Managing Componentware Development – Software Reuse and the V-Modell Process*

Dirk Ansorge[1], Klaus Bergner[1], Bernd Deifel[1], Nicolas Hawlitzky[1], Christoph Maier[2], Barbara Paech[3,**], Andreas Rausch[1], Marc Sihling[1], Veronika Thurner[1], Sascha Vogel[1]

[1]Technische Universität München, 80290 München, Germany,
{bergner | deifel | rausch | sihling | thurner | vogels}@in.tum.de
as@iwb.mw.tum.de, Hawlitzky@ws.tum.de
[2]FAST e.V., Arabellastr. 17, 81925 München, Germany
cma@fast.de
[3]Fraunhofer Institute for Experimental Software Engineering,
67661 Kaiserslautern, Germany
paech@iese.fhg.de

Abstract. We present the characteristics of component-based software engineering and derive the requirements for a corresponding development process. Based on this, we propose changes and extensions for the V-Modell, the German standard process model for information systems development in the public services. Following this model, we cover not only systems engineering, but also project management, configuration management, and quality assurance aspects.

1 Motivation

Componentware provides the conceptual and technological foundation for building reusable components and assembling them to well-structured systems. However, the problem of creating and leveraging reusable software assets cannot be solved by technology only. It requires a methodological and organizational reuse infrastructure, integrating software engineering and project management issues. This includes, for example, the organization of a company-wide reuse process and the estimation of the costs and benefits involved with reusing or building a certain component. Furthermore, reuse inherently extends the management domain from a single project to multiple projects and from the context of a single company to the context of the global component market.

In this paper, we suggest some solutions for an integrated treatment of components in information system development. We identify the main characteristics that distinguish component software from non-component software, and sketch the resulting requirements for a component-oriented software engineering process. To be able to describe the necessary extensions and changes in a coherent way, we discuss the German standard for information systems development in the public service, called V-Modell [VMod], as a specific example for a "traditional" single-project process

* This work was supported by the Bayerische Forschungsstiftung under the FORSOFT research consortium.

** The work was mainly carried out while the author was at the TUM.

M. Jarke, A. Oberweis (Eds.): CAiSE'99, LNCS 1626, pp. 134-148, 1999.

model. The V-Modell is highly suitable as a reference model for our purpose as it covers all aspects of traditional software engineering, but was not designed with the componentware paradigm in mind. Although our proposals for extensions and changes are strongly related to the V-Modell, we think that they also apply to many other process models used today. We have, therefore, tried to keep our presentation general enough to be understood by and profitable to people who are not familiar with the V-Modell.

The context of typical software engineering projects ranges from the ad-hoc development of individual information systems within a small company over the distributed development of global information systems to the development of standard software for the global market. Furthermore, different application areas, like distributed systems or production systems in the area of mechanical engineering, impose additional constraints on the development process. In elaborating our proposals, we have tried to leverage the experience and knowledge within the interdisciplinary FORSOFT research cooperative, consisting not only of computer scientists and researchers from mechanical and electrical engineering, but also of economy experts and practitioners from leading companies in each of these four fields.

The paper is structured as follows: Section 2 identifies the main characteristics and requirements associated with componentware, both from the perspective of component users and component producers. Section 3 describes the V-Modell and sketches the overall approach we took in extending and adapting it. The next two main sections contain detailed proposals – Section 4 deals with single projects, while Section 5 treats the multi-project issues brought up by component reuse. A short conclusion rounds off the paper.

2 Componentware – Perspectives and Requirements

In order to discuss the issues involved with componentware process models, it is sufficient to define components as reusable pieces of software with clearly defined interfaces. To be reusable, a component has to be understandable, capturing a generally useful abstraction. Furthermore, its integration into many different, heterogeneous system contexts must be possible in a cost-effective way, implying tool support and interoperability standards, such as [Sam97, Nin96].

To discuss reusability in an appropriate way, two different perspectives must be taken into account: The perspective of the component (re-) user and the perspective of the component producer (cf. Section 2.1, 2.2).

If we try to relate the user and producer perspectives to traditional software engineering and development process concepts, we can find a strong correspondence to the notions of top-down versus bottom-up development. The top-down approach starts with the initial customer requirements and refines them continually until the level of detail is sufficient for an implementation with the help of existing components. Conversely, the bottom-up approach starts with existing, reusable components which are iteratively combined and composed to higher-level components, until a top-level component emerges which fulfils the customer's requirements.

