A Multi-variant Approach to Software Process Modelling

Wolfgang Hesse[1] and Jörg Noack[2]

[1] c/o FB Mathematik/Informatik, Philipps-University Marburg/Germany
hesse@informatik.uni-marburg.de
[2] Informatikzentrum der Sparkassenorganisation (SIZ) Bonn/Germany
joerg.noack@siz.de

Abstract: In this article we present a new approach to software process modelling for a large banking organisation. In the past years, the main software development methods and tools of this organisation have migrated from structured to object-oriented technology. Presently, the software process is completely being redefined and adapted to the new goals and requirements. Since there are many kinds of projects differing largely in their goals, requirements and constraints, a two-level approach has been taken: On the base level, the ingredients of processes - activities, results, techniques and tools - are listed and described. These are composed in various ways to form a set of process variants which are defined on the second level. Each variant serves as a sample process for concrete project work. This multi-variant approach meets the requirements of the project managers and developers who demand for a flexible model covering a wide spectrum of projects.

Keynotes: Object-oriented software development, process model, model variants, activities, prototyping, component-based development, phase-oriented development, evolutionary development.

1 Introduction

In the German Savings Banks Organisation (short: GSBO, comprising about 600 savings banks and 13 state banks) information technology (IT) is primarily provided by 10 major development centres. SIZ, the IT coordination centre, is a service enterprise offering support for these centres. In order to improve organisational and process maturity as well as to enhance software quality and development performance, SIZ is publishing and maintaining an electronic project handbook called the *application development model* (AD model). This handbook documents the software processes and best practices collected from several development projects in GSBO and from the literature [11].

During the last years the AD model has continuously been improved to fulfil the requirements of a large banking organisation. A first issue of the handbook covered a waterfall-like process model, structured development techniques like entity-relationship-diagrams or functional decomposition and conventional programming in

M. Jarke, A. Oberweis (Eds.): CAiSE'99, LNCS 1626, pp. 210-224, 1999.

COBOL or C. It has primarily been used for large-scale mainframe projects building back-end systems, for example for clearing or accounting.

Meanwhile, the object-oriented approach to software development is becoming more and more popular. Concentration of the banking business has led to many reengineering projects which make it necessary to migrate or redevelop existing monolithic legacy application systems. The growth of the Internet opens up new dimensions of business action like electronic commerce and electronic banking. In order to meet these requirements, SIZ has started a methodology project aiming at a complete new version of its AD model. This version covers object-oriented software development for modern multi-layered software architectures using methodologies, languages and architectures like OMT, UML, JAVA and CORBA.

One kernel piece of the AD model is the object-oriented system life cycle (cf. [4]) described by a process model. In order to support projects of various kinds, goals, size and environments, a customisation approach has been taken: It is based on a toolbox for the construction of processes – the *reference framework* describing activities, results, techniques and roles. On top of this framework, so-called *model variants* act as guided tours through the OO development process. This *multi-variant* approach goes insofar beyond the existing OO methodologies (cf. for example [1], [9], [13] and [6] for a comparison) as it encompasses their model variants and embeds them into a general, customisable process model. This way, a *defined process* (cf. [8]) is achieved which does not force all projects into one single obligatory standard.

Recently, a first release of the AD model has undergone a major revision reflecting the evaluation results from several OO pilot projects in the GSBO. It is still too early to report on experiences of applying the multi-variant approach in a real project environment. However, the experiences of applying the first release of the AD model are discussed in a separate paper (cf. [12]).

2 The Overall Structure of the Development Process

In this section we present the overall *process architecture* (cf. [2]) underlying the AD model. There are several groups of people involved in the software development process which are distinguished by their different tasks and views on a software development project. We use the concept of *roles* to distinguish between these groups and their (complementary) views. To each role belongs an own process – thus the overall software process can be seen as a bunch of *concurrent sub-processes* which are synchronised by means of (milestone-like) revision points (cf. [5], [7]).

