# Frameworking

Stefan Roock[a] & Henning Wolf[b] & Heinz Züllighoven[c]
1roock@informatik.uni-hamburg.de[a]
1wolf@informatik.uni-hamburg.de[b]
zuelligh@informatik.uni-hamburg.de[c]
Software Engineering Group
Dept. of Computer Science
University of Hamburg
Vogt-Kölln-Straße 30
+49 40 5494 2306

## Abstract

*Frameworks play a role of increasing importance in today's software development, especially in the construction of large systems. They are seen as software products used to gain a higher degree of re-usability. This paper claims a new perspective on frameworks complementing the actual product oriented view: the process oriented view. To underline this change of view we coined the term 'frameworking' to focus on the process aspects of creating and using frameworks.*

*With this frameworking perspective we examine several aspects of the creation and usage of frameworks. The results of the examination direct us to the insight that frameworks are not simply software too. They have to be considered as something special, demanding new or at least improved process models for software development and management.*

**Keywords:** Framework, Frameworking, Software Development Process, Management Process

**BRT-Keywords:** FA, EE, EF

# Motivation

Frameworks are a promising approach for achieving a higher level of re-use resulting in more quality and efficiency (Nulden, 1997). They provide a convenient way for re-using both concepts and code in software projects. First experiences with frameworks are accumulated in different contexts (Bäumer, Gryczan, Knoll, Lilienthal, Riehle, Züllighoven, 1997, Bohlmann, Fricke, Lippert, Roock, Wolf, 1998, Roger 1997). While the benefits of using frameworks are not controversial, the development process and the process of using frameworks are still unclear. Recent work focuses on frameworks as products and related aspects like architecture (Bäumer, Gryczan, Knoll, Lilienthal, Riehle, Züllighoven, 1997). We need to consider the general process of framework development and its impact on the development of application software. Since frameworks are not simply software, we cannot directly re-use the knowledge gathered about the software development process. As we will show, there are many unanswered questions concerning the process of creating and using frameworks.

The overall question we are focusing on in this article is: What are the impacts of the development and usage of frameworks on the application development process? The question will not be answered to its full extent within this article. The first goal is to find and discuss the open issues caused by introducing frameworks into the development

process.

To underline this change of view we use the term *frameworking* when dealing with the process aspects. The duality of the terms *framework* and *frameworking* is very similar to the duality between the terms *prototype* and *prototyping*. It expresses the duality of product and process in their specific contexts.

# Introduction

The term ‘framework’ is commonly used in our context in the sense of a software framework that is constructed in an object-oriented way using widespread programming languages like C++, Java or Smalltalk. In fact software frameworks are a common choice when we aim at a high level of code re-use. But reducing the term ‘framework’ to a piece of software is a somewhat narrow view.

We can distinguish several types of frameworks by their usage within the development process. There can be conceptual frameworks during analysis, design frameworks during design and software frameworks during the implementation of a software system.

What have these different types of frameworks in common? They provide support for the different activities of the software development process. Therefore, it is of secondary importance to the degree to which frameworks are represented as software themselves. It is far more crucial that the various frameworks we use should enable a seamless modeling and development process. Then, we can reach a structural similarity between the application domain and the respective application software. As Bäumer (1998) has shown, structural similarity is an essential precondition to understandability, maintainability and the various aspects of usage quality (cf. Lilienthal, Züllighoven, 1997, Meyer 1997, Walden, Nerson, 1995).

But in order to achieve this structural similarity, it seems obvious that the frameworks used during the development process must show appropriate characteristics as well. At least it must be easy to map the concepts of the different frameworks to each other.

Looking at the situation at hand we have identified two levels of discourse: we are dealing with the application software and the frameworks for building these applications. On each level we have the duality of product and process. For object-oriented application software a lot of experience has been gathered and published in the literature covering both the product and the process aspect. With frameworks the situation is different. They are not merely software. Within recent years we have gained an initial understanding of what frameworks as products are. But, as we have hinted, these concepts need to be elaborated and refined. With respect to the process view, however, not much has been systematically collected so far.

The rest of this paper, therefore, tries to fill a few of the already identified gaps. First, we will present a more detailed discussion of the terms and concepts related to frameworks for software development. Then we will look at the role of frameworks within the development itself.

# The term ‘framework’

In the introduction we claimed that it is useful to distinguish different types of frameworks in the development of a software system. Let us look at it in more detail:

- A *conceptual framework* helps us to identify items of relevance and interest in the application domain. They shape our vision of the future system. An example of a conceptual frameworks is the tools and material metaphor (Riehle, Züllighoven, 1995). Other conceptual frameworks emerge in almost every application domain (e.g. Tang, Leifer, 1988).
- A *construction (design) framework* helps to transform our model of the application domain into a technical design of a software system. A set of design patterns (see Gamma, Helm, Johnson, Vlissides, 1994) or an architectural style (see Monroe, Kompanek, Melton, Garlan, 1997) may serve as a framework during the design of software systems.
- *Software frameworks* are used while technically constructing and implementing an application system. Small to medium scale software frameworks provide a domain-specific or generic service. Commercially available software frameworks deal with the technical aspects of software construction. They usually provide a container library, base classes for dialog construction and a binding to the user interface (e.g. Bohlmann, Fricke, Lippert, Roock, Wolf, 1998, Vlissides, Linton, 1990, Gamma, 1992).