Obviously, a pure bottom-up approach is impractical in most cases because the requirements are not taken into account early enough. However, the top-down approach also has some severe drawbacks: Initially the customer often does not know all relevant requirements, cannot state them adequately, or even states inconsistent require-

ments. Consequently, many delivered systems do not meet the customer's expectations. In addition, top-down development leads to systems that are very brittle with respect to changing requirements because the system architecture and the involved components are specifically adjusted to the initial set of requirements.

In our eyes, the central requirement for a component-oriented development process is to combine and reconcile the top-down and the bottom-up approaches within a unified process model. In order to capture the requirements for such a process model, the next two subsections will first sketch the perspectives of component users and producers in more detail. The third subsection will discuss possibilities and scenarios for the integration and combination of both viewpoints.

2.1 Using Components for Application Development

Usually, the development process will start with eliciting and analyzing the requirements of the customer. Then, the overall system architecture is designed, and specifications for the involved subsystems and base components are evolved. When a candidate component has been identified, the central question for a component user is whether to make it or to buy it. This decision critically depends on a comprehensive search for existing components. For their assessment, understandable specifications of the involved syntactic and behavioral interfaces are needed.

Sometimes, the available components will not match the customer’s requirements or the system architecture exactly. If it is possible to adapt such components with costs lower than the costs for making a substitute, they should be used anyway. The components’ properties may also trigger changes to the system architecture or even to the customer’s requirements, for example, when the customer decides to standardize his business processes according to a best-practice approach realized by an available standard component.

Using a component bears chances as well as risks. On the one hand, application developers have the chance to focus on their core competencies instead of wasting resources on re-creating readily available components for base services or for the technical infrastructure. By only providing added value on top of existing building blocks, they can considerably reduce their overall costs and their time-to-market. On the other hand, using available components may imply the strategic risk of dependence from certain component producers, and it may mean to give up valuable in-house knowledge in a certain area.

2.2 Producing Marketable Components

The business of a component supplier is the construction and selling of reusable components on the global component market. The customers and system contexts for such commercial off-the-shelf (COTS) components are not known a priori and may differ very much from each other. [CaB95] and [Dei98] present the basic problems arising in the development and requirements engineering of COTS in general.

The large number of potential customers means that the development costs can be distributed over all sold components, and allows the component producer to specialize in a certain area. The cost-effectiveness of component production therefore depends on the reusability of the produced software components. This implies two things:

First, component producers have to know the market and the requirements of their possible customers very well. COTS producers, therefore, have to perform a comprehensive and careful risk analysis and market studies during their requirements elicitation and product definition phase. To receive periodical feedback from their customers and to react directly to their needs, COTS suppliers normally develop their products in short release-cycles. Typically, they also try to acquire and retain customers by offering various license models, for example, for evaluation, non-commercial, and commercial usage.

Second, COTS components should be reusable in as many contexts as possible, and the adaptation costs for customers should be low. This implies that components should be very robust and must be thoroughly tested - implementation and documentation errors will be fatal for many customers and eventually also for the producer. Furthermore, support like design-time customization interfaces, adaptation tools as well as different variants for different hardware and software standards or platforms should be offered.

2.3 Integrating User and Producer Perspectives

Analogous to other markets, there are many possibilities for reconciling and integrating the idealized viewpoints sketched in the previous two subsections. Generally, the distinction between component users and component suppliers is not a strict one.

When components consist of other, lower-level components, both roles blend into each other, as the supplier of such a component also acts as a component user. There may also be companies that are neither true users nor true suppliers because they only act as link between both worlds. An example would be a company that acts as a component broker, identifying the requirements of many customers and elaborating specifications for components that result in orders to dedicated component developers.

Another common example is an in-house component profit center that selects and produces components for a limited number of customers within a certain company. As such in-house producers are only concerned with reuse within the context of a company and its in-house compliance standards, the necessary generality of the components is reduced. In practice, there is often no clear separation between in-house profit centers and COTS suppliers. Before a COTS supplier ships new components, he will usually reuse these components in-house to test them. Conversely, an in-house supplier may decide to sell a component that has been reused in-house successfully as a COTS component on the global market.

We expect that interface organizations like the ones mentioned will play an important role in a fully developed component marketplace, much in analogy to existing distributor hierarchies in established markets.

