TUM

INSTITUT FÜR INFORMATIK

V-Model Conform Software Development with Catalysis

Ruth Breu Wolfgang Schwerin

TUM-I0001
Januar 00

TECHNISCHE UNIVERSITÄT MÜNCHEN

TUM-INFO-01-I0001-0/1.-FI

Druck: Institut für Informatik der
Technischen Universität München

TUM

INSTITUT FÜR INFORMATIK

V-Model Conform Software Development with Catalysis

Ruth Breu Wolfgang Schwerin

TUM-I0001
Januar 00

TECHNISCHE UNIVERSITÄT MÜNCHEN

TUM-INFO-01-I0001-0/1.-FI

Druck: Institut für Informatik der
Technischen Universität München

V-Model Conform Software Development with Catalysis

TECHNISCHE
UNIVERSITÄT
MÜNCHEN

Dr. Ruth Breu

Wolfgang Schwerin

TU München

Abstract

Both Catalysis and the V-Model are process models supporting system development based on the notations of the Unified Modeling Language (UML). The subject of this study is the conformance of both methods. More precisely we develop a variant of the original Catalysis process with V-Model conform documents as outputs. In this way we provide the basis of a design environment in which a development team working with Catalysis tools and familiar with Catalysis notations and process can produce V-Model conform documents as e.g. required in governmental projects.

The integration is achieved in four steps ranging from the adjustment of basic notations and the generic product structure of the V-Model to the interpretation of the V-Model documents in the context of Catalysis and to the final integration of the adjusted V-Model documents as outputs of the Catalysis process steps.

This study has been sponsored by Platinum CA.

Acknowledgement

We wish to thank Dr. Walter Huber from Platinum CA for his great support and for his fruitful remarks to earlier versions of this study.

Contents

1. Introduction

Both Catalysis and the V-Model are software engineering methods supporting software development based on the notations of the Unified Modeling Language (UML). Catalysis, developed by D. D´Souza and A. Wills, provides an approach to object-oriented and component based development. The Catalysis process model supports a way to design software which is iterative, evolutionary and architecture-centric. Other principles Catalysis emphasizes are traceability through explicit modeling of refinement relationships and reuse through abstraction, decoupling and pattern-oriented design.

The V-Model is the Development Standard for IT Systems of the Federal Republic of Germany (EStdIT). It is a generic process model for the development and maintenance of systems which are predominantly based on software.

The core of the V-Model is a definition of a set of activities, documents and methods. Activity types state what is to be done, document types describe which kind of information is to be produced within the activities. The methods provide a basic kit of notations (comprising the UML) to be used within the activities. Additionally, the so-called product structure provides concepts to decompose a system into hierarchical building blocks.

Being generic and independent of a specific technology in its kernel, the V-Model is complemented with scenarios combining activities to development lifecycles. A tailoring of the V-Model to object-oriented technology is covered by the scenario for object-oriented development. This scenario supports incremental development and assigns object-oriented notations such as the UML use case and class modeling to activities and documents.

The subject of this study is the conformance of Catalysis and the V-Model. It is explored how a software development team used to develop software within the Catalysis approach can develop systems in a way that conforms to the V-Model with least change of notations and development process.

As Figure 1 illustrates the result is a variant of the Catalysis process with V-Model documents as produced artifacts. To achieve this result, the study proceeds in several steps.

First, as a basis, the notational UML variant Catalysis uses is checked against the notations of the V-Model methods kit. Moreover, the concepts of the V-Model product structure are mapped to the Catalysis concepts modeling static system structure (i.e. type, class, package, component).

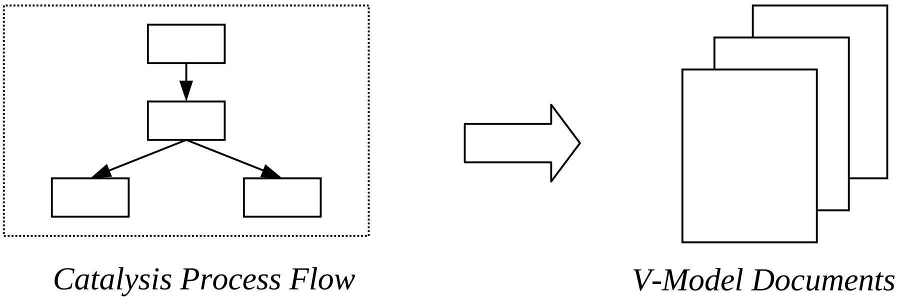


Figure 1 V-Model Conform Development with Catalysis

In a subsequent step, the V-Model documents (also called products) are adjusted to the Catalysis artifacts. Some of the artifacts correspond directly to each other, for others integrational work has to be done. In this way we obtain a set of V-Model documents interpreted in Catalysis terms and associated with Catalysis notations. Finally, the V-Model documents are integrated in the Catalysis process flow as inputs and outputs of the Catalysis design activities.

As a result we achieve a framework in which a development team working in a Catalysis environment produces V-Model conform design documents as e.g. required in governmental projects.

In this study we concentrate on the V-Model design activities and documents concerning the submodels of system development and quality assurance. The parts concerning configuration management and project management are outside the scope of this project.

Our study aims at an integration of V-Model conform output in design with Catalysis. A general comparison of the two methods has not been in the focus of the study. However, as a by-product, we summarize our observations concerning corresponding and differing activities within the software lifecycle of both methods.

The structure of this study is as follows. Section 2 gives an overview of the V-Model, introducing in the product structure (2.1) and basic methods (2.2), and sketches the scenario of object-oriented development (2.3) with its activities and document types.

Section 3 is the core of the study. Subsection 3.1 compares the underlying notations, subsection 3.2 deals with the interpretation of the V-Model product structure in Catalysis terms. In subsections 3.3 and 3.4 V-Model documents are adjusted to Catalysis artifacts and integrated in the Catalysis design process, respectively. Subsection 3.5 summarizes our observations concerning the lifecycle of both methods. Finally, section 4 gives a summary of our main results.

In this study we used a variety of sources of information. Concerning the "Component-based Development with Catalysis" process we referred to the HTML guide, version October 1999, of Computer Associates/PLATINUM Technology, and to [DW99]. Sources of information on the V-Model have been [IABG97], and [DHM98]. Besides that for information about the UML we referred to [UML97, UML99].

2. The V-Model

The V-Model is a generic process model for the development and maintenance of systems which are predominantly based on software.

In this model the development process is defined in terms of activities and documents. Activities state what is to be done. Documents describe which kind of information is to be produced by activities. Based on the so called product structure, which decomposes a system into hierarchical building blocks, such as segments and modules, the V-Model defines which document types are required for which kind of building block.

Adaptation of the V-Model to a specific project context and system class comprises the definition of a development strategy, tailoring and method allocation. By development strategies we understand the mapping of activities to development lifecycles. By tailoring we define accordingly to given guidelines which activities are to be performed and which documents are to be produced. Method allocation means the definition of “how” activities are to be enacted, that is which modeling concepts, strategies and notations are to be applied. These aspects are described in so called basic methods. In the V-Model’s method allocation manual, alternative and complementing methods for activities and documents are suggested.

So called scenarios are also part of the V-Model, providing possible development strategies. Besides that, some scenarios cover adequate method allocations for certain kinds of systems, too. The scenario of incremental development is assumed to be the common case concerning the development strategy.

An adaptation of the V-Model to object-oriented technology is covered in the scenario for object-oriented development. This scenario is based on incremental development and assigns object-oriented notations and methods, such as UML use case and class modeling, to activities and documents.

In the following sections we introduce the basic elements of the V-Model being relevant in the context of this study. In subsection 2.1 we describe the so called product structure which defines the architectural elements of the V-Model. In subsection 2.2, we outline the basic methods that are proposed for object-oriented development. Finally, we sketch the object-oriented development cycle in subsection 2.3, comprising a description of the lifecycle, the documents, and the activities.

2.1 Product Structure

The V-Model's generic product structure defines the architectural elements for the description of an IT system's hierarchical structure. A system's architecture is built of instances of the product structure's elements. With each of these elements certain kinds of documents are associated, which have to be produced during development.

In the following we characterize the elements of the V-Model's generic product structure. Figure 2 illustrates the elements of the generic product structure and their hierarchical relationships.

2.1.1 IT-System

The V-Model defines an IT system to be a uniform whole consisting of elements, such as hardware, software, installations, and persons, having the ability to fulfill certain tasks. IT systems are characterized by the fact that they are predominantly realized by the use of IT.

2.1.2 Segment

Segments allow the developer to structure a system into parts which comprise software, hardware, and other not IT related system elements, for example furniture. A typical example of a segment would be a certain hardware unit together with the software units running on it.

Segments can be built of other segments as well as of HW and SW units. The use of segments in a system architecture is optional.

2.1.3 HW/SW Unit

Compared with a segment, a unit represents either solely hardware or software, but not a combination of both. By means of units a SW or HW system, respectively, can be decomposed into manageable units. In particular, a SW unit can be built of SW components, SW modules, and databases.

2.1.4 SW Component

The element SW component of the generic product structure is an optional structuring element. A SW component may contain other SW components, SW modules, and databases. Typical examples of components are standard libraries of software development environments.

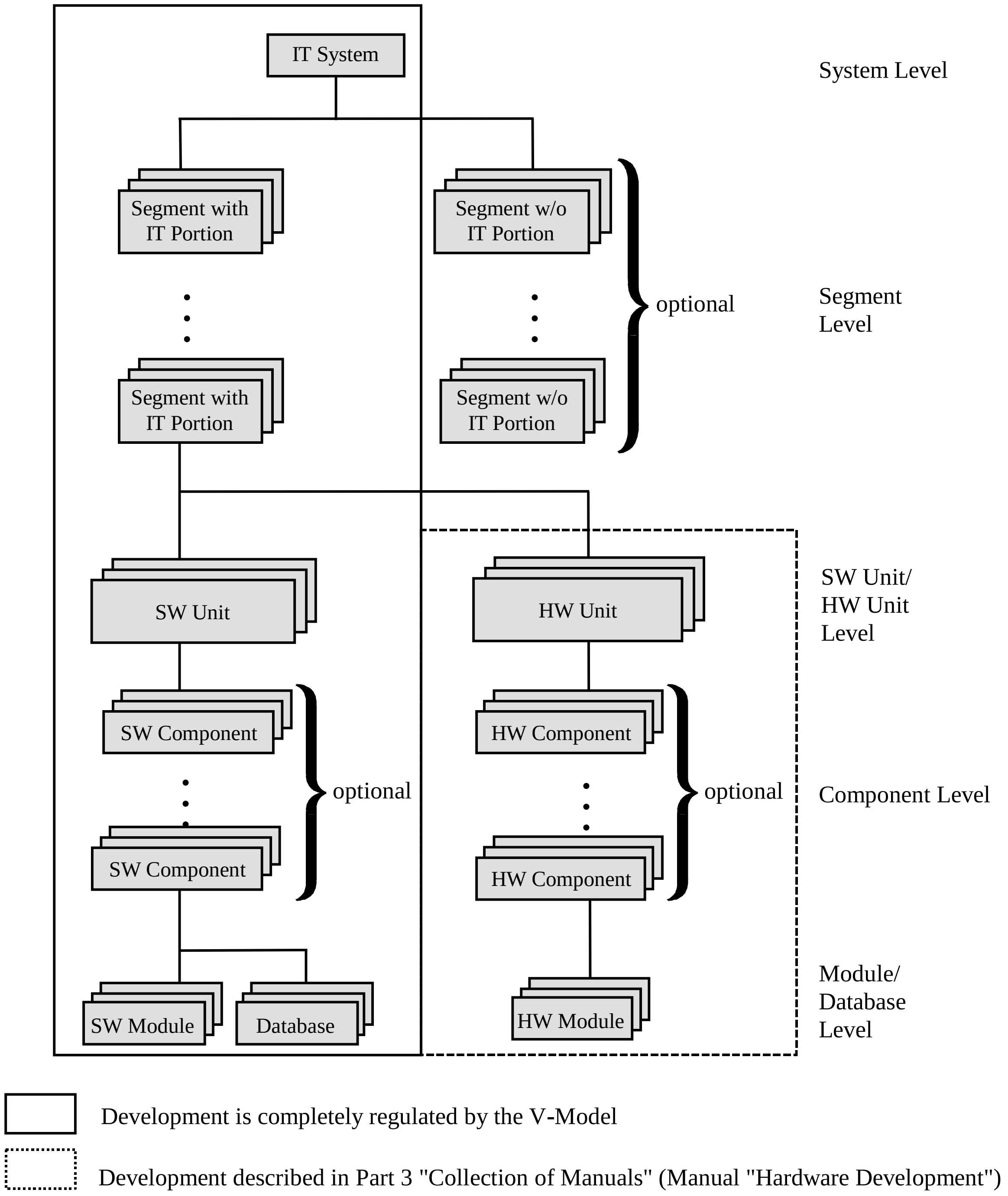


Figure 2: Generic Product Structure of the V-Model

2.1.5 SW Module

The V-Model defines SW modules to be the smallest programmable software components in SW Units. For implementation of modules, for example the programming language Ada offers the concepts package, task, and subprogram. In C++ a module is typically defined by a specification file with extension "h" together with an implementation file with extension "cpp", containing generally several classes, functions, and data.

