FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

The Creation of a Methodology for the Architectural Design Phase of a Scenario-Driven Software Development Environment

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Magister

im Rahmen des Studiums

Informatikmanagement

eingereicht von

Thomas Bruckmayer
Matrikelnummer 0225696

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung: Univ.Prof. Dipl.-Ing. Dr.techn. Hermann Kaindl
Mitwirkung: Projektass. Dipl.-Ing. Dr.techn. Jürgen Falb

Wien, 13.09.2011

(Unterschrift Verfasser) (Unterschrift Betreuung)

FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

The Creation of a Methodology for the Architectural Design Phase of a Scenario-Driven Software Development Environment

MASTER'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Master

in

Computer Science Management

by

Thomas Bruckmayer
Registration Number 0225696

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Univ.Prof. Dipl.-Ing. Dr.techn. Hermann Kaindl
Assistance: Projektass. Dipl.-Ing. Dr.techn. Jürgen Falb

Vienna, 13.09.2011

(Signature of Author) (Signature of Advisor)

Erklärung zur Verfassung der Arbeit

Thomas Bruckmayer
Heiligenstädter Straße 107-109/8, 1190 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit - einschließlich Tabellen, Karten und Abbildungen -, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

(Ort, Datum) (Unterschrift Verfasser)

Abstract

The increasing complexity of modern software applications encourages the further development of sophisticated software development processes. However, developers are often required to write software programs from scratch because reuse is still not sufficiently handled. The objective of the ReDSeeDS project, supported by the European Union, was to realize a scenario-driven software development system. With the help of this development system, developers are able to retrieve solution information (models and code) of the most similar problem and adapt the retrieved information for further reuse. The ReDSeeDS project supports the entire software development life cycle including the phases Requirements Engineering, Architectural Design, Detailed Design and Implementation. The ReDSeeDS processes for each phase differ in the usage of ReDSeeDS-specific activities in order to enable companies to adapt the ReDSeeDS Framework step by step. For instance, one ReDSeeDS process provides the specific notation but reuse is not yet supported, whereas another ReDSeeDS process provides full reuse support including model transformations.

During the Architectural Design Phase the basis and the framework of the solution are created. Therefore, requirements are transformed into definitions of software components and their interfaces. The Architectural Design Phase is the first phase of the software development life cycle which turns the requirements into a solution, and therefore, all the activities have to be carried out in an elaborated way to provide a solid basis for the Detailed Design and the following Implementation Phase. While the application of the ReDSeeDS project significantly improves each phase, it also adds additional activities that require attention. Detailed guidance for each phase is provided by the ReDSeeDS Methodology which has been created by several developers from different European countries. Developing such a methodology is a complex task, and therefore, this thesis describes in detail the ReDSeeDS Architectural Design Phase and its documentation within the ReDSeeDS Methodology.

In addition, the ReDSeeDS Methodology is tightly coupled to the ReDSeeDS Tool in order to bridge the gap between the high level description of the methodology and the detailed description to perform certain tasks within the tool. For instance, the methodology describes that a certain output has to be produced within a particular activity. The textual description of this activity directly connects to the tool and instantly presents additional guidance on the level of user interface interaction to achieve the described goal. The theoretical principles and the technical implementation of this method–tool coupling approach are described in this thesis as well.

Kurzfassung

Die zunehmende Komplexität moderner Software-Anwendungen fördert die ständige Weiterentwicklung von anspruchsvollen Software-Entwicklungsprozessen. Dennoch müssen Entwickler nach wie vor in vielen Fällen Software-Programme von Grund auf neu schreiben, weil Wiederverwendung im Entwicklungsprozess nicht ausreichend behandelt wird. Das Ziel des ReDSeeDS-Projekts mit Unterstützung der Europäischen Union, war die Schaffung einer auf Szenarien basierenden Software-Entwicklungsumgebung. Diese Umgebung ermöglicht Software-Entwicklern Informationen (Modelle und Code) zu ähnlichen Problemstellungen abzurufen, und die abgerufenen Informationen für die weitere Wiederverwendung anzupassen. Das ReDSeeDS-Projekt unterstützt den gesamten Software-Lebenszyklus einschließlich der Phasen Anforderungsanalyse, Architekturentwurf, Designentwicklung und Implementierung. In jeder Phase unterscheiden sich die ReDSeeDS-Prozesse hinsichtlich der Nutzung von ReDSeeDS-spezifischen Aktivitäten, und ermöglichen dadurch eine schrittweise Einführung des ReDSeeDS-Frameworks für Unternehmen. Zum Beispiel unterstützt ein ReDSeeDS-Prozess die spezifische Notation, wobei Wiederverwendung noch nicht zum Einsatz kommt. Ein anderer ReDSeeDS-Prozess behandelt die vollständige Wiederverwendung einschließlich Modelltransformationen.

Die Architekturentwurfsphase ist die erste Phase des Entwicklungszyklus, die die Anforderungen in Lösungskomponenten verwandelt, und deshalb ist eine anspruchsvolle Ausführung der einzelnen Aktivitäten zwingend erforderlich. Die Anwendung des ReDSeeDS-Projekts erweitert jede Phase um weitere Möglichkeiten, das wiederum bedeutet, dass zusätzliche Aktivitäten koordiniert werden müssen. Eine ausführliche Anleitung für die Abwicklung jeder Phase wird durch die ReDSeeDS-Methodik, die von mehreren Entwicklern aus verschiedenen europäischen Ländern erstellt wurde, zur Verfügung gestellt. Die Entwicklung einer solchen Methodik ist eine komplexe Aufgabe und daher beschreibt diese Magisterarbeit im Detail die ReDSeeDS-Architekturentwurfsphase und deren Dokumentation in der ReDSeeDS-Methodik.

Darüber hinaus ist die ReDSeeDS-Methodik eng an das ReDSeeDS-Tool gekoppelt, um die Lücke zwischen der allgemeinen Beschreibung der Methodik und der detaillierten, toolspezifischen Beschreibung zu überbrücken. Zum Beispiel beschreibt die Methodik, dass ein bestimmtes Resultat innerhalb einer bestimmten Tätigkeit erstellt werden muss. Die allgemeine textuelle Beschreibung dieser Tätigkeit ist direkt mit dem Tool verknüpft, um die konkrete Vorgehensweise im Tool durch automatisierte Benutzeraktionen aufzuzeigen. Diese Arbeit befasst sich auch mit den theoretischen Grundlagen und der technischen Umsetzung dieses Methodik-Tool-Kopplungsansatzes.

Contents

CHAPTER 1

Introduction

1.1 Software Engineering

Modern software applications improve the daily life of millions of people to such an extent that their omnipresent existence is taken for granted but often the immense effort to realize them remains unnoticed. The diversity of multiple applications and the individual use through numerous users are only an outline of factors which cause the immense complexity of developing software. To handle this complexity engineering disciplines like software engineering and standardized processes emerged. A definition of software engineering can be found in [2]:

> The application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software; that is, the application of engineering software.

Software Engineering can also be described from a more economic point of view, like a comment in [1]:

> Computer software has become a driving force. It is the engine that drives business decision making. It serves as the basis for modern scientific investigation and engineering problem solving. It is a key factor that differentiates modern products and services. It is embedded in systems of all kinds: transportation, medical, telecommunications, military, industrial processes, entertainment, office products . . . the list is almost endless. Software is virtually inescapable in a modern world. And as we move into the twenty-first century, it will become the driver for new advances in everything from elementary education to genetic engineering.

Moreover, descriptions of the term “software” from different perspectives can be found in [15]. There a “software process” is described as a “road map or a series of predictable steps that help developers to create a timely and high-quality result”. Furthermore it is stated that

the software development process is adapted by software engineers and their managers to serve their needs. Moreover the actual work products are the programs, documents and data produced by the software engineering activities defined by the development process [15]. In order to get a better understanding what Software Engineering is all about, the following components and aspects have to be considered [2]:

- development methods
- tools
- set of measures
- standards
- experience

Development methods provide the technical how-to descriptions and specify tasks which handle various aspects of Software Engineering like requirements analysis, design, and implementation. The exact phases which have to be covered by the development methods are described in Chapter 3 but it is important to state that software engineering methods rely on a set of basic principles depending on the field and the technology. These methods are supported by software engineering tools. In general tools are integrated in the software development so that information created by one tool can be used by another tool. A set of measures provide a quantitative indication of the extent, amount, dimension, capacity, or size of some attribute of a product or process. Therefore, a software engineer uses metrics to get an insight if the software development process is on the right track or needs to be adjusted. Specified standards provide additional guidance during the software development process. A set of development criteria eases the communication among all the involved people and avoid lack of quality which almost surely results if standards are not followed [15]. Experience resulted from previous software development processes is extremely valuable and has to be included into the ongoing process in order to avoid mistakes from the past. In [15], Software Engineering is described as a layered technology as shown in Figure 1.1.

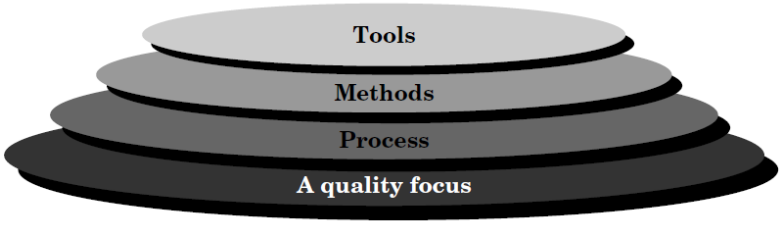

Figure 1.1: Software Engineering Layers taken from [15]

1.2 Software Development Process

A software development process can be roughly described as a set of activities which lead to the development of the desired software solution. As described above, a software engineer must incorporate a development strategy including methods and tools. Moreover, several models exist in order to streamline the development process as described in [15].

The Linear Sequential Model

The linear sequential model (also often referred as waterfall model) provides a process model where one phase is strictly followed by the other and is one of the oldest and widely used software development paradigms. One fundamental problem which is not addressed by this model is the fact that real world projects in the industry hardly follow a strictly sequential flow. On the contrary, changes occur quite often and therefore additional iterations are required. Furthermore, customers frequently add additional requirements later on because it is often not applicable to gather all the requirements explicitly at the beginning of the project. Another drawback of the linear sequential model is the fact that customers will only be able to view the first prototypes late in the software development life cycle. In order to partially encounter these drawbacks the waterfall model by Boehm (Figure 1.2) introduces feedback to the immediately preceding phase.

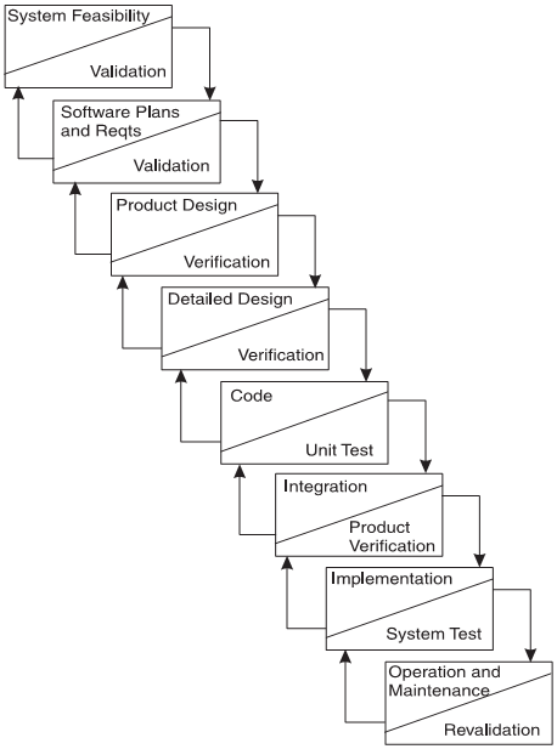

Figure 1.2: Waterfall Model by Boehm taken from [11]

The Prototyping Model

In many business scenarios the basic customer is not an expert when it comes to the definition of the requirements. Therefore, in many situations a prototype supports the gathering of the requirements to a great extent. As depicted in Figure 1.3, a draft design already takes place after the first meetings between the customer and the persons who are responsible for the requirements engineering.

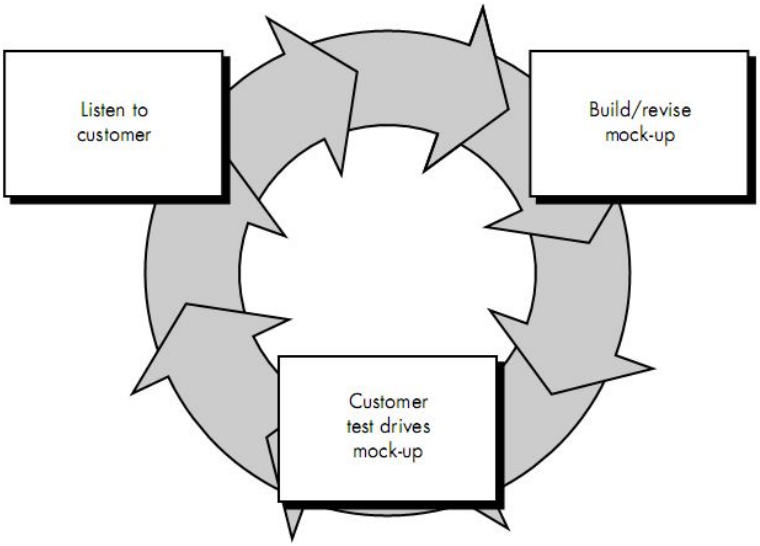

Figure 1.3: The Prototyping Model taken from [15]

After the first draft implementation the prototype is evaluated by the customer and provides a feasible basis for refining the requirements. The biggest drawbacks of this approach are the lack of customer awareness that the actual solution still has to be developed (because the prototype was only for gathering the requirements and cannot be used for solving the problem) and that the developers maybe get affected by the bad prototype design and don't develop all the components of the solution according to best practice approaches.

The RAD Model

In general, customers demand that the software development process is as short as possible and at the same time the development tools and methods are constantly improving. These circumstances lead to the creation of the Rapid Application Development (RAD) principle. Past projects show that the RAD model is used primarily for the development of information systems and encompasses the phases shown in Figure 1.4.

During the first phase the information exchange between business functions is modeled. This information flow is refined into data objects. These data objects are used in processes modeled in the Process modeling phase. The applications supporting the processes are generated through automated tools within the Application generation phase. During the last phase the created

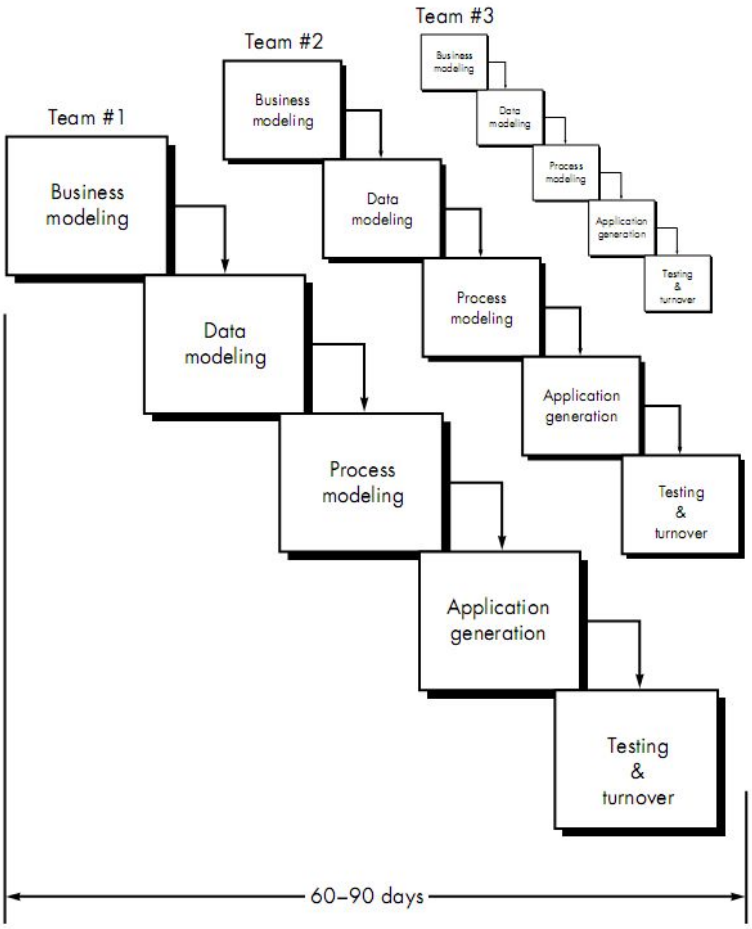

Figure 1.4: The RAD Model taken from [15]

components are tested and deployed. As mentioned in [15], the RAD approach can only be used if sufficient human resources are available to create the required number of RAD teams and if requirements do not rely on certain new technologies, which limit the automated generation of applications.

The Spiral Model

The spiral model focuses on the evolutionary character of software development processes and defines six basic task regions which are iterated in cycles as shown in Figure 1.5.

The software engineering team moves around the spiral clockwise, starting at the center. The first circuit may result in a project specification; subsequent circuits lead to the development of the first prototype, till the solution is implemented and deployed. The spiral model is a realistic approach to enterprise engineering of large-scale systems but it might be to sedate for smaller

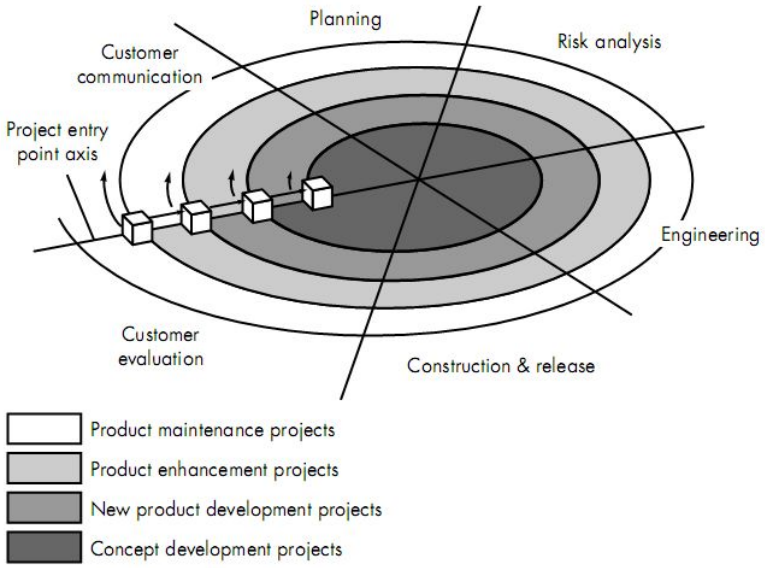

Figure 1.5: The Spiral Model taken from [15]

projects and not all customers may be open for this evolutionary approach.

Software Development Phases

These software development models substantiate the fact that the creation of a Software Development Process is a highly complex task and that numerous definitions are available. To sum things up, most of them describe the Software Development Process as a process which consists of the following basic phases:

1. Requirements Analysis
2. Design
3. Implementation
4. Test
5. Deployment
6. Maintenance

Moreover, it is important to define specific achievements for every single phase. In a strict sequential model, a phase can only be engaged when the previous one is completed. This eases the software development from an organizational point of view but is not really applicable in practice because in many cases the results affect the previous phase. As a consequence the idealized sequential model is often replaced by non sequential phase models.

Software models ease the software development process without any doubts but numerous specific challenges regarding software development need additional guidance and tool support.

1.3 Problem Statement of the Thesis

In order to enable reuse during the software development process on a very large scale the ReDSeeDS project with the following main objective was introduced [16]:

> The main objective of the project is to create an open framework consisting of a scenario-driven development method (precise specification language and process for the "how-to"), a repository for reuse and tool support throughout. The basic reuse approach is case-based, where a reusable case is a complete set of closely linked (through mappings or transformations) software development technical artifacts (models and code), leading from the initial user's needs to the resulting executable application.

