Virpi Mäkinen

ANALYSIS OF USE CASE APPROACHES TO REQUIREMENTS ENGINEERING

Master's thesis in
Information Systems Science
19.6.2003

University of Jyväskylä
Department of Computer Science and Information Systems
Jyväskylä

ABSTRACT

Mäkinen, Virpi Anna Helena
Analysis of Use Case Approaches to Requirements Engineering/ Virpi Mäkinen
Jyväskylä: University of Jyväskylä, 2003.
144 p.
Master's thesis

In this study we analyse use case approaches that have been crafted for requirements engineering. Requirements engineering is a process, where visions about a system are established in a context. The use case approach is widely used to capture functional requirements for a system. The use case technique of the UML is regarded as de facto standard in systems development. In the literature, several problems concerning the use case technique have been reported. Motivated by the deficiencies, a large variety of use case approaches have been introduced.

We analysed use case approaches on two levels. First, we carried out an overall analysis by classifying use case approaches through a framework. After that, we analysed the approaches in more details in terms of metamodeling. In addition, we made a comparative evaluation of three use case classification frameworks and modified one of them. The study is theoretical and is based on the literature. The study investigates the ways the large variety of the use case approaches covers requirements engineering and, moreover, the characteristics, concepts, differences and similarities the use case approaches have. Therefore, the study helps to understand the achievements gained from recently developed use case approaches.

KEYWORDS: requirements engineering, use case, systems development, UML, framework, metamodeling

CONTENTS

1 INTRODUCTION

Requirements engineering is widely considered to be the most important and difficult part of systems development. The cost of making a change into a system due to a requirements problem is much greater than the cost of repairing implementation or coding errors. It is quite justified to state that requirements engineering is a critical success factor in systems development (e.g. Hofmann & Lehner 2001, 58). As presented by Brooks (1986, 1074): 'The hardest single part of building a software system is deciding precisely what to build. -- No other part of the work so cripples the resulting system if done wrong. No other part is more difficult to rectify later.'

A *requirement* is a condition or capability needed by a user to solve a problem or achieve an objective (IEEE 1990, 62). Requirements are divided into functional and non-functional. According to Pohl (1994, 247) a *functional requirement* specifies, what a system must do. A *non-functional requirement* represents constrains on the services offered by a system (Pohl 1994, 247). For instance, 'A user must be able to check his account balance.' is an example of a functional requirement. Response time and memory usage represent non-functional requirements (see IEEE (1984, 21-23) for categories for non-functional requirements).

Requirements engineering (RE) aims at systematizing the first part of the systems development process, during which the definition of a system is derived. Pohl (1994, 245) describes requirements engineering as a process, where visions about a system are established in a context. In addition, Sommerville (1998, 64) defines requirements engineering as a process, where the services, which a system must provide, and the constraints under which it must operate are established. The goal of requirements engineering is to define the needs for and the behaviour of a system (Davis & Hsia 1994, 14).

Requirements engineering is vital for both software engineering and information systems development. *Software engineering* is the application of systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software (IEEE 1990, 67). *Information systems development* is a change process taken with respect to an object system in a set of environments by a development group using tools and an organized collection of techniques collectively referred to as a method to achieve or maintain some objectives (Tolvanen 1998, 32). To cope with the both development fields we use the general term 'systems development'.

Jacobson, Christerson and Overgaard (1992) introduced the concept of a use case in conjunction with the Objectory method. Ever since, the use case technique has become a popular and widely used technique for capturing and describing functional requirements for a system. In fact, use cases seem to be one of the best known and most widely accepted tools in systems development (e.g. Jaaksi 1998, 58). The use case technique has attracted considerable attention in systems development, human computer interaction (HCI) and business process re-engineering (BPR). Furthermore, use cases have been applied into several phases of systems development process.

The use case technique has been integrated in Unified Modelling Language (UML), which has emerged as the dominant modelling language in systems development (e.g. Phillips, Kemp & Kek 2001, 49). Therefore, the use case model of the UML is regarded as de facto standard.

According to the UML 1.5 (OMG 2003, 2-137) a use case defines the behavior of a system or other semantic entity without revealing the internal structure of the entity. A use case specifies a sequence of actions, including variants, which the entity can perform when interacting with actors. The sequences of action are called scenarios, and actors represent the world outside the entity. Use cases are

modeled in a use case diagram, which shows use cases and actors together with their relationships. In addition, use cases are described in a textual form. In the literature, the terms scenario and use case have often been used synonymously (see e.g. Dano, Briand & Barbier 1997, 80; Glintz 1995, 254). In this study we consider the terms as synonyms. However, we prefer the term use case.

The use case technique is said to be the most controversial technique of the UML (e.g Siau, Lee, Korhonen 2001, VIII-2). In the literature, several problems and weaknesses concerning the technique has been reported. For instance, the lack of object-orientation is said to be one of the drawbacks of the technique. Motivated by the deficiencies of the technique, a large variety of use case approaches have been proposed. For instance, use cases have been introduced in several forms (e.g. textual forms, sequence diagrams, state machines and Petri nets) and applied for different purposes (e.g. validation and verification of requirements, usage-oriented RE).

Neither the use case technique of the UML (OMG 2003) nor the original use case technique (Jacobson et al. 1992) does cover the entire requirements engineering. Hence, we may ask, in which way the large variety of the use case approaches cover requirements engineering and what kind of characteristics, concepts, differences and similarities the approaches have. Particularly, it is important to emphasise the purposes for and forms in which the use cases have been proposed. In order to answer the questions, we need a framework to investigate the role of the use case technique in requirements engineering.

In this study we will analyse use case approaches that have been suggested for RE. The study highlights the characteristics, concepts and concept constructions of the use case approaches. The study situates the current practise of the use case technique and helps to understand the achievements gained from the recently developed use case approaches.

The results of the study may be used when choosing a use case approach for a systems development project. Understanding the different forms the use cases may take and the benefits of applying these forms in practise are valuable. Furthermore, the study has the following theoretical implications. A comparative evaluation of three use case classification frameworks is made. Therefore, the study situates the practise of the use case classification frameworks and assist researches to develop new ones. An existing use case classification framework is modified and used to analyse recently developed use case approaches. In addition, metamodeling assist in gaining a more detailed picture of the approaches. Consequently, the study assists the researchers to find gaps and deficiencies among the use case approaches and to develop new ones. The study is theoretical and is based on literature.