The roles and corresponding sub-processes are (cf. fig. 1):

- Development
- Project management
- Quality management
- Configuration management and support
- Use and evaluation

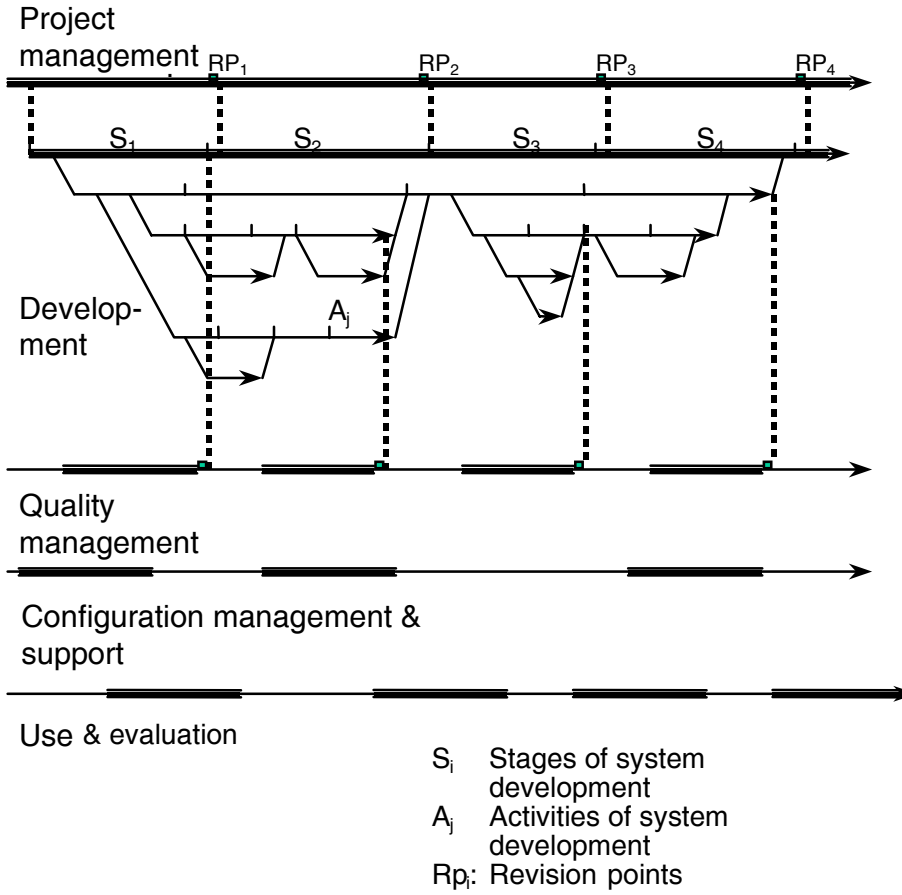

Fig. 1. The structure of a typical software process

3 The Process Building Tool Box

Since GSBO is a heterogeneous organisation covering savings banks, state banks, leasing and insurance companies etc. the AD model has to deal with heterogeneous kinds of projects, partners, and applications with special requirements which concern e.g. safety constraints or demands for reuse. For such an organisation a uniform process model is not a realistic option.

On the one hand, tasks should be manageable, components should be exchangeable and projects should be comparable – arguments which suggest to standardise the process as far as possible. Reuse, exchange and sub-contracting of results and components is a major issue for the target institutions – which makes a "real OO process" mandatory for many projects. On the other hand, there are several projects

rooted in the tradition of the structured approach which put less emphasis on exchange, reuse or subcontracting. Some projects have relatively fixed and established requirements – others experiment with new solutions or even explore new applications. Projects vary in size, stability of the requirements, complexity of the resulting system and many other factors.

These obviously conflicting requirements have driven our specific approach to process modelling. It is a two-level approach consisting of

(a) a *base level* which comprises all the raw material used for building OO development processes ("the reference model") and

(b) a *composition level* offering several *process variants* based on the ingredients of the base level.

The *reference model* consists of five main components:

- a set of paradigmatic descriptions of *activities*
- a set of descriptions for resulting documents – briefly called *results*,
- a set of *techniques* supporting the elaboration of results,
- a set of *role* descriptions for the various groups of stakeholders in a software development project.
- a set of *guidelines and rules* which are meant to support reading and application of the handbook.

These components are linked by several relationships – as shown in fig. 2. In the following subsections, we briefly summarize the components. Their original presentation consists mainly of (German) text, including graphical figures, tables and examples.