The simple existence of a framework and tools is not enough to enable sophisticated software development. Moreover, the developers have to know how to use the provided framework and the corresponding tools. The development of a well elaborated methodology is a complex task and requires a fastidious planned proceeding. This master thesis describes the main components and the development of a software development methodology by means of a concrete example, namely the Architectural Design Phase of the ReDSeeDS Methodology which has been created by a team of the Institute of Computer Technology at the Technical University of Vienna. The author of this thesis was part of the creation team.

1.4 Structure of the Thesis

After the introduction, this thesis is divided into the following chapters. Chapter 2 provides a general overview of the basic concepts of a software development methodology, including single components and models for visualization. Chapter 3 illustrates these concepts through a real world example created by a development team including the author of this thesis, namely the ReDSeeDS project. More precisely, this Chapter includes a brief description of each phase of the ReDSeeDS project, as well as a more detailed description of the Architectural Design Phase. Chapter 4 describes a methodology – tool coupling approach developed by a team including the author of this thesis.

CHAPTER 2

Software Development Methodologies

According to [21] a software development methodology is a framework that is used to structure, plan, and control the process of developing an information system. Furthermore one specific software development methodology framework is not necessarily suitable for use by all projects. Too many factors like the project organization and the used technologies play an important role and each of the available methodology frameworks was created for specific purposes. One recommended approach is to take the best suited methodologies, extract the important parts and to create a brand new tailored methodology for the given problem domain. As a consequence, well established software development frameworks are often bound to some kind of organization, for instance, the Rational Unified Process from IBM or the Agile Unified Process by Scott Ambler [21]. But before introducing the Rational Unified Process a brief overview of software development best practices from [10] is provided.

2.1 Software Development Best Practices

Develop software iteratively. As pointed out in Section 1.2 software development bears an evolving character which can be handled by iterative and incremental processes. For instance, following an iterative model allows the development team to react cost effectively to changes in the requirements specification because this phase is not only traversed once. As described in the next chapter the ReDSeeDS approach encompasses multiple ReDSeeDS related tasks which can be integrated step by step in different software development approaches [10].

Manage requirements. Requirements may change during the software development project and, therefore, the key activities, namely eliciting, organizing and documenting the system's required functionality and constraints deserve special attention. The effective engineering of requirements is one key component of the ReDSeeDS approach. As described in the next chapter a specific notation for requirements has been introduced which enables the automatic storage and retrieval of requirements. Moreover requirements can be automatically transformed into system components which are used through subsequent phases [10].

Use component-based architectures. The system's architecture reflects the organization of a software system regarding the input of different stakeholders which are involved in the development process. Section 2.5 elaborates more on the basic characteristics of software architecture like the identification of structural elements, their interfaces and their behavior in the system. Regarding best practices it is important to mention that software architecture is not only concerned with structure and behavior but also with usage, functionality, performance, resilience, reuse, and many other aspects. In order to enable reuse a component-based development (CBD) is an indispensable approach. Moreover, CBD follows the separation of concerns principle, which provides flexibility and interchangeability to certain components of the software system. Once again, the ReDSeeDS approach described in the next chapter of this thesis takes the component-based development practice a few steps further by providing a framework for reusing parts of the architectural components automatically [10].

Visually model software. As stated in [10] a model is a simplification of a real world example that describes a system from a particular perspective as depicted in Figure 2.1. As a consequence, models provide a better understanding of complex systems.

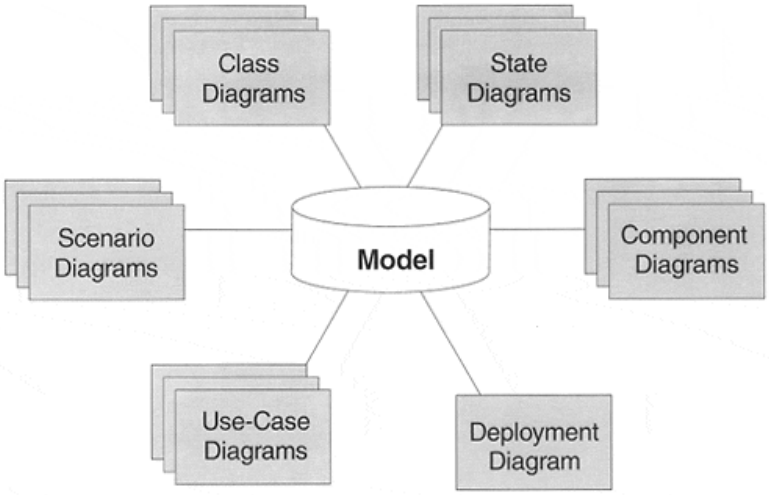

Figure 2.1: Modeling a system from different perspectives from [10]

In order to ease communication it is essential to use a standard modeling language such as UML (Unified Modeling Language developed by the Object Management Group [1]). Moreover, the visual modeling of software has a strong support through the industry and therefore several tools facilitate the management of visual models. This best practice is also covered by the ReDSeeDS approach. Diagrams modeled in an extended subset of UML can be automatically queried and retrieved from a common repository. Additionally, models can be transformed to artifacts used by other software development phases, for instance model and code transformations are supported.

[1] http://www.uml.org/

Continuously verify software quality. It is well known that software problems are much more expensive to find and to correct after deployment. Therefore, a continuous verification of the software quality during the development process is extremely important. This can be achieved by a detailed planned and iterative testing phase for each achieved milestone [10]. The ReDSeeDS framework improves the software quality because of the automatic reuse of well elaborated components but the verification process is not yet explicitly improved through automation.

Control changes to software. Multiple developers are working on the same software development project at the same time. Their activities have to be coordinated and their produced results collaboratively managed [10]. The ReDSeeDS Methodology clearly identifies certain roles and allocates them specific activities. Furthermore, the main ReDSeeDS tools support multiple users and version control.

2.2 The System Development Method SEM as an Idea Provider for the ReDSeeDS Methodology

As stated in [9], the methodological development of software is not only an issue in scientific research, but also bears great challenges in industrial practice. The system development method SEM ("Systementwicklungsmethode" in German) provides guidance for a software development project and is used by Siemens AG Austria. SEM points out the importance of dividing the software development process into precisely specified phases, which are supported through the techniques mentioned in 1. Moreover, SEM forms the basis for derivation of concrete methods, which are practically applied in projects. One example for a concrete method is ooSEM for object-oriented software development projects. According to ooSEM, a software development project encompasses the following phases:

- Initiation
- Definition
- Design
- Implementation
- Application
- Completion

In addition to this, SEM points out the possibilities using Hypermedia and Networks. The advantages of online documentation compared to traditional printed manuals are described in [9]:

- The contents of an electronic manual are available immediately at the workplace and can be used directly in the work process.
- A central intranet solution can be updated more often and more easily.

- A central Web-based intranet solution can be easily connected with other information.
- The communication and the information exchange among users are supported, and therefore improves the further development of the guiding method.
- The different technological platforms within the project are bridged with a Web-based solution, because no additional browser software is required.
- A Web-based documentation requires a sophisticated design, and therefore, the SEM Web representation is phase-oriented. As shown in Figure 2.2 users are able to access the desired information for every single phase, as well as phase specific information.

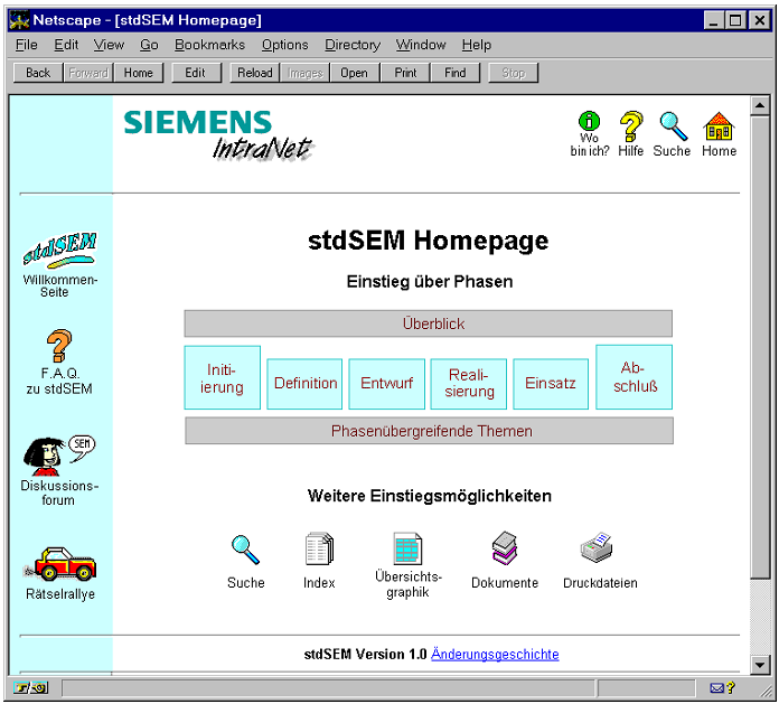

Figure 2.2: Screenshot of a SEM Website taken from [9]

Due to the fact that SEM documents rules and methods, the development of a particular solution is coordinated within a defined process. For each mentioned phase, standardized results and milestones are specified. Within the Initiation Phase, the feasibility of the project is analyzed. The main goals of the Definition Phase are the creation and the verification of the requirements, the creation of the project plan and quality assurance plans. The achievements of the Design Phase are a specified and verified solution and a verified testing plan. During the implementation Phase the product is developed, verified and finally accepted. The Application Phase ensures a successful product launch and the Completion Phase guarantees the correct archiving of every project-relevant information for possible later use.

2.3 The Rational Unified Process in Comparison with the ReDSeeDS Methodology

As described in Chapter 3 the RedSeeDS Methodology consists of several processes regarding software development. Before describing these innovative processes in detail, an overview of a well established software development process, namely the Rational Unified Process (RUP) is provided at this point. As IBM acquired Rational in 2003 one of the largest IT companies became responsible for maintaining the RUP and for providing updates. Moreover according to [10] one can see the RUP as a software product itself because it is delivered using Web technology, it can be tailored and configured to suit the specific needs of a development organization and it is integrated into many software development tools of the IBM Rational Suit. The fact that the RUP is treated more as a software product and not published as a book enables the developer to follow the process with the benefits provided by modern Web technologies like search functionalities and through hyperlinks accessible content as shown in Figure 2.3.

As described in [10] and shown in the overview section of the RUP HTML representation, the Rational Unified Process has two dimensions or structures as shown in Figure 2.4.

The two dimensions are described in the RUP as the following:

1. The horizontal axis represents time and shows the lifecycle aspects of the process as it unfolds. This first dimension illustrates the dynamic aspect of the process as it is enacted and expressed in terms of phases, iterations, and milestones.

2. The vertical axis represents workflows that logically group activities by nature. This second dimension portrays the static aspect of the process in terms of process components, disciplines, activities, workflows, artifacts, and roles.

It is worth mentioning that the expressions for classification objects which contain activities may vary. For instance the overview diagram of the RUP HTML representation labels the vertical axis with “disciplines” instead of “workflows” as in [10] but on subpages the items like Business Modeling, Requirements, Analysis and Design, etc. are labeled as workflows as well.

The phases organized along time address once again the importance of an iterative development as stated in 1 and are fully supported by the ReDSeeDS approach. More information regarding the dynamic iterative development from a RUP point of view can be found in [10]. Additionally [10] depicts the importance of project management at this point. The RUP project management workflow provides a framework for managing software projects and risks and provides guidelines for monitoring projects. On the other hand the Rational Unified Process cannot cover all the manifold aspects of project management like managing human resources (for instance hiring, training, coaching) or managing budgets and contracts [10].

When it comes to planning an iterative project [10] suggests two levels of plans - first the phase plan (coarse-grained) and second the iteration plans (a series of fine-grained plans). There should be only one phase plan per software development covering the major milestones, the required resources and the dates of the minor milestones. The iteration plans are built on traditional planning techniques and tools, in many cases classic Gantt charts are used. It is important

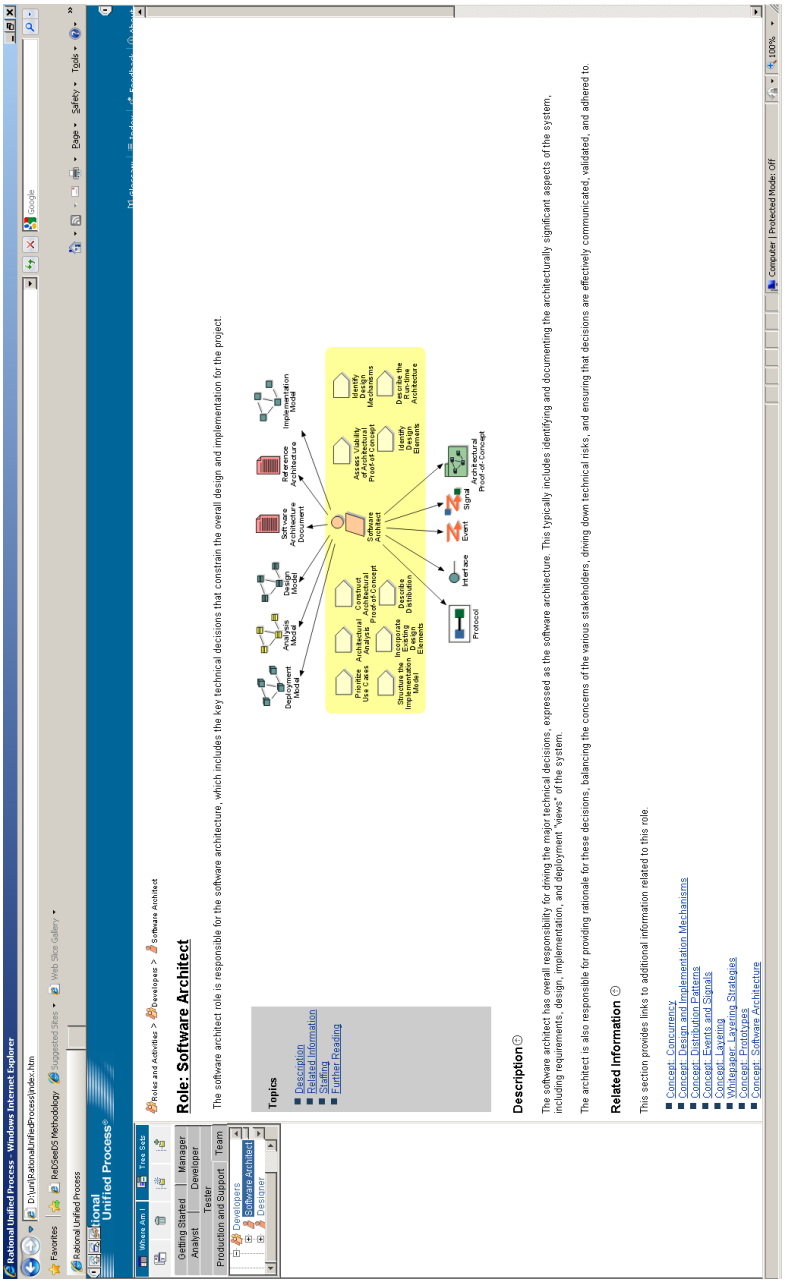

Figure 2.3: Screenshot of RUP Web representation

to know that two iteration plans usually become active at the same time, namely the current iteration plan and the iteration plan for the next pending iteration. The reasons for that are simple, because the next iteration plan is created during the second half of the current iteration [10].

The static aspects of the second dimension are basically represented through the primary

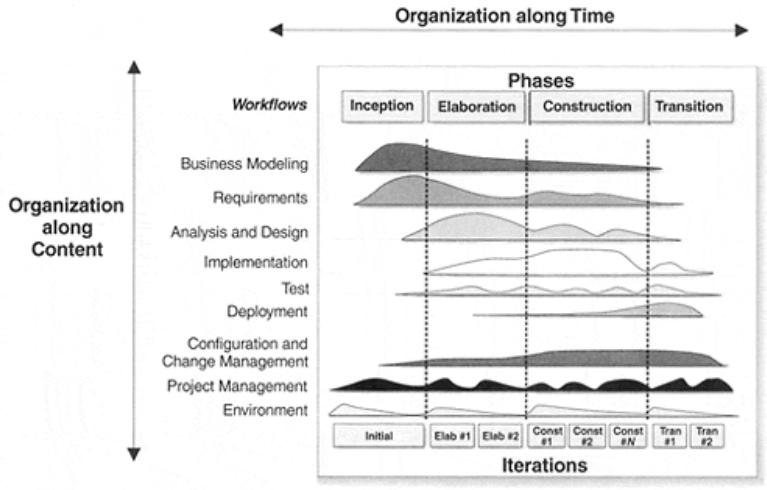

Figure 2.4: RUP – two dimensions taken from [10]

model elements.

2.4 Primary Model Elements

The process components or workflows describe who is doing what, how and when. The following primary modeling element description is taken from [10]. Table 2.1 gives an overview of the primary modeling elements regarding the ReDSeeDS Methodology elements used through the next chapter.

	Rational Unified Process	**ReDSeeDS Methodology (based on SPEM)**
the who	Workers, Roles	Roles
the how	Activities	Work Definition, Activities
the what	Artifacts	Work Products
the when	Workflows	Work Definition and Activities ordered on diagrams in Process Packages

Table 2.1: Primary Model Elements

The RUP defines a worker as follows [10]:

> A worker defines the behavior and responsibilities of an individual or a group of individuals working together as a team. The behavior is expressed in terms of activities. The worker performs, and each worker is associated with a set of cohesive activities. "Cohesive" in this sense means those activities best performed by one person. The responsibilities of each worker are usually expressed in relation to cer-

tain artifacts that the worker creates, modifies, or controls. Examples for workers are System Analyst, Designer, and Test Designer.

SPEM defines a role as method content as follows [12]:

> A Role Definition is a Method Content Element that defines a set of related skills, competencies, and responsibilities. Roles are used by Task Definitions to define who performs them as well as to define a set of Work Product Definitions they are responsible for. A Role Definition defines a set of related skills, competencies, and responsibilities of an individual or a set of individuals. Roles are not individuals or resources. Individual members of the development organization will wear different hats, or perform different roles. The mapping from individual to Role, performed by the project manager when planning and staffing for a project, allows different individuals to act as several different roles, and for a role to be played by several individuals.

The ReDSeeDS Software Development Methodology Deliverable [22] defines as follows:

> A role is responsible for a set of work products and is performer of activities. A process role is not a person. A given person may be acting in several roles and several persons may act as a single given role.

The RUP defines activities as follows [10]:

> Workers have activities, which define the work they perform. An activity is a unit of work that an individual in that role may be asked to perform, and that produces a meaningful result in the context of the project. The activity has a clear purpose, usually expressed in terms of creating or updating artifacts, such as a model, a class, or a plan. Every activity is assigned to a specific worker. The granularity of an activity is generally a few hours to a few days. It usually involves one worker and affects one or only a small number of artifacts. An activity should be usable as an element of planning and progress; if it is too small, it will be neglected, and if it is too large, progress will have to be expressed in terms of the activity's parts. Activities may be repeated several times on the same artifact, especially from one iteration to another as the system is refined and expanded. Repeated activities may be performed by the same worker but not necessarily the same individual. Activities are broken into steps. Steps fall into three main categories:
>
> - Thinking steps: The worker understands the nature of the task, gathers and examines the input artifacts, and formulates the outcome.
> - Performing steps: The worker creates or updates some artifacts.
> - Reviewing steps: The worker inspects the results against some criteria.

SPEM defines a Work Definition as follows [12]:

> Work Definition is an abstract Classifier that generalizes all definitions of work within SPEM 2.0. Work Definition defines some default associations to Work Definition Parameter and Constraint. Work Definitions can contain sets of pre- and post-conditions defining constraints that need to be valid before the described work can begin or before it can be declared as finished. Note that general UML constraints inherited via Classifier can be used to define additional constraints and rules for Work Definitions.