The study is organized as follows (see FIGURE 1 for illustration of the inter-dependencies between the chapters). In Chapter 2, we will discuss requirements engineering. First, we will examine the four aspects of systems development and describe the development aspect through the waterfall model. After that, we will concentrate on the processes and artefacts of the requirements engineering in the traditional systems development and the UML. The latter is described in the forms of the Unified Process and the Rational Unified Process.

In Chapter 3, we will discuss the use case technique. We will first introduce the use case model of the UML 1.5 and the Unified Process. After that, we will concentrate on the pros and cons of the technique. In Chapter 4, we will make a comparative evaluation of three use case classification frameworks and modify one of them to suit our purposes. We will first analyse the classification concepts, the coverage of the three dimensions of the RE framework and the requirements criteria in the frameworks. Secondly, we will inspect the concepts of the development and system worlds. Thirdly, we will choose the most suitable framework. Lastly, we will modify the framework.

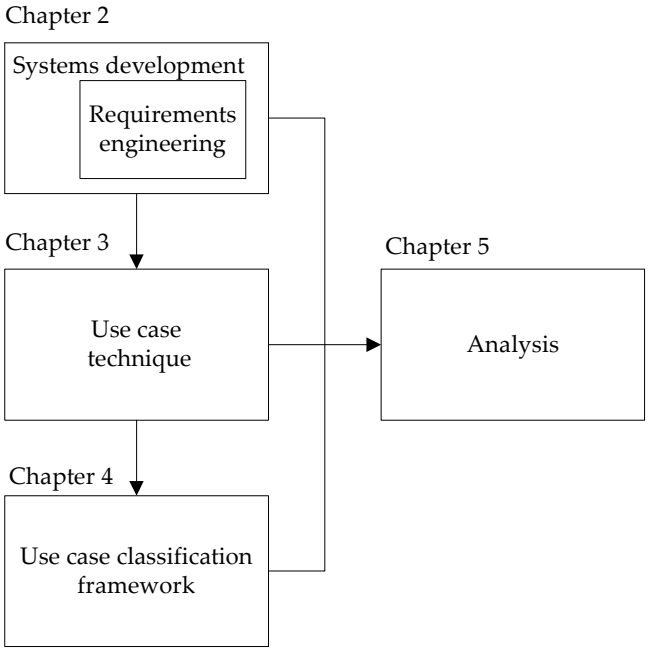

FIGURE 1. Chapters and their inter-dependencies.

In Chapter 5, we will analyse use case approaches that have been suggested for RE on two levels. We will first carry out an overall analysis in order to get a general picture of the use case approaches. In the analysis, we will apply the use case classification framework (Chapter 4) as a tool. After that, we will carry out an in-depth analysis of the use case approaches in terms of metamodeling. The study ends with conclusions in Chapter 6.

2 REQUIREMENTS ENGINEERING IN SYSTEMS DEVELOPMENT

In this section, we will investigate requirements engineering. In order to gain a clear picture of the position of requirements engineering in systems development, we will investigate the processes and artifacts of requirements engineering in the two main systems development approaches. The approaches are the traditional systems development (Sommerville 1998) and the UML (OMG 2003). The latter is described in the forms of the Unified Process (Jacobson, Booch & Rumbaugh 1999) and the Rational Unified Process (RUP, Kruchten 2001).

The chapter is organized as follows. We will first discuss systems development and the traditional requirements engineering. After that, we will discuss the Unified Process and the Requirements workflow in it. Lastly, we will examine the Rational Unified Process and its Requirements workflow.

The use case technique consists of a use case model and a procedure to make use case models. The use case model consists of a use case diagram and several use case descriptions. In this chapter, the Requirements workflows describe the use case modelling procedure of the Unified Process and of the RUP. In Chapter 3 we will present the use case model in the UML 1.5 and the Unified Process.

2.1 Requirements engineering in the traditional systems development

In this section, we will first examine the four aspects of systems development and describe the development aspect through the waterfall model. After that, we will concentrate on the requirements engineering process and its artefacts. The aim of the section is to describe the position, process and artefacts of the requirements engineering in the traditional systems development.

2.1.1 Four worlds of systems development

NATURE Team (1996, 517-518) presents systems development as a means of four worlds. The worlds are subject world, usage world, system world and development world (FIGURE 2; NATURE Team 1996, 517). The *usage world* describes, how systems are used to achieve work. The world includes agents – i.e. stakeholders, users and organisational contexts - acting in roles and carrying out tasks and work activities to produce artefacts. Systems development distinguishes between two types of tasks. User tasks are carried out in the usage world, while system tasks are performed in the system world. The usage world is the main source of the user-defined requirements and goals.

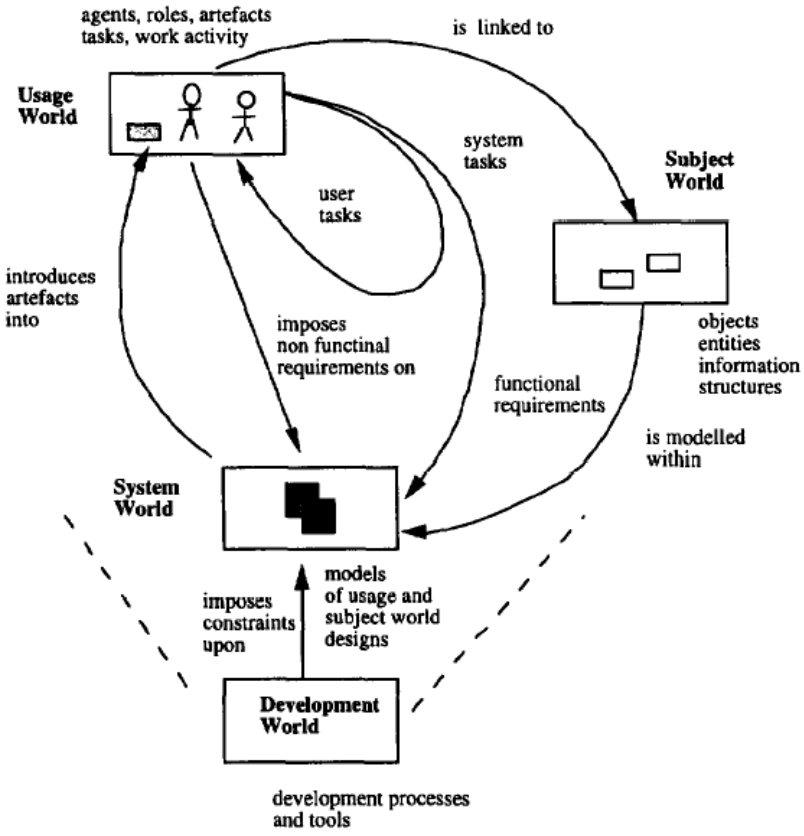

FIGURE 2. The four worlds of systems development (NATURE Team 1996, 517).