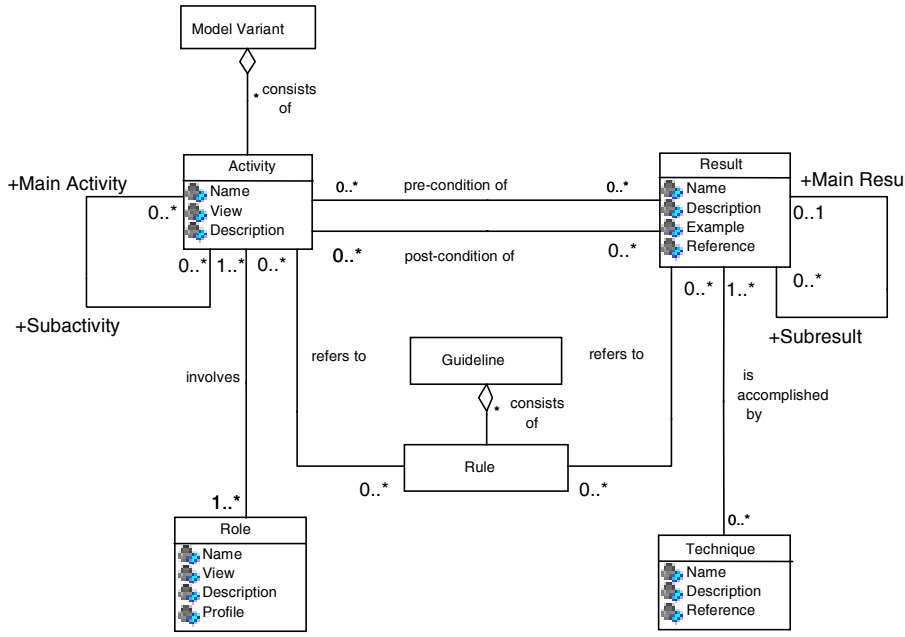

Fig. 2. Metamodel showing the overall structure of the reference model.

3.1 Activities

Activities are the essential building blocks of the process model. In general, their effect is to produce results (which are separately described in the result section). An activity can be decomposed into *subactivities*. These have either to be performed in sequential order or may be carried out in parallel (or in any arbitrary order).

Activities at the first level of the hierarchy are called *main activities*. In its present form, our activity model comprises 24 main activities. Examples of main development activities are *Analyse requirements, Build component, Build architecture, Integrate and test system.* Examples of main management or quality assurance activities are *Start project, Terminate project, Check quality*.

Activities are aggregated forming a tree-like hierarchy. However, sub-activities may be *used* by several main activities - not just by their predecessor activity. All activities are described in a uniform manner following a scheme which contains (among others) the following items:

- Name
- View (it belongs to, cf. above)
- Description
- Pre- and post conditions
- Roles (concerned by the activity)

3.2 Results

Similar to the activities, *results* are described in a schematic way using a form which contains (among others) the following items:

- Name
- Description
- Example
- Techniques (references to the corresponding section)
- Literature references

Typical kinds of results are *project plan, requirements statement* or *class structure model*. A result can consist of several *subresults*, e.g. descriptions of *classes* and *relationships* are parts of the result *class structure model*.

3.3 Techniques

In this part, the main *techniques* are listed and described which are recommended for elaborating the results. Techniques include various kinds of diagrams as, for example, offered by the Unified Modeling Language (UML) [14], test procedures, management forms etc.

3.4 Roles

In this part, the main *roles* of people involved in a software development project are listed and described. The description includes a profile, i.e. a list of skills and qualifications associated with that role. Typical roles include the *project manager,* the *analyst,* the *designer,* the *programmer,* the *quality specialist,* the *support specialist,* the *user*.

3.5 Guidelines and Rules

Process modelling is just one part of the AD model. Another part consists of *guidelines* which aim to support the activities and to ensure readability, quality, and portability of development results. Examples are guidelines for object-oriented analysis, design, programming, building a software architecture etc. Each guideline consists of several *rules*. In the descriptions of the activities and techniques these rules are addressed in order to support a uniform, comparable and manageable application of the whole framework.

4 The Model Variants: Four Guided Tours through the Development Process

With the ingredients presented in the previous section, processes can be individually composed and cast to the particular project situation they are to be used and to the requirements resulting from that situation. However, experience shows that it is helpful (and sufficient for the major part of situations occurring in practice) to concentrate on a few "typical" processes which cover most of the real-life cases. For these selected processes, a possible line of processing can be predefined and used as a sample for running a concrete project. We call such a sample process a *model variant*.

Given a concrete project with its particular goals, requirements and environment, the most suited model variant can be selected using the criteria given in the subsequent section. The selected variant can then be modified and adapted to the specific needs of the project. In order to cover the majority of current project practices in GSBO, we have selected the following four model variants for detailed presentation in the AD model:

- Incremental development (INC),
- Component-based development (CBD),
- Phase-oriented development (PHA),
- Evolutionary Prototyping (EVP).