2.1.6 Database

For a database the V-Model defines that it consists of a number of data items and is organized accordingly to the formal rules of the appropriate database schema and administrated by a database system.

2.2 Basic Methods

In this section, we summarize those basic methods of the V-Model which are assigned to activities and documents in the scenario for object-oriented development. In Table 1 we list all these methods.

Basic Methods of the V-Model for Object-Oriented Development
ACC (Analysis of covert channels), category of methods
COM (Class/Object Modeling)
CRC (Class Responsibility Collaboration)
DVER (Design Verification), category of methods
FMEA (Failure Mode Effect Analysis)
FS (Formal Specification)
IAM (Interaction Modeling)
MODIAG (Module Diagrams)
ODT (Object Design Technique)
PRODIAG (Process Diagrams)
RELM (Reliability Models)
SSM (Subsystem Modeling)
STMO (State Modeling in the Object-Oriented Field)
UCM (Use Case Modeling)

Table 1: Basic Methods of the V-Model for Object-Oriented Development

In the following, for each basic method

- we outline its modeling concepts,
- we describe at which points in the development process the method is suggested to be applied, and

- we show relationships with other basic methods.

We derive this information from the description of basic methods within the V-Model manual “Method Allocation”. Yet, we do not use V-Model specific terms for the description of how and when to apply the named methods in order to provide an characterization which facilitates the comparison of V-Model's basic methods with the techniques of Catalysis in subsection 3.1.

2.2.1 ACC and RELM

Method category ACC covers specific methods aiming at the discovery of information flows contrary to security requirements. Examples of such methods are Shared Resource Methodology and Information Flow Analysis. ACC is applied in technical system design and in the realization of a system. The method is only of significance with regard to IT security.

The idea of method category Reliability Models (RELM) is to cover the definition and proof of reliability requirements for the functions of a system as a whole and of coarse grained architectural elements, such as SW/HW units in the V-Model. Generally, different reliability characteristics can be considered, such as mean time between failure or failure rate. The V-Model suggests to apply this method when reliability requirements are high.

2.2.2 COM

The method Class/Object Modeling (COM) refers to modeling of classes as well as objects and their relationships as it is defined in the UML [UML97]. Important concepts of this method are class and object specifications, covering pre/post condition style specifications of operations, association and aggregation between classes and objects, as well as inheritance relationships between classes. The definition of constraints is one example of further concepts covered by the method.

Method COM can be used for modeling static logical aspects of an existing system, of user requirements, and of the system to be developed. For modeling the static physical structure of a system, the V-Model suggests to use module diagrams (MODIAG) and process diagrams (PRODIAG). For dynamic aspects usage of state modeling (STMO) and interaction modeling (IAM) is proposed.

In the case of complex systems method COM can be applied together with subsystem modeling (SSM), where a system is decomposed hierarchically into subsystems, and COM is applied on these subsystems. Thereby the system can be structured into segments, SW/HW units, SW Components, and Modules.

By using COM on all architectural levels, ranging from the user-level system structure (where the whole system is seen as a black box and its functionality as well as its interfaces are described from a user point of view) to the description of SW Components, Modules and object-oriented databases, the description of (static aspects of) interfaces of all these architectural elements is covered.

2.2.3 CRC

The V-Model describes the Class Collaboration Method accordingly to [Wir93]. This method is supposed to be applied in combination with method COM. It aims at the definition of actions and responsibilities and their assignment to defined classes. Moreover the way how classes collaborate is specified in order to meet the assigned responsibilities.

CRC is an informal technique which is used in CRC sessions as a means to explore the business domain and the tasks of the system to be developed. CRC cards have been proved to be an appropriate basis for communication between developers and clients or experts of the business domain.

2.2.4 FS and DVER

The V-Model defines a formal specification to be one written in a formal notation, which is based on well-founded mathematical concepts. Formal specifications can be used in order to supplement informal specifications for critical elements of system interfaces and architectural elements of all kinds. Examples for important classes of formal specifications are pre- and post-condition specifications of operations, and specifications based on traces or temporal logic.

Formal specifications are the prerequisite for the application of design verification (DVER). Formal analysis of covert channels (ACC) is also based on FS.

DVER is a method category covering formal verification methods. The objective of DVER is to mathematically prove refinement relationships between formal specifications. A DVER method must provide proof rules or transformation rules for the related formal specification method.

This method can be applied on specifications of the system as a whole as well as specifications of finer grained architectural elements such as SW Units. In particular, consistency of specifications of architectural elements on different architectural levels can be checked, for example between the system architecture and its SW Units, and between a SW Unit and its constituents, such as modules and databases.

2.2.5 FMEA

The method FMEA (Failure Mode Effect Analysis) [MIL80] is used for the identification of potential error types in order to define their effects on the examined objects, such as the system, segments or software units. Prevention of errors and of weak points in design is the method’s aim. The definition of corrective measures and test cases is also part of this method.

The V-Model suggests to apply this method for the definition of criticality levels, but only in case of high reliability requirements for a system and architectural element, respectively.

2.2.6 IAM

The method Interaction Modeling (IAM) is based on interaction modeling as it is defined in the UML [UML97]. The method's objective is to describe interactions and interaction sequences between objects. In interaction descriptions we define and relate occurrences of events and exchange of messages. As notation, both sequence and collaboration diagrams of the UML can be used.

Similarly and in addition to modeling static aspects with method COM, IAM can be used for aspects of interaction concerning an existing system, user requirements, and the system to be developed on all architectural levels. In particular, message oriented requirements for any kind of architectural element can be formulated. Besides that, IAM can be applied in order to model the realization of use cases. In the context of state based modeling of objects (STMO) interaction specifications can be used as starting points.

2.2.7 MODIAG

The method Module Diagrams (MODIAG) is defined in [Boo94] and integrated in a generalized form in the UML by means of component diagrams. In the context of this method, the term "module" must be understood as an element of the physical software structure which is expressed in the vocabulary of a certain programming language. A major objective of this method is the documentation of compilation dependencies.

The developer can use method MODIAG in combination with methods COM, SSM and PRODIAG. Whereas COM, SSM and PRODIAG are applied in order to structure a system and architectural elements from a conceptual point of view, method MODIAG is applied in order to allocate conceptual units, such as classes, to modules in the aforementioned physical sense. Accordingly to the method's objective compilation dependencies are an important part of the information covered in interface descriptions of modules.

2.2.8 PRODIAG

Similar to MODIAG this method was defined in [Boo94] and respected in the UML in terms of deployment diagrams. PRODIAG is applied in order to specify a view on systems and architectural elements concerning the configuration of processors and external devices as well as the allocation of processes running on processors and devices.

The developer can specify such execution platform views for an existing system, for the system to be developed, and for architectural elements on all levels.

In combination with method COM active objects are modeled as processes in process diagrams.

2.2.9 SSM

Subsystem Modeling (SSM) is a method defined in [SM92], where a variety of diagram types is also given. In the UML, the concept “package” supports subsystem modeling.

The method Subsystem Modeling allows the introduction of system structures based on classes and objects defined by application of method COM. These structures can be built with respect to different criteria, for example with respect to domain aspects, defining so-called user subsystems. Other kinds of subsystems are service subsystems, architecture subsystems and implementation subsystems. In object-oriented development with the V-Model, SSM allows the developer to structure a system’s architecture into subsystems of the types given in the V-Model’s product structure, such as segments, HW/SW units, SW components and SW modules.

2.2.10 STMO

Method State Modeling in the Object-Oriented Field (STMO) is based on state modeling as it is defined in the UML [UML97], originating from [Har97]. The method's objective is to describe the lifecycle of an object in terms of states and transitions between these states. Transitions between states can be triggered by events, guarded by conditions, and lead to the generation of events and state changes. States can be decomposed hierarchically.

Similarly and in addition to modeling static aspects with method COM and interactions with IAM, the developer can use STMO for state-transition based behavior specifications of an existing system, user requirements, and the system to be developed on all architectural levels. Interaction specifications and use cases can be the starting point for behavioral specifications with STMO.

2.2.11 UCM

Method Use Case Modeling was introduced in [Jac92]. The method's objective is the elicitation and representation of functional requirements of a system from the user point of view. Characteristics of use cases are that they are initiated by some kind of user, i.e. external actor, in order to achieve a certain task. In a use case, the interactions between external actors and the system which are relevant for the achievement of the related task are described. The focus is put on the external actor point of view and not on how the system works internally.

The method is intended for the description of use cases in text form. Use case diagrams as they can be found in the UML allow the developer to specify relationships between use cases and actors, as well as relationships among use cases.

By means of use cases not only functional user requirements for the system to be developed can be specified, but they can also be used for documentation and analysis of usage and operation of actual systems. In combination with method COM according class and object definitions can be developed from the requirements formulated as use cases. Together with methods IAM and STMO use cases are realized by according interaction and state-transition based behavior specifications of objects.

2.3 Object-oriented Development Cycle

In this section, we outline the object-oriented development cycle as it is given in the V-Model's scenario for object-oriented development. First of all, we characterize the underlying incremental life cycle. Then we sketch the major documents to be incrementally produced in the life cycle. Finally, we describe the most significant activities which are to be enacted in an iterative way.

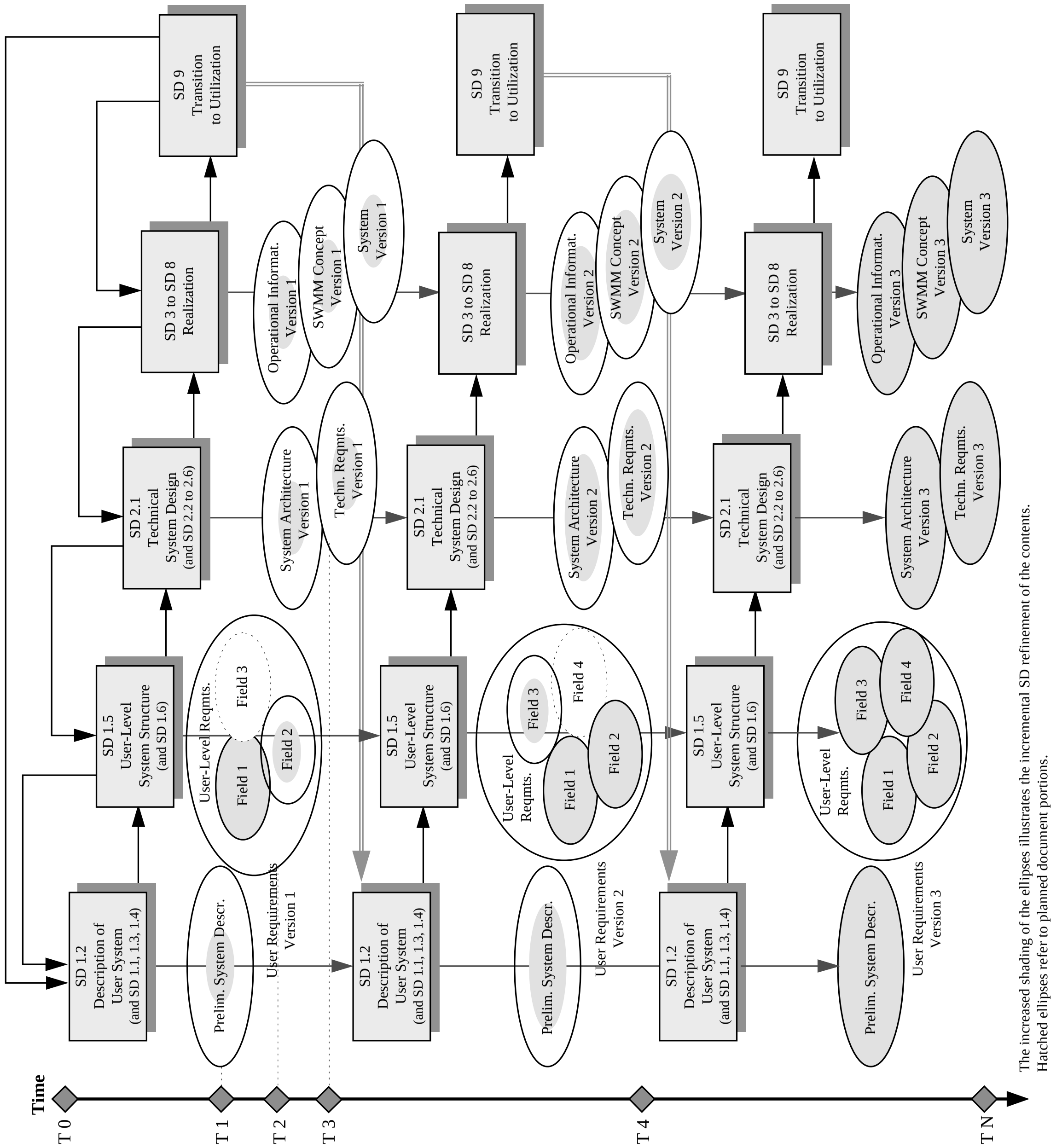
2.3.1 Life Cycle

The scenario for object-oriented development is based on the incremental development scenario. The basic idea of an incremental development life cycle is that a system is planned as a whole but realized in parts. Development activities are enacted repeatedly in order to realize the system incrementally. Thereby incremental realization must not lead to change of given requirements but only to their refinement and eventually extension. Initial requirements are binding. Consequently the general scope of a system must be known from beginning. Figure 3 illustrates the object-oriented lifecycle, covering the incremental completion of documents and the iterative enactment of activities.