  In order to support the implementation of an entire system there are emerging sets of software frameworks which are called *application frameworks* (see Rogers, 1997, Bohlmann, Fricke, Lippert, Roock, Wolf, 1998, Vlissides, Linton, 1990, Gamma, 1992). They are still few and mainly in-house products. Domain-specific frameworks use or contain generic or technical frameworks. One such framework is described by Bäumer, Gryczan, Knoll, Lilienthal, Riehle and Züllighoven (1997).

Having identified several types of frameworks, we can now look at a definition of the term *framework*. In the domain of application frameworks we often find definitions similar to that given by Rogers (1997):

> A framework is a class library that captures patterns of interaction between objects. A framework consists of a suite of concrete and abstract classes, explicitly designed to be used together [...] (Rogers, 1997, p.4)

This is true for software or application frameworks but is not sufficient, since every object-oriented program consists of collaborating classes. The definition does not help us to distinguish between a framework and an application program. The definition shares the shortcomings of another even shorter definition:

**A framework is a set of collaborating classes.**

Beside the mentioned aspect, both definitions deal only with software frameworks and do not cover analysis or design frameworks which contain no actual classes or implemented code. First we extend the given definition to all kinds of frameworks in the software development context. Since classes should be implementations of concepts we simply substitute these terms and get:

**A framework is a set of collaborating concepts.**

Now we have the necessary condition for frameworks but are still in search of the sufficient ones. When do collaborating concepts form a framework? When we say that these concepts are explicitly built for re-use we arrive at our definition of the term framework:

***A framework is a set of collaborating concepts shaped for re-use.***

This definition is both necessary and sufficient and covers conceptual frameworks as well as software frameworks. In software frameworks we may look at classes as implemented concepts. This way every software framework contains concepts as well. So it is possible to match classes of software frameworks to concepts of conceptual

frameworks.

Now that we have a precise definition of the term ‘framework’ we can look at its role within the application development process. The general relations between application domain, application development and framework development are shown in Figure 1.


**Figure 1: General relations between application domain, application development and framework development**

Within the application domain, requirements for a software system emerge. These lead to the initiation of an application development project. The operative software is fed back as a solution to the application domain. In order to gain an efficient application development process or to create re-usable components for succeeding projects, it may be decided to development applications based on frameworks. Within the application project requirements for a new or existing framework emerge and are a starting point for the (further) development of the framework. The framework components as outcome of the framework development process are taken by the application developers as solutions to their requirements. This way the framework development process is connected with the application domain in an indirect way. Although this connection is only indirect the application domain affects the framework development process and the framework development affects the application domain through the application system built upon the framework components.

# Software Development, Management Activities and Frameworking

We now discuss the various activities of software development and management in relation to frameworking. As an outline structure we use the different activities of the development process. The traditional waterfall model was introduced as a prescription for how to organize and manage a software project. (Boehm, 1976, Royce, 1970). A lot has been said about why this approach is neither conceptually nor technically feasible. Evolutionary, cyclic models today mark the state of the art in this respect (cf. Floyd, 1993). But as a common denominator for discussing the various activities a software developer has to cope with during analysis, design and implementation, the waterfall model may serve as well. The waterfall model is shown in Figure 2.

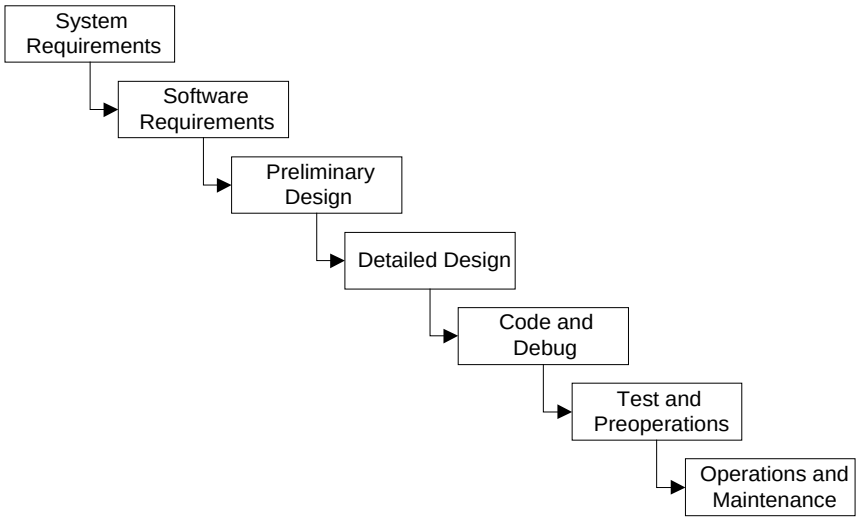


**Figure 2: The activities of a software development process as depicted by the waterfall model**

We will use this activity structure for discussing the process aspects of frameworking. Therefore there is a section discussing each activity.