3 Adapting and Extending the V-Modell

The V-Modell [VMod] is one of the few examples of an integrated treatment of software engineering and project management issues. The overall development process is structured into four different sub-models, namely *Systems Engineering* (SE, “Systemerstellung”), *Project Management* (PM, “Projektmanagement”), *Quality Assurance* (QA, “Qualitätssicherung”), and *Configuration Management* (CM, “Konfigurationsmanagement”). Each sub-model includes a process and detailed guidelines for

how to produce the different development products, for example specification documents and source code. Together, all development products make up the so-called *Product Model* ("Erzeugnisstruktur").

In the following two subsections, we will describe the V-Modell's approach in defining the process and its overall structure, and discuss our proposals for general changes and extensions regarding component oriented development.

3.1 Flow-Based vs. Pattern-Based Process Description

The V-Modell's overall approach in defining the process is flow-based. Development products are elaborated within activities and flow into other activities that need them as input. This structure is reflected in the scheme for the activity description:

from		product	to	
activity	**state**		**activity**	**state**
Activity a1	accepted	development result r1	activity a2	processed
...	...	...	...	...

The scheme means that the corresponding development activity receives the development result r1 in state accepted as input from activity a1. After completion of the activity, the state of r1 is set to processed and the result is forwarded to activity a2.

In the V-Modell, this flow-based definition approach is combined with a rather traditional process which is more or less structured into sequential phases. As mentioned above, a sequential process does not fit very well to the bottom-up aspect needed for componentware development. Furthermore, the flow-based formulation of the process model limits the developers' freedom to react flexibly on unforeseen events because the prescribed activity flows must be followed.

The V-Modell tries to alleviate these problems by two measures:

- Feedback loops are introduced into the product flows. This allows to rework and correct products by repeatedly performing certain activities, resulting in an iterative process.
- Informal scenarios provide additional guidance in scheduling the order of activities in the process before the start of a project. With respect to componentware, the scenarios *Use of Ready-Made Components* ("Einsatz von Fertigprodukten") and especially *Object-Oriented Development* ("Objektorientierte Entwicklung") seem to be most adequate.

In our eyes, both measures can only partly solve the above mentioned problems. While the introduction of iterations does not fully remove the rigidity of the prescribed processes of the flow-based model, the scenario approach is very informal and does not match very well with the rest of the process model. Furthermore, the scenario has to be selected before the start of a project and provides no support for modifications during its runtime.

To remedy the deficiencies of the V-Modell, it must be reworked in a more principal way. We propose, therefore, to switch from the current flow-based description to a pattern-based model. Here, the state and consistency of the hierarchical product model together with an assessment of the external market situation form the context for so-called process patterns which recommend possible activities. This way, top-down and

bottom-up activities may be selected and performed based on the process manager's assessment of the current development situation. We have motivated and described our pattern-based approach in detail in [BRSV98a,BRSV98b,BRSV98c]. Apart from a high-level description of a suitable product model, we have elaborated a comprehensive catalog of process patterns for component-oriented software engineering.

We think that the V-Modell should be reworked in order to serve as the basis for a complete pattern-based process model with a detailed product model, spanning all activities of the development process. To achieve this, the basic description scheme should be changed. Instead of describing the flow of products between sequential activities, as shown above, the pattern of activities necessary to create products or establish consistency conditions between products should be described without relating to preceding or subsequent activities. The resulting scheme could look like this:

pre	
product	**state**
product p1	complete
...	...

process pattern p1
activity a1
...

post	
product	**state**
product p2	in progress
...	...

This scheme means that this process pattern is applicable when development product p1 is completed (the conditions related to the external situation are not shown here). After completion of the activities described in the process pattern, the development product p2 is created and set to state in progress.

3.2 Single-Project vs. Multi-project Development

In its current form, the V-Modell aims at the management of single projects, where software is developed more or less "from scratch". Componentware brings up two kinds of new issues:

- There must be support for the reuse of components within a single project. This requires new engineering activities, for example, searching for existent software components. Furthermore, existing activities have to be extended or must change their focus. Testing, for example, now has to deal not only with software that was developed in-house, but also with components from external suppliers.
- The process has to be extended in order to support multi-project issues. This relates, for example, to establishing a company-wide reuse organization, or to maintaining a repository containing components that can be reused by many projects.

The V-Modell uses so-called roles to describe the knowledge and skills necessary for carrying out certain activities appropriately. These roles are then assigned to the available persons on a per-project basis. Some example roles are *System Analyst*, *Controller*, or *Quality Manager*. Similar to our process-changes, we also had to both modify existing roles and introduce new ones in order to reflect the additional and changed skills. The *System Analyst*, for example, is now also responsible for assessing the business-oriented functionality of existing systems and components as a prerequisite for the elicitation of the customer requirements (cf. Section 4.1). An example for a completely new role is the *Reuse Manager* who has to identify the need for new components within a business organization, and propagates the usage of available components within development projects (cf. Section 5.2).