At the starting point of the object-oriented scenario binding user requirements are given. Based on these user requirements a preliminary system description is produced. This description covers the system's overall functionality. The functionality is then structured into user level fields. This structure is besides others used as a basis for the incremental development.

For those identified fields that are to be realized in the first increment, user level requirements are defined completely. For other fields the user level requirements may not be specified at all, or only incompletely. Basically, this first version of user level requirements must cover all relevant user level fields. The overall functionality that is to be covered in the final version must be clearly defined.

In a next step the technical realization of the first increment's fields is defined. This means that an according system architecture and technical requirements are specified.



The increased shading of the ellipses illustrates the incremental SD refinement of the contents.
Hatched ellipses refer to planned document portions.

Figure 3: Object-Oriented Development Cycle (cf. [IABG97])

One emphasized aspect of the object-oriented development scenario is the consideration and integration of reusable off-the-shelf parts, such as existing class libraries. Considering off-the-shelf products may influence user level structuring as well as the definition of the system architecture.

After realization, the first increment is made available to be used. Generally, reactions and feedback from use of a preceding increment can be taken into account when refining the preliminary system description for the next increment.

In subsequent increments, further user level fields are realized, based on further refinement and stepwise completion of user and technical requirements, as well as of the system architecture.

2.3.2 Products

In this section we sketch shortly the most significant document types, synonymously also called products, defined in the V-Model for system development (submodel SD). An in depth treatment of all document types considered in this study can be found in subsection 3.3. There, we discuss the contents of the different document types chapter by chapter, accordingly to the chapter structure that is defined in the V-Model for each document type.

Generally, V-Model documents cover information about certain kinds of elements of the product structure. Figure 4 illustrates these relationships between document types and product structure elements.

User Requirements

Generally, a User Requirements document describes aspects of the system from the user point of view and independently of a possible system architecture. In particular, this kind of document covers a system’s organizational and technical environment, and a preliminary system description which is supposed to give an overview of the envisioned system. Detailed descriptions of functional user requirements structured in fields, quality requirements, marginal conditions, IT security, and thread and risk analysis are further important contents of this document.

Technical Requirements

In contrast to user requirements, technical requirements are generally not independent of the system architecture. Rather, technical requirements refer to specific elements of the system architecture. Technical requirements that cannot be allocated to a certain architectural element are documented as general requirements.

In the Technical Requirements document technical requirements for the overall system, the segments, and the SW/HW units are defined. For each element, its overall functionality, its technical interface requirements, quality requirements, and requirements concerning its development and SWMM (software maintenance and modification) environment are defined.

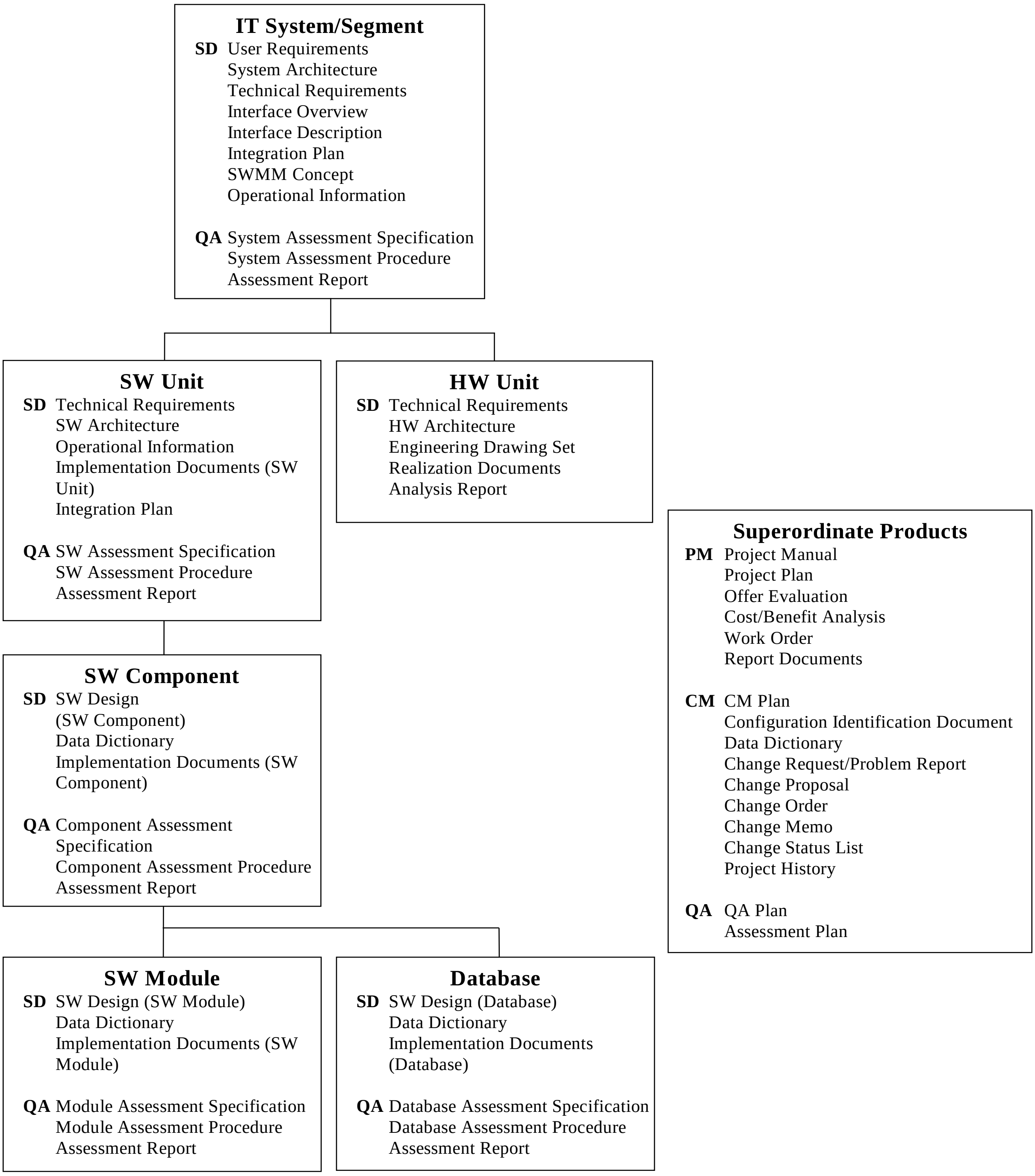


Figure 4: Product Structure Elements and Associated Document Types (Products)

System Architecture

A System Architecture document is associated with a system or a segment. In such a document the developer delineates a system's/segment's decomposition into segments and units, internal and external interfaces, and how the system/segment's constituents cooperate. Moreover, user requirements are allocated to constituents.

SW Architecture

SW Architecture documents refer to SW units. Similarly to a System Architecture document for a system/segment, in the Software Architecture the developer describes the unit's decomposition into SW components, SW modules, and databases, how these constituents collaborate, and interfaces to other units or segments as well as interfaces between the unit's constituents. Additionally, a description of how the technical requirements of a SW unit are allocated to its components, modules, and databases is contained.

SW Design

The document type for the description of SW components, SW modules, and databases is the Software Design. In the Software Design the developer describes the realization of a SW component, SW module, or database, respectively, down to programming specification level. A programming specification for a component or module covers information such as control and data flow, algorithms, and realization of data. In the case of databases according data schemata have to be specified.

Implementation Documents

Implementation documents refer to SW units, components, modules, or databases, respectively. They cover information about code as well as command procedures for compilation, linking, code generation, and others.

Integration Plan and SWMM Concept

An Integration Plan contains regulations concerning the technical aspects for the assembly of the system, of segments or units.

A Concept for Software Maintenance and Modifications (SWMM Concept) document covers information about a system's transition to utilization and its maintenance.

2.3.3 Activities

In this section we sketch the main software development activities that are to be enacted iteratively in the development cycle.

Description of User System

For this kind of activity an external specification from the customer together with a preceding and generally incomplete version of user requirements are the input. The aim of this activity is to define the Preliminary System Description and IT objectives of the given user requirements. In case of existing definitions these parts are refined.

User-Level System Structure

This activity refers to an external specification and user requirements as input and is based on the preliminary system description of the user requirements. One main concern of this activity is the description of functionality structured in separate fields. Criticality levels are assigned to these fields. Besides that, the system's organizational embedding has to be specified. This comprises for example the specification of business processes. Closely related to that, the system's external interfaces, that is user-system interactions are defined.

The fields that are to be realized in the according increment are specified completely, whereas other fields may have to be refined in further enactments of this activity.

Technical System Design

In this activity from a set of proposals one solution proposal for the system architecture is selected. Refinement of the selected proposal leads to the definition of the system architecture. This structural information is supplemented by the description of interactions between the architecture's elements.

Moreover, technical requirements of the architectural elements are derived from user requirements as far as possible. The technical requirements concern functionality, quality, and the development and maintenance environment (SWMM).

Realization

Realization comprises SW/HW Requirements Analysis, Preliminary and Detailed SW Design, SW Implementation, SW Integration, and System Integration.

The input for SW/HW Requirements Analysis are user requirements, the system architecture, and the previously derived technical requirements. In this activity the technical requirements are refined with respect to SW and HW units.

In the Preliminary and Detailed SW Design decomposition of SW units into SW components, SW modules, and databases is defined. The construction of each module and component must be described down to programming specification level. Databases are specified by according data schemata.

SW Implementation comprises the coding of SW modules and databases.

In a SW Integration activity SW modules and databases are integrated into SW components and units. In activity System Integration SW units, HW units, and possibly non-IT portions are integrated into the system. Integration of increments may imply changes to the existing increment.

Transition to Utilization

Transition to utilization comprises installation of a completed increment or system at the intended application site.

3. V-Model Conform Development with Catalysis

The conformance study of Catalysis and the V-Model proceeds in four subsequent steps.

- In the first step the subset of basic methods of the V-Model is identified that conform to the notations Catalysis uses (subsection 3.1).
- In the second step the V-Model's product structure is mapped to the notions of static system structuring in Catalysis (subsection 3.2).
- In the third step the V-Model documents are interpreted in the context of the Catalysis artifacts (subsection 3.3).
- In the fourth step the V-Model documents of step 3 are integrated in the Catalysis design process as input and output documents (subsection 3.4).

In this way we obtain an integrated Catalysis process with produced V-Model conform artifacts. We will refer to this tailored process as the *joint Catalysis/V-Model process* or, shortly, as the *joint process.*

Finally, in section 3.5 we summarize our observations concerning the software lifecycle of both methods.

3.1 Basic Methods

In this section we compare the basic methods that are suggested for object-oriented development in the V-Model with the techniques Catalysis provides.

3.1.1 COM — Type Specifications and Class-Layer of Design

In the V-Model's basic method COM classes, objects and various relationships are introduced as a means for the description of static aspects of user requirements as well as of architectural elements of all levels, ranging from the system as a whole to SW modules. Thereby, the authors of the V-Model refer to definitions of the UML.

Besides the specification of static aspects, behavioral aspects are also covered by the basic method, for example by means of operation specifications in a pre/post condition style.

This fits well with the techniques centered around the concepts "object", "type" and "action" in Catalysis. Similar to classes and objects in COM, we use objects and types in Catalysis for the specification of elements of the problem/business domain as well as for constituents on all scales of the system to be developed.

In contrast to COM and UML, Catalysis notations emphasize the difference between type and class, that is the difference between problem statement and solution. In Catalysis types represent specifications of object state and object behavior, being free of implementation. The interpretation of specifications is defined in a way which allows us to refine a given specification by specifying details that are left open. In contrast to types, classes are units of implementation providing a certain and definite solution to problems expressed in terms of related type specifications.

In UML the separation of class and type is not as strict as in the according Catalysis techniques. Whereas a UML class stereotyped with Type is similar to a type in Catalysis, and a UML class stereotyped with Implementation is similar to a class in Catalysis, in UML we need not necessarily assign one of these stereotypes to a class. In this case, a UML class may cover both, specification as well as implementation information. For example, the definition of a class' method, that is algorithmic steps in form of code, represents implementation information, whereas a pre/post condition description of the operation's effects is specification information.

