to be abandoned in the later project stages. Chains of decisions have to be extended (and altered, adapted to the current needs) from activity to activity, from revision point to revision point and from cycle to cycle.

- Software development is a *continuous process* which often lasts longer than the deadlines of particular projects. Development does not end with the installation of a "finished" software product. Further development activities (often called "maintenance") are common (and unavoidable) and demand for a knowledge transfer going beyond the limits of the particular project. This kind of transfer takes place whenever a new S-cycle of the EOS model is started.

5 Conclusions

In the previous sections, a new model for evolutionary, object-oriented software development has been presented and motivated by the results of an empirical project investigating the work conditions of software engineers, managers and user representatives. It has been argued that traditional phase models do no longer fulfill the needs of the people concerned but have to be replaced by more flexible models which take into account development cycles, composite processes, products of various size, re-development and reuse etc.

A forthcoming study on the life cycles models of some known object-oriented methodologies shows that these stand in the tradition of the waterfall model or still lack the required uniformness, orthogonality and flexibility (cf. [Hes 96]). The EOS model has been proposed to fill this gap and its implications on project management have been discussed in more detail.

Evolutionary software development does not facilitate project management but imposes new challenges on all project members - and particularly to project managers. For example, precise cost estimations and statements are hard to give in the early project stages. Therefore, the claim to predict precisely the outcome of a large, long-lasting software projects in its very beginning is likely to raise false expectations and to lead to major irritations and deceptions.

It is not always easy to explain this to impatient clients. This might even render the acquisition process more difficult. It is however true, that impreciseness is inherent and mostly unavoidable in this project stage. This fact has been hidden by the pseudo-precision of traditional planning methods.

By the end, the dynamic, evolutionary method is the more honest and the more precise one as it reflects the status of plans and achievements not only once (at project start) but at every point of a project. As an effect of continuous refinement, planning data become increasingly precise and facilitate a controlled termination of the project. In the industrial practice, the success of evolutionary development methods will depend on the readiness of all concerned people to accept the dynamics of projects, the versability of requirements and the necessity of a more realistic project planning.

Since its first publication, the EOS model has found positive resonance, particularly in the software practice field and at small and medium-sized software enterprises. It is now being implemented and evaluated by some pilot enterprises which cooperate in a regional German association with the general aim of software process improvement. Further work will focus on starting and supporting more pilot projects using the EOS method, including larger projects and enterprises. Feedback from these practical projects will be used for improving techniques and instruments for project management, quality assurance and for object oriented analysis and modelling.

References:

[BHS 95] U. Bittner, W. Hesse, J. Schnath.: Praxis der Software-Entwicklung, Methoden, Werkzeuge, Projektmanagement - eine Bestandsaufnahme, Oldenbourg 1995

[Boe 88] B.W. Boehm: A spiral model of software development and enhancement, Computer, May 1988, pp. 61-72

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

[B-F 94] Felix C. Brodbeck, M. Frese (Hrsg.): Produktivität und Qualität in Software-Projekten - Psychologische Analyse und Optimierung von Arbeitsprozessen in der Software-Entwicklung, Oldenbourg 1994

[BKMZ 84] R. Budde, K. Kuhlenkamp, L. Mathiassen, H. Züllighoven (eds.): Approaches to prototyping, Springer 1984

[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 Engineering Conference, LNCS 387, pp. 48-64; Springer 1989

[F-H 93] M. Frese, W. Hesse: The work situation in software development - Results of an empirical study, ACM SIGSOFT Software Engineering Notes, Vol. 18, No. 3, pp. A-65 - A-72 (1993)

[HBS 92] W. Hesse, U. Bittner, J. Schnath: Results from the IPAS Project: Influences of methods and tools, quality requirements and project management on the work situation of software developers; in: P. Elzer, V. Haase (Eds.): Proc. Fourth IFAC/IFIP Workshop on Experience with the Management of Software Projects, Annual Review in Automatic Programming, Vol. 16, Part II, Pergamon Press 1992

[Hes 84] W. Hesse: A systematics of software engineering: Structure, terminology and classification of techniques in: P. Pepper (Ed.): Program Transformations and Programming Environments, pp. 97-125, Springer 1984.

[Hes 96] W. Hesse: Life cycle models of object-oriented software development methodologies (forthcoming, 1996)

[HMF 92] W. Hesse, G. Merbeth, R. Frölich: Software-Entwicklung - Vorgehensmodelle, Projektführung und Produktverwaltung, Handbuch der Informatik, Band 5.3 Oldenbourg 1992

[H-W 94] W. Hesse, F. Weltz: Projektmanagement für evolutionäre Software-Entwicklung; in: Information Management 3/94, pp. 20-33, (1994)

[IS 92] Thema: Projektmanagement für objektorientierte Software-Entwicklung, Themenheft, Informatik-Spektrum, Bd. 15, Heft 5 (1992)

[Leh 80] M.M. Lehman: Programs, life cycles, and laws of software evolution, Proceedings of the IEEE, Vol. 68, No. 9, pp. 1060-1076 (1980)

[Mey 88] B. Meyer: Object-oriented software construction, Prentice Hall 1988

[W-O 92] F. Weltz, R. Ortmann: Das Softwareprojekt - Projektmanagement in der Praxis, Campus-Verlag 1992

Author's address:

Prof. Dr. Wolfgang Hesse
FB Mathematik/Informatik, Univ. Marburg
Hans Meerwein-Str.

D-35032 Marburg
Germany

Tel.: +49-6421-281515
Fax: +49-6421-285419

email: hesse@informatik.uni-marburg.de