In addition to the software development activities we have to consider the management of software development projects (cf. Figure 2). In line with management theory we have to deal with *planning*, *regulation* and *controlling* (Andersen, Kensing, Lundin, Mathiassen, Munk-Madsen, Rasbech, Sørgaard, 1990, Wöhe, 1993).

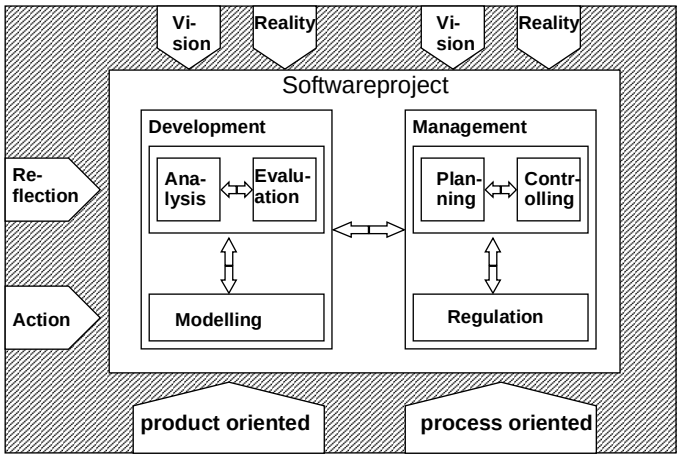


**Figure 3: Aspects of software development and management**

After the discussion of the activities of the software development process we discuss the management aspects.

Each of the following sections dealing with the activities of the development and the management process considers three questions:

1. How is this activity dealt with in conventional software development processes?
2. What are the impacts of using frameworks in this activity? This question deals with

the product view on frameworks.
3. What are the special requirements on the framework development process for this activity? This question considers the process view of frameworks.

## System Requirements

While analyzing and defining system requirements in individual projects we are looking at specific requirements at hand. Each project differs from every other project because it has different objectives, participants and conditions. But as different as the individual software projects are, as urgent is the need for guidance during this activity. Where to start, what to look at, how to describe the findings in what type of model? These are the questions for the software team.

Obviously, design or software frameworks are tools of little help for these activities of the application development process. Even when the new application will be based on an existing framework, these issues should be separated from the analysis and modeling of the application domain. Otherwise there is the danger of missing specific requirements of this special project because technical issues in terms of the existing software frameworks dominate the view.

But conceptual frameworks might help. They can guide our view of an application domain to the relevant tasks and objects of work. If they contain design metaphors, like tool and material (cf. Riehle, Züllighoven, 1995), they provide us with a language to communicate with users and build models that can be understood in the application domain. But still the individual project is in focus here.

The situation is quite different with system requirements for a framework development process. There we have to abstract from the different project specific requirements. The focus is on an entire application domain or at least on a generic technical service.

So the question is whether we can analyze and define system requirements for frameworks before any projects in this specific application domain or service area exist. This has been tried but with little success (cf. the Taligent story). The reason is obvious. Frameworks of every type aim at re-use. Obviously, as B. Meyer once said, ‘use’ comes before ‘reuse’. Without any specific project knowledge of at least two projects, you cannot decide what concept or software support a framework should provide. Or, at least, you are unable to say which requirements are domain and which are project specific. This leads to a framework development process which is based and partly combined with more than one application development process. Since it is always difficult to communicate experiences between developers, it is promising to share developers between the application and the framework development projects in an application domain. That way the framework developers experience the problems within the application system projects and have a better chance to identify useful abstractions for the framework.

## Software Requirements

Software requirements within an application development process aim at the individual software product and its general features or prerequisites. Here, similar to what we have said about system requirements, the individual project context is of primary relevance. But when looking at software requirements additional aspects, besides the application domain, come into focus. We have to realize that today no software is built from scratch or used in isolation. So there are more dimensions and contexts to consider. Bäumer

(1998) identifies the used and available technology and the existing system basis as contexts which are relevant as well as the application domain. Züllighoven (1998) points to the importance of identifying the appropriate ways and means of handling a software system (frequently called its look and feel).

A framework can support this activity insofar as it outlines the different contexts and dimensions to look for. For every application system we then have to decide where and to what extent to deviate from the existing framework, be it on a conceptual or a technical level.

For software requirements of the framework development process similar considerations apply as mentioned in the previous section. The various requirements from the different contexts and dimensions have to be generalized and abstracted. This can be done only on the background of several application projects. But here a specific problem comes into focus. Software requirements for frameworks have to be identified on two levels - the application context of the framework and the application context of the software systems built with the help of this framework. While the latter might for example be banking or health care, the application context of a framework is software development. So the structure of application frameworks at least has to reflect this. What does it mean? Software requirements for a framework have to reflect the fact that a family of applications in a specific or generic domain has to be built. Therefore aspects of re-use, unforeseen changes, potential variants and versions have to be considered besides the ‘conventional’ aspects discussed for application software requirements.