Based on the distinction of single-project and multi-project issues, we propose the following overall approach for extending the V-Modell with respect to component-oriented development:

- The existing V-Modell is slightly extended and modified in order to cope with the additional requirements for componentware. Leaving the structure of the V-Modell intact has the advantage that project managers do not have to adopt a totally new process model, but can build on existing knowledge.
- To deal with multi-project issues, new multi-project counterparts for the existing sub-models are added to the V-Modell. This extension reflects the deep impact componentware has on the organization of a company, but makes it possible for companies to introduce the necessary changes step by step and relatively independent from single projects.

The following figure visualizes the structure of the extended V-Modell:

	Single-Project Issues	Multi-Project Issues
Software Engineering		
Project Management		
Quality Assurance		
Configuration Management		

The next two main sections are organized as follows: Section 4 deals with the four sub-models in the context of a single project, while Section 5 is concerned about the multi-project issues.

4 Single Project Issues

Reuse enforces two basic principles on a software development project. On the one hand, standardization is necessary to support integration of components, and on the other hand flexibility of process and product is important in order to react to the dynamic component market. Standardization in software engineering activities requires, in particular, a well-defined component concept, standardized description techniques, separation between business-oriented and technical issues and a detailed assessment of components to be integrated. Flexibility implies a detailed search for adequate components, a close intertwining of component search, architectural design and requirements capture as well as an emphasis on adaptation and configuration in design and implementation. For project management flexibility in process definition and market activities is required and standards for supplier, contract and variant management. Configuration and quality management need to set standards for a component library, the integration of components and their test.

In the following subsections we discuss the impact of these principles on the activities and roles of the sub-models of the V-Modell.

4.1 Systems Engineering

The sub-model Systems Engineering (SE) subsumes all activities related to hardware and software development as well as the creation of the corresponding documents. It defines nine main activities which are carried out more or less sequentially: Starting

with *Requirements Analysis* (SE1), specific software and hardware requirements are analyzed. Then, the system is designed and split up into a hierarchy of logical and physical components (SE2 to SE5). These components are then implemented (SE6) and finally integrated (SE7 to SE8). Finally, the last main activity, *RollOut* (SE9), takes care of how to install the system in its specified environment, and how to configure and execute it properly.

Clear Component Concept: The V-Modell offers a variety of structuring concepts, reaching from *IT Systems* over *Segments*, *Software and Hardware Units*, *Software Substructures*, and *Components* to *Modules*. However, none of these concepts is defined clearly or even formally. This forces each development project to come up with its own incompatible, proprietary interpretations which hinders the reuse of existing components from other projects. To remedy this situation, we recommend to unify the structuring concepts by defining a uniform concept for hierarchical components. A first approach may be found in [BRSV98c].

Standardized Description Techniques: Apart from a clear component concept, reuse requires well-defined, standardized description techniques for the structure and behavior of components. This includes mostly so-called black-box descriptions of the component's external interfaces. During adaptation of pre-fabricated components in some cases also white-box descriptions of the internals of a component are necessary. Proposals for component description languages and graphical description techniques can be found in [UML97,SGW94,HRR98,BRSV98c]. The V-Modell yet only gives some recommendations on what to specify (for example, functionality, data management, exception handling of a component) and which description techniques may be used (for example, E/R diagrams or Message Sequence Charts), but it does not contain guidelines or requirements for component descriptions.

Reuse Activities: Although the V-Modell contains some remarks on reusing existing components in its activity descriptions, a comprehensive treatment of this aspect is missing - system development is mainly seen as starting out from scratch. To provide support for reuse, some activities have to be changed or to be added:

- *Search for Existing Components* should be an explicit activity. Its main purpose is to build an in-project component repository (cf. Section 4.3) based on selecting suitable components from public or in-house repositories (cf. Sections 4.3, 5.2). The V-Modell should provide standard activities for adding, deleting, editing, and searching components and the different versions and variants of these components.
- Based on the criticality of the involved existing components, detailed evaluations have to be carried out in order to assess their suitability and compatibility with the existing design. The *Component Evaluation* activity includes not only validation, verification, process and product tests, but also harmonization of the component and the system quality (cf. Section 4.4). To avoid costly workarounds it should be completed before making the definitive decision to use the respective component. A *Replacement Analysis* may additionally serve to be able to estimate the costs involved with changing architectures and using a substitute component.
- The *Design* and *Implementation* activities change their focus. Instead of designing and coding new software modules, componentware relies mainly on *Composition*, *Instantiation*, and *Adaptation* of existing components and infrastructure systems [Krue92]. In its current version, the V-Modell only covers the special case of a

central database component, which has to be realized and tested, based on the respective database schema.