SPEM defines an Activity as follows [12]:

> An Activity is a Work Breakdown Element and Work Definition that defines basic units of work within a Process, as well as a Process itself. In other words, every Activity represents a Process in SPEM 2.0. It relates to Work Product Use instances via instances of the Process Parameter class and to Role Use instances via Process Performer instances.

The ReDSeeDS Software Development Methodology Deliverable [22] defines an Activity as follows:

> An activity describes a piece of work performed by one role. Other roles may assist. An activity may consist of atomic elements called steps.

Definition of an Artifact in RUP as follows [10]:

> Activities have input and output artifacts. An artifact is a piece of information that is produced, modified, or used by a process. Artifacts are the tangible products of the project: the things the project produces or uses while working toward the final product. Artifacts are used as input by workers to perform an activity and are the result or output of such activities. In object-oriented design terms, just as activities are operations on an active object (the worker), artifacts are the parameters of these activities. Artifacts may take various shapes or forms:
>
> - A model, such as the use-case model or the design model
> - A model element – an element within a model – such as a class, a use case, or a subsystem
> - A document, such as a business case or software architecture document
> - Source code
> - Executables
>
> Note that artifact is the term used in the Rational Unified Process. Other processes use terms such as work product, work unit, and so on, to denote the same thing. Deliverables are only the subset of all artifacts that end up in the hands of the customers and end users. Artifacts can also be composed of other artifacts. For example, the design model contains many classes; the software development plan contains several other plans: a staffing plan, a phase plan, a metrics plan, iteration plans, and

so on. Artifacts are very likely to be subject to version control and configuration management. Sometimes, this can be achieved only by versioning the container artifact when it is not possible to do it for the elementary, contained artifacts. For example, you may control the versions of a whole design model or design package and not of the individual classes they contain. Typically, artifacts are not documents. Many processes place an excessive focus on documents and in particular on paper documents. The Rational Unified Process discourages the systematic production of paper documents. The most efficient and pragmatic approach to managing project artifacts is to maintain the artifacts within the appropriate tool used to create and manage them. When necessary, you can generate documents (snapshots) from these tools on a just-in-time basis. You should also consider delivering artifacts to the interested parties inside and together with the tool rather than on paper. This approach ensures that the information is always up-to-date and is based on actual project work, and producing it should not require additional effort.

Definition of a Work Product in SPEM [12]:

> Work Product Definition is Method Content Element that is used, modified, and produced by Task Definitions. Work Product Definitions can be related to other Work Product Definitions via the Work Product Definition Relationship. Work Products are in most cases tangible work products consumed, produced, or modified by Tasks. They may serve as a basis for defining reusable assets. Roles use Work Products to perform Tasks and produce Work Products in the course of performing Tasks. Work Products are the responsibility of Role Definitions, making responsibility easy to identify and understand, and promoting the idea that every piece of information produced in the method requires the appropriate set of skills. Even though one Role Definition might "own" a specific type of Work Product, other roles can still use the Work Product for their work, and perhaps even update them if the Role Definition instance has been given permission to do so.

Definition of a work product in the ReDSeeDS Software Development Methodology Deliverable [22]:

> A work product is anything created, used or modified by a process. Examples for work products are documents, models and source code. A work product can be associated with a responsible role.

The RUP defines a workflow as follows [10]:

> A mere enumeration of all workers, activities, and artifacts does not quite constitute a process. We need a way to describe meaningful sequences of activities that produce some valuable result and to show interactions between workers. A workflow is a sequence of activities that produces a result of observable value. In UML terms, a workflow can be expressed as a sequence diagram, a collaboration diagram, or an activity diagram.

As described in the next section, the ReDSeeDS Methodology uses activity diagrams to put all the work definitions and the related activities in a meaningful order. Within SPEM a process Package helps organizing these diagrams. A definition for a Process Package is provided [12] as well but in this case the definition is more technical. In simple words a Process Package provides a way to group process elements.

2.5 An Architecture-centric Process

The central role that architecture plays in the RUP helps to give an overview of architectural characteristics before describing the ReDSeeDS Architectural Design Phase. A large part of the RUP and the ReDSeeDS Methodology focuses on modeling because visual models help to understand the underlying problem and the developed software solution. To put it differently, models represent the system to be built or in some cases the so-called architecture of the system.

According to [13] architecture can be described as follows:

> The organizational structure and associated behavior of a system. An architecture can be recursively decomposed into parts that interact through interfaces, relationships that connect parts, and constraints for assembling parts. Parts that interact through interfaces include classes, components and subsystems.

Another definition of architecture can be found in [10]:

> Architecture is what remains when you cannot take away any more things and still understand the system and explain how it works.

According to [10] Architecture encompasses significant decisions about the following:

- The organization of a software system
- The selection of structural elements and their interfaces by which the system is composed, together with their behavior as specified in the collaboration among these elements
- The composition of these elements into progressively larger subsystems
- The architectural style that guides this organization, these elements and their interfaces, their collaborations, and their composition

Moreover, many different stakeholders are interested in architecture and they all want a personal understandable representation of it. Therefore the RUP suggests the 4+1 View Model of Architecture as depicted in Figure 2.5.

Regarding the architectural design process, the Rational Unified Process defines two primary artifacts related to architecture [10]:

- The software architecture description (SAD), which describes the architectural views relevant to the project

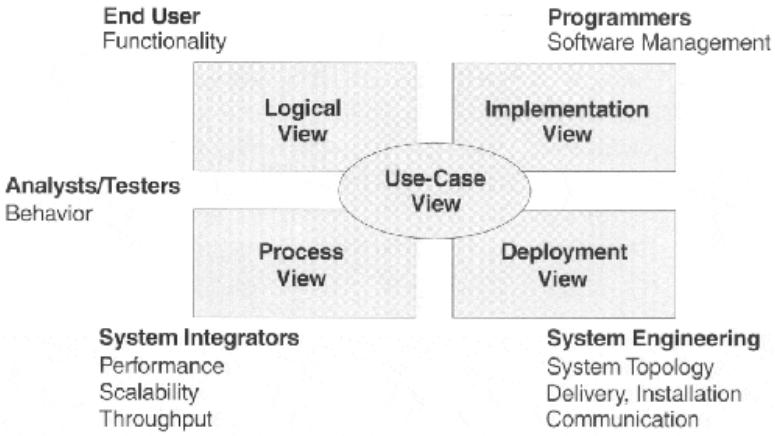

Figure 2.5: The 4+1 view model of architecture from [10]

- The architectural prototype, which serves to validate the architecture and serves as the baseline for the rest of the development

The ReDSeeDS Architectural Design Phase "only" defines one work product as outcome, namely the Architectural Design Document, which contains the detailed description of the system architecture. The development of a prototype is not mandatory at this point.

2.6 The Analysis and Design Workflow

According to [10], the main purpose of the Analysis and Design Workflow is the translation of the requirements into a specification that describes the implementation of the system. At this point the ReDSeeDS Methodology distinguishes between the Architectural Design Phase and the Detailed Design Phase. Within the Analysis and Design Workflow, RUP distinguishes between Analysis and Design. The former puts uses cases in the center of attraction in order to analyse the requirements, whereas the latter takes the result of this analysis to design the solution. Moreover, [10] tries to answer the question how detailed the system has to be designed during the workflow. At this point the simple answer is "detailed enough in order to implement the system unambiguously".

The Rational Unified Process expresses the analysis and design workflow in terms of workers, artifacts and activities. A detailed description of the ReDSeeDS Architectural Design Phase regarding different processes can be found in Chapter 3 but at this point the most obvious similarities and differences are shown. Figure 2.6 compares the two phases without ReDSeeDS-specific activities.

On the overview level, only the workflows or work definitions are comparable because the RUP depiction does not show the workers or the artifacts on this level. When selecting a subworkflow of the RUP description like Define a Candidate Architecture, the different structuring immediately becomes obvious. The RUP workflow combines the ReDSeeDS Architectural Design- and Detailed Design Phase and, therefore, splits the first analyzing tasks between two

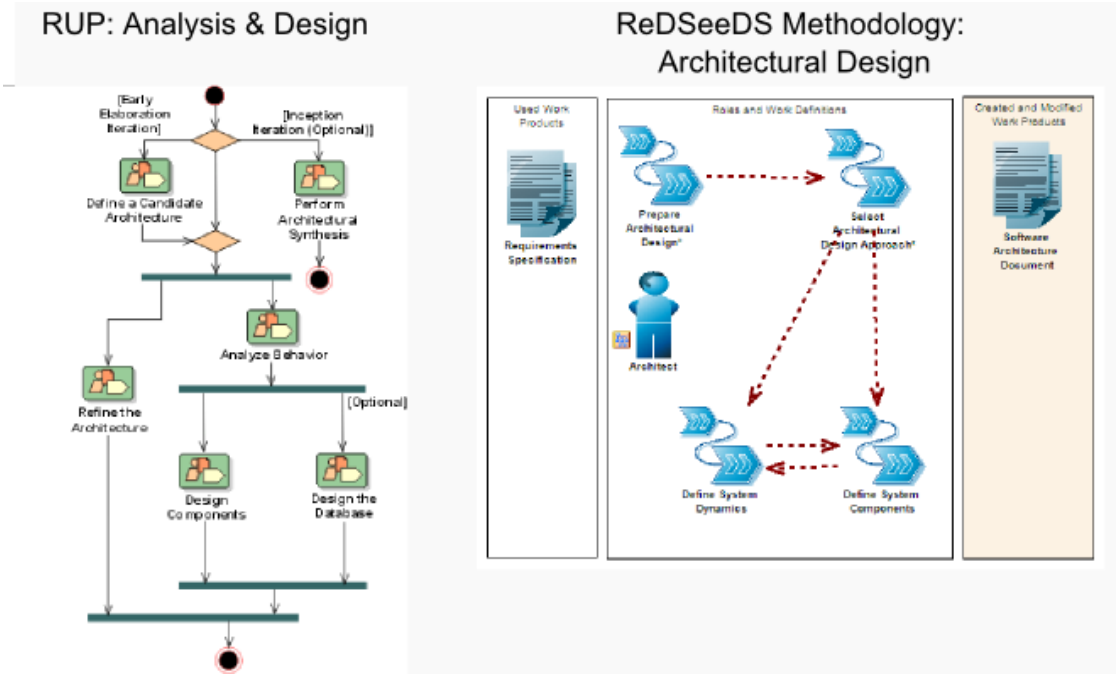

Figure 2.6: Comparision Architectural Design RUP and ReDSeeDS, taken from [10] and [22]

workers namely the Software Architect and the Designer. The former performs an Architectural Analysis and the latter a Use Case-Analysis. The Prepare Architectural Design work definition plans a single role for the requirements analysis and the creation of the architectural vision, namely the ReDSeeDS Architect. The RUP workflows Refine Architecture, Analyze Behavior and Design components handle the same architectural key principles like the RedSeeDS work definitions Select Architectural Design Approach, Define System Dynamics, Define System Components, namely the definition of the single components of the desired solution and their interactions. In general, the RUP description is on a more detailed level because the ReDSeeDS Methodology focuses on ReDSeeDS-specific principles. For instance, the RUP points out single artifacts like Project Specific Guidelines and the ReDSeeDS description covers the entire outcome in the Software Architecture Document. The reason for that is simple. As pointed out in Chapter 3, it is possible to introduce the ReDSeeDS framework in any company even if a software development process has not been established yet. Therefore, the RedSeeDS Methodology also provides guidance for not applying ReDSeeDS-specific activities at all. The guidance of the ReDSeeDS Slim process concentrates to pave the way for more complex ReDSeeDS processes and focuses on objects which are used in the more complex processes. For instance, more complex ReDSeeDS processes will add models which can be automatically transformed like a Component Model or an Interface Definition Model.

The only other eye-catching difference is the fact that RUP clearly addresses database design. Once again, the ReDSeeDS Methodology aims at a higher level of abstraction and does not cover specifically database design yet.

CHAPTER 3

RedSeeDS Methodology

3.1 Introduction to the ReDSeeDS Project

The "General Public Overview" of the official project website [16] describes the addressed problem, namely that developers often have to write software programs from scratch, regardless if something similar has already be done. In order to enable reuse, an automated way of searching a central software repository is required. Moreover, it should be possible to extract existing software "artifacts" from this repository for use in new systems. Therefore, the EU-funded ReDSeeDS project was established. Within this project a requirements specification language, which allows developers to use a single user interface to frame their queries, has been defined. In addition to this, query technologies which automate the query process have been developed and finally a repository technology which searches the repository and provides answers to queries has been designed. The Objectives Overview section of the official website [16] depicts the main objective of the project as follows:

> The main objective of the project is to create an open framework consisting of a scenario-driven development method (precise specification language and process for the "how-to"), a repository for reuse and tool support throughout. The basic reuse approach will be case-based, where a reusable case is a complete set of closely linked (through mappings or transformations) software development technical artifacts (models and code), leading from the initial user's needs to the resulting executable application. A new problem description in the form of a requirements model can be matched with previous requirements models. The solution information (models and code) of the most similar problem can then be taken for reuse and adapted to even only partially developed requirements. Unlike for other approaches, the effort associated with preparing reusable solutions with this framework is kept to the minimum.

An overview of the ReDSeeDS framework is given in Figure 3.1.

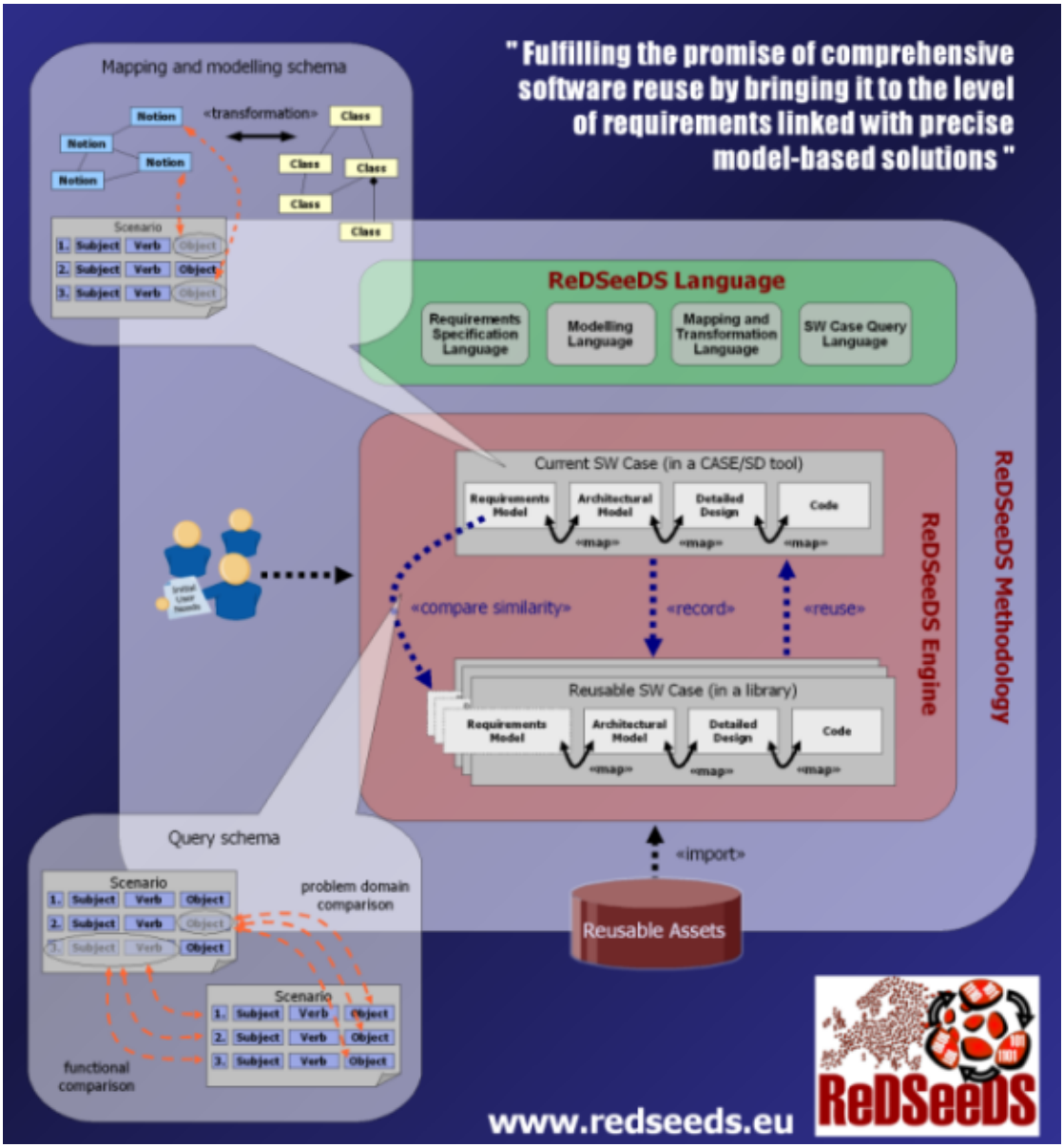

Figure 3.1: Overview of the ReDSeeDS framework from [16]

3.2 Introduction to the ReDSeeDS Methodology

Applying the ReDSeeDS approach in a software development project is not a trivial task and therefore systematic guidance is needed. The guidance comes in form of a development methodology and is described in [22]. The following chapter describes the basic approaches mentioned in [22]. The ReDSeeDS processes are modeled in detail and these models can be exported to a hypertext representation. Moreover, the ReDSeeDS Methodology addresses two different

groups of readers, namely Methodology Users and Methodology Managers. The former apply the methodology in a specific project, the latter introduce the ReDSeeDS Methodology in an organization. As stated in [22] both user groups are intended to use the hypertext representation of the processes in addition to the deliverable. For the Methodology Users the hypertext contains the detailed descriptions of software development activities. For the Methodology Managers the hypertext representation shows in detail which activities are affected when a new ReDSeeDS process is introduced into an organization.

In general the ReDSeeDs software development methodology consists of the following three components:

1. Notation
2. Processes
3. Techniques

Notation

The so-called Software Case Language (SCL) is the used notation within the ReDSeeDS Methodology. This language is used to model all artifacts in a software case. These artifacts can be various objects, for instance requirements, architectural models, source code and even model transformations between those. Therefore the SCL is powerful enough to represent all these artifacts and relationships. More precisely, the SCL consists of several sub-languages:

- The Requirements Specification Language (RSL), which allows the specification of requirements on different levels of abstractions.
- The Software Development Specification Language (SDSL), which is a selected subset of UML for describing the architecture and detailed design of a software case.
- The Model Transformation Language MOLA, which enables transformations between models in RSL, SDSL and code.

Processes

Another main component of the ReDSeeDS Methodology next to the notation are the processes. According to [22] the ReDSeeDS Methodology addresses three approaches which are based on each other. Therefore it is not compulsory to follow the whole functionality provided by the ReDSeeDS framework; on the contrary an organization is encouraged to adopt the approaches one after another. Therefore the RedSeeDS Methodology provides tailored guidance for every approach. A precise notation (SCL) is the centre of attraction in the first approach, the second approach introduces the concept of a RedSeeDS software case and enables reuse, and the third approach adds model-driven development [22]. Figure 3.2 depicts an overview of the ReDSeeDS processes in more detail.