In UML as well as in the notations of Catalysis, a realization relationship connects specifications with implementations.

The notions of attributes and associations of classes and types, respectively, are similar in UML and Catalysis notations. Operations in UML correspond to localized actions in Catalysis.

For structuring a system with respect to various criteria, for example in order to express a system's architecture in terms of the V-Model's product structure, we are advised to apply COM in combination with SSM (subsystem modeling). As we will see below, in Catalysis, similarly to UML, we can use packages in the sense of subsystems as proposed in the basic method SSM (subsystem modeling). Besides packages, type refinement of Catalysis is well suited to structure the system to be developed accordingly to the V-Model's product structure, that is in segments, SW components and SW modules. This results from the fact that refinement is more expressive than aggregation or composition as defined in UML, because refinement covers a mapping of elements of the abstract type to their realization by the elements making up the refinement. By this mapping between abstract type and its refinement we can express for example the allocation of requirements, as it is required by the V-Model products SW Architecture and SW Design.

3.1.2 CRC

In the basic method CRC actions and collaborations and their assignment to classes are the main concepts. We can find these concepts in the Catalysis techniques, where they have a precise meaning. In Catalysis we find different kinds of actions, such as abstract action, directed action, localized action, which differ with respect to the details they cover about an action, as well as their relationships with collaborations and with operation specifications of types. This means, that the technique of CRC card session can be used to develop Catalysis models.

3.1.3 IAM — Actions and Collaborations

Catalysis comprises interaction modeling as it is defined in IAM and the UML. Within the Catalysis techniques however, this form of interaction modeling which defines collaboration on the fine grained level of message exchanges, is only the most detailed form of describing how objects work together.

Less detailed forms are collaborations with joint actions, where we merely state an action's effect in a pre/post condition style and abstracting from concrete interactions. We can add more detail when we additionally specify the sequential composition of actions. These actions can still be undirected. Adding further information concerning an action's direction and optionally whether an action is based on synchronous or asynchronous communication, we achieve the level of detail represented in method IAM.

The different levels of detail the notations of Catalysis provide for the specification of collaborations allow us to choose the adequate level for documents required by the V-Model. For example, for the description of aspects of collaboration in the Preliminary System Description of document User Requirements, we can choose only to define joint actions, whereas for the Dynamic Sequence Model in document SW Architecture we may choose to specify sequences of directed synchronous/asynchronous actions.

3.1.4 MODIAG Programming Language Packages

Because of modules in the context of MODIAG are to be interpreted as elements of the physical software structure which is expressed in the vocabulary of a certain programming language, we map the modules of MODIAG to programming language packages of Catalysis.

When we use package diagrams in MODIAG as proposed in Catalysis instead of using module diagrams of [Boo94] or UML's component diagrams as proposed in the V-Model we do not loose information. We can map from component diagrams to (programming language) package diagrams by substitution of components with packages. Dependency relationships between components can either be mapped to dependencies between the according packages, or they can be refined to import/extension or import/usage relationships.

3.1.5 PRODIAG

In Catalysis we define object locality and link implementation similar to PRODIAG. The notational basis are in both cases deployment diagrams of the UML.

3.1.6 SSM — Horizontal and Vertical Slices, Domain Packages

The basic method SSM corresponds to the application of packages for modeling vertical and horizontal slices, and different domains in Catalysis.

3.1.7 STMO — State Modeling

In Catalysis, state-modeling is applied similarly to STMO. Due to the fact, that static invariants are an elementary part of type specifications in Catalysis, state-modeling and type specification, that is COM and STMO, are well integrated.

3.1.8 UCM — Use Case as Joint Action

In Catalysis we can use joint actions in the sense of use cases. This implies for example, that these joint actions have to involve an external actor and that the actions' granularity must correspond to a meaningful and complete task from the actor's point of view. Modeling use cases as joint actions gives us the possibility to describe the effect of a use case by means of an action specification, that is in a pre/post condition style. We can refine this abstract specification to an interaction based view. The transition from an abstract joint action to a collaboration and to localized actions of types together with the according type specifications represents the integration of methods UCM, COM, IAM, and STMO required by the V-Model.

3.1.9 DVER and FS

In general method DVER is based on the application of method FS.

The pre/post condition specification style as proposed in Catalysis for the specification of actions is well suited for the application of formal specification languages. However, OCL which is proposed in Catalysis is not yet a formal specification language in the sense of the V-Model, because it has no semantics defined on the basis of a mathematical model. Besides that, until now, there exists no proof and transformation rules for OCL which prevents it from building a basis for method DVER.

3.1.10 ACC, FMEA, and RELM

Methods ACC, FMEA, and methods classified by the method category RELM are not contained in Catalysis but can be integrated accordingly to the V-Model.

3.2 Product Structure

In this section we relate elements of the V-Model's generic product structure with structural elements, such as component, class and package of Catalysis.

3.2.1 System/Segment — Component

In the V-Model we decompose an IT system into manageable units, that is segments and HW/SW units. The way in which we decompose a system is determined by quality goals we have to meet, such as maintainability and portability. Building a system by composition of components as they are defined in Catalysis aims at the same quality goals. Consequently, we map V-Model segments and HW/SW units to Catalysis components.

In the V-Model systems/segments are characterized as architectural elements which are directly related with user requirements and which may cover software as well as hardware and other units. Therefore we can map those top-level Catalysis components to segments which comprise not only software components, that is components implemented in software, but also hardware components and components modeling other elements of an IT system.

3.2.2 HW/SW Unit — Component

HW and SW units of the V-Model represent units which are solely implemented in hardware or software, respectively. Therefore, SW units map well with (software) components of Catalysis.

For HW units, information covered in documents Technical Requirements and HW Architecture can also be represented by means of Catalysis type specifications. Thereby, we can integrate HW and SW units on a type specification level.

3.2.3 SW Component — Class Layer Package

Similar to classes in Catalysis, in the V-Model SW components, SW modules, and databases are architectural elements for which we do not only provide declarative specifications, but also operational definitions and implementations. In particular, a V-Model SW component represents a grouping mechanism for SW components, SW modules, and databases. In Catalysis we can use packages for grouping of classes.

3.2.4 SW Module — Class

Because of SW modules are defined to be the smallest programmable element in a SW unit, we map SW modules to Catalysis classes.

3.2.5 Database

We can map a V-Model database to a Catalysis package covering the realization of classes as logical database schemata.

3.3 V-Model Documents in the Context of Catalysis

In this section we interpret the V-Model document types shortly sketched in section 2 in the context of the artifacts and notations of Catalysis. In the first part of the section the document types are considered in an isolated setting (sections 3.3.1 to 3.3.12). In each subsection, first, the V-Model document is briefly characterized, followed by its interpretation in the context of Catalysis. A summary in section 3.3.13 completes the first part. In the second part the document types are put in the context of incremental development (section 3.3.14).

Table 2 lists all V-Model document types concerning system development (called *SD-documents*) and quality assurance (called *QA-documents*) together with a reference to the respective subsection. Documents adhering to configuration management and project management are not considered in this study.

The adjustment of V-Model document types to Catalysis will not only interpret the documents in the Catalysis context, but will also identify mismatches. These mismatches concern on the one side information in Catalysis artifacts that is not captured in V-Model documents and, vice versa, information in V-Model documents that is not captured in Catalysis artifacts. Concerning the first case, we will integrate, as far as possible, additional Catalysis information in adequate V-Model documents. Concerning the latter case we will extend the Catalysis process outputs by the required V-Model documents (in section 3.4).

In the sequel, mandatory documents within the V-Model will be marked as such (where a mandatory document with subchapters means that there is at least one mandatory subchapter). In order to avoid misunderstandings we will use the term *document* in the context of the V-Model and the term *artifact* in the context of Catalysis. Moreover, *chapters* refer to the structure of V-Model documents, *sections* refer to the structure of this study.

SD Documents	***section***
User Requirements	3.3.1
System Architecture	3.3.2
Technical Requirements	3.3.3
Interface Overview	3.3.4
Interface Description	3.3.4

Integration Plan	3.3.5
SWMM Concept	3.3.5
SW Architecture	3.3.6
SW Design	3.3.7
Data Dictionary	3.3.8
Implementation Documents	3.3.9
Information for the User Manual	3.3.10
Information for the Diagnosis Manual	3.3.10
Information for the Operator Manual	3.3.10
Other Application Information	3.3.10

Table 2 – List of V-Model SD-Documents

QA Documents	***section***
QA Plan	3.3.11
Assessment Plan	3.3.12
Assessment Specification	3.3.12
Assessment Procedure	3.3.12
Assessment Report	3.3.12

Table 3- List of V-Model QA-Documents

3.3.1 User Requirements (mandatory)

The User Requirements describe the user-level aspects of the system independent of any implementation. Conceptually, the User Requirements comprise all Catalysis artifacts that are defined at the domain/business level or describe the external behavior of the target system. In particular, the User Requirements comprise both the Catalysis Business Models and the System Type Specification.

In more detail, the User Requirements consist of six chapters (*UReq_2* to *UReq_7*[1]). In the sequel these sections are shortly characterized and associated with Catalysis artifacts.

[1] In each V-Model document chapter number 1 (here: *UReq_1*) is reserved for general information (name, version, date, person in charge, etc.)

UReq_2 Actual Status and Current Analysis

In this chapter the organizational and technical environment is described on the basis of the actual status. The description typically includes the following items.

- recording of user environment (structure of organization, official regulations, laws and the like)
- recording of user-level and technical factors that cannot be influenced
- recording of available IT equipment
- detection of weak points

In a conform Catalysis/V-Model environment the role of the Actual Status and Current Analysis is played by the Catalysis As-Is Business Model. In particular, business collaborations and scenarios identify the actors in their domain and their interaction. These interactions model the problem domain in the current status. Additional text may treat the items listed above.

UReq_3 IT Security Objective

In this chapter it might be specified which requirements with regard to availability, integrity or confidentiality exist for certain functions or information.

In the current version of Catalysis, security aspects at this high level of abstraction are not covered. Hence, chapter *UReq_3* will complement the Catalysis process in safety critical applications.

UReq_4 Threat and Risk Analysis

The threats relevant for the system have to be identified, and, by taking into consideration the probability factors of damages to be expected, the connected risks to be evaluated. The results of the threat and risk analysis are the basis for the formulation of requirements for IT security.

As above, chapter *UReq_4* should complement the Catalysis process in safety critical applications.

UReq_5 IT Security

In this chapter IT Security requirements are specified in more detail. In particular, the total effects of measures in the system environment (such as access control or radiation protection) and measures in the system itself (such as password protection or protocol logging) have to be specified. Moreover, the threats from the treat and risk analysis must be allocated to the requirements.

Again as in the two chapters above, chapter *UReq_4* is not covered in Catalysis artifacts and will complement the Catalysis process in safety critical applications.

UReq_6 **User-Level Requirements (mandatory)**

This chapter consists of seven subchapters *UReq_6.1* to *UReq_6.7*.

UReq_6.1 **Preliminary System Design**

The preliminary system description must fulfil criteria in the form of an overall horizon, comprising e.g.

- the user-level tasks to be supported by the system
- the application concept in order to derive the technical and organizational application environment
- quantitative estimations (e.g. about network organization, computing power)

This chapter is intended to give an overview – more precise information e.g. about user-level tasks is contained in subsequent sections. Concerning the Catalysis context, this V-Model chapter corresponds to the Project Scope within the Project Initiation stage. The Project Scope comprises e.g. the business objectives, a data model diagram, user tasks and an outline solution.

UReq_6.2 **Organizational Embedding**

This chapter contains a description of the organizational sphere of the system: the structure and process organization of the user, user classes, responsibilities and competence during operation, and the additional equipment of the user.

In Catalysis this kind of information is covered within the business models (as business collaborations). If not dominant within this phase of development (e.g. due to a complex organizational structure) we recommend to skip this subchapter within the joint Catalysis/V-Model process.

UReq_6.3 **Utilization**

This chapter contains a description of the requirements for operation and use of the system, e.g. for mobile or stationary operation, operation times, communication between user and system and the extent of the services to be automatically made available by the system.

In Catalysis this kind of information is covered by the Non-Functional Requirements. The Utilization chapter will be integrated as an optional document in the joint process.

UReq_6.4 **Criticality of the System**

The criticality of the system is specified. In Catalysis, criticality capturing is part of the Business Case within the Project Initiation Stage.

UReq_6.5 External Interfaces (mandatory)

Requirements for the external interfaces to neighboring systems and, in particular, for the man-machine interface have to be documented.

Conceptually, this chapter corresponds to the System UI Specification within the System Specification step of Catalysis. The System UI Specification details the mechanisms by which the user can invoke system operations. The UI portion of the increment is seen as a black box that receives user events and transforms them into business events (and vice versa). Techniques used in the system specification are e.g. a system context model with actions at the UI dialog abstraction level, screen mock-ups and storyboards.