Controlling: As mentioned in Section 2, existing components may trigger changes to the architecture and even the requirements of the customer. The V-Modell tries to support this via the so-called *Requirements Controlling* activity ("Forderungscontrolling") which is a bottom-up activity in the sense of Sections 2 and 3.1. Based on eventual preliminary architectural considerations, the requirements may possibly changed after the *Requirements Analysis* phase. With componentware requirements should be controlled regularly, especially after the system architecture or key components have been elaborated or changed. Furthermore, if for example new components, standards, or technologies arise on the market also a *Architecture Controlling* activity should be performed

Separation of Business-Oriented and Technical Issues: The V-Modell structures the design into three main activities: During *System Design* (SE2), the overall architecture is designed. *SW/HW Requirements Analysis* (SE3) extends the requirements for the involved hardware and software units. *High-Level SW Design* (SE4) is concerned with designing the internal architecture of software units, and *Low-Level SW Design* (SE5) specifies the interfaces and implementations of the involved software components in detail. With respect to componentware, this scheme has major flaws:

- The combined specification of hardware and software units in the system architecture makes it impossible to separate the business-oriented software design from the technical architecture, consisting of the involved hardware environment and the middleware infrastructure. This is inconsistent with one of the key features of modern component approaches [EJB98], namely, the possibility to employ business-oriented components within different technical system contexts [HMPS98].
- It is not possible to postpone the decision whether a certain component is to be implemented by means of software or by means of hardware.

We therefore propose to abolish the activities SE2 to SE4, replacing them with two activities for *Business-Oriented Design* and for *Technical Design*, which are clearly separated. The hardware/software mapping and the reconciliation and unification of these two architectures into a single, overall architecture may then be done in the subsequent *Low-Level Design* activity, as shown in [BRSV98a,BRSV98b].

4.2 Project Management

The sub-model *Project Management* (PM) consists of three kinds of activities: project initialization activities, planning activities, and execution activities. The *Project Initialization* activity (PM1) comprises basic steps that are performed once before the project starts, like the definition of project goals and the selection of tools. Furthermore, roles have to be assigned to persons and the V-Modell has to be "tailored" with respect to project-specific needs. During the tailoring activity, the project manager decides which activities of the full V-Modell may be discarded. The initialization phase results in an overall *Project Plan*, including a rough expenditure estimate and a time schedule.

Afterwards planning and execution activities start. Typical planning activities are performed periodically for each sub-phase of the project. They include the *Allocation of External Orders* (PM2), the *Management of Suppliers* (PM3), *Detailed Planning*

(PM4) of resources, milestones and expenditures, and C*ost/Value Analysis* (PM5). The resulting data serves as a basis for *Go-/No-Go Decisions* (PM6) for parts of the project. Concurrently, *Risk Management* (PM7), *Controlling* (PM8), and *Reporting* (PM9) activities are performed. Execution activities pertain to *Training and Briefing of Employees* (PM10 and PM13), *Preparation of Resources* (PM11), and *Allocation of Tasks* (PM12).

The final *Project Completion* activity (PM14) results in a final report about progression and project results. *Project Management* roles defined in the V-Modell include the *Project Director*, the *Project Manager*, the *Controller*, the *Legal Adviser,* and the *Project Adviser*.

Flexible Process Redefinition: In principle, the V-Modell already contains all activities necessary for componentware. However, changes on the component market require much more flexible approaches to planning and controlling. Following the pattern-based process model sketched in Section 3.1, the *Project Manager* and the *Project Director* may redefine the project during its runtime by selecting adequate patterns. Similar to the V-Modell's tailoring process, the conditions for the pattern application may be defined before the start of the project.

Dynamic Negotiation of Results: The current V-Modell already provides some facilities for bottom-up information flow and iterative elaboration of results, as described in Section 3.1. The additional flexibility brought by the pattern-based process model leads to a new understanding of the process management role: instead of a pure controlling activity, managing can now be seen as mediating between the customer, the developer, and the external component producers. Mediation relies on activities like *Requirements Controlling* in order to dynamically elicit and define the customer requirements, or *Architecture Controlling* in order to dynamically elaborate a suitable system architecture, taking into account the possible risks, benefits and costs of applying and integrating available components (cf. Section 4.1). The following issues mainly deal with the relation to the external component suppliers.