All variants are embedded in a general scheme for project management activities depicted in fig. 3:

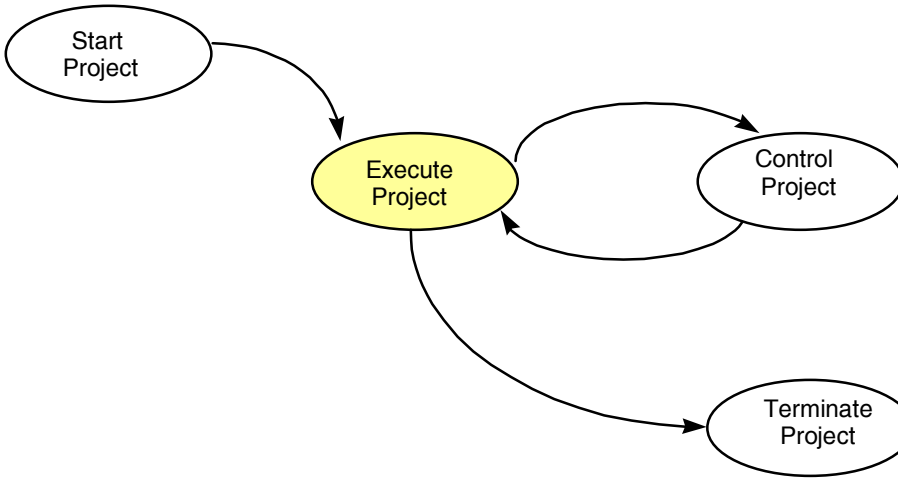

Fig. 3. General management scheme for all model variants

In fig. 3 we have used a very simple ad-hoc diagram technique (so-called *bubble-charts*) to illustrate a process consisting of activities (ovals), subactivities defined at other places (highlighted ovals) and sequence relations (arrows). The same bubble-chart technique will be used as well in the subsequent figures.

In the following sections, the four model variants are depicted by diagrams and briefly explained.

4.1 Incremental Development

This variant is based in the notion of increment. An *increment* is a piece of software which is added to an existing system in order to enhance its functionality or performance. To develop a system incrementally means to start with a relatively small kernel and enhance this kernel step by step by adding increments until the required functionality is reached. An increment may be – but is not required to be - a stand-alone executable unit.

Fig. 4 and 5 show the main steps of incremental development. The development process as a whole is embedded in management activities as shown in fig. 4. Analysis is done as a system-wide analysis (covering all increments) and may (optionally) be supported by developing an explorative prototype (fig. 4).

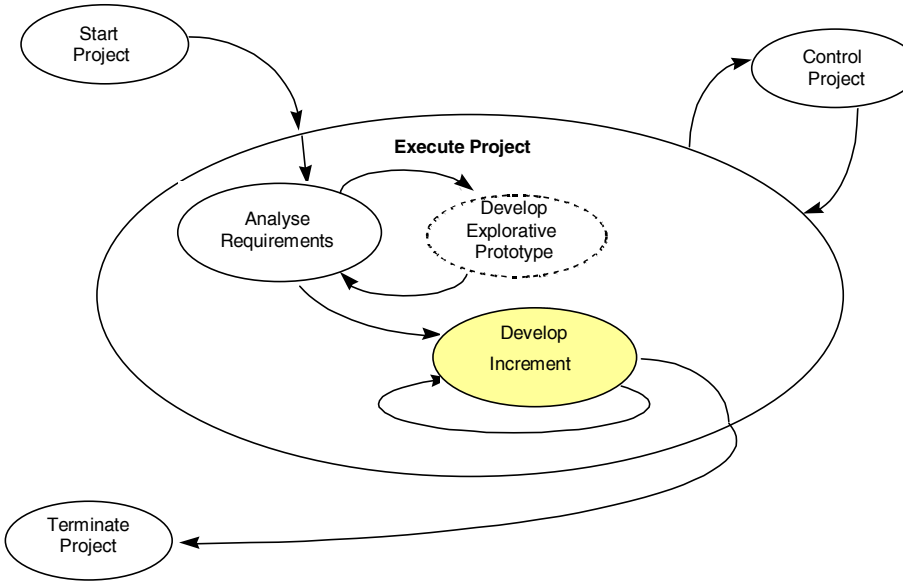

Fig. 4. Incremental development: overall project structure

Analogously, each development of an increment is considered as a kind of "project" with starting, controlling and concluding management activities (fig. 5). The central activity *Build increment* is refined to activities *Model increment, Implement increment* and *Integrate increment into application system* which are main activities of the reference model.