## Preliminary Design

The preliminary design is used to build first application-oriented models of the future system. Therefore the software developer has to analyze and understand the concepts of the application domain. With respect to this analyzing activities some authors name this phase *Analysis*. What is often overlooked is that, besides the domain specific aspects, a usage model has to be designed as part of this activity, which means a model of how a user can understand and work with the application system in a comprehensive and uniform way.

Both aspects of preliminary design are greatly supported by an appropriate conceptual framework. It links the concepts and activities of the existing application domain to the model and use situation of the future system. It can help to establish a structural similarity not only with respect to software components and application concepts but also between means and objects of the existing work situation and the future system usage.

Using software frameworks within the development process we can distinguish two situations: When we use software frameworks with few conceptual or application specific parts, these frameworks have no major impact on the preliminary design. When we use software frameworks with many conceptual parts we have to deal with the impacts. The models built during the preliminary design have to match the concepts found in the used framework(s). To match the concepts it is first necessary to understand the concepts of the framework. This is, especially for the domain specific parts of the framework, often very hard and costly in terms of time and money. We think that this is not only caused by incomplete and hard to understand documentation. We take the size and complexity of domain specific frameworks to be the reason for this problem. A framework has to be built by very experienced developers sharing a common culture in development and a very profound understanding of the application domain. Such a framework cannot simply be bought and used like an ordinary commodity. In order to

understand the framework fast and to get the most out of it, a framework should never be given as an isolated ‘resource’ or ‘tool’ to the application development project. A framework should be accompanied by a massive transfer of application know-how and technical or conceptual knowledge. So, support from one or more framework developers or an experienced senior software architect is needed to instruct and coach the application system developers.

At least for the construction of software frameworks we have to create a preliminary design as well. As was said in the previous section, we have to deal with a special application domain: the software construction itself. This may seem an easy task for a software developer since that is what he does all the time - build software systems. But the everyday experience does not automatically imply any kind of explicit knowledge. This is similar to the insider-outsider dilemma described by Bodker and Strandgaard Pedersen, (1991): First the developer is an outsider to the application domain. To build a useful system he has to gain deeper insight into the domain. When the developer not only gains knowledge about the application domain but also shares the workplace culture, he becomes an insider. As an insider he has the ability to work within the domain but has problems reflecting the cultural rules of the domain. But this distance is needed to question existing work procedures and build innovative and useful systems. This is similar to the situation of a developer of a generic technical framework, since he has to build models about his own work. Therefore it is in fact a mistake with possibly severe consequences to assume the creation of a technical framework to be an easy job.

The situation is aggravated when dealing with application frameworks. Here, the application domain knowledge has to complement the technical knowledge of the developer. New design problems have to be solved. In (Bäumer, Gryczan, Knoll, Lilienthal, Riehle, Züllighoven, 1997) we have identified the need to design an entirely new level of abstractions, which we have called the business domain level, in order to ensure the collaboration of application components on an adequate semantic level. These issues are not yet widely considered but seem to us to be of primary importance for the success of large scale application frameworks.

## Detailed Design

During the detailed design the models built within the preliminary design are used to create formal models for the computer system. These models focus on the concepts needed to build an executable system while the models of the previously discussed activities focus on the application domain. The mapping between the application domain and the computer system is done during the detailed design.

Using frameworks can be of substantial help during detailed design. Design patterns and an architectural style help to shape the macro and micro structure of the future system. Software frameworks, especially application frameworks, contain a specific software architecture and a flow of control which influence or even define the structure and dynamics of the application software (see Bohlmann, Fricke, Lippert, Roock, Wolf, 1998, Vlissides, Linton, 1990, Gamma, 1992). Here it is essential that the concepts of the construction frameworks and the software frameworks fit. This means that at least the concepts behind software frameworks have to be made explicit. This has rarely been done by software frameworks until now. Here the discussion of the previous section also applies. ‘Using’ an experienced framework developer or software architect within the application development project is a convenient mechanism to communicate the concepts of the framework.

The detailed design of a framework itself does not require any specific basic

methods or techniques beyond the ones used for the detailed design of application systems. But a few items have to be dealt with in addition. First of all, the detailed design has to deal with a higher demand for flexible and scalable frameworks. Architectural issues have to be dealt with explicitly. Often a multi-dimensional layering structure is needed. In order to connect the different layers, design patterns are used. So, we have design patterns within one layer and as connectors among layers. First ideas and experiences have been published in this area (cf. Shaw, Garlan, 1996, Bäumer, Gryczan, Knoll, Lilienthal, Riehle, Züllighoven, 1997). Understandability is perhaps the most important aspect for frameworks, since the number of developers using frameworks outsizes the number of developers involved in a single application software process. In general, the detailed design of frameworks leads to a new level of complexity and a lot of still open questions.

## Code and Debug

During implementation the universe of discourse is restricted by the chosen implementation language and the experience of the implementing developers.