Market Analysis and Marketing: The initialization phase of sub-model PM has to be supplemented by a market analysis for the corresponding system or component. Component users need information about existing systems mainly in order to elicit the requirements of their customers. For component suppliers, detailed knowledge of the market is a vital precondition, as described in Section 2.2. Therefore, they usually start their development projects with a market study, scanning the market for similar components and analyzing requirements of potential customers. Depending on the results, the supplier estimates the potential income and the strategic value of the planned component. This assessment will in turn be used to further refine the features of the component in order to find an optimum cost/value ratio. In case of novel components and so-called enabling technologies, time-to-market is usually the most important factor – even if not all desirable features are present, the component may open up a new market. In case of more mature markets where components with similar features already exist, the component supplier also has to differentiate the component from those of competitors, either by providing additional features or by offering it for a lower price. Due to the importance of all activities concerned with the market, we propose to add the new role *Marketing Manager* to the V-Modell.

Supplier Management: While marketing is mainly concerned with the possible customers of a system or component, supplier management deals with producers of external components that are to be used within a project. While the V-Modell already includes an activity for *Supplier Management*, it is mainly intended for subcontractor

management. Dealing with independent component suppliers involves some additional issues. Besides an evaluation of the price and the features of the offered components, a supplier's reliability and continued support are important to component users, especially in case of critical, non-standard components. During *Risk Management*, corresponding technical and strategic risks must, therefore, be assessed carefully.

Contract Management: After a user has decided to buy a certain component, a contract has to be concluded. Conversely, suppliers must prepare a licensing model for their products. The license model directly influences the profitability of component reuse, for example, by regulating how often license fees have to be paid or whether new releases of components may be bought with certain discounts. Redistribution and copyright issues must also be clarified. It is also important whether the supplier commits himself to a certain roadmap for further development, beyond the usual responsibility for removing errors. Furthermore, the supplier's liability in case of damages caused by a component has to be clarified. Finally, precautions for the bankruptcy of the supplier should be made, for example, by stating in the contract that the source code of the supplied product must be handed over to the user in that case. The management of supplier contracts widens the job of the *Legal Advisor*.

Variant Management: Another important point for both users and suppliers is component variant management. Users should strive to minimize the number of variants, for example, by preferring to reuse existing components against buying modified or compatible ones (cf. Section 4.3). Users also have to check if any of their existing components may be suitable for reuse in a new development project by reviewing the project requirements. For COTS-suppliers, the situation is more complex. On the one hand, having many variants rises the number of potential customers. On the other hand, suppliers can usually only manage a limited range of variants with justifiable effort. To coordinate the development of the variants and especially to avoid duplicated work, we propose the new role *Variant Manager* for component suppliers.

4.3 Configuration Management

The primary goal of the sub-model *Configuration Management* (CM) is to ensure that each part of a product as well as the product itself is identifiable at every time. It includes four main activities: In *Configuration Management Planning* (CM1), guidelines for the configuration, change, build, and archive management have to be established. *Product and Configuration Management* (CM2) deals with the creation, deletion, and change of the product's entities, the product itself, and configurations of the entities. This includes especially the transfer of reusable entities to the central *Configuration Management Services* activity. *Change Management* (CM3) handles error reports, problem reports, and suggestions for improvements. At last, the *Configuration Management Services* (CM4) provide common services like product catalogues, data administration, access control administration, and interface management.

Integration of Component Descriptions: In addition to the traditional hierarchical structure of product models componentware demands for a clear separation of architectural requirements for a certain abstract component from the description of existing components. This allows component users to integrate different, encapsulated components along with their descriptions from external suppliers. Note that the ex-

change of standardized component descriptions relies on the standardization of the description format as well as of the involved description techniques (cf. Section 4.1).

Component Library Management: Apart from the developed products, *Configuration Management* must also archive components that have been assessed or used in a certain development project. It is also necessary to manage a list of external repositories in order to help developers in searching for reusable components. Note that the management of a component library for a single project doesn't pertain to building a global component repository used by multiple projects (cf. Section 5.3).