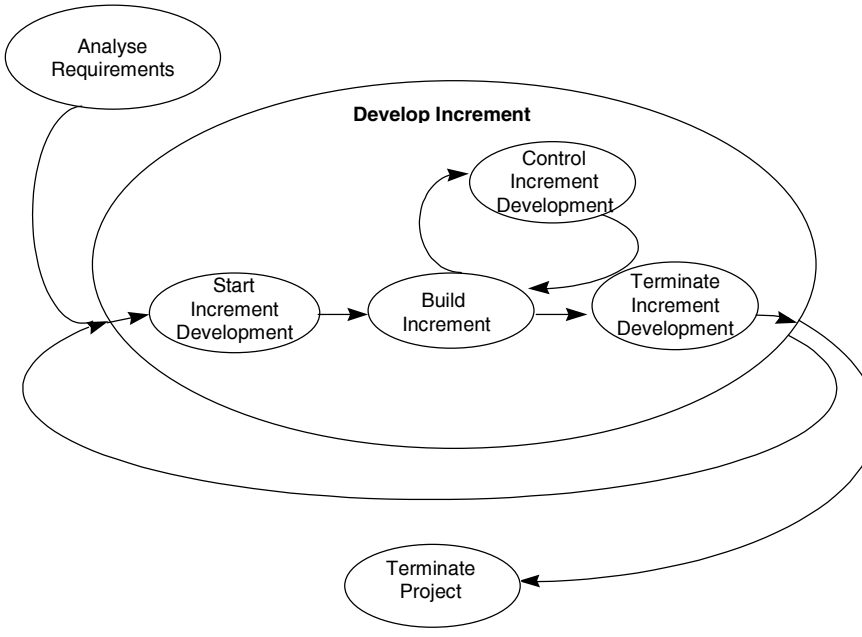

Fig. 5. Development of an increment

4.2 Component-Based Development

A piece of software which has a well-defined interface, may be used in various contexts and can easily be replaced by an equivalent piece (i.e. one having the same or a very similar functionality) is called a *component*. To base a development on components means to develop an architecture of relatively independent units which can be built separately or even borrowed from other projects or from a component library.

Fig. 6 shows the overall structure of a component-based development process. As in the previous variant, the development process is embedded in management activities (*Start project, Control project, Terminate project*). Again, the (system-wide) analysis may be supported by building an explorative prototype. It is followed by the activity *Build architecture* which is most central for this variant and implies the definition and delimitation of its components.

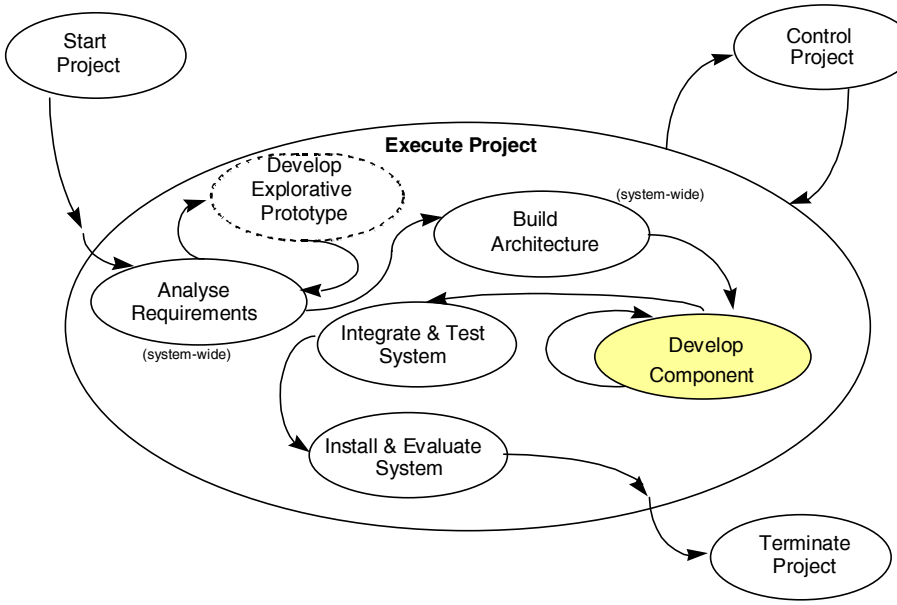

Fig. 6. Component-based development: overall project structure

Released components are integrated to an application system and then installed, used and evaluated (fig. 6). In principle, each component may be developed independently from the others. Therefore it is viewed as a subject of an own (sub-) project as illustrated in fig. 7.