The usage world needs information about the *subject world*, which is the real-world domain that a system is intended to maintain information about. The subject world includes the real-world objects, entities and information structures that are described in conceptual modelling. The subject world is the main source of domain-imposed requirements, which are facts of nature and form the connection between the real world and a system.

The usage and subject worlds are modelled within the *system world*, which contains descriptions of events, processes and technical entities. Moreover, the world involves the mappings from the conceptual specifications to design and implementation of a system. The subject and usage worlds impose requirements on and the development world defines constraints upon the system world. Moreover, the system world introduces artefacts into the usage world. Lastly, the *development world* contains the processes and tools for creating systems and changing the knowledge of the other worlds into specifications and designs.

The worlds are embedded in a cyclical process. Consequently, the development world investigates and contains models of the subject, usage and system worlds as they are created during the process. Next, we will investigate the development world, i.e. the systems development process in a greater detail.

2.1.2 Systems development process

The oldest and the most widely known process model of systems development is the waterfall model (Royce 1970), which is also called the classic line cycle or the linear sequential model. The process model suggests a systematic, sequential approach to systems development (FIGURE 3; cf. Pressman 1997, 32; cf. Sommerville 1998, 9). The approach cascades through analysis, design, implementation, testing and maintenance phases. In practise, the phases of the

waterfall model overlap and feed information to each other. (cf. Pressman 1997, 33; Sommerville 1998, 10.)

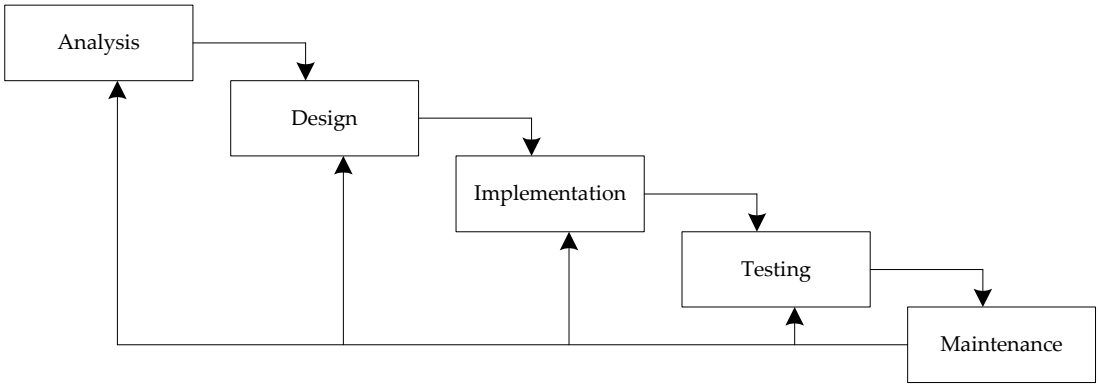

FIGURE 3. The waterfall model (cf. Pressman 1997, 32; cf. Sommerville 1998, 9).

During the analysis phase, which is also called the requirements analysis and definition, the requirements for a system are gathered and documented. The phase involves co-operation with customers and usage of different elicitation techniques. In the design phase, requirements are described as structured presentations. The design phase is a multistep process, which focuses on data structure, software architecture, interface representations and procedural detail dimensions. (Sommerville 1998, 10; Pressman 1997, 34, 270.)

During the implementation phase, the design is translated into a machine-readable form by coding. After the implementation, the system is tested during the testing phase to make sure that it meets requirements. The testing process focuses on the logical and functional internals of the software. The logical internals testing assures that all statements are tested. The functional internals testing uncovers errors and ensures that the input of the system will produce required results. (cf. Pressman 1997, 34.) The longest phase of the process model is the maintenance. During the phase, the system will still undergo enhancements resulting from external environment changes or new requirements (Sommerville 1998, 10.)

The waterfall model has many weaknesses (see e.g. Pressman 1997, 35; Sommerville 1998, 11), but it provides a template into which the methods for the analysis, design, implementation, testing and maintenance phases can be placed (Pressman 1997, 35). The waterfall model reflects the current practise. Moreover, other process models, such as the spiral model, are based on this approach (cf. Sommerville 1998, 11). The model presents the functional parts of the development process, which are used as building blocks in other process models.

As described above, requirements engineering is a process, where visions about a system are established in a context (Pohl 1994, 245). Therefore, the analysis phase stands for the requirements engineering in the traditional systems development process.

2.1.3 Requirements engineering process

According to Pohl (1994, 247-255) requirements engineering may be presented as three dimensions - specification, representation and agreement - that all have diverse goals (FIGURE 4; Pohl 1994, 249). Input to the requirements engineering process is opaque personal views of a system expressed in informal representation formats. Desired output of the process is a formal, complete system specification on which an agreement has been reached.

The aim of the s*pecification dimension* is to transform the unclear understandings of a system into a complete system specification. Standards and guidelines are used to guide the process. The *representation dimension* aims at changing the informal representations into formal or semi-formal. According to Pohl (1994) the requirements engineering should support different representation formats and the transition from one format to another.