Variant Management: For the management of different variants of a single product, one can use concepts and tools for version management that are already available. Branches capture different variants and their history, while configurations may be used to model different internal structures of a component variant. However, due to the complexity introduced by variants, dealing with variants methodically remains an open research question.

4.4 Quality Assurance

The *Quality Assurance* sub-model consists of five main activities: Similar to the project initialization activity in the *Systems Engineering* sub-model, the *QA Initialization* (QA1) results in an overall *QA Plan* which describes the planned tests. *Test Preparation* (QA2) provides a more detailed plan, including the planned measures, test cases, and test procedures. These planning activities are complemented by performing the actual test and quality assurance measures – *Process Tests of Activities* (QA3) corresponds to reviewing and assessing the processed activities, while *Product Test* (QA4) deals with the development documents. *QA Reporting* (QA5) represents the information flow to *Project Management* activities which are informed in case of problems detected during *Quality Assurance*. The structure of activities QA1 to QA5 is similar in that each of them considers constructive as well as analytical measures, in order to both avoid and correct failures.

Component Assessment: Existing QA activities of the V-Modell not always suffice for ensuring the quality of components. As their source code may not be available, the assessment techniques for externally developed components might differ from those employed for assessing software developed in-house. In many situations, one has to resort to black-box tests based on a specification of the component's desired behavior. The *Supplier Management* main activity may also provide some additional information about the quality level of certain component producers. Sometimes, it will even be possible to access the source code, for example, with Open Source Software [Ray98], or when the supplier agrees to provide it. Finally, in contrast to tests of internally developed components, tests of external components should not be delayed until system integration. Rather, they have to be performed during the elaboration of the architecture and the interfaces in order to detect hidden flaws that could require costly workarounds during implementation or might even compromise the whole project (cf. Section 4.1).

5 Reuse Aspects

Reuse inherently expands the development context from a single project to multiple projects. Building a reusable component generally requires more effort which is only worthwhile if it can be sold several times. Therefore it is vital for component users to investigate related current and potential development projects for potential reuse opportunities. Rather than being forced into the organizational framework of a single development project, multi-project activities are generally carried out continuously over longer periods of time by a reuse organization. This organization is responsible for the coordination of all development efforts of the company, and it administers the company's component repository.

In the following, we describe some of the issues involved in the management of such a reuse organization. As sketched in Section 3.2, the principle structure, organization, and processes resemble very much their counterparts in the existing V-Modell. As an example, consider the roles of the new sub-model *Reuse Management* which are more or less isomorphic to the roles of the existing sub-model *Project Management*, only that their context comprehends not only a single project, but also multiple projects and that their tasks usually are more long-term oriented.

The following sections will, therefore, not repeat these foundations again, but only provide some additional recommendations and insights. As with the other activities of the V-Modell, nothing is said about the actual organizational settings – the reuse organization may be a dedicated department or a group of distributed component caretakers working on ordinary projects most of the time.

5.1 Systems Engineering

Reference Architecture and Company Standards: A reuse organization may not only develop reusable components, but also elaborate the company's overall IT architecture and company standards. Clear guidelines help to avoid compatibility problems and ease the work of application developers, as they can start with proven and accepted solutions. Advantages for component suppliers are similar appearance and a common corporate identity on the market.

Identification and Generalization of Reusable Components: Candidate components for (re-)use in multiple projects may be found by examining single in-house projects or by harmonizing product lines based on an overall domain analysis, but also by assessing the properties of components on the component market. Identified component candidates have to go through a reuse analysis which estimates the potential degree of reuse. Normally, in-house components from projects must first be adapted and generalized in order to be usable in a wider context. This may also lead to truly reusable components which may evolve to a stand-alone product for the component market themselves.

5.2 Reuse Management

Assistance for Systems Engineering: Normally, reference architectures and company standards are only proposed and elaborated by *Systems Engineering*. The final decision is usually met by *Reuse Management*. Similar considerations apply for the *Identification and Generalization of Reusable Components*.

Component Propagation: Multi project management has to propagate the usage of components within development projects by actively marketing them within the company. First and foremost, this means to announce already existing components, for example, by organizing systematic publicity campaigns and by informing projects about the company's global component repository.

Rewarding Reuse: Building a reusable component within a project leads to increased costs and efforts. Reuse management therefore has to create a system of incentives for this purpose, for example, by granting additional financial resources to project managers or developers.