The implementation acitivity seems to be the home of software frameworks because frameworks are very often seen as re-usable class libraries. (As we have shown, a general concept of frameworks goes beyond that notion). Software frameworks contain re-usable software components and already implemented abstractions. They extend the basis of the implementation beyond language and experience. So they can support a quicker, cheaper and easier development resulting in higher quality software products.

But frameworks are not the only re-use method. Not every class that is used by two others should be part of a framework. Blaschek and Pomberger (1996) differentiate between internal and external re-use. Internal re-use is the re-use of parts developed in the same project; external re-use is the usage of parts developed in other projects, for example a framework development project. Nulden (1997) discusses several other aspects concerning re-use. One topic is the reuse of persons. As we have shown in previous sections it seems promising to reuse the experience of framework developers and software architects in application development projects.

An important support on the level of micro structures are programming patterns (cf. Riehle, Züllighoven, 1996).

Implementing software frameworks first of all means implementing software. But what makes the implementation more complicated is that software frameworks are at best partially operative by themselves. They have to be abstracted from and used with concrete applications. For application frameworks the sheer size of the software sometimes poses a major problem. Without proper architectural design the compiling and linking process, even after minor changes, can be really tedious.

## Test and Preoperations

In application development processes the implemented components are tested before the integrated system is tested. Most developers underestimate the importance of testing. This is sometimes also true for project managers since testing may become quite expensive in large projects. Not testing is in most cases even more expensive.

So the good news while using frameworks in an application development is that you develop fewer components so there is less to test. We assume that framework components are tested by framework developers and not by the application developer (see next section). But the integration of components and testing of integrated components

might be a complicated task because the framework concepts must be undestood by the application developers. This task is particularly complicated because frameworks mostly encapsulate control flow. The latter aspect makes testing and debugging sometimes really complicated since major (and complex) parts of the running system (i.e. the frameworks) may not be available as source code and detailed documentation is often missing.

The news is even worse for framework developers. Because of the generic and derived requirements of frameworks (see System and Software Requirements), the framework components have very universal contexts. This fact makes testing more complicated. Unfortunately this often means that the good news for application developers is short lived: When they use a framework component in a new context that has not been foreseen by the framework component developer the used component might be faulty. This is another reason for using cyclic development processes for framework development. Errors found in the framework by the application developers must be corrected immediatly and the corrections must be made available to all users of the framework.

On this background it is quite logical to change the point of view towards errors. While traditionally, errors are looked upon within a product as disturbing; we look quite differently at errors within the process view: They are welcome during the development process because they can deepen the insight into the application context and its reconstruction within the software system.

## Operations and Maintenance

Operation as an activity of the application development process is often combined with maintenance. In waterfall process models the maintenance phase covers all further development needs of the application. In cyclic and evolutionary process models operation will lead to a new development cycle and maintenance is seen as development.

When using frameworks, problems may occur because application code and framework code evolve rather independently. This is because the framework development process is combined being linked to more than one application development project. So, without a versioning strategy for frameworks, even a re-compilation of an application might fail.

Another observation is that there will be new requirements on the application during operation. These might lead to new requirements on the framework as well. Using frameworks in an application development process generates new requirements for the framework. Just as application users detect new requirements because of new possibilities through the usage of an application program, application developers will find more cases where they would like to be supported by a framework.

As indicated, an important aspect of framework development is the handling of versions. Since frameworks are used for the development of different software projects often located in different developer organizations, it is not always possible to synchronize the production of the framework with the production of the applications. Then, frameworks have their own problems and demands on the development process. What seems a small requirement from the application programmer’s viewpoint may call for a major reconstruction of a framework. Therefore frameworks evolve rather independently of their applications. But they have to fit with their applications and their further development.

Using frameworks which evolve over time leads to a major problem with the otherwise extremely useful open-closed principle introduced by Meyer (Meyer, 1997). The open-close principle means that in an object-oriented system the interface of a class

can be fixed (i.e. closed) for re-use by other clients; at the same time the interface can be evolved (i.e. open) through subclassing for further development. This well-established principle has to be revisited when further developing software frameworks:

Applications using software frameworks usually have to subclass several classes of that (so-called white-box-) framework. If a framework evolves this might mean a change of the superclasses for its applications, thus invalidating the application programs.

A possible solution to this problem is to introduce a life cycle of a framework. This life cycle means that a framework version goes through three stages over time:

1. The current version
2. The outdated version
3. The coming version.

At each point in time, a framework version is in one of the above three stages. The *current version* is the version that should be used by 'active' application projects. It is supported and maintained but the interfaces do not change. Any application shipped to a customer has to be based on the current version of a framework.

Applications in operation may use the *outdated version* of a framework. The outdated version is still supported but no longer enhanced. Developers maintaining an application which is based on the outdated version know that they have to migrate to the current version.