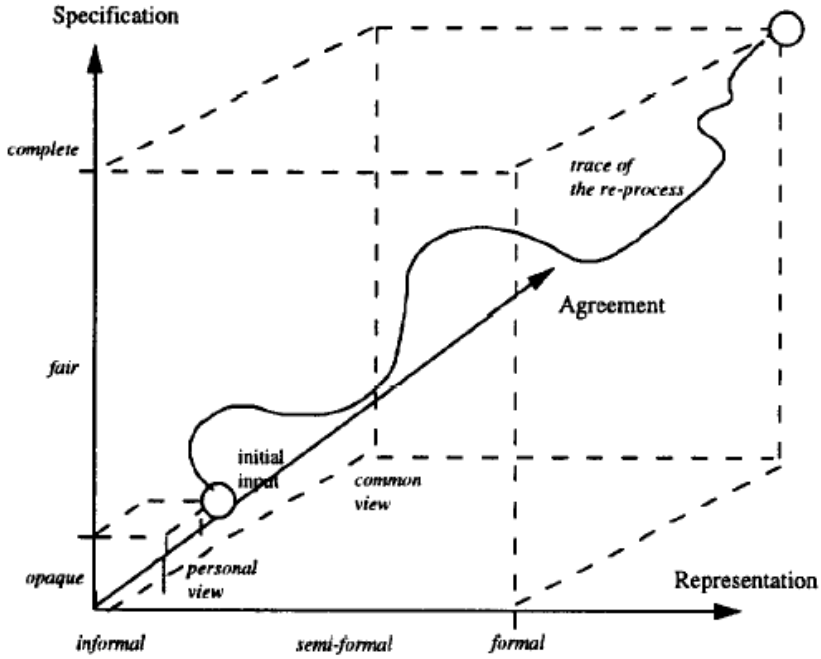

FIGURE 4. The requirements engineering process within three dimensions (Pohl 1994, 249).

The representation formats are divided into informal, semi-formal and formal. Informal representations are used in the every-day life and have a very high expressive power. They include e.g. natural language, sounds, animation and examples. Diagrams represent semi-formal representations. They are clear and provide a good overview of a system. Lastly, formal representation formats have a well-defined semantics and involve e.g. specification and knowledge representation languages.

The *agreement dimension* deals with the degree of an agreement reached on a specification. In the beginning of the requirements process, divergent views about a system are common and have positive effects on RE process. On the other hand, the goal of the dimension is to transform the personal opinions into a common agreement. Lastly, the *requirements engineering process* is the trace in the space formed by these dimensions.

2.1.4 Stages of the requirements engineering process

According to Sommerville (1998, 67-68) the requirements engineering process consists of activities that lead to the production of requirements definitions and requirements specifications. In addition, also other documents, such as software specifications, may be produced during the process. The requirements engineering process is composed of four stages. The stages are Feasibility study, Requirements analysis, Requirements definition and Requirements specification (FIGURE 5; cf. Sommerville 1998, 67). The stages are iterated during the RE process. Next, we will describe the RE process according to Sommerville (1998).

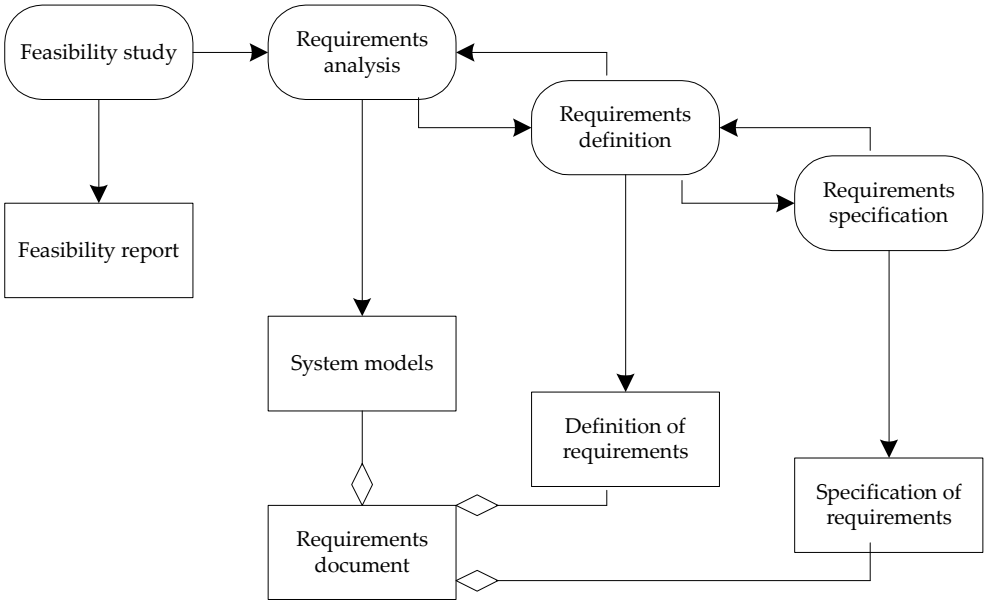

FIGURE 5. The requirements engineering process (cf. Sommerville 1998, 67).

The first, and sometimes the only stage of requirements engineering process is the *Feasibility study*. The study investigates, whether it is cost-effective to develop the proposed system. The stage should be quick and cheap. After the feasibility study, a decision is made, whether to go ahead with analysis phase or to abandon the development of the system. The results of the feasibility study are reported in a *feasibility report*.

The second stage of the process is the *Requirements analysis*. During the stage, the requirements are derived through discussions with potential users, observing existing systems, prototyping etc. Furthermore, one or more system models may be developed. A *system model* is an abstract description of a system. In general, system models are based on computational concepts, such as objects and functions. A data-processing model and a composition model are examples of the system models.

The requirements analysis consists of six activities, which are iterated in a cycle. (FIGURE 6; Sommerville 1998, 81):

1. Domain understanding: Developing understanding about the application domain.
2. Requirements collection: Interacting with stakeholders to discover the requirements. Domain understanding will also develop during this activity.
3. Classification: Organising requirements into groups.
4. Conflict resolution: Solving conflicting requirements that may result from the demands of several stakeholders.
5. Prioritization: Classifying requirements e.g. into mandatory and desirable or organizing them as organized set of increments.
6. Requirements validation: Checking the validity of requirements, i.e. that they are complete, consistent and equivalent to the needs of the stakeholders.

The process can be seen to start from the domain understanding and end at the requirements validation.

The *Requirements definition* is the third stage of the requirements engineering process. During the stage, the gathered information is translated into a brief definition of requirements that characterizes the requirements. The description is written for the customers as well as for the software developers, so it must be clearly understandable.