UReq_6.6 Description of the Functionality (mandatory)

This chapter describes the user-level structure of the system functionality from a user's point of view, modeling the following aspects:

- the tasks to be fulfilled by the system
- business processes for the determination of the required system functionality
- user-level structuring into *fields* as a basis for structuring the system

Conceptually, chapter *UReq_6.6* comprises both the To-Be Business Model (including an essential model and the corresponding refinement model) and the System Type Specification[2].

More precisely, the Description of the Functionality document in a joint Catalysis/V-Model context contains the following parts.

- the To-Be Business Model, eventually accompanied by an essential model and a refinement model - these models specify the business domain of the application.
- the System Type Specification modeling the system as a type, with a type model (attributes and associations) and a set of operations specified against that type model
- the Type Specification can be split across subject areas using a package structure – this corresponds to the definition of fields in the V-Model as described above.

UReq_6.7 Quality Requirements

This chapter specifies quality requirements with regard to

- reliability

[2] In this chapter we consider the documents in an isolated setting. When we put the documents in the context of iterative development, document type *UReq_6.6* will also correspond to type specification of increments. The same holds for other document types treated in this first part.

- user friendliness
- efficiency
- maintainability
- portability
- reusability, etc.

In the Catalysis context, this subchapter corresponds to the artifacts of the Non-Functional Type Specification and the Problem/Requirement List, both output of the task *Define Additional Requirements* within the System Specification.

UReq_6.7 Marginal Conditions

In this chapter technical and organizational marginal conditions are specified, such as

- technical requirements for interfaces
- specification of off-the-shelf products
- adhering to technical standards
- required communication, cooperation and coordination among instances in the user’s process organization
- execution rhythm and repetitiveness of business processes
- geographical distribution of users in the application area
- specifications from organization development for staff size, job plans, job descriptions etc.

This kind of information extends the more formal presentation of information of chapter 6.6. In Catalysis, the marginal conditions do not correspond to a specific document. In the subsequent section we will integrate chapter 7 in the joint process as an optional document.

3.3.2 System Architecture (mandatory)

The System Architecture describes the system structure based on the notion of segments and (SW) units of the generic product structure. The document System Architecture includes possible solution proposals, results from feasibility studies, the IT security concept, the IT security model, and the allocation between User Requirements and the elements of the System Architecture.

The System Architecture document may not only contain one specific solution, but may comprise several proposals. The level of abstraction the System Architecture is based on is technical, implementation-oriented. Therefore, in the context of Catalysis, the System Architecture corresponds both to

- the Technical Architecture Options (exploring possible solutions) and to
- the Technical Architecture being part of the System Architecture Definition.

Since the Application Architecture and the Technical Architecture play an equally important role in Catalysis, we recommend to integrate the Application Architecture in the V-Model's System Architecture document.

In more detail the System Architecture describes a specific architectural solution in five subchapters (these subchapters are elaborated in more or less detail for each architecture proposal).

SysArc_2 System Structure

This chapter describes the architectural kernel for the system implementation in two subchapters *SysArc_2.1* and *SysArc_2.2*. Chapter *SysArc_2.1* contains the representation of the technical system structure. Here the structuring of the system into segments and units is defined and off-the-shelf products are identified (subchapter *SysArc_2.1.1*). Moreover, interfaces of segments and units are defined (subchapter *SysArc_2.1.2*) and the User Requirements (cf. section 3.3.1) are allocated to the elements of the System Architecture (subchapter *SysArc_2.1.3*). Additionally, in chapter *2.2* the cooperation of the technical elements is explained.

Conceptually, chapter *SysArc_2* corresponds in Catalysis to the Technical Architecture Definition (or a Technical Architecture Option, respectively). The Technical Architecture describes the component design of the software system and its realization. A Technical Architecture in Catalysis comprises a class and package structure (for static dependencies) and collaborations (for interactions between components) corresponding to the subchapters *SysArc_2.1.1* and *SysArc_2.2*, respectively. We recommend to extend chapter *2* by an additional subchapter which contains the Application Architecture in terms of interacting components.

In order to obtain full conformance with the V-Model's generic product structure, the Catalysis component structure has to be mapped to the segment/unit structure of the V-Model. Doing this, the Requirement Allocation (subchapter *SysArc_2.1.3*) directly corresponds to the Refinement Model in Catalysis relating the Application Architecture with the System Specification (i.e. User Requirements in the terms of the V-Model).

In our opinion the definition of interfaces (subchapter *SysArc_2.1.2*) becomes superfluous in the chosen approach since interfaces are already part of the component structure of the Application Architecture. Thus, in the joint process, the interface specification is replaced by the Application Architecture.

SysArc_3 Realization

In this chapter specific solution proposals are elaborated (*SysArc_3.1*) and evaluated in Feasibility Studies (*SysArc_3.2*). Conceptually, the concept of a feasibility study corresponds to (architectural) prototyping in Catalysis. Thus, chapter *SysArc_3* in the Catalysis context comprises both an Architecture Prototype Specification (corresponding to *SysArc_3.1*) and the Prototype itself together with the Prototype Results (corresponding to *SysArc_3.2*).

SysArc_4 IT Security Concept and *SysArc_5* IT Security Model

These two chapters describe the IT security strategy and the IT security model, respectively. Within the security model it is necessary to prove that the objectives of the IT security concept are met by means of the modeled IT security functions and the units external interfaces.

Since the modeling of security aspects is not fully integrated in Catalysis, subchapters *SysArc_4* and *SysArc_5* do not have a direct corresponding in Catalysis. Security aspects at a similar level of abstraction are roughly captured in the System Architecture Requirements in Catalysis.

3.3.3 Technical Requirements (mandatory)

This document accompanies the System Architecture specifying technical requirements on the level of the whole system and, more detailed, on the level of segments and units. Each level of abstraction (systems, segment, unit) forms an own chapter in the Technical Requirements document with a uniform structure.

Each such chapter consists of the following parts.

- Identification of the Element (i.e. its name)
- Overall Function of the Element

  The functionality of the element is represented as a whole.
- Technical Requirements of Interfaces

  This chapter describes the technical realization of user interfaces and other interfaces. This might include the definition of masks, windows and dialog sequences.
- Quality Requirements

  In this chapter quality requirements concerning criticality, technical requirements for the IT security, reliability, portability, user friendliness and others are specified.

In the context of Catalysis the quality requirements clearly correspond to the System Architecture Requirements produced within the System Architecture Investigation. In the Catalysis framework the overall function of elements is intimately connected with the structural definition of the application architecture and technical architecture through type models, invariants and specification of operations. Similarly, the technical requirements of user interfaces are integrated in the user interface specification.

Summarizing we recommend to reduce the Technical Requirements document in the joint Catalysis/V-Model process to the chapter of Quality Requirements. The other chapters are deferred to the System Architecture (overall function of the segments/units) and the External Interface Chapter *UReq_6.5* of the User Requirements (technical requirements of user interfaces).

3.3.4 Interface Overview and Interface Description

These two documents specify interfaces of elements from the segment level to the module and database level serving as a kind of interface catalog of the whole design. The content of both documents is produced in several phases of the design (e.g. interfaces of segments are defined in an earlier activity than interfaces of modules).

While the Interface Overview merely identifies system-external and system-internal interfaces, the Interface Description contains for each element (segment, unit, component, module or database) the following parts.

- Use of the interface – purpose of the interface and participating elements
- Syntax of the interface – technical structure of the interface, e.g. parameter list or communication protocol
- Semantics of the interface – the information and operations exchanged via the interface

The interface overview and the interface specification integrate information produced in several stages of the design. Since in the Catalysis process there is no equivalent, the information required in these documents is spread over several types of Catalysis artifacts: the Application and Technical Architecture for the segment and unit level, Timebox Design for the component and module level and Database Design for the databases.

In case that the Interface Overview and the Interface Description are mandatory, the required information has to be generated out of the above Catalysis artifacts. In the other case we recommend to skip these two chapters.

3.3.5 Integration Plan and SWMM Concept

The Integration Plan contains regulations governing the technical aspects for the assembly of the system, the segment or unit. The strategy to be applied, the measures to be realized, possible restrictions and dependencies as well as the required resources and tools must be defined.

In the context of Catalysis the Integration Plan roughly corresponds to the Development Environment artifact produced within the System Architecture Definition.

The Concept for Software Maintenance and Modifications (SWMM Concept) contains regulations about how the system can be transferred into the state where it can be used, which organizational measures must be taken for the later SWMM, and which resources will be required.

The Catalysis process does not explicitly treat maintenance and modification issues that exceed evolutionary development of the intended software system. The SWMM Concept document may be skipped in the joint process.

3.3.6 SW Architecture (mandatory)

The Software Architecture is defined for each SW unit. This document contains proposals for possible SW architectures and for the selected decomposition of the SW unit into components, modules and databases. In more detail the Software Architecture consists of the following parts.

SwArc_2 Solution Proposals

This chapter includes a rough description and the evaluation of possible SW unit architectures.

SwArc_3 Modularization/Database Design

The modularization describes the static decomposition of a SW unit into SW components and modules as well as the real-time specific connections. Furthermore, this chapter includes a preliminary design of the databases of the SW unit. The subsections of this chapter are as follows.

SwArc_3.1 Overview of SW Components, SW Modules, Processes and Databases

SwArc_3.2 Individual Descriptions

SwArc_3.3 Dynamic Sequence Model – description of the dynamic interrelations of the elements

SwArc_3.4 Criticality of the Elements

SwArc_3.5 Other Design Decisions

SwArc_4 Interfaces

The interfaces of the elements are integrated in the Interface Overview (cf. section 3.3.4).

SwArc_5 Requirements Allocation

This chapter sets up the relationships to the Technical Requirements (cf. section 3.3.3).

In the context of Catalysis the Software Architecture corresponds to the Timebox Definition artifact. In each timebox one or several Catalysis components are designed, implemented and tested. The Timebox Definition describes the structure of the components that are realized in the timebox.

In more detail chapter *SwArc_3.1* and *SwArc_3.2* correspond to type and class models defining the structure and behavior of each component. Chapter *SwArc_3.3* should contain collaboration diagrams defining how classes will interact. Chapter *SwArc_5* corresponds to a refinement model relating the Timebox Definition with the Increment Specification. Chapters *SwArc_2*, *SwArc_3.4* and *SwArc_3.5* do not have a direct correspondence in Catalysis and may be elaborated or skipped in the joint process as required by the application. The specification of alternative proposals for Timebox Definitions is not supported in Catalysis.

Though we are aware that timeboxes in Catalysis are of an organizational nature, we relate timebox-specific artifacts in Catalysis with unit-specific documents in the V-Model. This kind of relationship is justified by the fact that Catalysis advises developers to implement in each timebox one or several system components.

What concerns the database design, Catalysis pursues the database design for the whole system (increment) rather than a database design for a specific component. Thus, database design in general is not covered in the SW Architecture, but in a separate document (SW Design, see section 3.3.7) in the joint process.

3.3.7 SW Design

A Software Design document is associated with each component, module or database and describes the detailed design of the element. In case of a component or module the SW Design consists of the following parts.

- Characterization of the Task
- Environment
- Interfaces
- Realization of the Component/Module (algorithms, pre-/postconditions, etc.)
- Local Data
- Exceptional Behavior
- Characteristic Quantities (like resource and time requirements and precision of processing)

In the context of Catalysis SW Design corresponds to Timebox Design. The document contains collaboration diagrams defining how classes will interact. Moreover, the class model (as defined in the SW Architecture document) is validated and extended by adding classes, operations and parameters and modifying classes as required.

For databases the SW Design consists of three parts.

- DBMS Name
- Schema Definition
- Characteristic Quantities

As already mentioned in section 3.3.6, the SW Design document corresponds to the Database Design artifact in Catalysis. The database design is developed for the whole system (increment).

3.3.8 Data Dictionary

The Data Dictionary contains all information about the data used in SW units. It can be generated manually or with the help of a tool. The Data Dictionary consists of the data description and the data realization.

In the object-oriented approach of Catalysis the role of the Data Dictionary to collect information about data types spread over several units becomes superfluous. A useful substitution of the Data Dictionary in the joint process could be a dictionary of classes. The concept of a dictionary Catalysis supports is a dictionary containing model elements together with an informal description forming a glossary.

3.3.9 Implementation Documents

The Implementation Documents contain references to the code of the SW module, the database, the SW component or SW unit, as well as command procedures for code generation, installation, etc, and also listings and protocols of compile, link and load runs.

In Catalysis the equivalent artifacts to the Implementation Documents are the builds developed in the timeboxes and the database, together with the whole installation package.

3.3.10 Information for the User Manual, Diagnosis Manual, Operator Manual and Other Application Information

The Information for the User Manual includes all information required by the user of the system to operate the system properly and to be able to react appropriately when problems arise. In Catalysis this document corresponds to the User Guide developed in the step of Education Development.

The Information for the Diagnosis Manual supplies the information required for the SWMM and diagnosis activities with respect to the system. The function of the system and the diagnosis environment are described, thereupon each individual diagnosis feature and error message is referred to. Catalysis has no corresponding kind of artifact.