The *coming version* is the next coming framework version. It has a well-defined interface and functionality which are published when a version is declared to be the new current one. Software projects may use the coming version as a specification from the start in order to access new interfaces or functionality of a framework. The prerequisite is that they can ship their application only after the coming version has become the current one. The version life cycle of application programs and framework is shown in Figure 4. In our example version 3 of the framework is coming in July 1998, version 2 is the current version and version 1 is outdated.

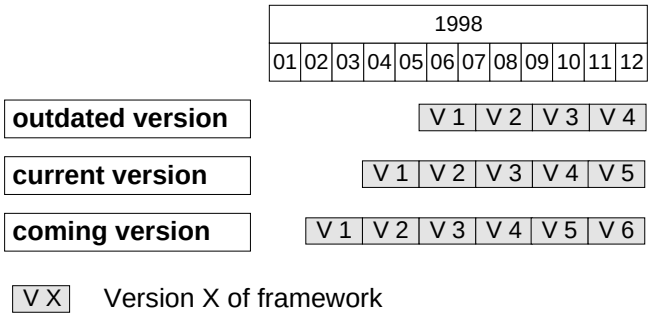


**Figure 4: Example versions of a framework**

## Planning, Regulation and Controlling

Planning is a complex task within software development. Surprisingly it is as often ignored as the tasks of regulation and controlling. Many authors seem to think that the management of software projects is not worth mentioning. Perhaps it is thought of as being superfluous since developers are highly qualified experts. Some seem to expect that they should manage themselves without any supporting concept or method.

We use several documents and concepts for the planning, regulation and controlling of the development process. First we use project stages for time planning. A project stage defines which consistent and comprehensive components of the system should be available and when. Base lines are used to detail the planning within a project stage. They do not focus on schedules but define what has to be done, who does it and

who controls it (cf. Krabbel, Wetzel, Ratuski, 1997).

In line with the project stages we divide the software system into a *core system* with *extension levels* (cf. Krabbel, Wetzel, Ratuski, 1997). The core system is a usable part of the software system which addresses certain important needs. It is developed first and then supplemented by *special systems*. Since the core system is usually still quite complex, it is subdivided into extension levels which are built upon each other. An example of a core system with extension levels is shown in Figure 5 (taken from the domain of hot rolling mills). The upper extension levels use the functionality of the lower extension levels. This way we get an application-oriented structure which is useful for planning the time schedule. It is obvious that the lowest extension level has to be created first, followed by the next higher one and so on.

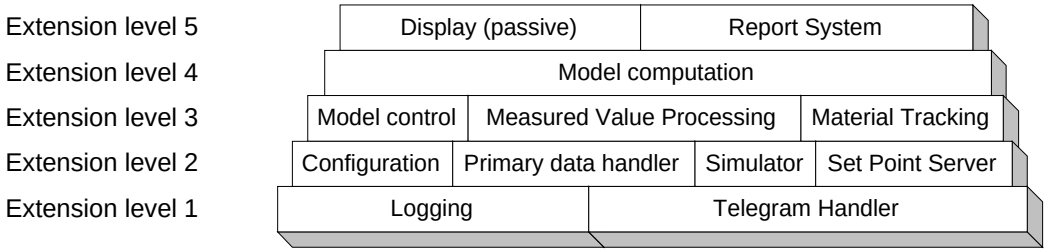


**Figure 5: Example core system with extension levels**

The special systems just add functionality to the core system. An example of a core system with special systems is shown in Figure 6 (again taken from the domain of hot rolling mills). The special systems are drawn as circles.

As special systems only complement the core, it is possible to deliver an operative and useful core system very early and get feedback from the users. In parallel maybe, different software teams can build special systems. Special systems should only use the core system but not the other special systems. This way, we achieve a maximum of independence among the special systems. They can be created in any order or even in parallel. Obviously the core system has to provide the basic functionality for the whole system, since it is the only possibility for the special systems to exchange information. Therefor the core system will usually provide a set of basic communication mechanisms allowing information transfer between different parts of the whole system.

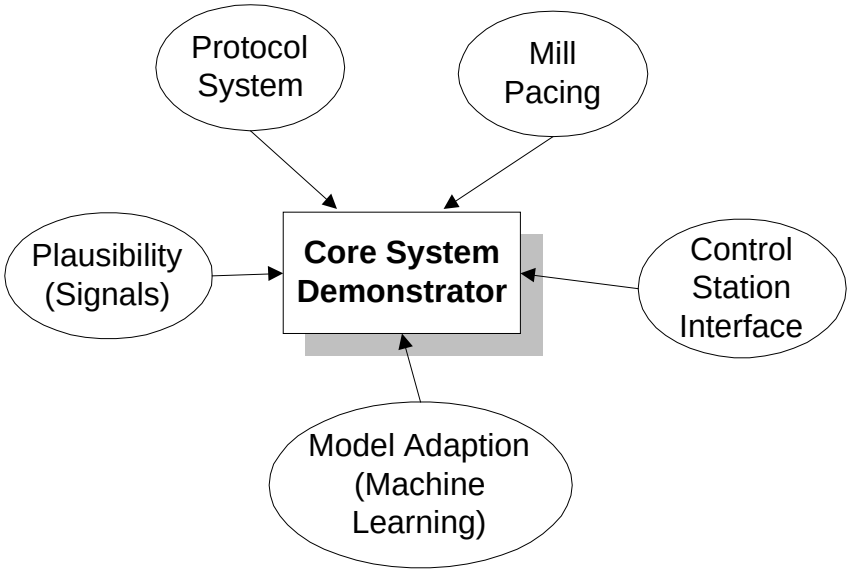


**Figure 6: Example core system with special systems**

Another important aspect of the management activities is the definition of the goal of the