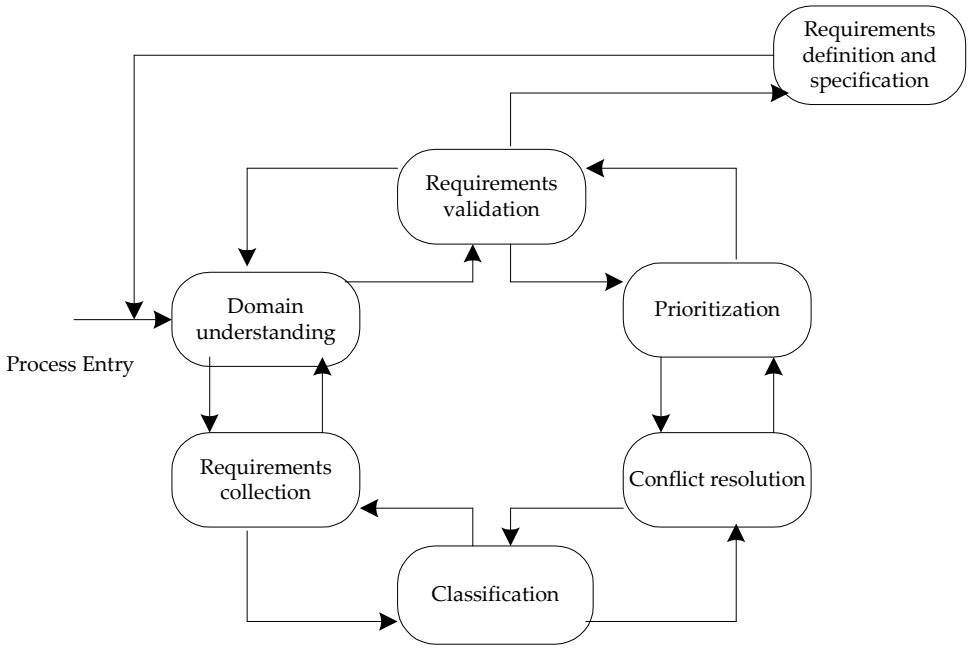

FIGURE 6. The requirements analysis process (Sommerville 1998, 81).

The last stage of the RE process is the *Requirements specification*. A detailed and precise specification of requirements is produced during the stage. Because the requirements and design activities influence each other, the description is usually written simultaneously with the high-level design. It is recommendable that the description serves as a contract between the customers and system developers.

Requirements evolution is a very important feature in requirement engineering process. As stated by Jacobson et al. (1999, 9): 'It is one of the constants of systems development that the requirements change.' This inevitability of change should be taken into account when producing requirements documents. Requirements should not been frozen prematurely because that results in a poor design. Requirements change over time e.g. due to better understanding of a system and the changes in the environment of a system (e.g Davis & Hsia 1994, 14; Berry & Lawrence 1998, 28; Sommerville 1998, 73).

From the evolutionary point of view, requirements may be categorized as enduring and volatile requirements. Enduring requirements are stable and relate directly to the system domain - they can sometimes be derived from the domain models. Volatile requirements are likely to change during systems development or after it due to organizational, political or technical changes. Particularly, the non-functional requirements are affected by changes in the technology.

Volatile requirements fall into categories of mutable, emergent, consequential and compatibility requirements. Mutable requirements change because of changes in the environment of an organisation. Emergent requirements emerge as requirement process goes on, and consequential requirements result from the introduction of a system into the systems context. Compatibility requirements depend on the processes of the system or business processes within an organization.

In practice, it is nearly impossible to define a complete and consistent set of requirements. One of the reasons for that is that stakeholders have different requirements and priorities, which may conflict. Usually customers impose requirements because of budgetary and organizational constraints, which may conflict with the requirements of the users. Therefore, requirements are inevitably a compromise.

Also the requirements validation is a significant issue in requirement engineering. The *requirements validation* means ensuring that the requirements define what stakeholders want. If the validation fails, the errors in the requirements will be propagated to the design and implementation of the system. Thereby, errors in requirements are very expensive to correct after a system is ready. The cost of making a change into a system due to a problem in the requirements is much greater than the cost of repairing errors in the implementation or coding.

Requirements validation should be done all the way during the requirements process, not just after the software requirements document has been written. In addition, validity of the requirements should be checked in requirements reviews, during which multiple readers verify the requirements document. In addition, prototyping is a commonly used and significant validation technique.

In the requirements validation, the requirements criteria must be checked. The criteria are as follows:

1. Validity: Requirements are what users really need.
2. Consistency: Requirements should not conflict with each other.
3. Completeness: All functions and constraints must be defined.
4. Realism: Requirements must be realizable.
5. Verifiability: Requirements should be testable.
6. Comprehensibility: Requirements are properly understood.
7. Traceability: The origin of the requirement must be clearly stated. Requirements evolve and sometimes it is necessary to access the impact of change.
8. Adaptability: Requirements should not have large-scale effects on other requirements.

2.1.5 Artefacts of the requirements engineering process

According to Sommerville (1998, 64-65, 68-69) requirements definitions, requirements specifications and software specifications are the descriptions produced during the requirements engineering process. A *requirements definition* is a high-level abstract statement of the services a system must provide and the constraints it must meet. It is expressed in a natural language and diagrams and targeted at the managerial level, customers and software developers. The definition is based on customer-supplied information.

A *requirements specification* (also called a *functional specification*) is a structured description describing the services the system must meet in a more detailed manner. The description may serve as a contract between software developers and customers. The requirements specification expands the requirements definition and is targeted at project managers and software developers. As an example of a requirements definition and a requirements specification, we present the requirement Presenting and accessing external files (FIGURE 7; Sommerville 1998, 65).

Requirements definition

1. The software must provide a means of presenting and accessing external files created by other tools.

Requirements specification

1. The user should be provided with facilities to define the type of external files.
2. Each external file type may have an associated tool which may be applied to the file
3. Each external file type may be represented as a specific icon on the user's display.
4. Facilities should be provided for the icon representing an external file type to be defined by the user.
5. When a user selects an icon representing an external file, the effect of that selection is to apply the tool associated with the type of the external file to the file represented by the selected icon.

FIGURE 7. A Requirements definition and a Requirements specification for the requirement Presenting and accessing external files (Sommerville 1998, 65).

Usually requirements definitions and requirements specifications are the textual descriptions written during the requirements engineering process. Moreover, a software specification may be produced. However, the description is not obligatory. A *software specification* is even more detailed document, which is written for the software developers involved in developing a system. It is an implementation-oriented, abstract description of a system and acts as a basis for the Design and Implementation phases. The specification may be expressed in a formal notation or in a design language.