The Information for the Operator Manual describes, for a functional unit (system, segment, SW unit), the measures that are required for starting the operation, the execution and control of the operation, and the interruption or termination of the operation. The structure of the functional unit and the security regulations have to be described, too.

Catalysis aims at a consistent set of artifacts. Therefore, the specification documents at various levels of abstraction (such as the System Type Specification, the Increment Specifications and the Timebox Designs) can take the role of the Operator Manual Document. In this case, the Operator Manual can be skipped in the joint Catalysis/V-Model process.

The document “Other Application Information” combines all those information for the user with regard to transfer, implementation and operation of the system. This might refer to topics like training, configuration options, data transition, infrastructure, etc. In the context of Catalysis this kind of document may include various artifacts produced in the Iterative Deployment stage (e.g. training material) or may be skipped.

3.3.11 QA Plan (mandatory)

The QA Plan contains the general definitions with regard to prevention and proof activities (methods, tools, sequences and operation) valid for the entire project. In more detail, the QA Plan consists of the following parts.

- Desired Quality and Risks in the Project
- QA Measures according to Criticality and IT Security – reference to existing standards and criteria
- Quality Assurance during Development – listing of products and activities (of the other V-Model's submodels) to be assessed
- Special Control Measures – e.g. entry control of off-the-shelf products, controlling subcontractors, tasks with regard to documentation, assessment and acceptance control

In the context of Catalysis the QA Plan corresponds to the Project Control Procedures being part of the Project Initiation Report.

3.3.12 Assessment Plan, Assessment Specification (mandatory), Assessment Procedure and Assessment Report (mandatory)

The Assessment Plan defines the objects to be assessed as well as the tasks and responsibilities of the assessments, the planning schedule, and the resources required for the realization. The Assessment Plan determines the state in which products and activities have to be assessed, also when, by whom, and by what means they have to be assessed.

In Catalysis the Assessment Plan roughly corresponds to the Test Strategy defined within the task *Establish Development Standards and Procedures* belonging to the System Architecture Definition.

The Assessment Specification contains the description of assessment requirements and goals, assessment methods, assessment criteria derived from the requirements and the test cases. With the help of the Assessment Specification it must be possible to decide if the assessment has been successful or not. An Assessment Specification is generated for each object to be assessed.

The Assessment Procedure is an instruction and contains exact statements for each individual assessment. The Assessment Procedure defines the individual assessment steps, the expected assessment results as well as the regulations about assessment preparation and postprocessing.

Conceptually, in Catalysis Assessment Specification and Assessment Procedure together correspond to the Increment Test Plan (also the V-Model description states that Assessment Specification and Assessment Procedure can be combined in one document). In a refined version the Assessment Specification also comprises the Catalysis Test Scripts and,. together with the Assessment Procedure, also the Timebox Test Plans.

The Assessment Report contains the assessor-originated recordings about the course of the assessment, especially the comparison between expected and actual results. There is an Assessment Report for each assessment object and each assessment.

In Catalysis the Assessment Report corresponds to the Test Result artifact within the task *Test the Build* in a timebox. In particular, Catalysis associates test results with components.

3.3.13 Summary

The following two tables summarize the rough correspondences between V-Model documents and Catalysis artifacts (Table 4) in our interpretation. In the joint process the Catalysis artifacts will be replaced by the respective V-Model Documents. Table 5 lists the main V-Model documents that do not have a direct correspondence in the Catalysis process. These document types will complement the outputs of the joint process. For each V-Model document in the tables the corresponding subsection in this study is given.

V-Model Document	*section*	*corresponding Catalysis artifact*
User Requirements	3.3.1	Business Models Project Scope System Type Specification System UI Specification Non-Functional Type Specification Problem/Requirement List
System Architecture	3.3.2	Technical Architecture (Option) Application Architecture (Option) Architecture Prototype Specification Prototype Prototyping Results
Technical Requirements	3.3.3	Architecture Requirements
Integration Plan	3.3.5	Development Environment
SW Architecture	3.3.6	Timebox Definition
SW Design	3.3.7	Timebox Design Database Design
Implementation Documents	3.3.9	Build Installation Package
Information for the User Manual	3.3.10	User Guide
QA Plan	3.3.11	Project Control Procedures
Assessment Plan	3.3.12	Test Strategy
Assessment Specification	3.3.12	Increment Test Plan Timebox Test Plan Test Script
Assessment Procedure	3.3.12	Increment Test Plan Timebox Test Plan
Assessment Report	3.3.12	Test Result

Table 4 Correspondences between V-Model Documents and Catalysis Artifacts

V-Model Document	*section*
User Requirements	3.3.1
Chapter 3 – IT Security Objective Chapter 4 – Threat and Risk Analysis Chapter 5 – IT Security	
System Architecture	3.3.2
Chapter 4 – IT Security Concept Chapter 5 – IT Security Model	
Interface Overview	3.3.4
Interface Description	3.3.4
SWMM Concept	3.3.5
Data Dictionary	3.3.8
Information for the Diagnosis Manual	3.3.10
Information for the Operator Manual	3.3.10

Table 5 V-Model Documents without Correspondence in Catalysis

3.3.14 Documents within Incremental Development

Up to now we have interpreted the V-Model documents in an isolated setting. Put in the context of incremental development, several instances of the document types exist – each one associated with a specific increment. Figure 5 illustrates the kernel documents in the object-oriented development scenario which is based on the scenario of incremental development.

Comparing the structure of documents produced in the object-oriented development scenario with the structure of output artifacts in Catalysis, a major difference can be observed. While in the V-Model each document instance is associated with a specific increment, Catalysis has a global part that specifies the overall system structure and behavior.

The Catalysis global part essentially consists of the Business Models, the System Specification and the System Architecture Definition. The global artifacts are kept consistent within the whole design. A new increment specification is derived both from the global requirements (e.g. specifying the additional functionality to be provided in the increment) and from the structure and behavior of the just finished increment.

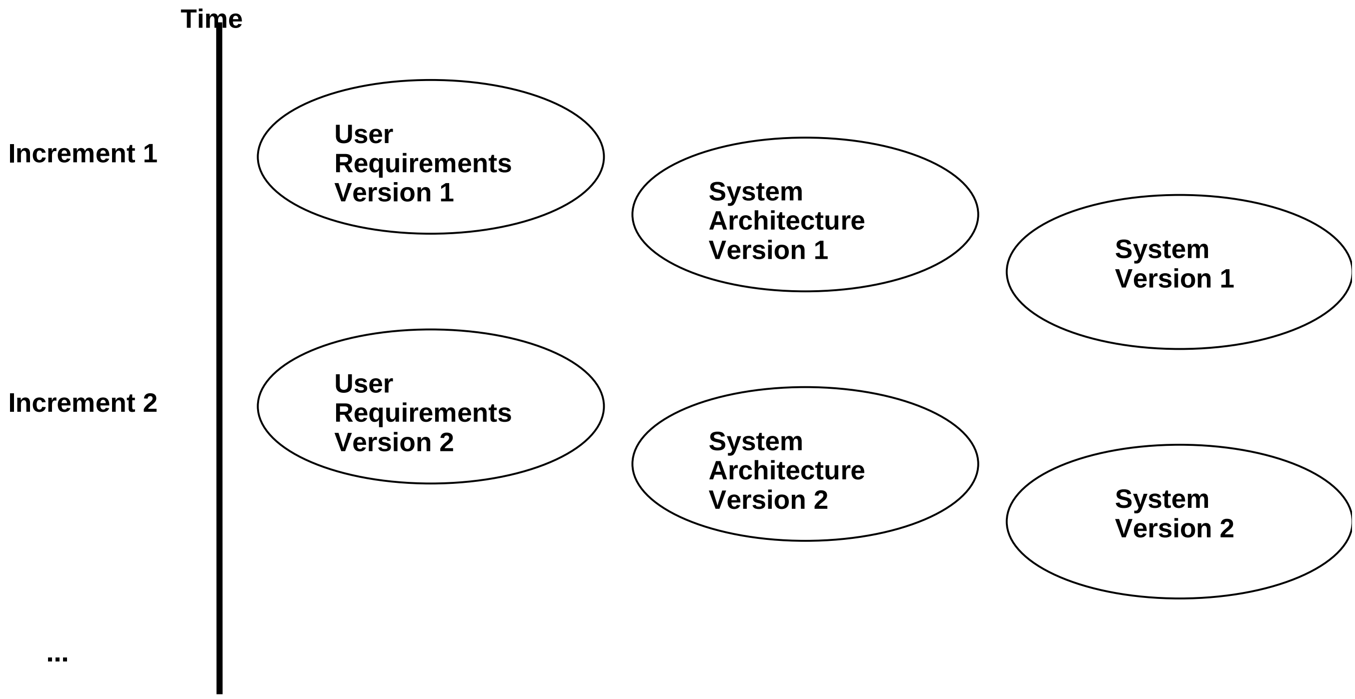

Time
Increment 1
Increment 2
...
User Requirements Version 1
System Architecture Version 1
System Version 1
User Requirements Version 2
System Architecture Version 2
System Version 2

Figure 5 Produced V-Model Documents in Object-Oriented Development

As stated in the V-Model's description, the first increment in the V-Model takes both the role of the global system specification and the specification of the increment. However, a clear separation of these parts or a directive to keep the initial increment specification consistent with later increment specifications is not provided.

As a solution we recommend to extend the scenario of object-oriented development in a way that is conform to the Catalysis process. Since the usage of scenarios is not mandatory in the V-Model, we do not leave the framework of the V-Model through this extension.

Figure 6 illustrates the rough structure of V-Model documents in the extended scenario.

In the sequel we will use the terms *User Requirements* and *System Architecture* for the document instances associated with the global system and the terms *Increment User Requirements, Increment System Architecture* and the like for the document instances associated with a specific increment.

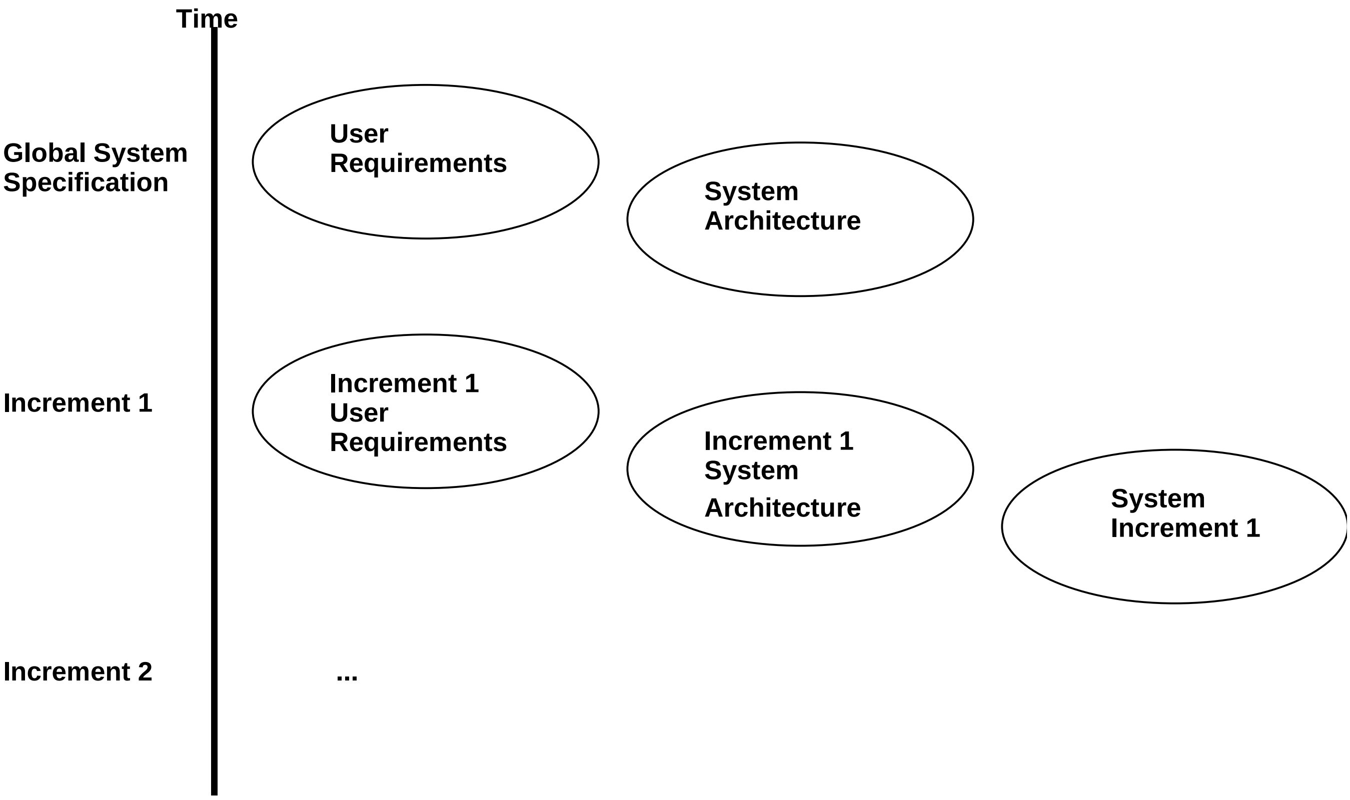

Time
Global System Specification
User Requirements
System Architecture
Increment 1
Increment 1 User Requirements
Increment 1 System Architecture
System Increment 1
Increment 2
...