The Slim Process provides a very general description of the software development activities and prepares the user for the ReDSeeDS-related activities. In addition, it enables organizations

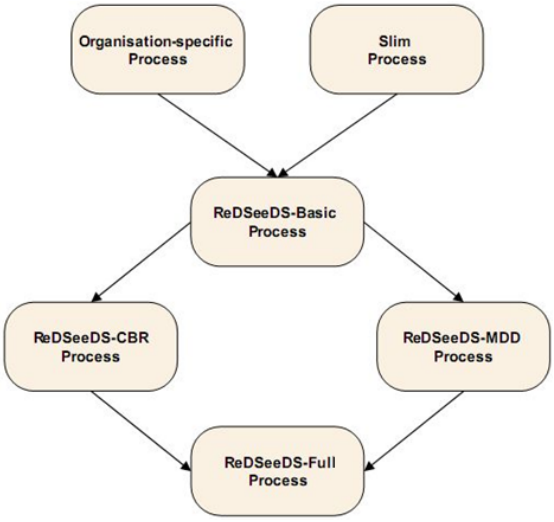

Figure 3.2: Overview of the ReDSeeDS processes taken from [22]

which have not used a software development process yet to bridge the gap between activities that are not influenced by the ReDSeeDS approach and ReDSeeDS-related activities. Therefore the ReDSeeDS processes can be used by companies which aim to build a new software development process from scratch and by companies which want to enhance their established software development methodologies by adopting the ReDSeeDS processes [22].

The ReDSeeDS-Basic Process adds ReDSeeDS-specific support and introduces the Requirements Specification Language (RSL) and the Software Development Specification Language (SDSL). Therefore general software development phases are substituted by specific ReDSeeDS phases, which are described in the next chapter. Furthermore ReDSeeDS-compliant work products are generated. In general the artifacts produced by the Requirements Engineering and the User Interface Specification phase are described in RSL and the artifacts generated by the Architectural Design- and Detailed Design phase are described in SDSL. A more detailed description for the ReDSeeDS-Basic Process can be found in [22].

The next step after introducing a ReDSeeDS-specific notation is the enabling of case-based reuse which is described in the ReDSeeDS-CBR Process ("CBR" is the abbreviation of case-based reuse). In a nutshell, this phase enables reuse of all artifacts in all phases. Therefore, elements of former created Requirements Models and similar software cases can be retrieved as blue prints for the Architectural and Detailed Design phases. Again a more detailed insight is provided by the official deliverable addressing this topic [22].

Finally the ReDSeeDS-MDD Process uses all the previosly mentioned three sub-languages of SCL. So, transformations enable the creation of the specific software development artifacts. For instance, requirements can be transformed to design models and the models can be finally generated into code skeletons. The process is described in detail in [22].

Techniques

In general, the techniques presented in the ReDSeeDS Methodology are descriptions of individual activities. These activities are performed within the processes by specific roles (people) and operate on specific artifacts (work products, documents, models) [22].

3.3 Presentation Options

Software Process Engineering Metamodel (SPEM)

As mentioned in Section 3.2, the ReDSeeDS Methodology consists of several related processes (e.g. ReDSeeDS-Basic Process, ReDSeeDS-CBR Process) and therefore the model representation has to support the consistent and effective maintenance of related processes. According to [12] SPEM provides mechanisms for process engineers to manage a set of related processes. As a consequence, the ReDSeeDS processes are presented using the Software Process Engineering Metamodel (SPEM).

SPEM was founded by the Object Management Group (OMG), a consortium for computer industry standards. The OMG is responsible for well-known modeling specifications like UML and for Middleware Specifications like CORBA/IIOP. A process engineering meta-model like SPEM provides the necessary concepts for modeling, documenting and exchanging methods and processes. Moreover, SPEM provides a clear separation between the method content and the application of the method content in a specific development process. For instance, the method content "Prepare Architectural Design with SDSL" can be defined once and used in multiple processes like the ReDSeeDS-CBR Process or the ReDSeeDS-MDD Process.

According to [22] the core idea of SPEM can be described as follows:

> The core idea of SPEM is that a software development process is a collaboration between active entities called process roles that perform operations called activities on concrete entities called workproducts.

This core idea is illustrated in Figure 3.3:

Enterprise Architect

Not only a mighty meta-model like SPEM was tremendously important for the creation of the ReDSeeDS Methodology but also a tool which enables Europe-wide cooperation during the development process. In the end, Enterprise Architect from Sparx Systems was chosen as the modeling tool. The Reviewer's Guide to Enterprise Architect [18] describes their tool as "a visual platform for designing and constructing software systems, for business process modeling, and for more generalized modeling purposes." Moreover, Enterprise Architect is based on the UML 2.3 specification. The following subsections provide an overview of the most important topics regarding the development of the ReDSeeDS Methodology with this tool.

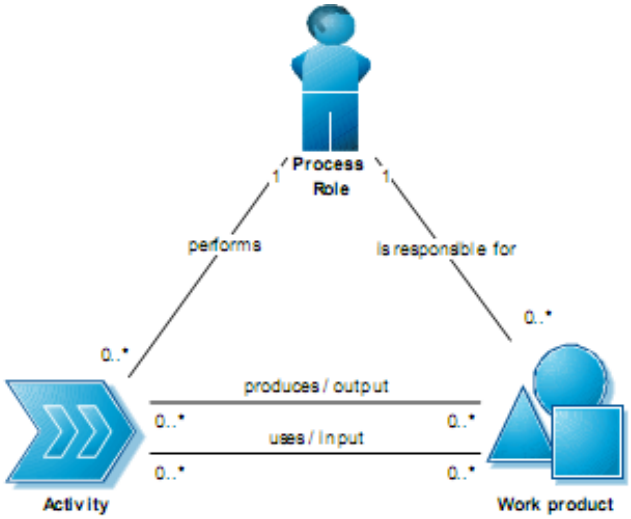

Figure 3.3: The core idea of SPEM according to [22]

Structuring the Visual Presentation

As stated in [5], every concept and the corresponding diagram including its elements should reside in an extra package. The arrangement in packages becomes even more important when several people are working on the same model. The ReDSeeDS Methodology is arranged in several packages. The first package layer represents the different processes. Every process package contains a second package layer including the different phases. Last but not least there exists an extra package which contains general and introductory information. The structuring is illustrated in Figure 3.4.

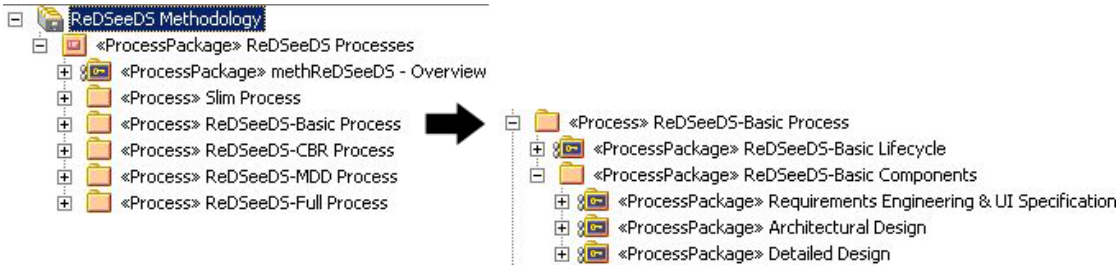

Figure 3.4: Structuring in Enterprise Architect

Enabling SPEM in Enterprise Architect

There exists a UML Profile for SPEM for Enterprise Architect which can be downloaded from [20]. After downloading and extracting the archive, the XML profile file can be easily imported as described in Figure 3.5.

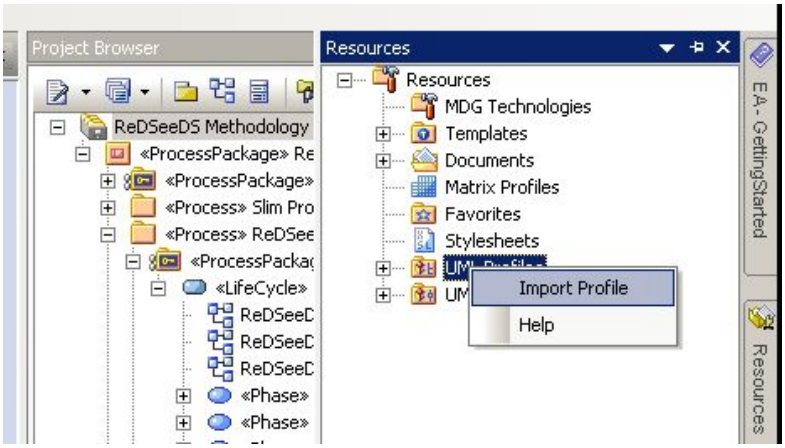

Figure 3.5: Importing the SPEM Profile into Enterprise Architect

After the import, the SPEM model elements are available in the Toolbox.

Version Control with Enterprise Architect

Due to the fact that the ReDSeeDS Project was a European-wide project several people have been working on the ReDSeeDS Methodology at the same time. As stated in the Version Control Best Practices for Enterprise Architect White Paper [19], Enterprise Architect applies version control to packages which are the primary organizational construct for UML models. According to [19] Enterprise Architect supports two primary ways of version controlling packages:

1. Model Baselines
2. Version Control Integration

The first approach stores point-in-time snapshots of a Package in the model repository itself, whereas the second approach supports integration with third-party version control systems including Subversion, CVS, Microsoft's TFS and SCC compatible tools, such as Accurev and Visual Source Safe. The second approach is preferable when not only roll-back purposes are needed but the full collaboration functionality like checkout mechanisms. For the development of the ReDSeeDS Methodology, Subversion has been chosen as the version control system. In this case it is important to mention that Enterprise Architect must use the Subversion command line client to communicate with the Subversion server (other clients such as TortoiseSVN do not work in this case). After installing the Subversion command line client, the Enterprise Architect Version Control Settings have to be configured. Figure 3.6 depicts a shared working Enterprise Architect environment.

Traceability

With the help of version control, the development of the ReDSeeDS Methodology has been traceable among multiple developers. In addition, it is important to trace model elements through

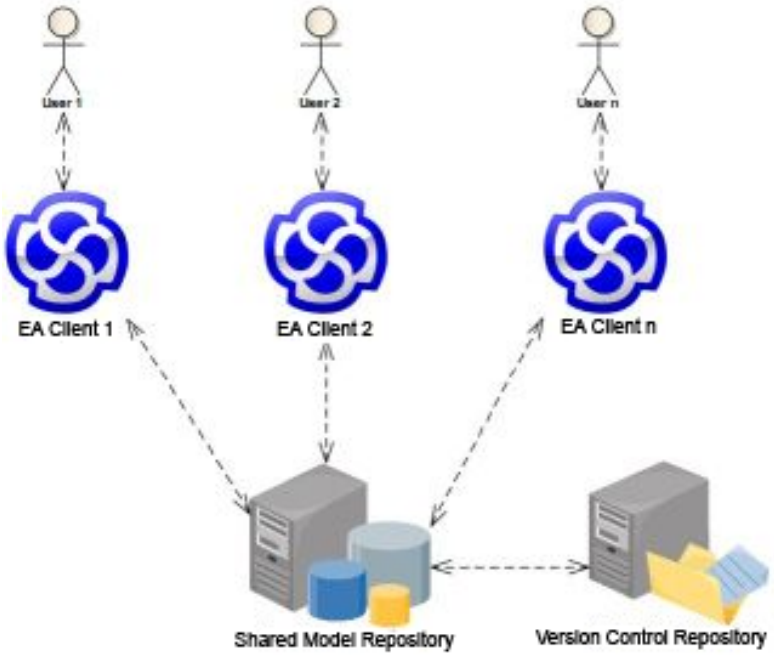

Figure 3.6: Shared working environment with Enterprise Architect taken from [19]

numerous models. Enterprise Architect provides mechanisms for this purpose as well. One feature worth mentioning is the "In Diagrams..." functionality which is available from the context menu in the project browser. For instance, a developer can quickly find out in which scenarios the activity "Decompose the system into components with SDSL" is used. Furthermore, the Element List and the Model Search functionality help to find specific elements. For instance, it is possible to create a list which displays the last modified activities. Moreover, with the help of Model Views it is possible to create user defined views. For instance, one could create a view for displaying only elements which are relevant for the Architectural Design Phase or use built-in search criteria like the "orphans criteria" to get a list of model elements which are not used in diagrams so far. Figure 3.7 depicts an example of a custom model view to list all SDSL relevant elements.

Built-in Document Editor and Linked Documents

With the help of Linked Documents it is possible to link an RTF document to any UML element in the model. Within the ReDSeeDS Methodology detailed guidance for activities is provided with the help of Linked Documents. In most cases this detailed guidance includes a clearly specified goal and specific steps to achieve this specified goal. A sample Linked Document is shown in Figure 3.8.

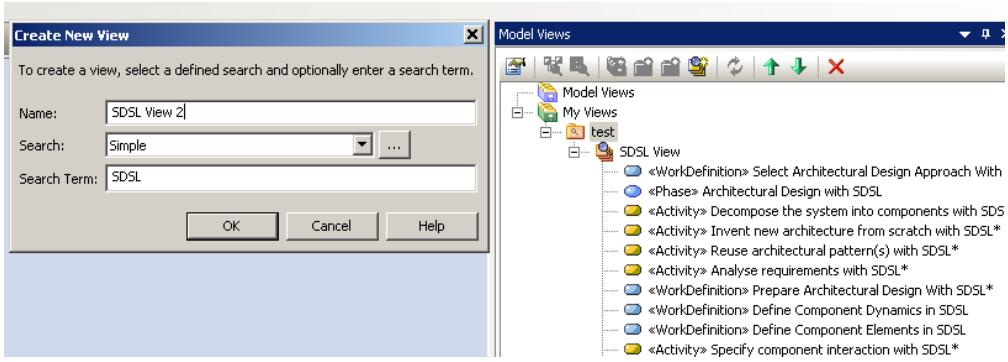

Figure 3.7: Sample custom model view in Enterprise Architect

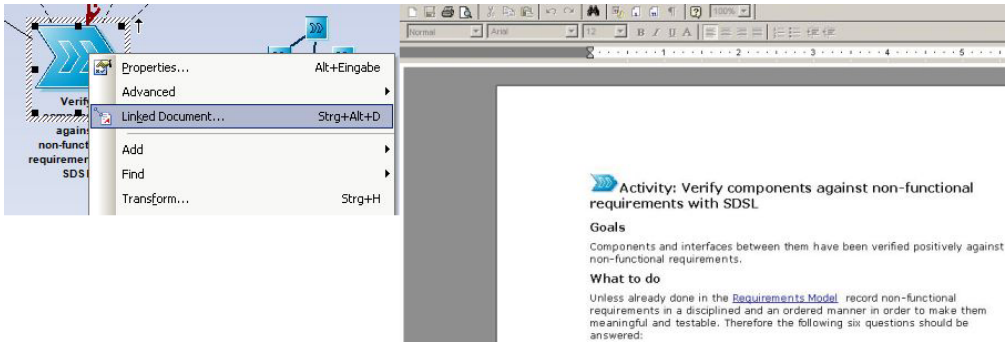

Figure 3.8: Linked Documents in Enterprise Architect

The HTML Report Writer

With the help of the HTML Report Writer, the ReDSeeDS Methodology developers have been able to export the entire model or a single branch of the model to HTML Web Pages. The HTML report provides a detailed model tree with hyper-linked elements which enable comfortable browsing. Furthermore, the HTML documentation is based on user-customizable HTML templates, so it is possible to generate tailored webpages. As depicted in Figure 3.9 the HTML Report Assistant is easily accessible from the context menu in the Project Browser.

3.4 Description of the Phases

The following six phases were defined during the creation of the ReDSeeDS Methodology:

1. Requirements Engineering
2. User Interface Specification
3. Architectural Design

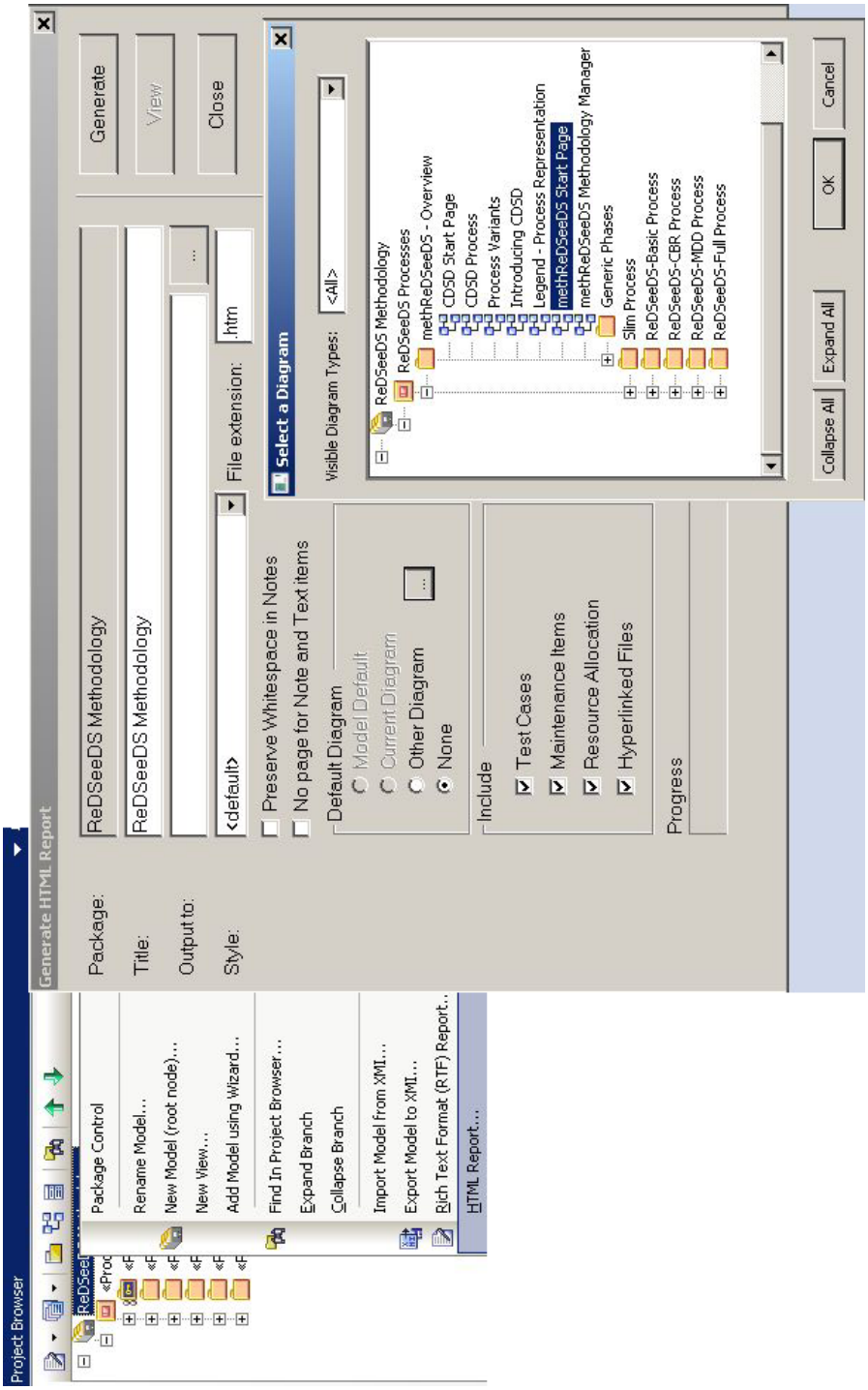

Figure 3.9: The HTML Report Writer in Enterprise Architect

4. Detailed Design

5. Implementation

6. Testing

In general, requirements are the basis for every project and formulated by the stakeholders in combination with a "Requirements Engineerer". Requirements provide the basis for all the following development work. Formulating requirements is challenging because the requirements have to be understood by all the people involved in the project and without conflicting each other. As a consequence, detailed guidance can be found in the literature, for instance [6] and [23]. The User Interface Specification Phase is strongly related to the Requirements Engineering Phase because the stakeholders' interaction with the new system is specified. For the Architectural Design Phase, which is the focus of this Master's thesis, the output of these two preceding phases is important. As mentioned above, the ReDSeeDS Methodology distinguishes between different ReDSeeDS processes. Table 3.1 gives an overview of the Work Products which are generated during the Requirements Engineering and the User Interface Specification Phase and are used by the Architectural Design Phase.