project. It is crucial to notice that the production of the software system is *not* the primary goal. The goal is always derived from the application domain — often it means raising the work efficiency. To achieve this goal it can be appropriate to build a software system. So a software system may be the means to achieve the project goal but it is normally not the only one.

In mission critical projects, risk handling is an important aspect. The risks within the project have to be analyzed and assessed carefully.

We have experienced some success with core systems, extension levels, project stages and base lines in the context of framework development. These mechanisms are appropriate for planning the development process of frameworks as well.

The mechanisms described are also useful for regulation. Project stages are used to control the project progress and timeliness.

The goal(s) defined during the planning of the projects have to be controlled continuously. Therefore it is crucial to define criteria which allow the testing of goal attainment. Some kinds of metrics may be helpful.

The risks defined during the planning have to be monitored. If a risk becomes a problem, the project management has to react in a appropriate way.

We have used the concepts of core system with extension levels and special systems successfully within the REFORM (REusable Framework For Rolling Mills) project to plan the development of a demonstrator application. This application is used to show the usefulness of the REFORM framework.

# Conclusion and Outlook

We have argued that frameworks are seen from a product viewpoint today but application frameworks are not products in the strict sense. We opted for a complementary process view calling it *frameworking*. We discussed several aspects of frameworking from the development and the management point of view. The results of this discussion make clear that frameworks are not simply software. This statement holds true for the framework as a product as well as for the process of developing and using frameworks — the frameworking. Frameworks can only be (re-)used within a given background. This means that knowledge and understanding of the use context have to be transferred as well. A promising approach is re-using the framework together with framework developers and software architects. The deployment of experienced framework developers or software architects in application development projects is a suitable way to transfer knowledge and experience into the application development team. And it will also help to transfer and better understand new requirements from application development towards the framework.

Neither the optimal usage nor the development of frameworks is an easy task. Both are challenging and call for further research work.We see a need for research work on the development of an integrated application and framework process model. Solutions to the problems which we have described above have to be part of this model. New constructive methods have to be developed to avoid some of the problems.

Another important issue of our further research work will be the organizational aspects of the frameworking process. Which organizational structures are necessary? What are the qualifications that are necessary for application developers? Do they differ from qualifications for framework developers?

# Related Work

The REFORM project funded by the E.C. aims at developing a domain specific framework for hot rolling mills. The JWAM framework of the Software Engineering Group at the University of Hamburg is a Java framework supporting application development according to the tools and material metaphor (cf. Riehle, Züllighoven, 1995, Züllighoven, 1998). We will work on the processes of developing and using these frameworks.

ET++ (see Gamma, 1992) is a well known and very successful object-oriented framework written in C++. It is also a source of several successful patterns (see Gamma, Helm, Johnson, Vlissides, 1994). Like most other publications about frameworks, the articles and books about ET++ focus on the product view.

Bäumer (1998) describes a layering architecture for large domain specific frameworks. This layering structure is also the topic of Bäumer, Gryczan, Knoll, Lilienthal, Riehle and Züllighoven (1997).

Birrer, Bischofberger and Eggenschwiler (1995) describe different technical possibilities for re-using through frameworks like white-box and black-box frameworks. They also make clear that using frameworks is expensive and will not repay costs in the short-term, but will in medium-term. Another aspect of their work concerns the organizational conditions necessary for successful framework-based software development.

# Acknowledgement

We are grateful to the E.C., who is funding the research project REFORM (REusable Framework For Rolling Mills) of which this work is a part and we would like to thank our project partners for their inspiring discussions on the presented topics. Thanks also to Dirk Bäumer, whose experience with the development of large frameworks was a major source of many ideas in this paper.

# References

N. E. Andersen, F. Kensing, J. Lundin, L. Mathiassen, A. Munk-Madsen, M. Rasbech, P. Sørgaard. Professional Systems Development. New York: Prentice Hall, 1990.

D. Bäumer: Softwarearchitekturen für die rahmenwerkbasierte Konstruktion großer Anwendungssysteme. Disserationsschrift zur Vorlage am Fachbereich Informatik der Universität Hamburg, Januar 1998. German.

A. Birrer, W. R. Bischofberger, T. Eggenschwiler. Wiederverwendung durch Framework-Technik - vom Mythos zur Realität. OBJEKTSpektrum Nr. 5, 1995, München, SIGS Conferences GmbH, pp. 18-26. German.