Figure 6 Produced V-Model Documents in the Joint Process

3.4 The Joint Catalysis/V-Model Process

In this section we are now able to integrate the adjusted V-Model documents of section 3.1 in the Catalysis process flow. In particular, we will define the output documents of the joint process as a variant of the output documents in the original version of Catalysis (the input documents are not defined explicitly, but can be deferred in a straightforward way from the output document definition).

On the one hand, the output documents of the joint process are those V-Model documents that correspond to the output Catalysis artifacts of the original process. On the other hand, we will integrate additional V-Model documents in appropriate steps which have no direct correspondence in Catalysis.

Catalysis artifacts which have no correspondence in the considered part of the V-Model remain in the joint process. Since we do not consider the V-Model parts of project and configuration management in this study, the specification of output documents contains Catalysis artifacts which have a correspondence in these parts of the V-Model.

In the following subsections we define the output documents of the joint process in tabular form. V-Model documents are shown off in italics. Section 3.4.1 treats the Project Initiation stage, section 3.4.2 treats Solution Definition. Section 3.4.3 is concerned with Evolutionary

Delivery and section 3.4.4 with Iterative Deployment. For each V-Model document type we refer to the corresponding Catalysis artifacts in the original process and to the respective section in this study.

3.4.1 Project Initiation

Outputs

Product	*Original Product*	*mandatory*	*section*
Project Initiation		Yes	
UReq_6.1 Preliminary System Design	Project Scope	Yes	3.3.1
UReq_6.4 Criticality of the System	Business Case	Yes	3.3.1
QPl QA Plan	Project Control Procedures	Yes	3.3.11

The Project Initiation Report consists of the following parts.

- Project Schedule and Budget
- Project Organization
- Product/Object Management Procedures
- Business Case

The remaining parts of the original Project Initiation Report (Project Scope, Project Control Procedures) are replaced by V-Model documents.

3.4.2 Solution Definition

Step 01 – Stage Management

This part remains unchanged.

Outputs

Product	*mandatory*	*section*
Commitment Calendar	No	
Exception Plan	No	
Issue Log	No	
Progress Report	Yes	

Step 02 – Business Modeling

Outputs

Product	*Original Product*	*mandatory*	*section*
UReq_2 Actual Status and Current Analysis	As-Is Business Model	No	3.3.1
UReq_3 IT Security Perspective		No	3.3.1
UReq_6.2 Organizational Embedding		No	3.3.1
UReq_6.3 Utilization		No	3.3.1
UReq_6.6 Description of the Functionality	To-Be Business Model	Yes	3.3.1

Documents *UReq_2* and *UReq_6.6* together correspond to the original Catalysis Business Model. The other V-Model document chapters are extensions.

Step 03 – System Specification

Outputs

Product	*Original Product*	*mandatory*	*section*
UReq_4 Threat and Risk Analysis		No	3.3.1
UReq_5 IT Security		No	3.3.1
UReq_6.5 External Interfaces	System UI Specification	Yes	3.3.1
UReq_6.6 Description of the Functionality	System Type Specification	Yes	3.3.1
UReq_6.7 Quality Requirements	Non-Functional Type Spec. Problem/Requirement List	Yes	3.3.1
UReq_7 Marginal Conditions		No	3.3.1

The V-Model documents *UReq_6.5*, *UReq_6.6* and *UReq_6.7* have the same scope like the original Catalysis System Specification produced in this step. The other documents are extensions.

Step 04 – System Architecture Investigation

Outputs

Product	***Original Product***	***mandatory***	***section***
SysArc_2 System Structure	Application Architecture Option Technical Architecture Option	Yes	3.3.2
SysArc_4 IT Security Concept		No	3.3.2
SysArc_5 IT Security Model		No	3.3.2
TReq Technical Requirements	System Arch. Requirements	Yes	3.3.3

Document *SysArc_2* corresponds to the Application Architecture Option and the Technical Architecture Option in the original Catalysis process. Document *TReq* corresponds to the System Architecture Requirements. Documents *SysArc_4* and *SysArc_5* are extensions.

Step 05 – Prototyping

Outputs

Product	***Original Product***	***mandatory***	***section***
Issue Log		No	
UReq_6.7 Quality Requirements	Problem/Requirement List	No	3.3.1
SysArc_3 Realization	Prototype Specification, Prototype Prototyping Results	Yes	3.3.2

The V-Model document *Realization* replaces the original Catalysis artifact related with a prototype, in particular the Prototype Specification and the Prototyping Results. Prototyping may lead to an extension of the document *Quality Requirements.*

Step 06 – System Architecture Definition

Outputs

Product	***Original Product***	***mandatory***	***section***
SysArc_2 System Structure	Application Architecture Option Technical Architecture Option	Yes	3.3.2
IfOv Interface Overview		No	3.3.4
IfDe Interface Description		No	3.3.4
IntPl Integration Plan	Dev. Standards and Procedures	Yes	3.3.5
SWMMC SWMM Concept		No	3.3.5
AsPl Assessment Plan	Test Strategy	Yes	3.3.12

The *System Structure* corresponds to the Application Architecture Option and Technical Architecture Option, the *Integration Plan* corresponds to the Development Standards and Procedures artifact. The *Assessment Plan* corresponds to the Test Strategy. *Interface Overview*, *Interface Description* and *SWMM Concept* are extensions.

Step 07 – Increment Planning

This part remains unchanged.

Outputs

Product	***mandatory***	***section***
Incremental Delivery Plan	Yes	

Step 08 – Stage End Assessment

Outputs

Product	***mandatory***	***section***
All documents of the Solution Definition in a consolidated version	Yes	
Stage Report	Yes	

3.4.3 Evolutionary Delivery

Step 01 – Stage Management

This part remains unchanged. In this step the same artifacts as in Step 01 of the Solution Definition are produced.

Step 02 – Initial Increment Definition

Outputs

Product	*Original Product*	*mandatory*	*section*
UReq_6.6 Incr. Description of the Functionality	Increment Spec.	Yes	3.3.1
UReq_6.5 Increment External Interfaces	Increment UI Spec.	Yes	3.3.1
SysArc_2 Increment System Structure	Incr. Architecture Def.	Yes	3.3.2
IfOv Interface Overview		No	3.3.4
IfDe Interface Description		No	3.3.4
AsSpec Assessment Specification	Increment Test Plan	Yes	3.3.12
AsProc Assessment Procedure	Increment Test Plan	Yes	3.3.12
SWDes SW Design (Database Design)	Database Design	Yes	3.3.7
UI Prototype		No	
Timebox Plan		Yes	
SWArc SW Architecture	Timebox Definition	Yes	3.3.6

Documents *SysArc_2, UReq_6.5, UReq_6.6, SWDes, AsSpec* and *ASProc* replace the corresponding original Catalysis artifacts, the *Interface Overview* and the *Interface Description* are optional extensions. Timebox Plan and UI Prototype remain unchanged.

The *SW Architecture* associated with each V-Model unit replaces the Catalysis Timebox Definitions associated with each timebox (for the relationship between V-Model units and timeboxes we refer to the remark in section 3.3.6).

Step 03 – Evolutionary Development Cycle

Outputs

Product	***Original Product***	***mandatory***	***section***
Problem Report/Change Request		Yes	
SWArc Software Architecture	Timebox Definition	Yes	3.3.6
AsSpec Assessment Specification	Test Scripts	Yes	3.3.12
	Timebox Test Plan	Yes	
SwDes SW Design	Timebox Design	Yes	3.3.7
ImplD Implementation Documents	Build, Database	Yes	3.3.9
IfOv Interface Overview		No	3.3.4
IfDe Interface Description		No	3.3.4
AsProt Assessment Report	Test Result	Yes	3.3.12
Baseline		Yes	

In this step the timeboxes are realized. The *SW Architecture* corresponding to the Catalysis Timebox Definition is refined. The *SW Design* replaces the Timebox Design, the *Assessment Specification* corresponds to the Catalysis Test Scripts and the Timebox Test Plan, and the *Implementation Documents* contain the builds and databases. The *Assessment Report* replaces the Catalysis Test Result. *Interface Overview* and *Interface Description* are completed by the interfaces of components and classes. The original Catalysis artifacts Problem Report/Change Request and the Baseline remain unchanged.

Step 04 – Increment Finalization

Outputs

Product	*Original Product*	*mandatory*	*section*
ImplD Implementation Documents	Build, Database	Yes	3.3.9
Inspection Summary Report		No	
Problem Report/Change Request		No	
SysArc_2 System Structure	System Architecture Def.	Yes	3.3.2
Walkthrough Report		Yes	
Framework		No	
Reusable Component		No	
Baseline		No	

This step remains in wide parts unchanged. Merely the V-Model *Implementation Documents* replace the installation package and the *System Structure* document replaces the System Architecture Definition which has to be revised in this step.

Step 04 – ED Stage End Assessment

This part remains unchanged.

Outputs

Product	*mandatory*	*section*
Stage Report	Yes	

3.4.4 Iterative Deployment

This step is concerned with the deployment of the application components into the production environment. The V-Model documents of system development and quality assurance we consider in this study are not involved in this step – apart from the *Informations for the User Manual*, *Diagnosis Manual* and *Operator Manual*. While the User Manual replaces the Catalysis User Guide developed in this stage, the other two kinds of documents are extensions. In the other parts the output products defined for this stage remain unchanged.

3.5 Observations Concerning Both Lifecycle Models

In the previous sections we developed a variant of the Catalysis process in which V-Model conform documents are produced. In this way we obtained a framework in which a development team used to work with Catalysis can produce a V-Model conform design, as e.g. required in governmental projects.

For the integration the V-Model documents and the generic product structure have been in the center of our considerations. The activities and scenarios of the V-Model only played a role in so far as they influence the document structure (e.g. incremental development producing multiple instances of specific document types). A general comparison of the lifecycle models of both methods has not been the aim of this study.

In this section, as a by-product, we summarize our observations concerning the process models of Catalysis and the V-Model in a loose sequence of remarks. In these observations we state the differences between the two methods. Since the V-Model is a generic framework which is intended to provide effective support only together with a tayloring to specific teams and projects, we avoid in wide parts valuations of our observations.

Business Models

Catalysis contains a step in which the business models (as-is model, to-be model and essential model) are developed. In the V-Model the development of business models is spread over several activities. In particular, the to-be model is part of the description of the system functionality (*UReq_6.6*). An own chapter for the to-be model before the system functionality description would be advisable.

Security

Security plays an important role in the V-Model. Several document (chapters) describe security aspects at different levels of abstraction. In Catalysis the specification of security aspects has not yet been fully integrated. Security is only one among other criteria in artifacts specifying requirements (e.g. the Architectural Requirements), but is not yet supported in a systematic way.

Product Structure

The generic product structure of the V-Model is based on a hierarchical structuring of the system (in levels such as segments, units or modules). In contrast, the structuring concept of Catalysis with the notions of types, classes, packages and components supports general relationships between the elements like export, import, refinement and inheritance.

Implementation Options

Catalysis supports a clear process in which several architecture options are described, evaluated by prototypes and one solution is chosen for the final realization. In the V-Model this process is less clear. In particular, during the activity of Technical System Design the proposals for the system architecture are developed (and documented in the System Architecture), however no solution is chosen explicitly or marked in the document as such.

Database Design

The V-Model pursues a coherent design and refinement of structuring elements. Units are decomposed into components, modules and databases, these elements are designed and subsequently implemented. In general, Catalysis supports a similar approach but assigns a central role to the database design. In Catalysis the database is designed and implemented for the whole increment, possibly before the specification and implementation of the timeboxes.

Refinement

In Catalysis refinement and traceability is a major concern supported throughout the process. On the one hand, this concerns the explicit specification of refinement models. On the other hand, separate steps deal with the actualization of specification artifacts in order to keep the whole design in a consistent state. Moreover, each element in the focus of the development (the whole system, a specific increment or a timebox in an increment) is always equipped with a set of models defining the portion of the system this element is adhered to (e.g. each timebox definition contains a type specification which defines the part of the increment type specification the timebox is concerned with).

Traceability is also a concern of the V-Model, however not supported in the same rigorous way as in Catalysis. As shown in section 3.1 the requirement allocations in the V-Model correspond to the refinement models of Catalysis. Activities that preserve the consistency of the design are not part of the V-Model. Moreover, the relationships between system portions (units, increments) and the whole system are not as explicitly modeled as in Catalysis (e.g. a unit is not equipped with that part of the functional behavior of the system this unit realizes).