Requirements are expressed in a *software requirements document*, which is also called a *software requirements specification (SRS)*. The document includes requirements definitions, requirements specifications and system models. Usually, requirements definitions and requirements specifications are presented separately. However, in some cases these two are incorporated into a single description. The requirements should be produced to enable the final system design to be traced into the requirements.

The software requirements document should satisfy six quality criteria. It should describe only external system behaviour, specify constraints on the implementation and be easy to change. Furthermore, it should serve as a reference tool for the system maintainers, characterize acceptable responses to undesired events and record forethought the life cycle of a system. The structure of a software requirements specification is presented in many standards, i.e. in IEEE Standards 830 and 1233 (1984, 1998a, 1998b). Also Pressman (1997, 307) and Sommerville (1998, 69) give generic structures for Software requirements specifications.

In this section, we have discussed the four aspects of systems development and described the development aspect through the waterfall model. In addition, we have described the requirements engineering process and its artefacts in the traditional systems development. In the next section, we will investigate the RE in the Unified Process.

2.2 Requirements engineering in the Unified Process

The Unified Software Development Process (also called the Unified Process) is a use-case driven, iterative and incremental systems development process presented by Jacobson, Booch and Rumbaugh (1999). In this section, we will first discuss the Unified Process. After that, we will go on with the

Requirements workflow of the Unified Process and compare it to the traditional requirements engineering process (Sommerville 1998). The aim of the section is to investigate the position, process and artefacts of the RE in the Unified Process.

2.2.1 The Unified Process

According to Jacobson et al. (1999, 11-13) the Unified Process consists five workflows, which take place over four phases (FIGURE 8; cf. Jacobson et al. 1999, 11; Wallnau 2000, 12). The workflows are Requirements workflow, Analysis workflow, Design workflow, Implementation workflow and Test workflow. The phases are Inception phase, Elaboration phase, Construction phase and Transition phase. In FIGURE 8, the curves approximate the extent to which the workflows are carried out during each phase. The phases are divided into iterations and terminate in milestones. A typical iteration goes through all the five workflows.

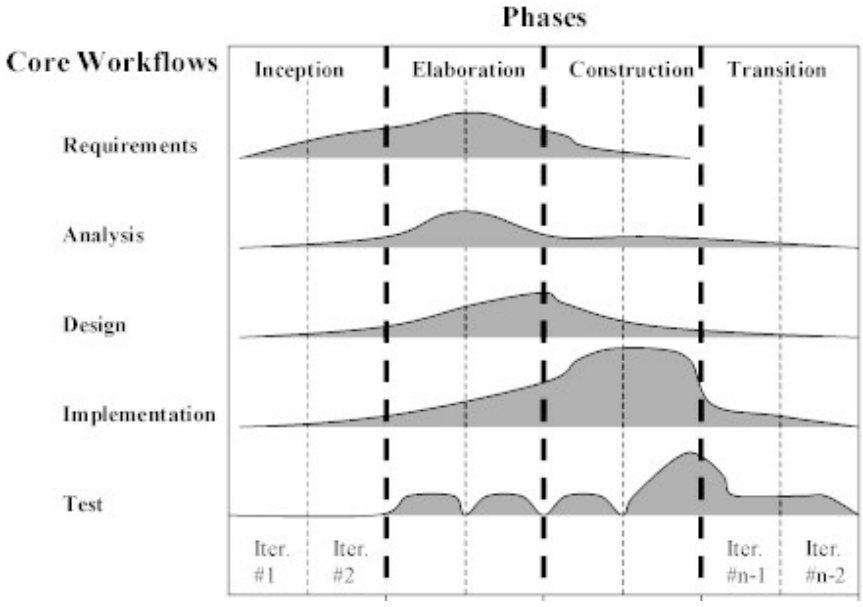

FIGURE 8. The Unified Software Development Process (cf. Jacobson et al. 1999, 11; Wallnau 2000, 12).

The workflows are repeated with various emphasis and levels of intensity during every iteration. Each iteration follows a pattern similar to the waterfall model and contains activities from all workflows. Therefore, the workflows are the building blocks in the process and, in a way, correspond the phases in the waterfall model. Next, we will shortly discuss the workflows. After that, we will concentrate on the phases of the Unified Process.

The Requirements workflow consists of stages, which concern establishing requirements. The stages involve finding use cases and actors, prototyping user interface and detailing and structuring the use cases. The Analysis workflow involves architectural analysis and activities for analysing use cases, classes and packages. During the Design workflow, the architectural design is made. The Implementation workflow includes coding, performing unit testing and integrating the whole system. The Test workflow involves planning, designing and performing integration and system tests as well as evaluating the test results.

During the first phase of the Unified Process, Inception, a vision of a system is produced. The phase answers three questions: What is the system doing for each user? What could the architecture look like? What will it cost to develop the product and what is the project plan like? A simplified use case model is build to answer the first question. At this stage, the system architecture is tentative and typically just a sketch. The most important risks are identified and prioritised, the elaboration phase is planned in detail and the whole project is roughly estimated. The Inception phase includes features of the analysis and design phases of the waterfall model.

In the Elaboration phase, the system architecture is designed and most of the use cases are specified in detail. The architecture is expressed as a use case model, an analysis model, a design model, an implementation model and a deployment model. During the phase, the most critical use cases are realized

and an architecture baseline is build. The Elaboration phase includes activities from the analysis and design phases of the waterfall model.

The system is programmed during the Construction phase. The system may not be totally free of defects, which should be found and fixed later during the Transition phase. The Construction phase corresponds to the implementation phase of the waterfall approach. In addition, it includes some functionalities of the testing phase. The Transition phase involves activities such as manufacturing, training customer personnel, providing on-line assistance and correcting defects found after delivery. The Transition phase includes activities of the maintenance and testing phases of the waterfall model.

In conclusion, the Requirements workflow represents requirements engineering in the Unified Process. Next, we will take a closer look at the Requirements workflow.

2.2.2 Requirements workflow

In this sub-section, we will discuss the Requirements workflow as presented by Jacobson et al. (1999). The Requirements workflow consists of four stages. The stages are *List candidate requirement, Understand system context, Capture functional requirements* and *Capture non-functional requirements*.