	Slim Process	**ReDSeeDS-Basic Process**	**ReDSeeDS-CBR Process**	**ReDSeeDS-MDD Process**	**ReDSeeDS-Full Process**
Requirements Specification	x	x	x	x	x
User Interface Specification	x	x	x	x	x
Requirements Model		x	x	x	x
User Interface Model		x	x	x	x

Table 3.1: ReDSeeDS Architectural Design Phase used Work Products overview

The created and modified work products of the Architectural Design Phase are listed in Table 3.2.

Generally spoken, the Detailed Design Phase uses the Software Architecture Document generated by the Architectural Design Phase to develop a detailed design of the system. The output of this phase results in the Component Design Document which serves as the basis for the Implementation Phase. During the Implementation Phase the software development on code level is started and the requirements described through models in the previous phases are turned into a software system to provide the desired solution. Regarding the ReDSeeDS Methodology the Implementation phase draws attention when model transformation comes into play and the static and dynamic detailed design models are generated into code skeletons and methods.

	Slim Process	ReDSeeDS-Basic Process	ReDSeeDS-CBR Process	ReDSeeDS-MDD Process	ReDSeeDS-Full Process
Software Architecture Document	x	x	x	x	x
Component Model		x	x	x	x
Interface Definition Model		x	x	x	x
System-level Interaction Model		x	x	x	x
Query			x		x
Slice		x	x		x
Retrieved Software Case		x	x		x
Requirements to Architecture Transformation Specification				x	x
Requirements to Architecture Transformation Rules				x	x

Table 3.2: ReDSeeDS Architecutral Design Phase created and modified Work Products overview

Development of the Architectural Design Phase

According to [4] the Architectural Design Phase defines the software in terms of major software components and interfaces and therefore can be referred as the "solution phase" of the software development life cycle. For sure the developed Architectural Design must cover all the requirements previously defined in the preceding Requirements Engineering phase. In [15] Architectural Design is described as the following:

> Architectural design represents the structure of data and program components that are required to build a computer-based system. It considers the architectural style that the system will take, the structure and properties of the components that constitute the system, and the interrelationships that occur among all architectural components of a system.

The description of the resulting work product of the Architectural Design Phase states [15]:

> An architecture model encompassing data architecture and program structure is created during architectural design. In addition, component properties and relation-

ships (interactions) are described.

According to [4] the Architectural Design Phase can be organized into the following topics:

1. Examination of the SRD
2. Construction of the physical model
    a) Decomposition of the software into components
    b) Implementation of non-functional requirements
    c) Design quality
    d) Prototyping
    e) Choosing a design approach
3. Specification of the architectural design
4. Integration test planning
5. Planning the detailed design phase
6. The architectural design phase review

The Software Architects and the Developers should be involved in the Requirements Engineering Phase, but at the beginning of the Architectural Design Phase an explicit confirmation that the requirements are understandable should be provided. Furthermore, the requirements are analyzed at the beginning of the ReDSeeDS Architectural Design Phase.

The physical model describes the design in implementation terminology including the definition of the software components, the hierarchy of control and the data flow. It is important to mention that developers should be able to work in parallel and therefore the exact definition of interfaces ensures that every part of the system will fit into the whole solution. Moreover, a logical model has to be constructed before the physical model can be developed. The logical model depicts the solution without any physical concerns like developing technologies or human and machine resources. The ReDseeDS methodology does not intend the explicit differentiation of a logical and a physical model but a concrete example would be the following scenario. For instance, the logical model envisions that parts of the architecture should be queried and reused from a common repository. The physical model would include technical descriptions of the particular query mechanisms and repository details.

The decomposition of the software into components is an extremely important part of the creation of the physical model. The decomposition should be done top-down which means that the top-level software components are defined first, followed be the lower-level components. The construction of the physical model should be an iterative process until the architectural design is detailed enough to continue with the Detailed Design Phase. This is basically achieved when all the concerned members can continue developing the solution on module level.

In addition to the decomposition, the non-functional requirements have to be taken into account. Non-functional requirements can cover for instance performance requirements, documentation requirements or security requirements. It is worth mentioning that not every non-functional requirement has a direct impact on the architecture and therefore some can be deferred to the Detailed Design Phase but nevertheless, each requirement has to be considered during the Architectural Design Phase. Regarding performance requirements, the performance of each component has to be analyzed and evaluated for each technology option. Security requirements can affect different components but for instance decisions about access to particular capabilities of the system have to be made during the Architectural Design Phase.

After the system is decomposed into components regarding non-functional requirements, it is time to design the single components and the communication among them. The designs should be adaptable and maintainable, software metrics can help to measure the simplicity and the complexity. Generally spoken, designs should be modular which means that the cohesion within each component should be maximized and the coupling between components minimized. Therefore cohesion measures the degree to which activities within a component are related to one another. For instance, there should be a component responsible for reusing a software case and these functions should not be implemented in a "miscellaneous routine component".

3.5 Architectural Design Phase within the Slim Process

During the Slim Process the final output of the Architectural Design Phase is generated without any ReDSeeDS-specific activities. Figure 3.10 describes the used work products, the roles, the work definitions and the created and modified work products. A single work definition contains several activities which are performed by roles in order to create work products. The red dotted arrows between work definitions (and also between their activities on a more detailed level) illustrate that work definitions highly affect each other and therefore cannot be performed in a strict sequence ignoring each other.

The used work products of the Slim Process differ from the ReDSeeDS-specific processes because models are not required at this point. The Requirements Specification document describes the software product the customer intends to purchase. Moreover, it covers the essential information provided by the Vision Scope document and additionally specifies all requirements. The Architect is the person, team, or organization responsible for the system architecture. The Architect is responsible for all work products created during the Architectural Design Phase. Furthermore, the Architect also assists the Requirements Engineer in checking quality of requirements and the Transformation Designer in specifying architecture to design transformation rules. Moreover, the Architect should use tools for the creation of architecture artifacts if appropriate.

The Software Architecture Document describes the architecture of the system in detail and contains the results of several activities: Create an Architectural Vision, Select Architectural Design Approach, Define System Component and Define System Dynamics.

It has to be consistent with: Vision and Scope, Software Requirements Specification and the Requirements Model (in all ReDSeeDS processes except for the Slim process).

Furthermore, the document contains:

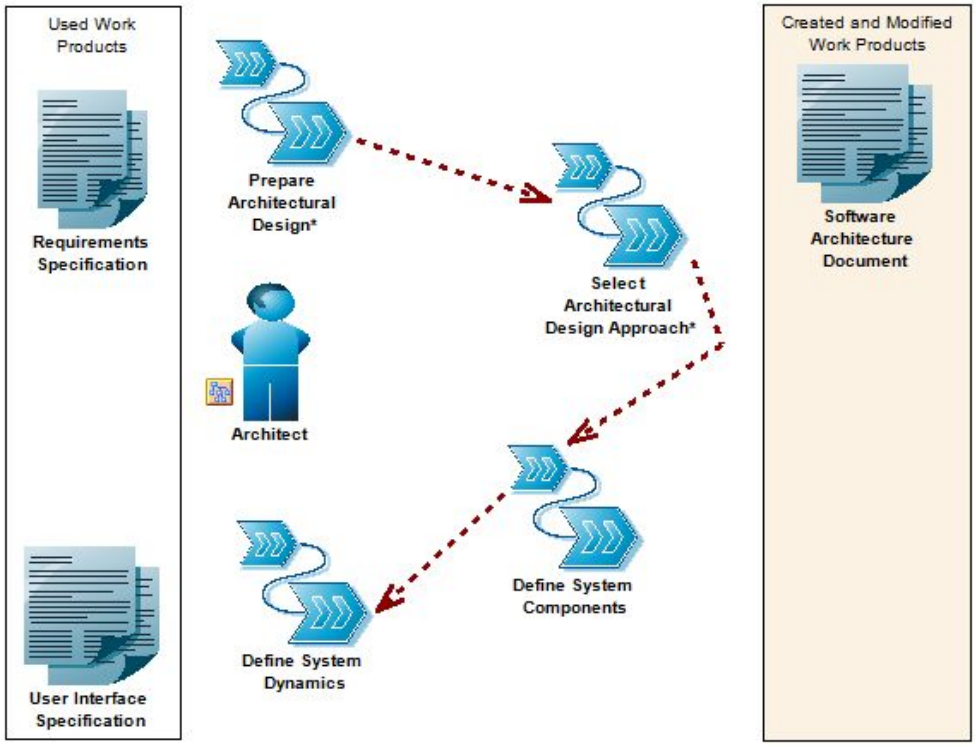

Figure 3.10: Overview of the Slim Processes taken from [22]

- Architectural Vision which includes architectural approach, overall system description, systems relations with the environment, rationale for choosing the architectural approach, etc.
- Specification of the software system structure. Structure contains the partition of the system into system components as well as connections between these components – connectors. It defines also the architectural style used in the architectural design (e.g., layered architecture, pipes and filters, repository) to assign roles of components and connectors in the architecture.
- Specification of the system behavior by defining the interactions between components.
- Physical structure and deployment of logical components onto it.

Additionally, the Software Architecture Document specifies how the architecture deals with non-functional requirements. The specification can contain models of structural description but also non-formal diagrams as well as natural language.

The first work definition, Prepare Architectural Design, prepares the Architectural Design Phase through analysis of the requirements and the initial creation of the Software Architecture Document. Figure 3.11 shows the activities in detail.

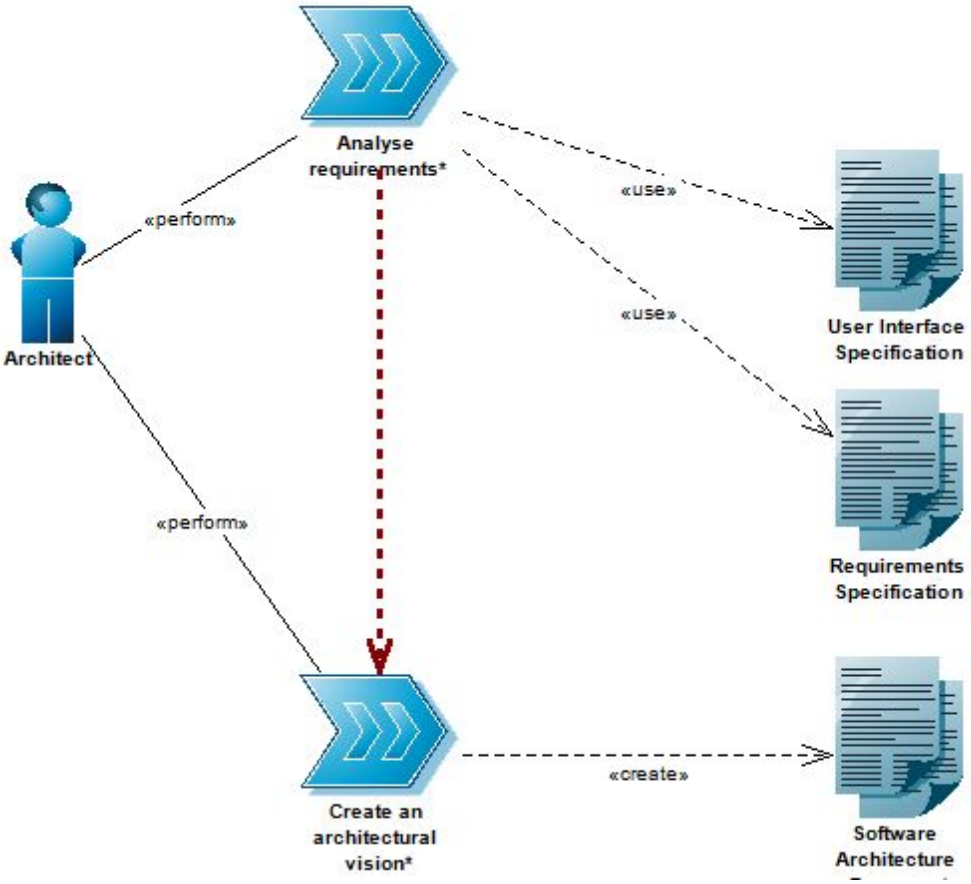

Figure 3.11: Slim Processes: Prepare Architectural Design taken from [22]

The main goal of the activity Analyse requirements is to understand the problem stated by the requirements. In order to create an architectural vision for a software solution, it is necessary to study the given problem. Therefore, the Architect is supposed to go through the Software Requirements Specification and the User Interface Specification systematically. Furthermore the Architect needs to create a mental model of the defined problem to make sure to understand clearly what has to be solved by the software, and what is outside the scope. Finally the initial version of the Software Architecture Document is created.

After the basic preparations for the Architectural Design Phase took place the Architect has to select an architectural design approach as shown in Figure 3.12.

Two approaches are possible, namely inventing a new architecture from scratch or to reuse one or more architectural patterns. For the former approach the Architect has to take into account the non-functional requirements and consider the following issues:

- Existing architectural styles (pipe and filter client-server, repository, etc.) including their tradeoffs (e.g., layered architecture facilitates security but may involve a performance

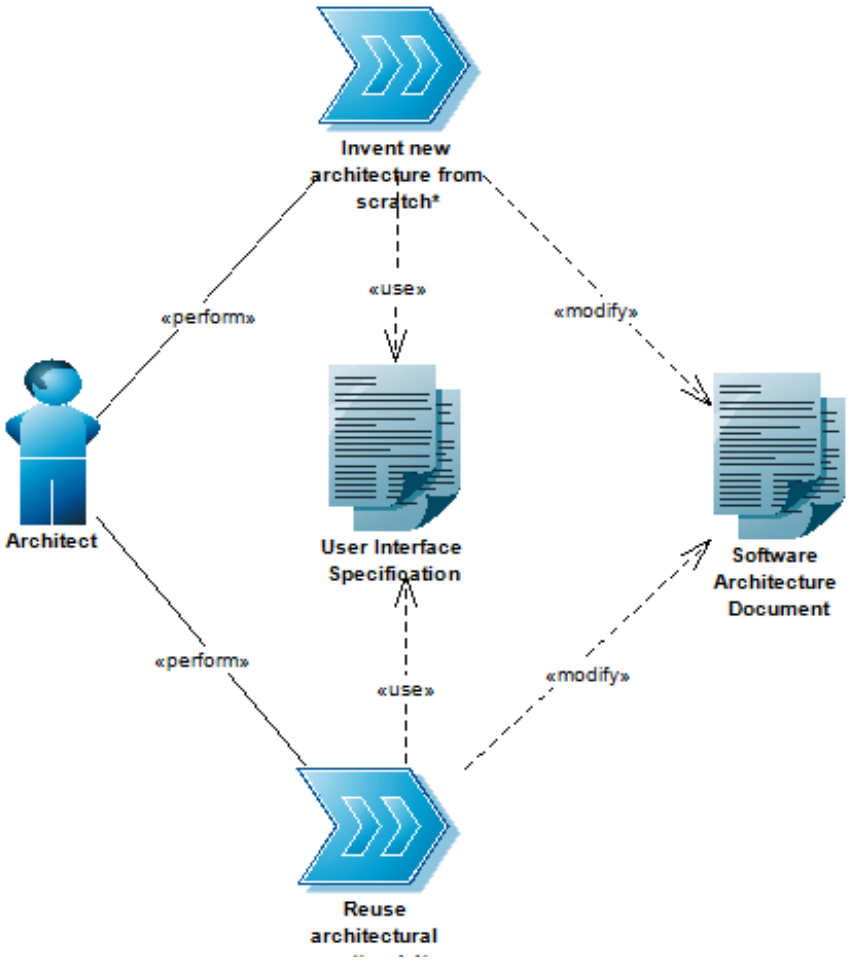

Figure 3.12: Slim Processes: Select Architectural Design Approach taken from [22]

overhead)

- Economic and organizational aspects (available “know-how”, etc.)
- Risks
- Impact of the architectural approach to development, testing and maintenance
- Logical structure and physical deployment

Furthermore, available frameworks and commercial, off-the-shelf (COTS) components have to be evaluated. Moreover, the Architect has to provide rationale for using or for not using specific artifacts. In addition, the Architect has to reason why a new architecture from scratch is preferred to existing architectural solutions. For the second approach, well-known architectural

patterns have to be reviewed and their applicability (based on their quality attributes) for the given problem as stated in the requirements has to be evaluated.

If more than one pattern is needed to solve the given problem, the Architect is supposed to link the chosen architectural patterns properly. If this is not possible, the candidate architectural patterns have to be merged to a new one by enhancing the resulting merged pattern or deleting unfeasible elements. Then the Architect has to evaluate the applicability of this new pattern with regard to the requirements. Both approaches are documented in the previously created Software Architecture Document.

As stated in [22] the main purpose of the Architectural Design phase is to engineer the software systems architecture regarding static and dynamic concerns. More precisely, the Architect has to define the system components, their interfaces and their interactions among them.

The activities related to the definition of system components are shown in Figure 3.13.

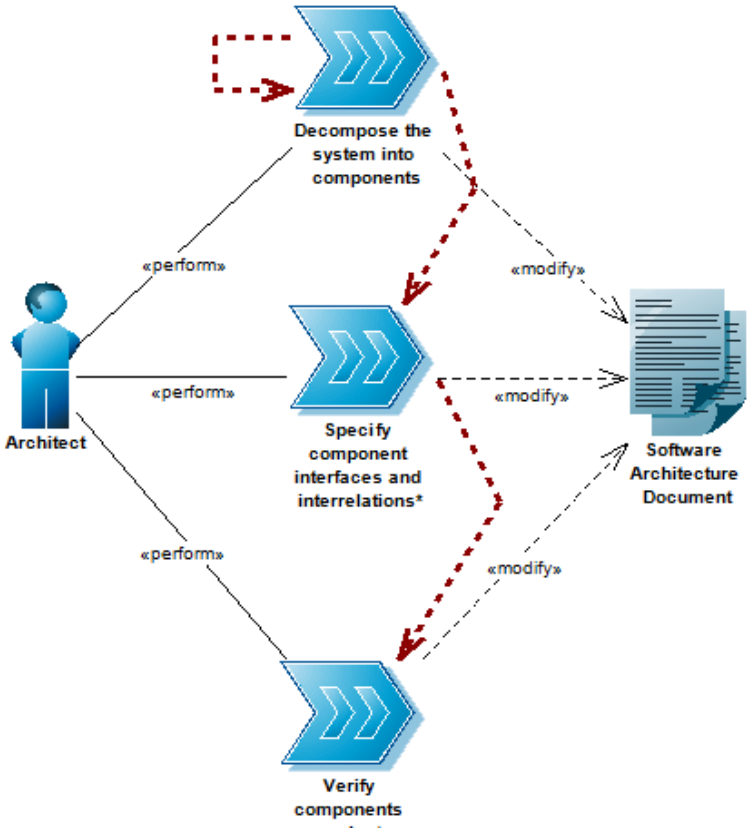

Figure 3.13: Slim Processes: Define System Components taken from [22]

In order to define the single components of a system, the selected architectural pattern has to be decomposed into single components. Therefore, a decomposition approach is required. This approach has to be selected by the Architect unless the selected architectural pattern predetermines the decomposition into components. With the help of an appropriate decomposition

approach, the Architect decomposes the system into components. This decomposition has to be done systematically and possibly recursively ("divide and conquer"). It is recommended to achieve minimal coupling (degree of interdependence between components) and maximum cohesion (degree of dependence within such a component). Moreover, for each crosscutting concern (like security or safety or other non-functional requirements) the Architect is supposed to define a single extra component. After defining the components, the Architect has to define the interfaces of each component. A component can have more than one interface and there is a distinction between provided and required interfaces. Therefore, the Architect has to consider which functionality is required from other components and which functionality of the component should be used by other components. Required interfaces are interfaces, which can be requested by another component. Provided interfaces provide the possibility for invoking functionalities of other components. Each required interface of the requesting component must be connected to a corresponding provided interface of the requested component. The Architect should try to define minimal interfaces for each functionality used by another component. The components and interfaces between them have to be verified positively against non-functional requirements. Therefore, the following six questions should be answered:

- Which entity is the source (actor) for stimulating the system?
- What is the request/intention of that source?
- What is the condition of the system when the request arrives?
- Which artifact of the system is affected?
- What is the response to this request?
- What is the measurable result for the response?