Coordinating Development Activities: Sometimes a certain project may intend to use a component which currently is under development in another project. In cases like this, reuse management has to coordinate the development of both projects by harmonizing both project plans accordingly. Especially when time-to-market is a decisive criterion for the success of a software product, component reuse may be profitable even if costs increase compared to in-house development. In such a situation, a component supplier may speed up development within certain limits, if enough resources are available. Reuse management may support the negotiation of costs and development time in order to optimize the total benefits for the company.

Management of Human Resources: Similar to building a portfolio of reusable components, *Reuse Management* may strive to build a portfolio of human resources. This may be achieved by tracking existing abilities of the employees, but also by identifying abilities that are likely to be used in multiple future projects.

5.3 Configuration Management

All issues applying to component *Configuration Management* have already be mentioned in Section 4.3 and apply to *Reuse Configuration Management* analogously, only that the context is the company and not a single project.

5.4 Quality Assurance

Establishing Component Assessment and Quality Standards: In-house assessment standards for components should be developed and established, and assessment rules should be given to the supplier to achieve a common quality level within the company. The standards and rules have to be based on an experience database that has to be built up and refreshed within each new project.

Reusing Test Results: Reuse can reduce the effort for QA activities within similar projects or similar usage contexts. The reason is that certain QA activities must be carried out only once while developing a component. Whenever the component is reused, the focus may then be on integration testing. Even the remaining efforts may be reduced more and more based on the increasing experience with the component.

6 Conclusions

In this paper we have argued that the process models used today, and the V-Modell in particular, is not yet suited to the requirements of a componentware development process. In particular, this concerns the lack of an organizational structure and a process allowing the exchange of information and software either between different, not

necessarily concurrent projects within a company or via an open market for components. We have, thus, proposed some modifications and enhancements for the current version of the V-Modell, mainly by introducing new roles and new activities, by evolving a new sub-model concentrating on reuse, and by switching to a pattern-based process definition instead of a flow-based one. In our view, this constitutes the necessary and sufficient foundation for a more detailed elaboration of the V-Modell for component-oriented development.

References

[BRSV98a] K. Bergner, A. Rausch, M. Sihling, A. Vilbig, "A Componentware Development Methodology Based on Process Patterns", Pattern Languages of Programs 1998 (PLOP98), Monticello, Illinois, 1998.

[BRSV98b] K. Bergner, A. Rausch, M. Sihling, A. Vilbig, "A Componentware Methodology based on Process Patterns", Technical Report TUM-I9823, Institut für Informatik, Technische Universität München, 1998.

[BRSV98c] K. Bergner, A. Rausch, M. Sihling, A. Vilbig, "An Integrated View on Componentware - Concepts, Description Techniques, and Process Model", IASTED International Conference Software Engineering '98, Las Vegas, 1998.

[CaB95] E. Carmel, S. Becker, "A Process Model for Packaged Software Development", IEEE Transactions on Engineering Management, Vol. 42, No. 1, 1995.

[Dei98] B. Deifel, "Requirements Engineering for Complex COTS", REFSQ´98, Pisa, 1998.

[EJB98] Sun Microsystems, "Enterprise JavaBeans Specification", Version 1.0, Sun Microsystems, 901 San Antonio Road, Palo Alto, CA94303, 1998.

[HMPS98] W. Hordijk, S. Molterer, B. Paech, Ch. Salzmann, "Working with Business Objects: A Case Study", Business Object Workshop, OOPSLA'98, 1998.

[HRR98] F. Huber, A. Rausch, B. Rumpe, "Modeling Dynamic Component Interfaces", in TOOLS 26, Technology of Object-Oriented Languages and Systems, pp. 58-70, Madhu Singh, Bertrand Meyer, Joseph Gil, Richard Mitchell (eds.), IEEE Computer Society, 1998.

[Krue92] Ch. W. Krueger, "Software Reuse", ACM Computing Surveys, 24,2, 1992.

[Nin96] J.Q. Ning, "A Component-Based Development Model", COMPSAC'96, pp. 389-394, 1996.

[Ray98] E. Raymond, "Open Source: The Future is Here", WWW page http://www.opensource.org, 1998.

[Sam97] J. Sametinger, "Software Engineering with Reusable Components", Springer 1997.

[SGW94] B. Selic, G. Gullekson, P. Ward, "Real-Time Object-Oriented Modeling", John Wiley and Sons Ltd, Chichester, 1994.

[UML97] UML Group, Unified Modeling Language Version 1.1, Rational Software Corporation, Santa Clara, CA95051, USA, 1997.

[VMod] IABG, "Das V-Modell", WWW page http://www.v-modell.iabg.de/, 1998.