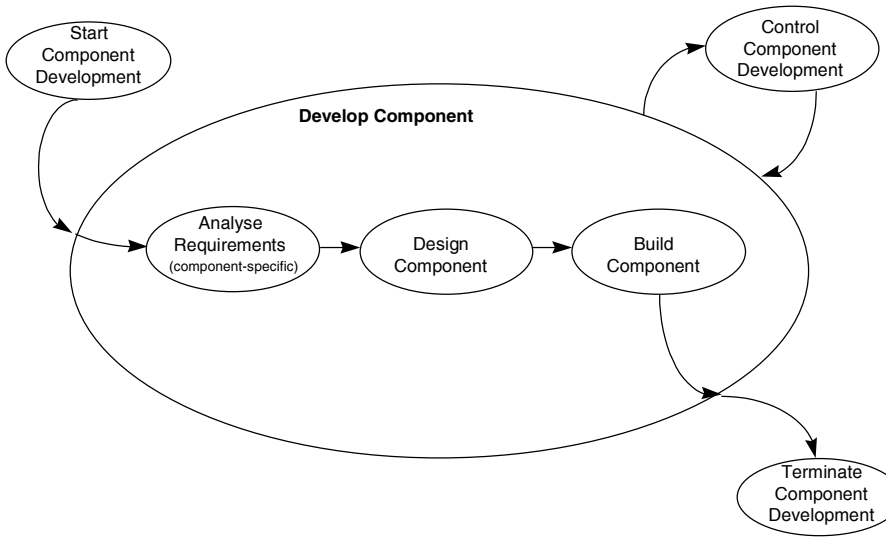

Fig. 7. Development of a component

4.3 Phase-Oriented Development

This is the most traditional of the four model variants. A project is assumed to deal with a rather monolithic system, i.e. one the structure of which is not explicitly reflected by the process structure. Basically, this structure is given by *phases,* i.e. temporal units which follow each other in a sequential manner. On the uppermost level, we distinguish four main phases which correspond to the activity categories introduced in section 3: *Analyse requirements* (with an optional development of explorative prototypes), *Design system* (with an optional development of experimental prototypes), *Build system, Install and evaluate system* (fig. 8).

4.4 Evolutionary Prototyping

Prototyping is a development technique which may be used at many places, in various situations and contexts. Thus we have to distinguish several kinds of prototyping. Following an earlier classification [3] we differentiate between *explorative, experimental and evolutionary prototyping*. Whereas the first two alternatives are rather viewed as supplementary activities supporting the analysis and design steps, resp., *evolutionary prototyping* is considered a technique which constitutes its own kind of process. Therefore, we have it included as a separate model variant.

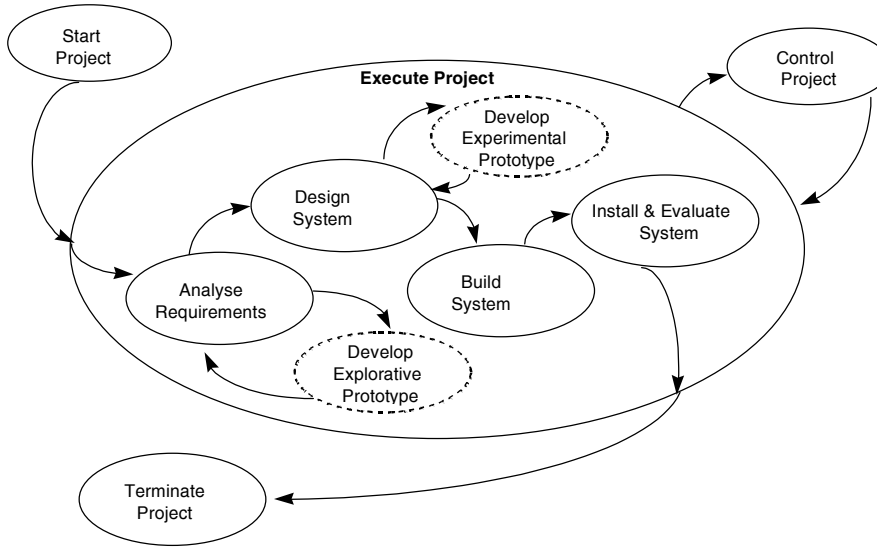

Fig. 8. Phase-oriented development: overall project structure

Evolutionary prototyping applies to projects in an unstable environment, with incomplete, unsafe or not yet defined requirements and constraints. Complex dynamic systems are characterised by the fact that they influence and change their environment which leads to new requirements and eventually results in a chain of feedback loops covering development and use steps [10]. This is reflected by our model variant (fig.'s 9 and 10):