The Architect documents the specified components and their interactions in the Software Architecture Document.

3.6 Architectural Design Phase within the ReDSeeDS-Basic Process

In the ReDSeeDS-Basic Process the application of the Software Development Specification Language (SDSL), one of the ReDSeeDS SCL languages affects the used work products, the roles and work definitions and the created and modified work products as shown Figure 3.14.

The usage of SCL within the process enables the Architect to retrieve more detailed input from the Requirements Engineering and the User Interface specification compared to the Slim Process. Moreover, the Architect explicitly uses models in the Basic Process which enables the usage of the Requirements Model as a work product. The Requirements Model contains all requirements-related information of one development project in the Requirements Specification Language (RSL). More precisely, the Requirements Model involves the following parts:

- Domain Specification model

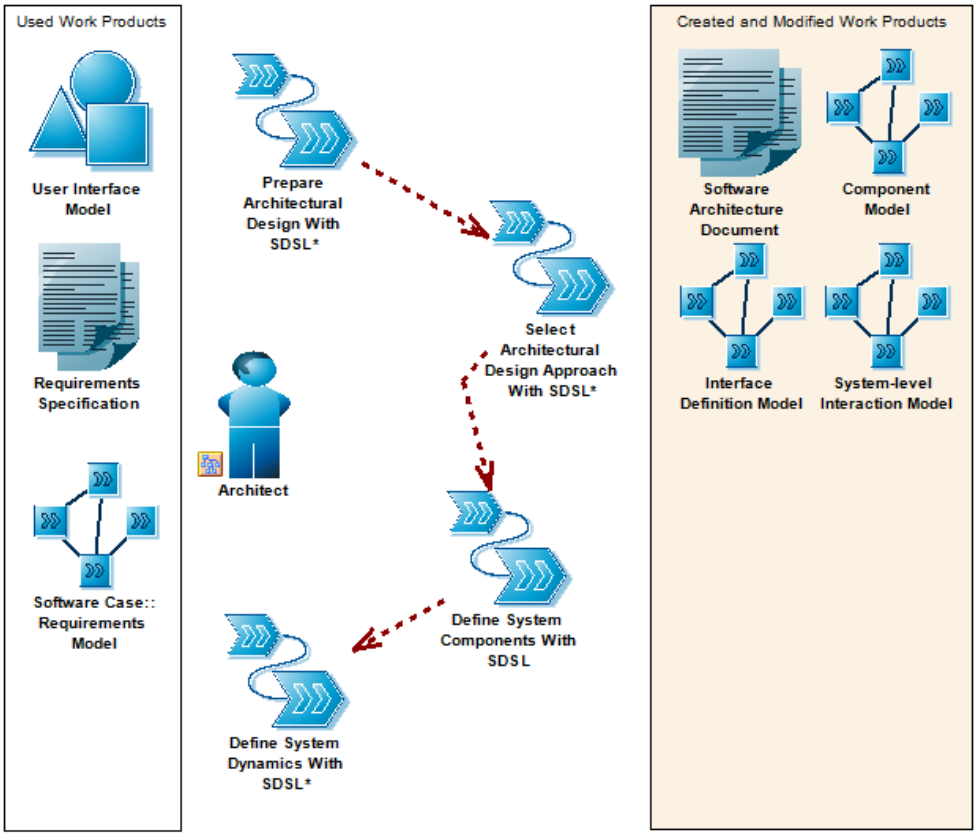

Figure 3.14: ReDSeeDS-Basic Processes: Overview taken from [22]

- Goals Specification model
- Requirements Specification model
- User Interface Specification

In the ReDSeeDS-Basic Process the Architect has to prepare the Architectural Design regarding SDSL as shown in Figure 3.15.

In order to create an architectural vision for a software solution, the Architect has to study the given problem. Therefore, the Architect has to alyse the Requirements Model, the User Interface Model and the User Requirements Specification systematically. Furthermore, the Architect has to create a mental model of the problem defined and focus on the most critical facets. The Architect has to understand clearly the difference between the requirements that can be solved through the software, and requirements which lie outside the solution scope.

Similar activities based on the Slim Process are required in order to select the appropriate architectural design approach but the Architect additionally needs to know SCL, in particular SDSL as shown in figure 3.16.

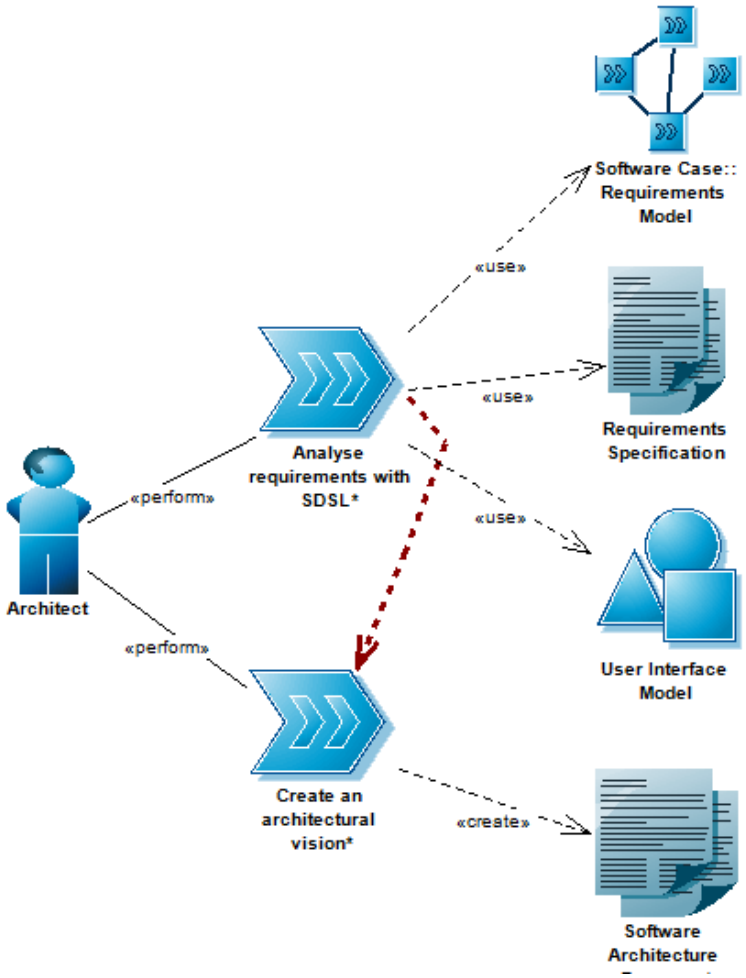

Figure 3.15: ReDSeeDS-Basic Processes: Prepare Architectural Design taken from [22]

If the Architect has to invent a new architecture from scratch, the notation in SDSL has to be considered. The invented architecture is based on the Software Requirements Model, the User Interface Specification and the architectural vision. Moreover, the Architect has to take into account especially the non-functional requirements (as defined in the Software Requirements Model). While defining a new architectural approach the Architect has to consider existing architectural styles including their tradeoffs analog to the Slim Process. Moreover, the reuse of architectural patterns and the linking between numerous chosen architectural patterns has to be considered analogously to the Slim Process as well.

Within the ReDSeeDS-Basic Process the definition of the system components requires additional modeling as shown in Figure 3.17.

Component diagrams offer one suitable way to document the component landscape. At this point the Architect has to keep in mind only to use the SDSL-compliant UML elements.

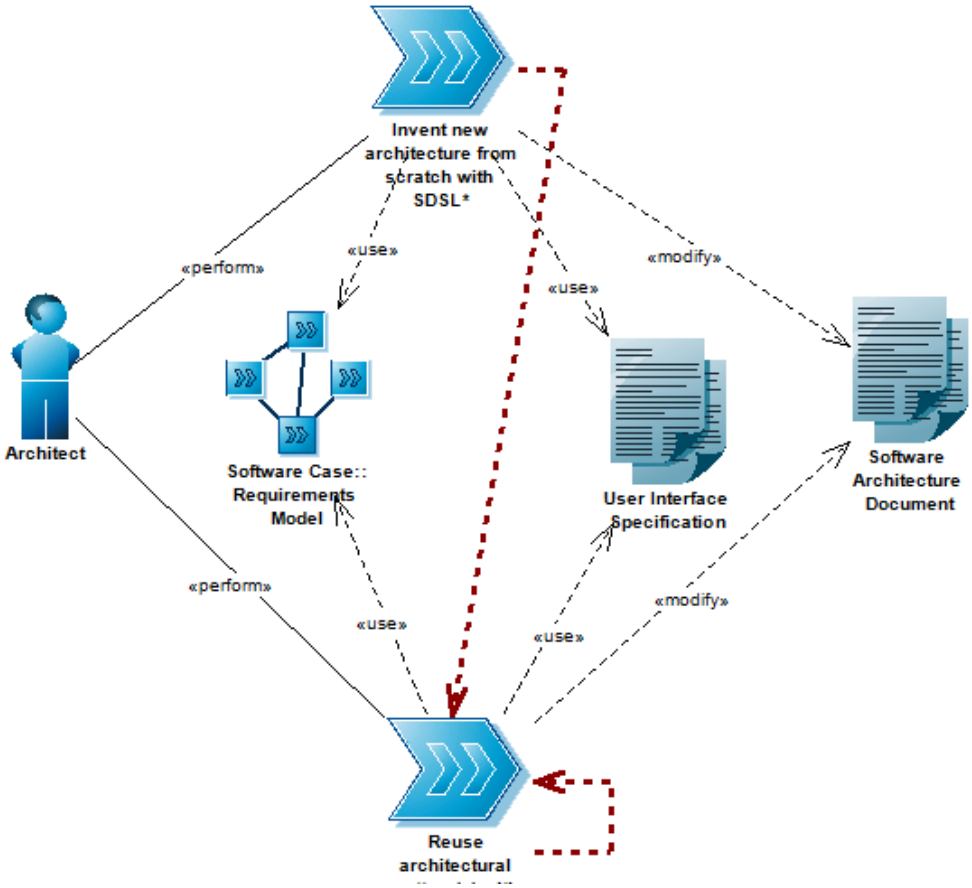

Figure 3.16: ReDSeeDS-Basic Processes: Select Architectural Design Approach taken from [22]

Suitable UML elements for a component diagram are:

- Component
- Interface
- Dependency
- Class
- Package

Defining the system dynamics regarding SDSL requires a slightly adopted approach compared to the SLIM Process as well and is described in Figure 3.18.

Kind of interaction diagrams offer a way to describe the dynamics of architecture. Compared to the Slim Process a System-level Interaction Model comes into play in the ReDSeeDS-Basic

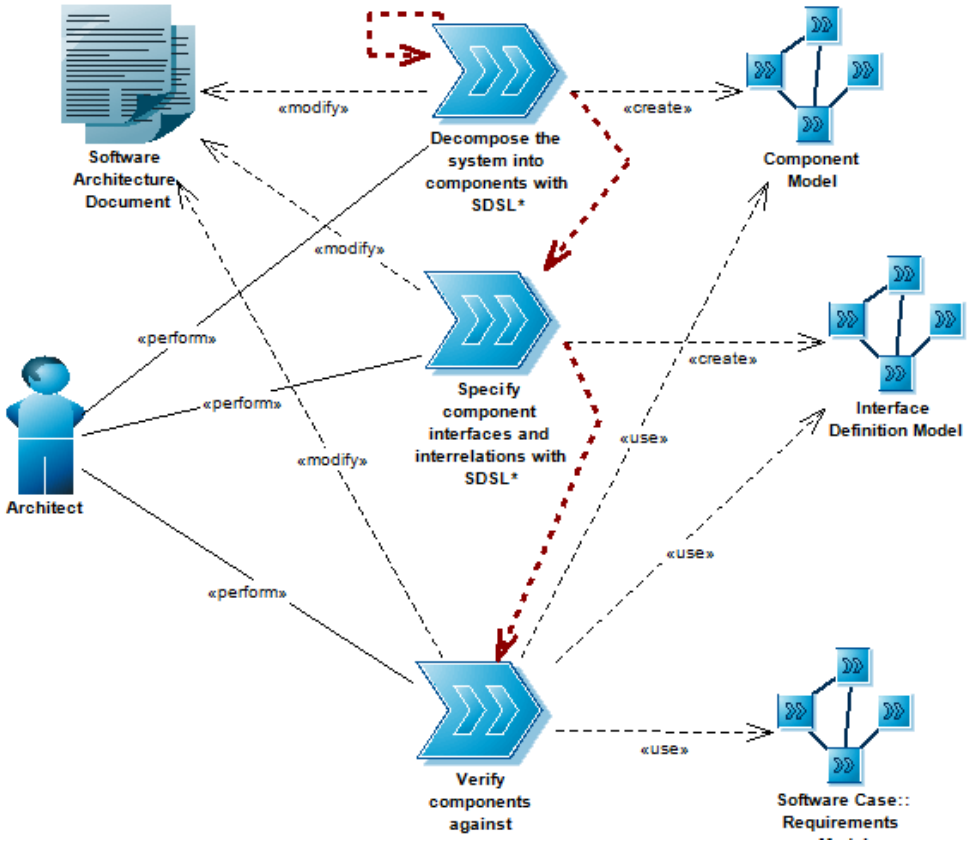

Figure 3.17: ReDSeeDS-Basic Processes: Define System Components taken from [22]

Process. This model shows the interactions of components defined in the Component Model via interfaces defined in the Interface Definition Model. Although interactions can be explained in plain text, sequence diagrams offer an easier to understand, graphical style to describe the interactions between components. Suitable elements are:

- Object
- Lifeline
- Message

The Architect is not limited to use these UML elements but only these will be considered for transformation. Again the output of the Architectural Design Phase is the Software Architecture Document which should be generated automatically as a report from the CASE tool used to create the Component Model, Interface Definition Model and the System-level Interaction Model. The report should be configured so that the sequence of chapters and sections reflect the division of the models into packages.

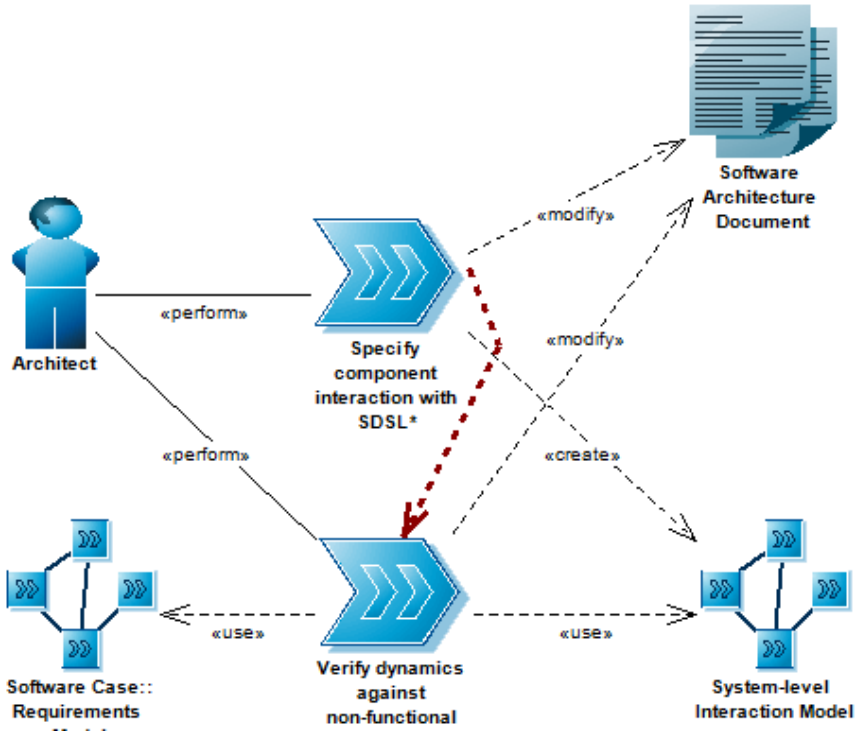

Figure 3.18: ReDSeeDS-Basic Processes: Define System Dynamics taken from [22]

3.7 Architectural Design Phase within the ReDSeeDS-CBR Process (Case-based Reuse)

This process deals with case-based reuse, which enables the Architect to query for similar software cases which have been saved into a global repository before. If the retrieved software case is similar enough to allow reusing it (or parts of it, in particular its architectural design) it can be used by the Architect for the development of the solution's architecture. The related activities and additional work definitions are illustrated in Figure 3.19.

The preparation of the Architectural Design Document involves the same activities as in the ReDSeeDS-Basic process but then the Architect has the possibility to create a query in order to retrieve a software case similar to the case described by the requirements as depicted in Figure 3.20. The specified query compromises a description of the current project's Requirements Model. Then the Architect uses this query to find similar Requirements Models in the Software Case Repository.

Depending on the given circumstances, the Architect can reuse an existing query or specify a complete new one. In order to use an existing query, the Architect has to check the queries that have been specified and saved earlier for one that meets the information needs. If the Architect is able to modify a suitable existing query, the Architect has to select the query and modify it by adding or deleting requirements (with their relations) and domain elements. With the help of the

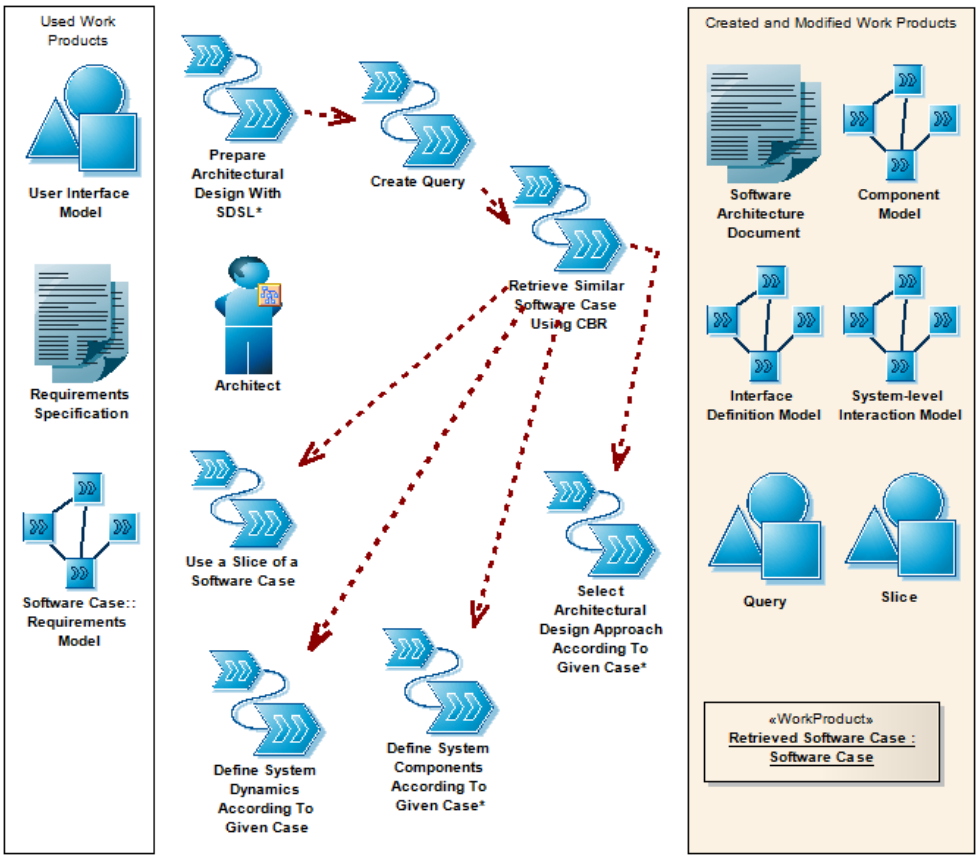

Figure 3.19: ReDSeeDS-CBR Process: Overview taken from [22]

created or modified query, the Architect is able to retrieve a similar software case.