Dirk Bäumer, Guido Gryczan, Rolf Knoll, Carola Lilienthal, Dirk Riehle, Heinz Züllighoven. Framework Development for Large Systems. Communications of the ACM, October 97, Vol. 40, No. 10, 1997.

Dirk Bäumer, Rolf Knoll, Guido Gryczan, Heinz Züllighoven. Large Scale Object-Oriented Software Development in a Banking Environment - An Experience Report. In: Pierre Cointe (Hrsg.): ECOOP'96 - Object-Oriented Programming, 10th European Conference, Proceedings, Springer Verlag, Linz, Austria, July 1996, pp. 73-90. 1996.

B.W. Boehm. Software Engineering. In: IEEE Trans. Comput., vol. C-25, pp. 1226-1241, Dec. 1976.

Keld Bodker, Jesper Strandgaard Pedersen. Workplace Cultures: Looking at Artifacts, Symbols and Practices. In: Joan Greenbaum, Morten Kyng (Hrsg.): Design at Work. New Jersey. 1991.

C. Floyd. STEPS - A Methodical Approach to Participatory Design. Communication of the ACM, June 1993/Vol. 36, No. 4, p. 83.

Erich Gamma. Objektorientierte Software-Entwicklung am Beispiel von ET++. Berlin, Heidelberg: Springer-Verlag, 1992. German.

Erich Gamma, Reinhard Helm, Ralph Johnson, John Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software. Reading, Massachusetts: Addison-Wesley, 1994.

Holger Bohlmann, Niels Fricke, Martin Lippert, Stefan Roock, Henning Wolf. Java Framework. University of Hamburg, Department of Computer ScienceSoftware Engineering Group. http://swt-www.informatik.uni-hamburg.de/~Software/JWAM. 1998. German.

Anita Krabbel, Ingrid Wetzel, Sabine Ratuski: Anforderungsermittlung für Krankenhausinformationssysteme: Definition von Kernsystem und Ausbaustufen, SoftKIS '97, W. Hasselbring. (Hrsg): Erfolgsfaktor Softwaretechnik für die Entwicklung von Krankenhausinformationssystemen, GI/GMDS-Workshop, Universität Dortmund, Krehl Verlag Münster, Februar 97, pp. 1-8. German.

Meir M. Lehmann. Programs, Life Cycles, and Laws of Software Evolution. In: Proceedings of the IEEE. Volume 68, Number 9. September 1980.

Carola Lilienthal, Heinz Züllighoven. Application-Oriented Use Quality, The Tools and Materials Approach. In: Interactions Magazine, ACM, November+December 1997, Volume IV.6.

Betrand Meyer. Object-Oriented Software Construction.. 2nd Edition. Prentice Hall. New Jersey. 1997.

R .T. Monroe, A. Kompanek, R. Melton, D. Garlan. Architectural Styles, Design Patterns, and Objects. IEEE Software, 14(1), January/February 1997, pp. 43 - 52.

U. Nulden. The Why, What, and How of Reuse in Software Development. In Kristin Braa, Eric Monteiro (editors), Proceedings of Iris'20: Social Informatics, pp.407-414. 1997.

Gustav Pomberger, Günther Blaschek. Software Engineering - Prototyping und objektorientierte Softwareentwicklung. 2nd Edition. München, Wien. Hanser-Verlag. 1996. German.

Gregory F. Rogers. Framework-Based Software Development in C++. Prentice Hall. New Jersey. 1997.

W.W. Royce. Managing the development of large software systems. In: IEEE WESCON. August 1970, pp. 1-9.

Dirk Riehle, Heinz Züllighoven. A Pattern Language for Tool Construction and Integration Based on the Tools and Material Metaphor. In: James O. Coplien and Douglas C. Schmidt (Eds..): Pattern Languages of Program Design. Reading, Massachusetts: Addison-Wesley. 1995.

D. Riehle, H. Züllighoven. Understanding and Using Patterns in Software Development. In: K. Lieberherr, R. Zicari (eds.): Theory and Practice of Object Systems, Special Issue Patterns. Guest Editor: S. Berczuk,Vol. 2, No.1, 1996, pp. 3-13.

G. Shaw, D. Garlan: Software Architecture. Perspectives on an emerging discipline. Prentice Hall, 1996.

John C. Tang, Larry J. Leifer. A framework for understanding the workspace activity of design teams. In CSCW'88 Proceedings of the Conference on Computer Supported Cooperative Work, Oregon, 1988.

John M. Vlissides, Marc A. Linton. UniDraw: A framework for building domain-specific graphical editors. In: ACM Transactions on Information Systems, 8(3): pp 237-268, July 1990.

Kim Walden, Jean-Marc Nerson. Seamless object-oriented software architecture : analysis and design of reliable systems. Prentice Hall. New York. 1995.

Günter Wöhe. Einführung in die Allgemeine Betriebswirtschaftslehre. Verlag Franz Vahlen München. 1993. German.

H. Züllighoven: Das objektorientierte Konstruktionshandbuch nach dem Werkzeug & Material-Ansatz. dpunkt Verlag 1998, to appear. German.