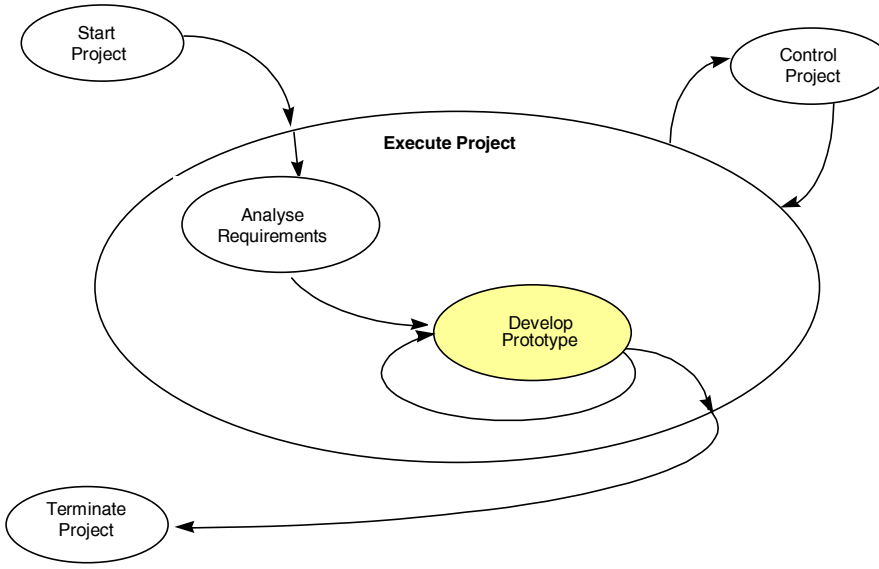

Fig. 9. Evolutionary prototyping: overall project structure

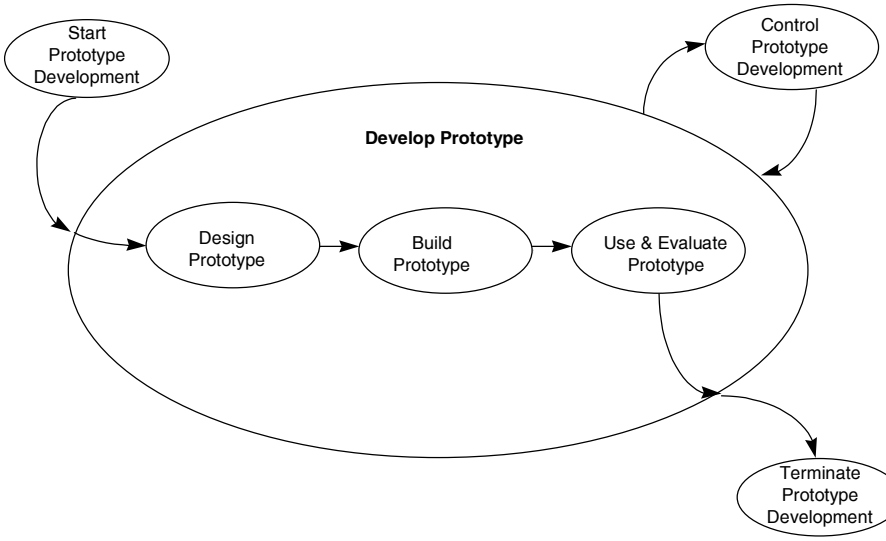

Fig. 10. Development of a prototype

5 Criteria for Variant Selection

In a given concrete situation of an evolving or starting project, it is often not easy to select the most appropriate model variant. In order to support this selection process, we have listed some criteria which normally influence the selection and scored the given model variants with respect to these criteria. Criteria have been classified in two groups:

- Project goals and requirements
- Process constraints

Note that the scores given in these tables are mainly based on experience and subjective assessment and are not verified by exact investigations or measurements. Thus they should be handled with care and rather taken as informal hints than as precise, objective rules.

In our present project practice, variants are used with different priority and frequency. Incremental development is often considered the most practicable and most generally applicable variant. Phase-oriented development has still a high preference, mainly for projects building on traditional application systems. Component-based development is considered a rather innovative alternative and still handled with some care and reserve. Evolutionary prototyping is the alternative which so far has least been experienced.

(1) Project goals and requirements	INC	CBD	PHA	EVP
First release, a running system does not exist	+++	+	+	++
Rapid delivery of a subsystem required	++	++	-	+++
Rapid delivery of complete solution required	+	++	+	?
Limited budget, high cost efficiency required	++	+	++	?
Stable environment and system constraints	+	?	++	-
Monolithic system with a few subsystems	+	?	++	?
Heterogeneous system with multiple independent functions	++	+++	?	+
Short change and maintenance cycles	++	+++	-	+
Unstable requirements; frequent change requests expected	++	+++	-	++
Distributed system	++	+++	-	+
High security requirements	++	++	+	?
High quality requirements	++	++	+	+

Fig. 11. Selection criteria: Project goals and requirements

Our selection criteria are rather new and cannot yet been judged reliably. At the moment, we are gathering feedback from the "users" of the AD model (mainly from the project managers and process engineers) in order to check and continuously improve the criteria and the scoring of variants.