The result of the activity Retrieve Similar Cases is a list of similar software cases. From this list the Architect can choose the most appropriate software case. In many scenarios not the complete software can be used for the given problem but only an excerpt as a slice. The ReDSeeDS Methodology provides guidance for using a slice of a software case as illustrated in Figure 3.21.

First the Architect has to select a slice of a retrieved software case. Therefore, the Architect has to select a specific slicing criterion and a particular slicing mode to import a slice marked for reuse into the current software case. Finally the imported slice is merged with the software case. The definition of the system components now has to consider the given case, as well as the definition of the system dynamics.

The interactions between the components are specified analogously to the given software case. Moreover, the verification against non-functional requirements has to consider the given software case as well at this point. The selection of the architectural design approach is done according to the given case as shown in Figure 3.22.

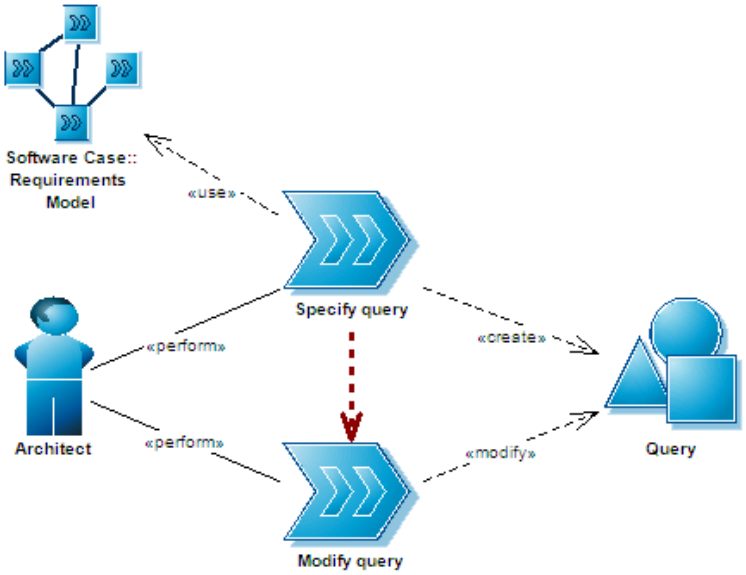

Figure 3.20: ReDSeeDS-CBR Process: Create Query taken from [22]

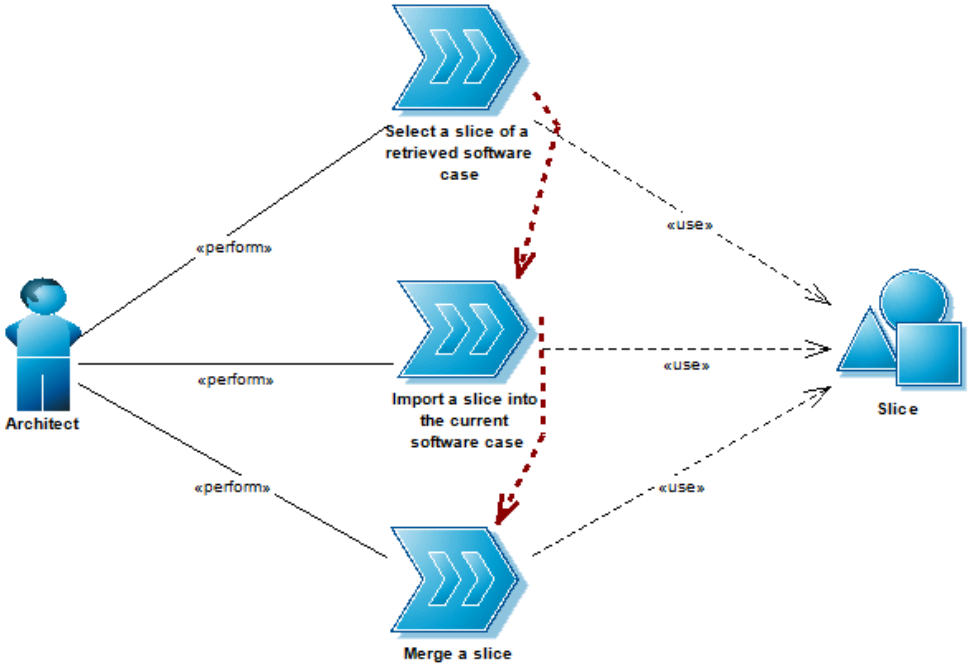

Figure 3.21: ReDSeeDS-CBR Process: Use a slice of a software case taken from [22]

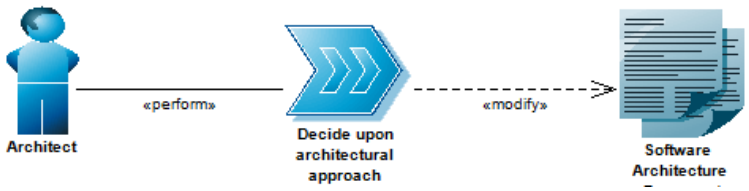

Figure 3.22: ReDSeeDS-CBR Process: Select Architectural Design Approach taken from [22]

3.8 Architectural Design Phase within the ReDSeeDS-MDD Process

As described in [22], this process takes model-based transformations from requirements to architecture into account. Therefore, a precise specification enabled by RSL and SDSL is required. This means that the ReDSeeDS-MDD Process can only be introduced after the ReDSeeDS-Basic Process has been introduced [22]. The ReDseeDS-MDD process also introduces an additional role, namely the Transformation Designer. He or she is responsible for formulating rules and specifying transformations between various models written using the ReDSeeDS languages (RSL, SDSL). Moreover, the Transformation Designer should have good knowledge of technologies used to develop the system. The Transformation Designer should be able to communicate with the Architect to determine transformation rules for the selected technologies.

The Transformation Designer should also have good programming and debugging skills to be able to write specifications (programs) in the MOLA language and should have experience in using MOLA tools. An overview of the Architectural Design Phase within ReDSeeDS-MDD Process is given in Figure 3.23.

The Work Definitions Prepare Architectural Design With SDSL and Select Architectural Design Approach With SDSL are performed as described in the ReDSeeDS-Basic process where SDSL is introduced, but this time the Architect is able to Create an Architectural Model Using MDA as depicted in Figure 3.24.

First the Architect has to consider the non-functional requirements of the software case and to decide which transformation profile should be used. A transformation profile can be described as a set of transformation rules written in MOLA. Moreover, the transformation profile has to be selected from a repository of transformation profiles. Furthermore the Architect is supposed to choose a security profile for systems where security is a dominant non-functional requirement and a real-time profile for systems where a guaranteed response time is a dominant non-functional requirement. Additionally, the Architect has to choose a performance profile for systems where performance is a dominant non- functional requirement. Within the selected profile the Architect has to choose which kind of applicable architectural style fits best the given requirements, for instance layered architecture, repository architecture or client-server architecture. After that the Architect has to modify the MOLA transformation rules of the transformation profile in the following ways:

- Add additional transformation rules to the chosen transformation profile for requirements

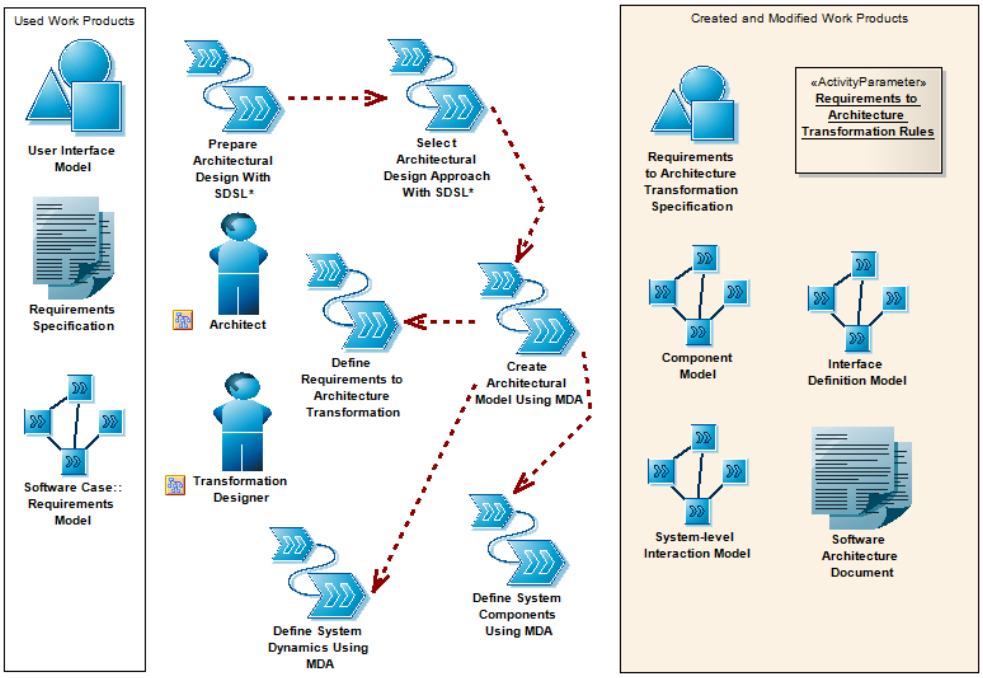

Figure 3.23: ReDSeeDS-MDD Process: Overview taken from [22]

which are not covered by this transformation profile per se.

- Modify transformation rules of the transformation profile to make them more specific for your given problem.
- Delete transformation rules of the transformation profile which have no corresponding requirements.

Furthermore the Architect has to ensure that the MOLA Transformation Rules cover the following aspects:

- Iterate over all use cases and define appropriate elements in the Architectural Design that realize these use cases (components).
- Iterate over all domain elements and define appropriate elements in the Architectural Design that realize these domain elements (components).
- Iterate over all scenarios and define appropriate elements in the Architectural Design that realize these scenarios (interfaces and interrelations).

Next the Architect is supposed to start the transformation process to execute the transformation profile for the given requirements model to generate a draft architectural model. Then the Architect is supposed to study the generated architectural model carefully for correctness and

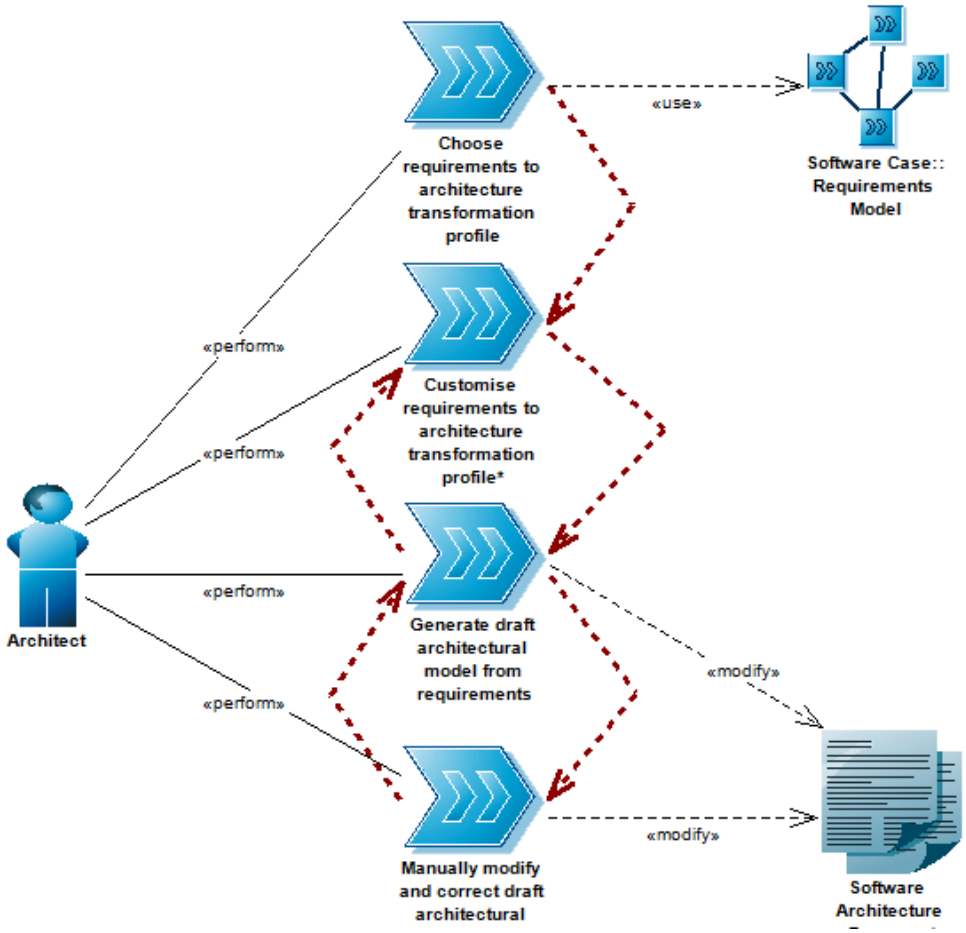

Figure 3.24: ReDSeeDS-MDD Process: Create Architectural Model

completeness, and manually modify it if necessary. The Creation of an Architectural Model Using MDA would not be possible without the Definition of Requirements to Architecture Transformation which is illustrated in Figure 3.25.

The Architect is supposed to specify rules for transforming the requirements models to architectural design models. The rules should precisely reflect the architectural design. Based on these rules, the Architect writes a transformation specification in MOLA. Moreover, the Architect has to create tests in order to make sure that the transformation specification written in MOLA performs automatic transformation according to the specified rules.

The model-driven creation process impacts the definition of the system components and the system dynamics. While decomposing the software system, the Architect has to keep MDA-specific considerations in mind (e.g.: how does this specific component or this interaction relate to the transformation profile).

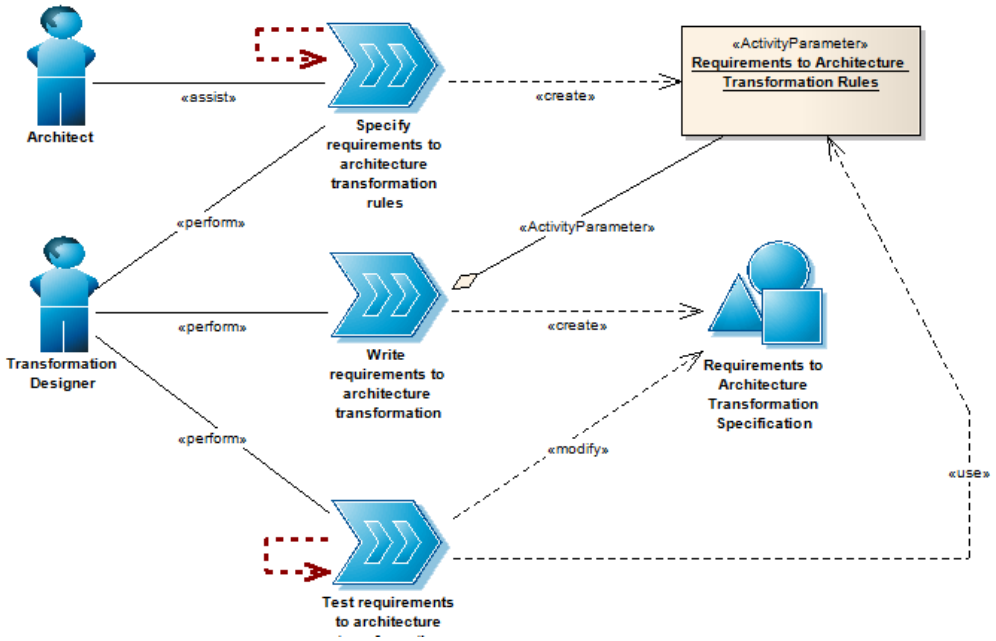

Figure 3.25: ReDSeeDS-MDD Process: Define Requirements to Architecture Transformation taken from [22]

3.9 Architectural Design Phase within the ReDSeeDS-Full Process

As the process name indicates, the ReDSeeDS-Full Process provides the "full" ReDSeeDS functionality, which means that all the ReDSeeDS languages and both, case-based reuse and model-driven development, are covered. The Architect prepares the architectural design and retrieves a similar software case with the help of the query functionality. The retrieved software case impacts all the other work definition including those which take model driven development aspects into account as shown in Figure 3.26.

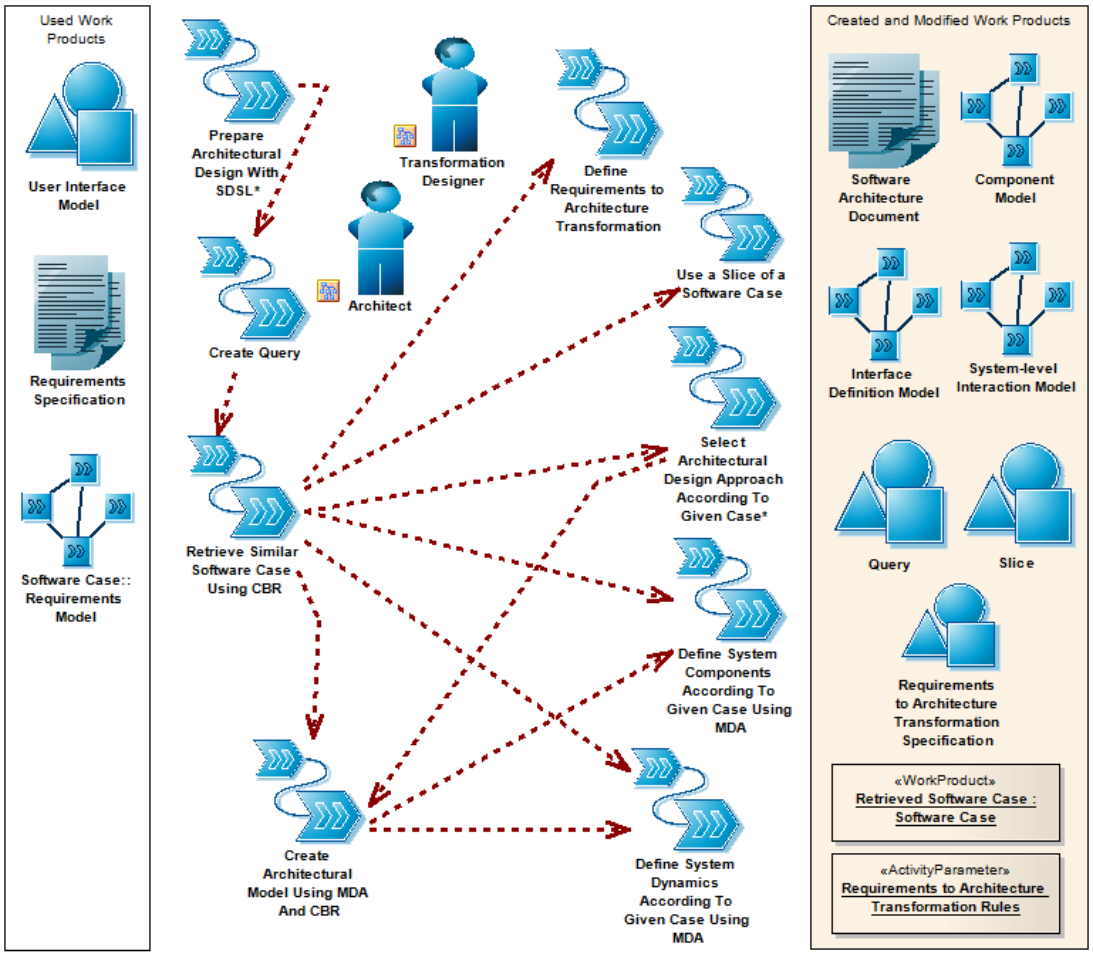

Figure 3.26: ReDSeeDS-Full Process: Overview taken from [22]

CHAPTER 4

Coupling Methodology – Tool

The ReDSeeDS Methodology bridges the gap between a method and the underlying tool. As described in [8] software developers often ask how to do what the method instructs them to do with a specific tool and, on the contrary, they do not know what to do with a specific tool without the describing method. In [8] the seperation between primary tasks and secondary tasks based on [14] is pointed out as follows:

> A primary task is a task that someone wants to (or is supposed to) carry out in the first place, and a secondary task is a task of mastering enough of a tool to accomplish a primary task.