The workflow proceeds as follows. Firstly, ideas about favourable features of a system are listed as candidate requirements in a feature list. After that, the context of a system is modelled in a domain model or in a business model. A domain model describes the important concepts of a system context as domain objects. A business model describes business processes and is a kind of a superset for a domain model. After that, functional requirements are captured in use cases. In addition, a user interface is specified. Finally, non-functional requirements are captured and connected either to a class in an analysis model,

in a use case or managed separately in a list of supplementary requirements. The non-functional requirements, which are specific to an individual use case, are represented in a context of the use case.

In conclusion, a use case model consisting of use case diagrams and use case descriptions and a list of supplementary requirements correspond to the software requirements document in the traditional systems development. A use case model correlates to a system model.

The functional requirements are captured during five activities. The activities are Find actors and use cases, Prioritise use cases, Detail a use case, Prototype user interface and Structure the use case diagram (FIGURE 9; cf. Jacobson et al. 1999, 143). To start with, a first version of a use case diagram is established by identifying actors and use cases. After that, architecturally significant use cases are recognized and prioritized. The prioritized use cases are described in detail. More or less in parallel with that, a user-interface is described in a prototype. Lastly, the use case model is restructured by defining relationships between the use cases.

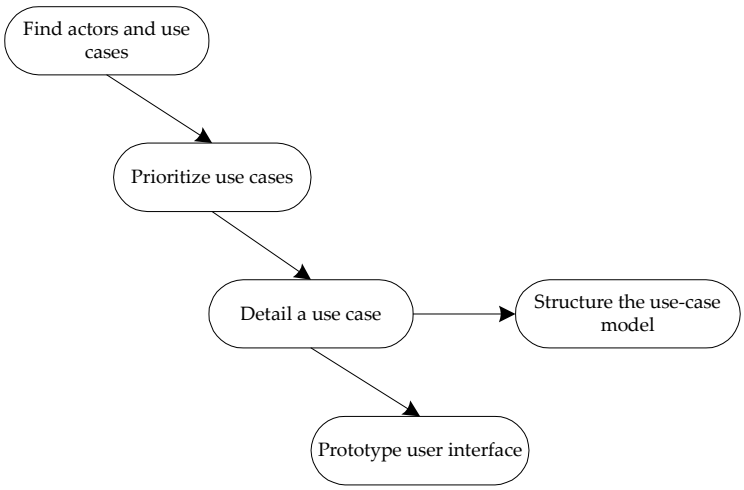

FIGURE 9. The stages for capturing requirements as use cases (cf. Jacobson et al. 1999, 143).

The activity *Find actors and use cases* consists of four steps. The steps involve finding the actors, finding the use cases, briefly describing each use case and describing the use case model as a whole. As a result, a use case model is developed. The purpose of the activity *Prioritize use cases* is to determine, which use cases need to be developed in early iterations and which of them may be developed later. The intention of the activity *Detail a use case* is to describe the flow of events in a more detailed manner. That includes describing how the use case starts, ends and interacts with actors. The outcome of the activity is a detailed description of use cases presented in text and diagrams.

The activity *Prototype user interface* consists of two steps. First, a logical user-interface design is made to find out, what is required from the user interface to enable use cases for each actor. After that, a physical user-interface design and prototypes are created to illustrate how users can use the system to perform the use cases. During the activity *Structure the use case model*, the use case model is structured to general and shared functional use cases that can be used by more specific use cases, and additional functional use cases that extend more specific use cases.

Next, we will compare the traditional requirements engineering process (Sommerville 1998) to the Requirements workflow of the Unified Process (Jacobson et al. 1999). The results are summarized in TABLE 1. The first stage of the requirements engineering process, Feasibility study, does not have correspondences in the Requirements workflow. The stages Requirements analysis, Requirements definition and Requirements specification are covered by the stages Capture functional requirements and Capture non-functional requirements.

Moreover, the activity Domain understanding corresponds to the stage Understand system context. Requirements collection is carried out during the stage List candidate requirements and the activity Find actors and use cases.

The activities Classification, Conflict resolution and Prioritisation are performed during the activity Prioritize use cases. Finally, Requirements validation is carried out during the activities Detail a use case and Prototype user interface.

TABLE 1. Correspondences between the traditional requirements engineering process (Sommerville 1998) and the Requirements workflow of the Unified Process (Jacobson et al. 1999).

Traditional RE process (Sommerville 1998)	**Requirements workflow in the Unified Process (Jacobson et al. 1999)**
Feasibility study	List candidate requirements
Requirements analysis	Understand system context
- Domain understanding	Capture functional requirements
- Requirements collection	- Find actors and use cases
- Classification	- Prioritize use cases
- Conflict resolution	- Detail a use case
- Prioritization	- Prototype user interface
- Requirements validation	Capture non-functional requirements
Requirements definition	
Requirements specification	

In conclusion, the stage Requirements analysis is involved in all of the stages of the Requirements workflow. At this point, we can specify the meaning of the stages Requirement definition and Requirements specification. Sommerville (1998, 82) explains that the separation of the Requirements analysis from the Requirements definition and Requirements specification is artificial and is done simply to allow the process to be discussed. In reality, it is very difficult to separate the activities of the Requirements analysis, Requirements definition and Requirements specification. Furthermore, the flow of events is quite equivalent in the traditional requirements engineering process and in the Requirements workflow.

2.3 Requirements engineering in the Rational Unified Process

The Rational Unified Process is a specific and detailed instance of the Unified Process. In this section, we will first briefly discuss the Rational Unified Process according to Kruchten (2001) and compare it to the Unified Process. After that,

we will investigate the Requirements workflow of the Rational Unified Process (Kruchten 2001) and compare it to the Requirements workflow in the Unified Process. The aim of the section is to investigate the position, process and artefacts of the RE in the Rational Unified Process.

2.3.1 The Rational Unified Process

The Rational Unified Process has two dimensions, which correspond to the dimensions of the Unified Process (FIGURE 10; Rational Software Corporation 1998, 3). As in the Unified Process, the RUP is expressed in terms of cycles, phases, iterations and milestones. In FIGURE 10 the horizontal axis represents the phases and the vertical axis the workflows. The phases of the RUP (Inception, Elaboration, Construction and Transition) are equivalent to the phases of the Unified Process.