(2) Process constraints	INC	CBD	PHA	EVP
Reusable software units or class library to be used	++	+++	?	+
Application is triggered by business processes	++	+	++	?
Business processes are (relatively) independent from each other	++	+++	-	++
Data-centred legacy system is to be integrated	+++	+	+	?
Complex, intertwined data structures	++	+	+	+
Many local, rather few global data	++	+++	?	?
Stable process conditions with a low probability of modifications	+	+	+++	-
Unstable process, requirements have still to be captured	++	++	-	+++
Large, complex system; high distribution of workload	++	+++	+	+
Cooperation project; work is widely shared with partners	+	+++	+	?

Fig. 12. Selection criteria: Process constraints

6 Conclusions

Faced with the requirements and goals of a wide range of different projects and partners in a large banking organisation we have come to the conclusion that there is no chance for a "uniform" software process. In order to cover a broad spectrum of projects we have presented a new approach to software process modelling based on a two-level framework: a base level defining all the raw material (e.g. sets of well-organised activities, techniques, results and roles) and a composition level combining the given material to process variants.

Well-known project variants like incremental, component-based, phase-oriented and evolutionary development can easily be described by using our simple bubble-chart notation. All variants are based on the same framework of activities, result types and techniques and are thus equally supported by the AD model.

Based on our experiences and subjective assessments we have developed an elaborated catalogue of criteria for the selection of the adequate process in a given situation. We believe that both the framework and the catalogue of criteria can easily be adapted to further process variants which might be included in the future. This way, the AD model and handbook constitutes an extensible piece of technology which can always easily be adapted to the current goals, requirements and practices.

The multi-variant approach reduces the average expenses for tailoring the process model to a broad range of projects in GSBO while it guarantees the applicability of several proven process patterns. The positive reaction of our user community, i.e. the project managers and engineers in the development centres in GSBO, is very promising and stimulating for our further work. In a next step we are going to implement our approach in a Web environment using HTML and JAVA for browsing and navigation.

Acknowledgement

We thank our colleagues Klaus Heuer for fruitful discussions and Stephan Düwel for his careful reading of the manuscript.

References

1. G. Booch: Object-Oriented Analysis and Design with Applications. Second Edition, Benjamin/Cummings Publ. Comp. 1994
2. A.M. Christie, A.N. Earl, M.I. Kellner, W.E. Riddle: A Reference Model for Process Technology. In: C. Montangero (Ed.): Software Process Technology, 5th European Workshop, EWSPT 96. Springer LNCS 1149, pp. 3-17 (1996)
3. C. Floyd: A systematic look at prototyping. In: R. Budde, K. Kuhlenkamp, L. Mathiassen, H. Züllighoven (eds.): Approaches to prototyping, Springer 1985
4. B. Henderson-Sellers, J.M. Edwards: Object-oriented software systems life cycle. Comm. of the ACM Vol. 33, No. 9 (1990)
5. 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)

6. W. Hesse: Life cycle models of object-oriented software development methodologies. In: A. Zendler et al.: Advanced concepts, life cycle models and tools for object-oriented software development. Reihe Softwaretechnik 7, Tectum Verlag Marburg 1997
7. W. Hesse: Improving the software process guided by the EOS model. In: Proc. SPI '97 European Conf. on Software Process Improvement. Barcelona 1997
8. W. Humphrey: Managing the software process. Addison-Wesley 1989
9. I. Jacobson: Object-Oriented Software Engineering - A Use Case Driven Approach. Revised Printing, Addison-Wesley 1993
10. M.M. Lehman: Programs, life cycles, and laws of software evolution. Proceedings of the IEEE. Vol. 68, No. 9, pp. 1060-1076 (1980)
11. J. Noack, B. Schienmann: Designing an Application Development Model for a Large Banking Organization. Proc. 20th Int. Conf. on Software Engineering, IEEE Computer Society Press, Kyoto, pp. 495-498 (1998)
12. J. Noack, B. Schienmann: Introducing Object Technology in a Large Banking Organization. IEEE Software (1999, to appear).
13. J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, W. Lorensen: Object-Oriented Modeling and Design. Prentice Hall 1991
14. Unified Modeling Language (UML) Version 1.1 Specification, OMG documents ad/97-08-02 – ad/97-08-09 (1997)