As a consequence, it would be a tremendous improvement for the methodology user if the high-level methodology includes specific tool descriptions as well and even links into the particular case tool. Figure 4.1 gives an overview of the coupling process differentiating between a development perspective on the left side and a runtime perspective on the right side [8].

Regarding the runtime perspective, project partners implemented a prototypic tool (the so-called ReDSeeDS Engine) which supports the creation of requirements models with the requirements language RSL [20]. Additionally this tool contains a reuse engine for the retrieval and reuse of past software cases [8]. To sum things up, the prototype of the ReDSeeDS Engine provides a lot of functionality which can be invoked through secondary tasks. Figure 4.2 shows an example screenshot of the ReDSeeDS Prototype.

Due to the fact that the ReDSeeDS Engine is built upon the Eclipse Framework, guiding Eclipse Cheat Sheets are available. According to [7] Chat Sheets enable the user to view interactive tutorials from within the Eclipse Workbench, or to put it differently, they guide a developer through a series of complex tasks to achieve some overall goal. Furthermore some tasks can be performed automatically, such as creating a new ReDSeeDS project. Figure 4.3 shows an example of a Cheat Sheet. The Cheat Sheet is clearly structured in steps, each including a textual description.

In order to realize the method-tool coupling, first the primary tasks and the Cheat Sheets describing the secondary tasks have to be designed. The primary tasks are exposed in the software

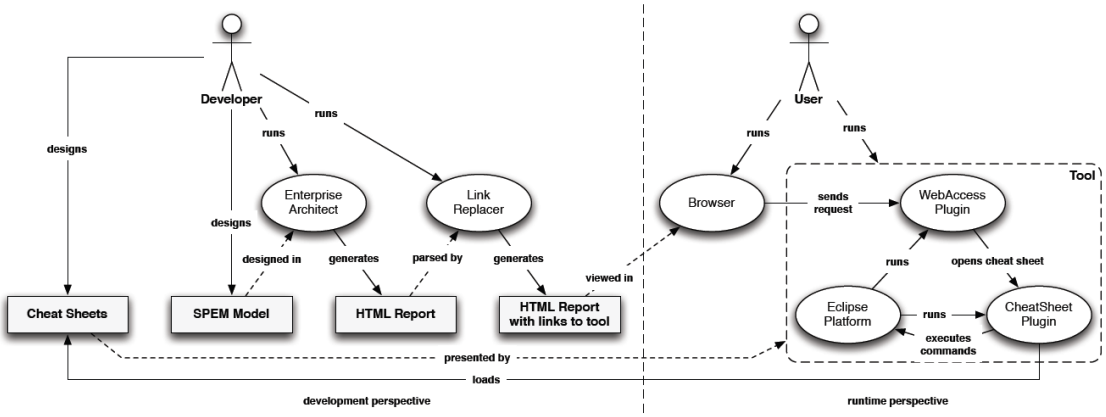

Figure 4.1: Coupling process taken from [8]

development method designed within Enterprise Architect in form of a SPEM model including Activities which are textually described by means of embedded Linked Documents in RTF (Rich Text Format). The secondary tasks presented by Cheat Sheets are specified in XML documents. The Eclipse platform processes these XML documents and presents them in form of a ToDo list to the user. Moreover, Cheat Sheets offer support for tight integration with the tool by enabling execution of commands via "click-to-perform links". Cheat Sheets can range from simple ones with just a sequence of steps to Composite Cheat Sheets with conditional execution of steps and redo support. All Cheat Sheets have to be packaged and added to the Eclipse platform in form of an additional plugin [8].

As described in [8] the actual coupling process consists of three steps:

1. Adding symbolic Cheat Sheet links to the activity descriptions within the method model.

2. Generation of the method's Web representation and conversion of the symbolic links to tool-specific links.

3. Connecting Cheat Sheets to actions within the tool

From a technical point of view, Step 3 has to be considered already during the implementation of the particular tool because the desired actions have to be exposed through interfaces in order to be invokable from Cheat Sheets. As shown in the XML representation the action attribute points to a specific class which has to implement the "IAction interface". Moreover, if the class for the action has to be aware of the state of the Cheat Sheet, the interface "ICheatSheetAction" can be optionally implemented as well. The official Eclipse Help section [3] provides more information on this topic.

Step 1 and Step 2 are pretty much straightforward. As described in Section 3.3 the textual information for Activities is realized with the help of Linked Documents. With the simple Editor provided by the Enterprise Architect, the developer can easily add a symbolic Cheat Sheet link in the specified format. At this point it is important to notice that every Cheat Sheet must be

uniquely identifiable by its ID. After the model has been enriched with symbolic Cheat Sheet links, the developer can create the HTML Web representation with the standard output possibilities offered by Enterprise Architect as described in Section 3.3. After the HTML output with the symbolic Cheat Sheet links has been generated, the CheatSheetLinkReplacer converts the symbolic links to tool-specific links. The conversion is achieved with the help of a Java application.

An implementation of a Web server is needed in order to handle the request from modified HTML pages and to display the guidance information. One key requirement for the method-tool-coupling approach is that the involved coupling mechanisms are transparent to the user and do not require any additional user interaction. Therefore, the Web server has to be automatically activated after the user opened the ReDSeeDS prototype.

The ReDSeeDS prototype is based on Eclipse, which uses OSGi as plugin-architecture. OSGi originally stands for Open Service Gateway Initiative in order to address the requirements of the embedded device market. This initiative was developed by multiple companies which used JAVA as their preferred language. The OSGi experts aimed at a system which was capable of being altered without interrupting the ongoing processes [17]. In order to achieve this goal, one of the OSGi architectural principles is that any framework that implements the OSGi standard provides an environment for the modularization of applications into smaller bundles. According to [17] each bundle is "a tightly-coupled, dynamically loadable collection of classes, jars, and configuration files that explicitly declare their external dependencies (if any)". To sum things up, due to the fact that the OSGi framework specification forms the basis of the Eclipse Runtime and the ReDSeeDS prototype is based on the Eclipse Runtime, it is possible to implement a server plugin whose functionality is automatically available after the startup process. The server plugin handles the request and extracts the identification number in order to display the desired Cheat Sheet.

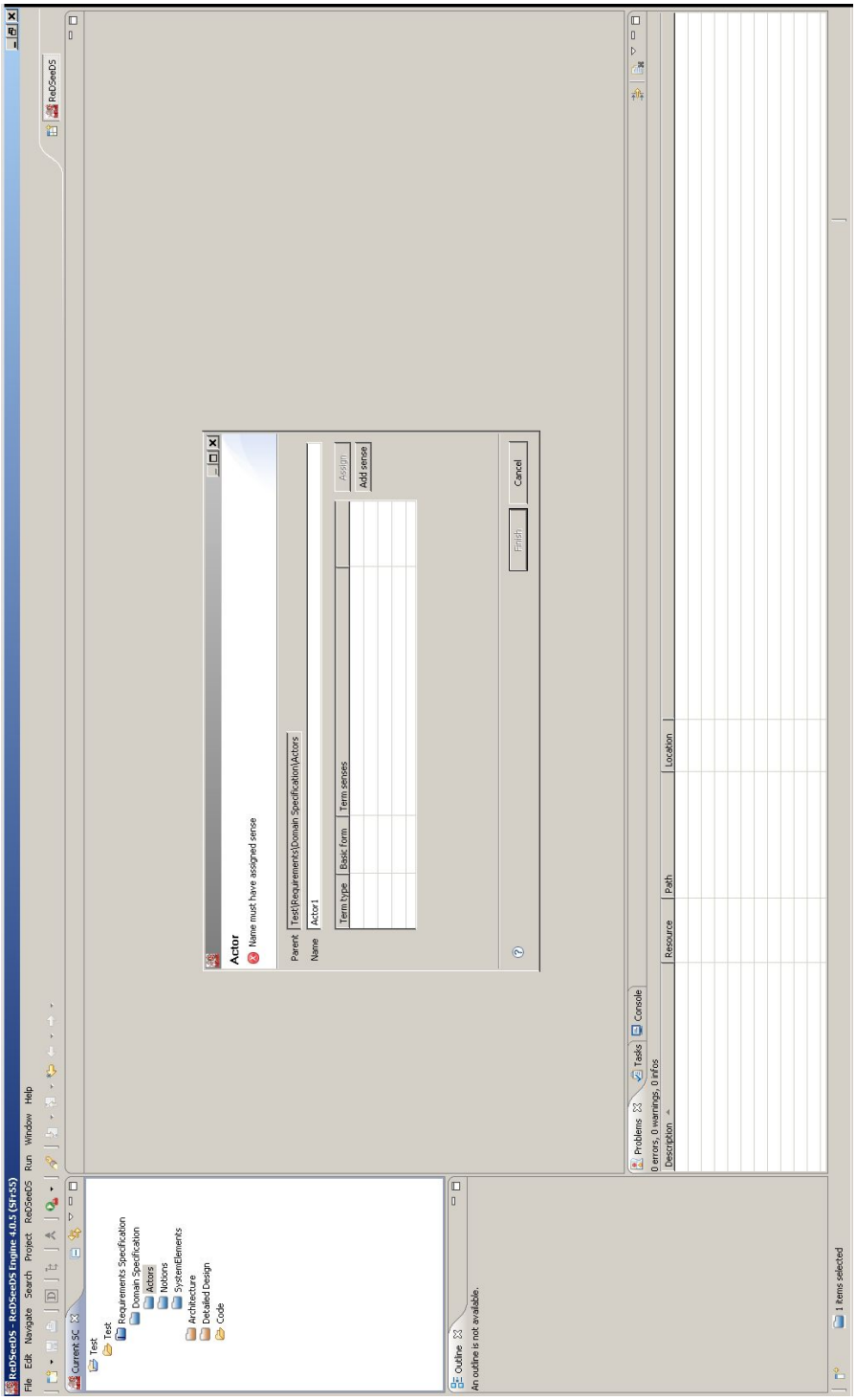

Figure 4.2: Screenshot of the ReDSeeDS Prototype

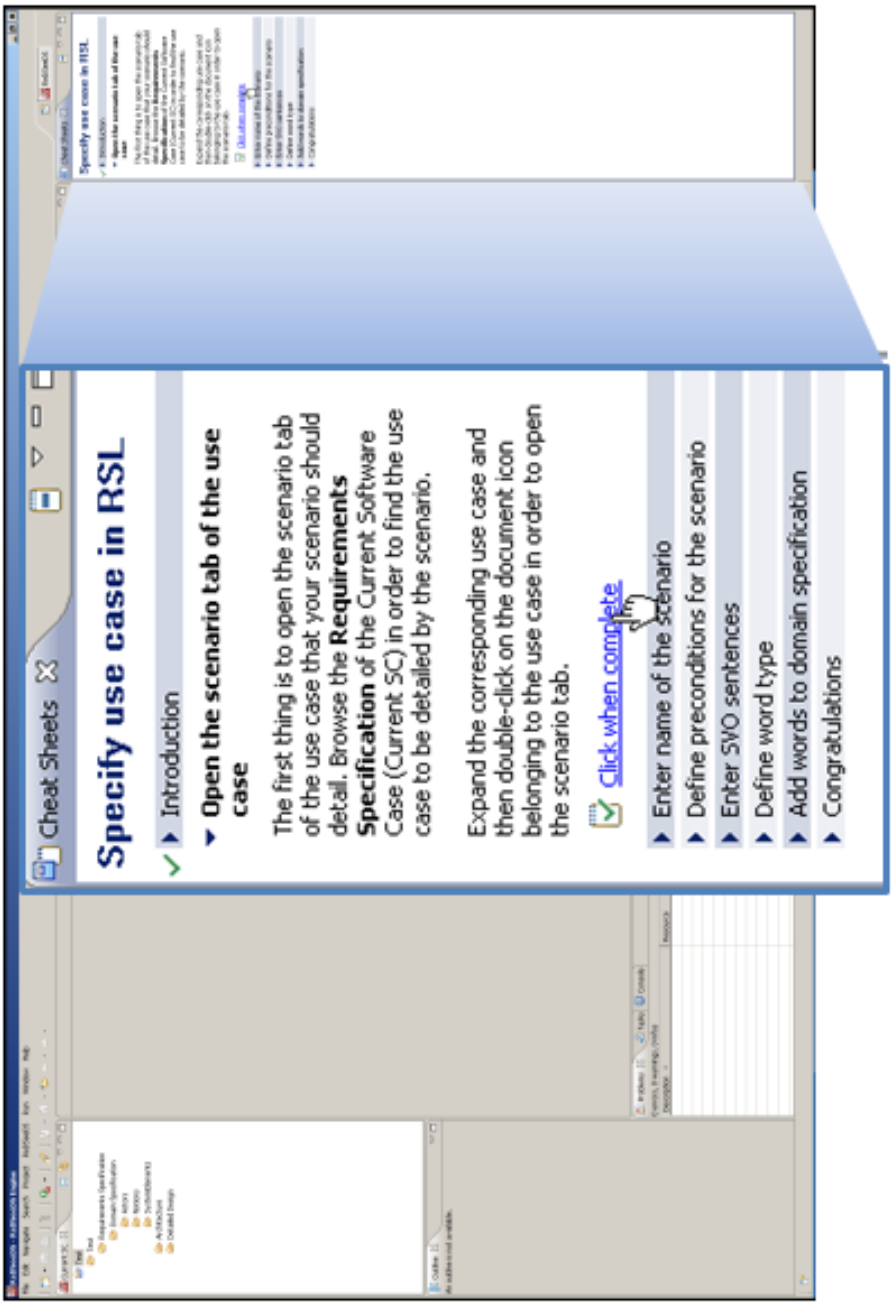

Figure 4.3: A sample Cheat Sheet

CHAPTER 5

Conclusion

It is a wide-spread problem that developers have to "reinvent the wheel" and do repetitive tasks because reuse is not integrated into their development process. Even though some common repositories or libraries where developers can reuse certain artifacts exist, it is still a long way to automated query mechanisms based on given problems and model-to-code transformations of the reusable parts. This is where the ReDSeeDS project comes into play. The ReDSeeDS project shows how modern software development processes can be tremendously enriched without demanding any changes in the existing, well-established development processes. This is achieved by delivering the ReDSeeDS Framework with a well-documented methodology which distinguishes between several different processes which apply ReDSeeDS-related activities step by step. However, it is important that every phase of the software development life cycle is covered by the methodology.

The Architectural Design Phase is tremendously important because the requirements are turned into a basic architectural concept of the solution. Moreover, the Detailed Design and the Implementation Phase strongly depend on the outcome of the Architectural Design Phase. In order to document the phase, the responsible roles have to be identified and their relevant activities classified. Moreover, in the first methodology development iteration the Architectural Design Phase was conceived without any ReDSeeDS-related activities, just the way this phase has been forged through the years of known software methodologies. In additional iterations the designed activities were enriched or replaced step by step with ReDSeeDS relevant activities. The first enhancements provide a specific notation for reuse, the second enhancement the automatic reuse, and the third enhancement automatic reuse with transformation capabilities.

A sophisticated representation of the ReDSeeDS Methodology is essential for the methodology development and for conveying the information to the user. Models enriched with textual information are an adequate representation of the ReDSeeDS Methodology. In order to exchange information, a standardized notation according to a specific metamodel has to be used. The Software Process Engineering Metamodel (SPEM) is one possibility. In addition, a tool has to be selected that supports modeling according to the chosen metamodel, team development and collaboration, a comfortable way for structuring and tracing elements, a way to link models,

an effective way for adding textual information to activities, and offers all the needed output creation possibilities.

In many methodologies there exists a gap between primary tasks (to achieve a certain goal) and secondary tasks (mastering a tool to accomplish the primary tasks). Often developers either know what has to be done or what they can do with a tool without having the big picture in mind. The ReDSeeDS Methodology describes the primary tasks with models and textual description but additionally provides guidance for secondary tasks with the help of Eclipse Cheat Sheets (the ReDSeeDS prototype is based on the Eclipse Framework). Because the ReDSeeDS Methodology can easily be exported to a Hypertext Web pages representation, it allows the user to automatically invoke Cheat Sheets in the ReDSeeDS tool with the help of hyperlinks.

Many things have to be considered when developing a methodology. However, in the end a sophisticated methodology ensures that new innovative solutions get the acceptance and the wide-spread use that they deserve.

Bibliography

[1] H. Baetjer. *Software as Capital: An Economic Perspective on Software Engineering*. The Institute of Electrical and Electronics Engineers, 1998.

[2] R Dumke. *Software Engineering: Eine Einführung für Informatiker und Ingenieure*. Vieweg+Teubner, 2003.

[3] Eclipse Authors. Help - Eclipse SDK. `http://help.eclipse.org`. Accessed: 2010-11-01.

[4] ESA Board for Software Standardisation and Control (BSSC). Guide to the software architectural design phase, 1995. ESA PSS-05-04 Issue 1 Revision 1.

[5] M. Fritz. *UML mit Enterprise Architect*. XEN Information Systems, 2007.

[6] E. Hull, K. Jackson, and J. Dick. *Requirements Engineering*. Springer, 2004.

[7] IBM. Building cheat sheets in Eclipse V3.2 Website. `http://www.ibm.com/developerworks/opensource/library/os-ecl-cheatsheets/`. Accessed: 2010-11-01.

[8] H. Kaindl, J. Falb, S. Melbinger, and T. Bruckmayer. An Approach to Method-Tool Coupling for Software Development. *Software Engineering Advances (ICSEA), 2010 Fifth International Conference*, page 101, 2010. IEEE.

[9] H. Kaindl, B. Lutz, and P. Tippold. *Methodik der Softwareentwicklung*. Vieweg+Teubner, 1998.

[10] P. Kruchten. *The Rational Unified Process: An Introduction*. Addison Wesley, 2000.

[11] P. Mohapatra. *Software Engineering: A Lifecycle Approach*. New Age International, 2010.

[12] OMG. SPEM Website. `http://www.omg.org/spec/SPEM/`. Accessed: 2010-11-01.

[13] OMG. *OMG Unified Modeling Language Specification*, 2001.

[14] J. Preece, Y. Rogers, H. Shap, D. Benyon, S. Holland, and T. Carey. *Human-Computer Interaction: Concepts And Design*. Addison Wesley, 1994.

[15] R. Pressman. *Software Engineering: A Practitioner's Approach*. McGraw-Hill Science/Engineering/Math, 2009.

[16] ReDSeeDS. ReDSeeDS Website. `http://redseeds.iem.pw.edu.pl`. Accessed: 2011-03-21.

[17] D. Rubio. *Pro Spring Dynamic Modules for OSGi Service Platforms*. Apress, 2009.

[18] Sparx Systems. Reviewer's Guide to Enterprise Architect. `http://www.sparxsystems.com/downloads/whitepapers/EAReviewersGuide.pdf`. Accessed: 2010-11-01.

[19] Sparx Systems. Version Control in Enterprise Architect. `http://www.sparxsystems.com/WhitePapers/Version_Control.pdf`. Accessed: 2010-11-01.

[20] Sparxsystems. EA SPEM Profile Website. `http://www.sparxsystems.com.au/resources/developers/spem_profile.html`. Accessed: 2010-11-01.

[21] Wikipedia Authors. Software Development Methodology. `http://en.wikipedia.org/wiki/Software_development_methodology`. Accessed: 2010-11-01.

[22] K. Wolter, T. Krebs, L. Hotz, M. Smialek, T. Bruckmayer, and H. Kaindl. ReDSeeDS Software Development Methodology, 2008.

[23] R. R. Young. *The Requirements Engineering Handbook*. Artech House, 2003.