Reuse

Reuse is considered in the V-Model by the integration of off-the-shelf products. Catalysis goes further in the respect that reuse is supported at many levels of abstraction. In particular, the framework-based approach of Catalysis comprises the use of design patterns, off-the-shelf products and process patterns. Moreover, separate design steps support the definition of project-specific frameworks and reusable components.

Incremental Development

As already discussed in section 3.3.14, in the object-oriented development scenario of the V-Model each document instance is associated with a specific increment. In contrast, Catalysis has a global part that specifies the overall system structure and behavior. This global system specification is kept consistent throughout the design. In the V-Model the specification of the first increment takes part of the role of the global system specification. However, neither global and increment parts are clearly separated in this model nor the consistency of the global specification is preserved during the development.

Moreover, also the documents and tasks concerned with the planning of incremental development in Catalysis (e.g. the Incremental Delivery Plan produced in the Solution Definition) do not have an equivalent in the V-Model.

4. Summary

In the previous chapters we developed a joint Catalysis/V-Model process. The joint process has been defined as a variant of the original Catalysis process with V-Model conform documents as outputs. In this way we provide the basis for a design environment in which a development team working with Catalysis tools and familiar with the Catalysis notations and process can produce V-Model conform documents as e.g. required in governmental projects.

The integration has been achieved in four steps ranging from the adjustment of basic notations and the generic product structure of the V-Model to the interpretation of the V-Model documents in the context of Catalysis and to the final integration of the adjusted V-Model documents as outputs of the Catalysis process steps.

A considerable amount of V-Model documents corresponded directly to Catalysis artifacts. For other document types integrational work had to be done interpreting the V-Model document in the spirit of Catalysis. In particular, we had to define a variant of the V-Model's object-oriented development scenario in order to obtain a conform view of document instances. In the joint process the V-Model documents replace the corresponding Catalysis artifacts as inputs and outputs of design steps and tasks.

A further set of documents has been identified which has no correspondence in Catalysis. Primarily, these documents are concerned with the specification of security and maintenance aspects or contain information that can be deferred from other documents. The joint process adds this kind of documents in appropriate design steps.

Those Catalysis artifacts that have no correspondence in the V-Model remain output artifacts in the joint process. These artifacts are primarily concerned with management and planning activities.

Summarizing we can state that in their kernel parts the document types of the both methods fit well to each other, though Catalysis provides more rigorous support of principles like refinement and evolutionary and architecture-centric design. Additionally considering the V-Model's submodels of project and configuration management, an even greater correspondence could be achieved.

As a result we obtain a framework in which V-Model conform documents can be produced in the context of a Catalysis environment without considerable additional effort, provided that the V-Model output document structure is supported in a similar way as in the original Catalysis process.

References

[Boo94] G. Booch: Object-Oriented Analysis and Design with Applications, Benjamin/Cummings Publishing Company Inc., Redwood City, CA, 1994.

[DHM98] W. Dröschel, W. Heuser, R. Midderhoff: Inkrementelle und objetorientierte Vorgehensweise mit dem V-Modell 97, R. Oldenbourg Verlag, 1998.

[DW99] Desmond F. D'Souza, Alan Cameron Wills: Objects, Components and Frameworks with UML; the Catalysis Approach, Addison-Wesley Object Technology Series, 1998.

[Har97] D. Harel: Statecharts: A Visual Formalism for Complex SystemsScience of Computer Programming (8), Elsevier, North Holland, 1987.

[IABG97] IABG: V-Modell 97, Entwicklungsstandard für IT-Systeme des Bundes, Allgemeiner Umdruck 250, 1997, http://www.v-modell.iabg.de/.

[Jac92] I. Jacobson: Object-Oriented Software Engineering - A Use Case Driven Approach, Addison-Wesley Publishing Company, Wokingham, England, 1992.

[MIL80] MIL-STD 1629A: Procedures for Performing a Failure Mode, Effects and Critically Analysis, 1980.

[SM92] Shlaer, Mellor: Object Lifecycles - Modeling the World in States, Prentice-Hall Inc., Englewood Cliffs, NJ, 1992 .

[UML97] G. Booch, I. Jacobson, J. Rumbaugh: Unified Modeling Language, Version 1.0, 1997, http://www.rational.com/.

[UML99] OMG: Unified Modeling Language Specification, Version 1.3 alpha R5, March 1999, http://www.omg.org/.

[Wir93] R. Wirfs-Brock: Objektorientiertes Software-Design, hanser Verlag, 1993.

Annex

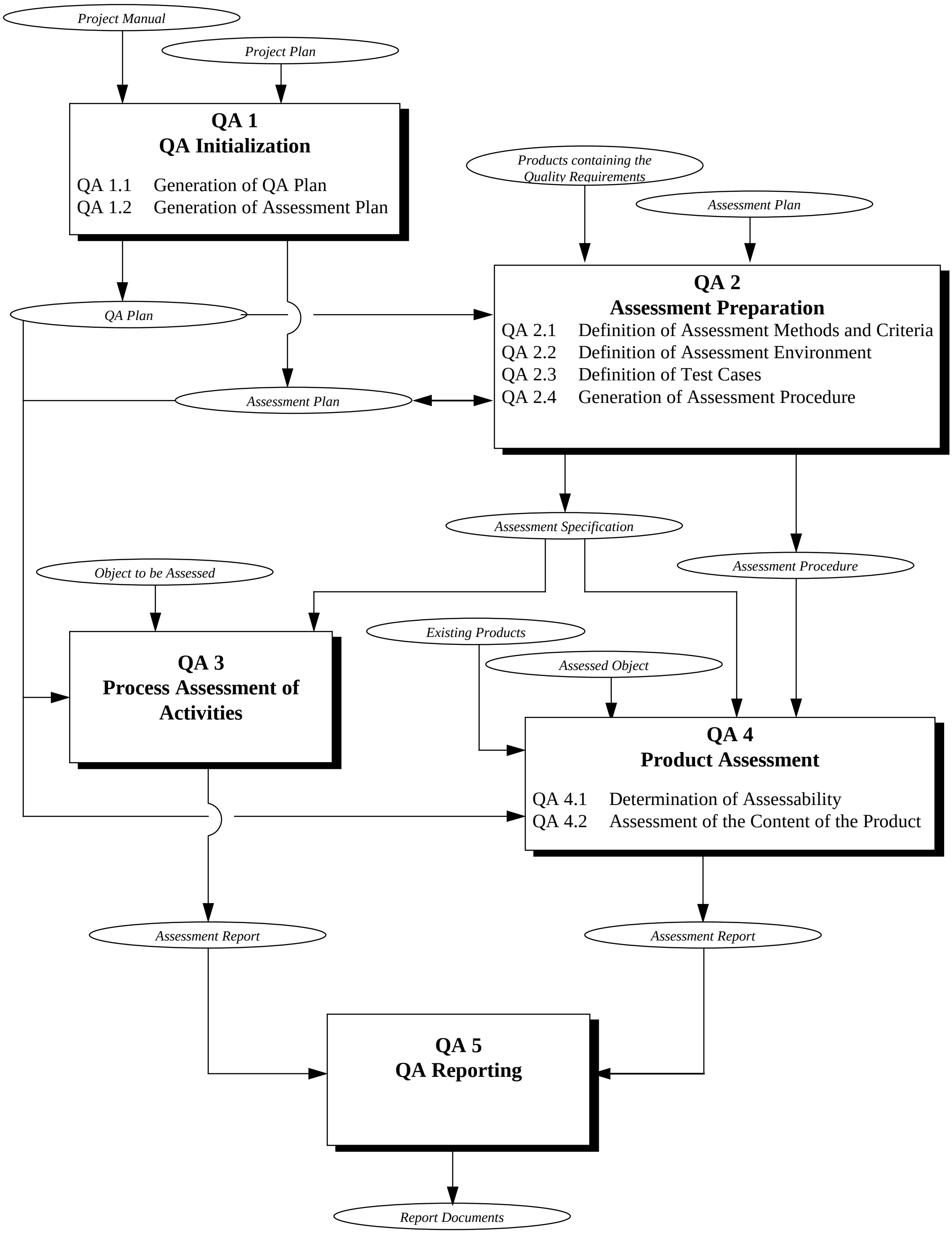

Project Manual
Project Plan
QA 1
QA Initialization
QA 1.1 Generation of QA Plan
QA 1.2 Generation of Assessment Plan
Products containing the Quality Requirements
Assessment Plan
QA Plan
QA 2
Assessment Preparation
QA 2.1 Definition of Assessment Methods and Criteria
QA 2.2 Definition of Assessment Environment
QA 2.3 Definition of Test Cases
QA 2.4 Generation of Assessment Procedure
Assessment Plan
Assessment Specification
Object to be Assessed
Assessment Procedure
Existing Products
Assessed Object
QA 3
Process Assessment of Activities
QA 4
Product Assessment
QA 4.1 Determination of Assessability
QA 4.2 Assessment of the Content of the Product
Assessment Report
Assessment Report
QA 5
QA Reporting
Report Documents

Figure 7: Overview of the V-Model's submodel QA (Quality Assurance) (cf. [IABG97])

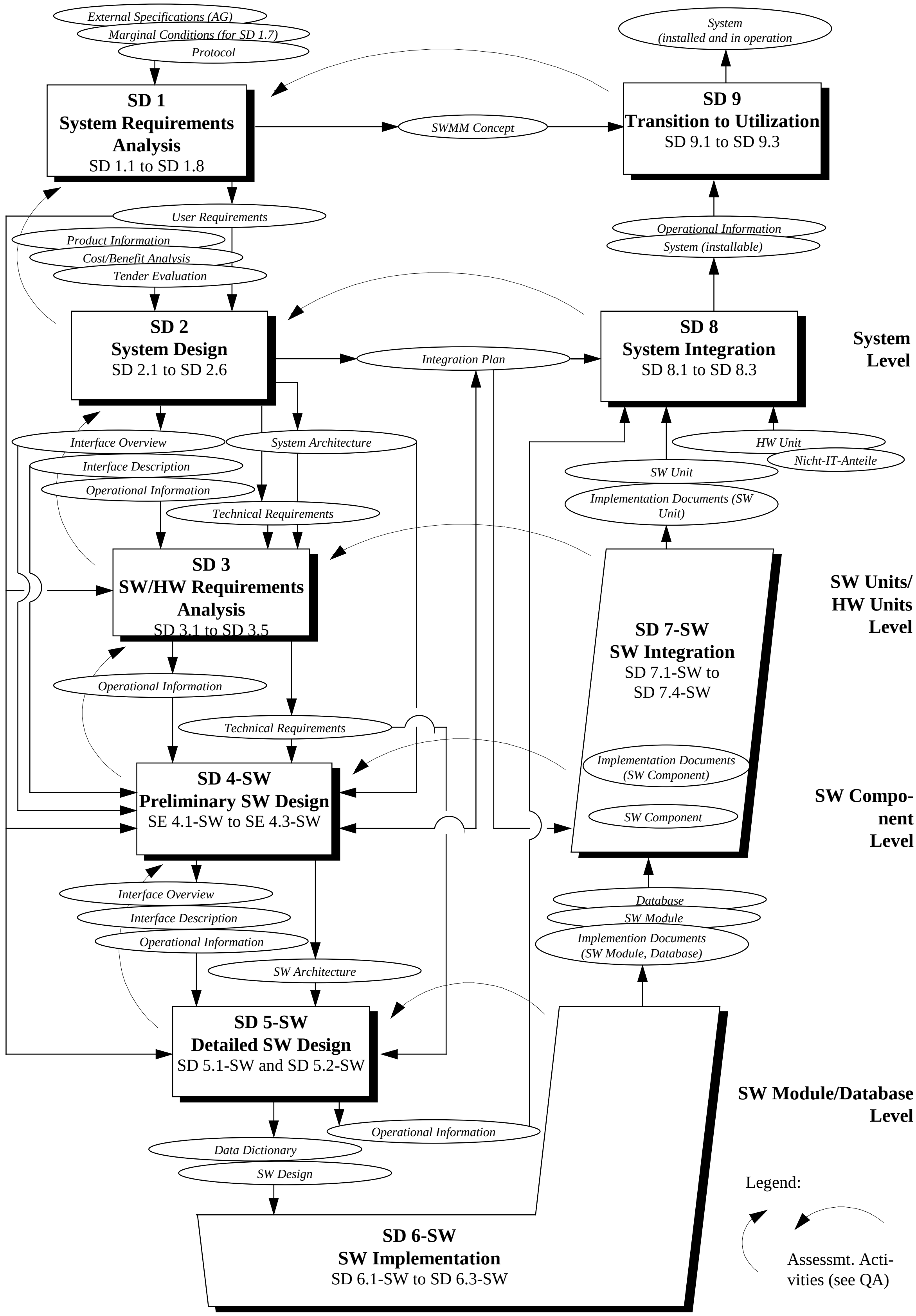


Figure 8: Overview of V-Model's submodel SD (System Development) (cf. [IABG97])