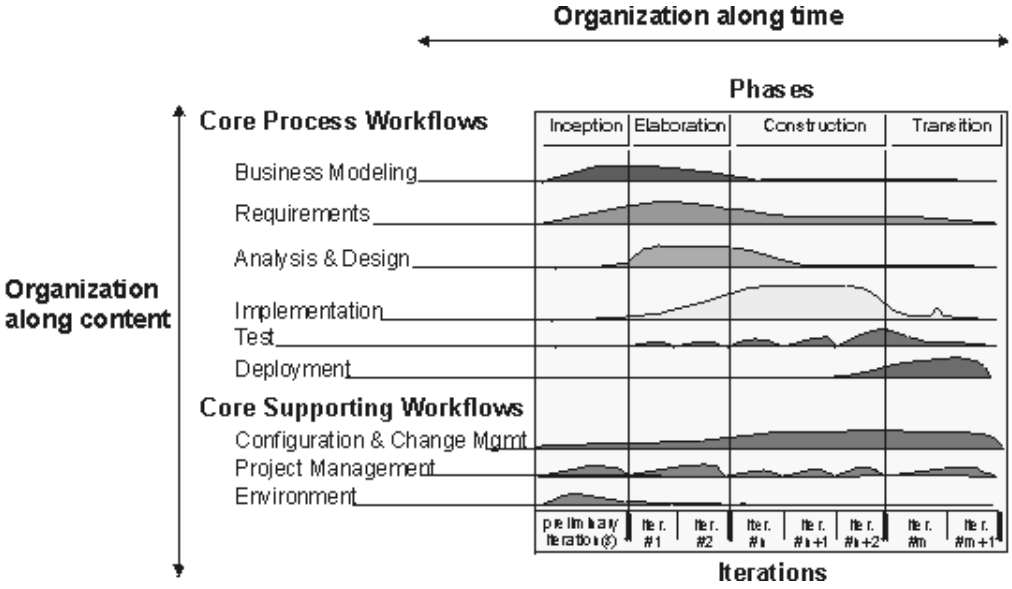

FIGURE 10. The Rational Unified Process (Rational Software Corporation 1998, 3).

The difference between the RUP and the Unified Process is in the workflows. The RUP introduces three core supporting workflows, which are Configuration

and change management, Process management and Environment. In addition, there are six core process workflows in the RUP (Business modelling, Requirements, Analysis & design, Implementation, Test and Deployment) instead of five in the Unified Process.

The Business modeling workflow involves modelling the context of a system. The Requirements workflow discusses capturing and managing requirements and designing a user interface. During the Analysis and design workflow, the requirements are translated into specifications (Kruchten 2000, 171). The Implementation workflow includes activities for decomposing a system into subsystems, implementing components, unit testing and integration of the whole system. The Test workflow involves testing the quality of a system and stabilizing the architecture. Finally, the Deployment workflow includes activities such as packaging, distributing, installing, training and testing. In conclusion, the Requirements workflow represents the requirements engineering in the RUP. Next, we will discuss the Requirements workflow in more details.

2.3.2 Requirements workflow

The Requirements workflow starts at collecting requests and wishes from the customers, marketing and other project stakeholders. The requests are used to develop a vision document that contains the needs of the stakeholders and users and the high-level features of a system. The high-level features express the services that must be delivered by a system. The features must be translated into detailed requirements, which are captured in a use case model and supplementary specifications. The supplementary specifications include the requirements that do not fit well in the use cases.

A glossary, a user-interface prototype and a use-case storyboard are developed in parallel with other requirement activities. A glossary defines common

terminology. A user-interface prototype and a storyboard describe a user interface and are built to act as feedback mechanism. Requirement attributes are used during the whole process to manage changing requirements. Like in the Unified Process, a use case diagram including use cases and supplementary specifications constitute the software requirements document. In addition, a glossary is included in the document.

The Requirements workflow consists of six stages, which are iterated in a cycle. The stages are Analyse the problem, Understand stakeholder needs, Define the system, Manage the scope of the system, Refine the system definition and Manage changing requirements (FIGURE 11; Rational Software Corporation 2001, 12):

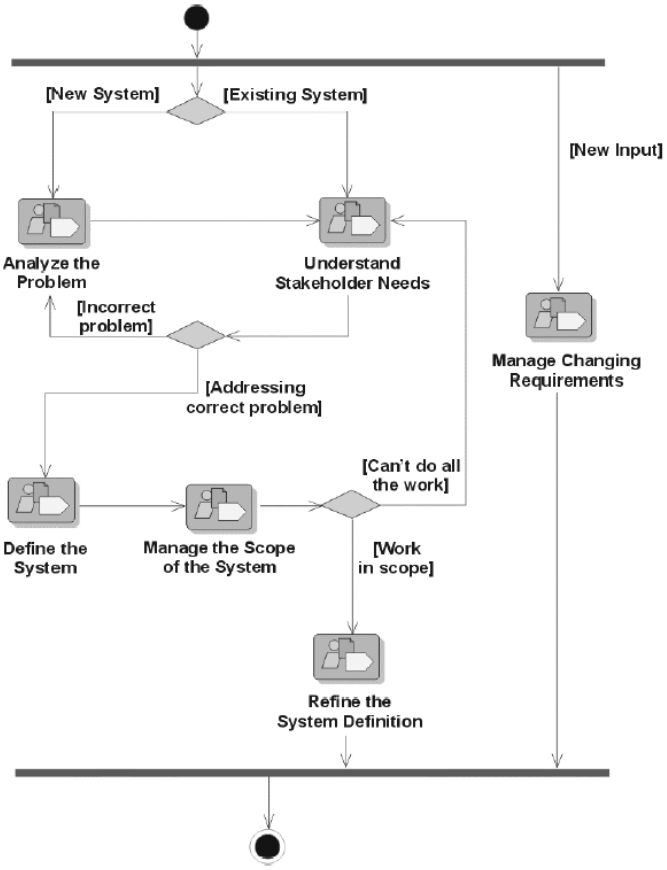

FIGURE 11. The Requirements workflow of the Rational Unified Process (Rational Software Corporation 2001, 12).

During the stage *Analyze the problem*, an agreement of a problem is written on a statement. Stakeholders, boundaries and constraints of the system are identified and a requirements management plan including a business case is developed. The stage *Understand stakeholder needs* includes the usage of different elicitation techniques to gather stakeholder requests. The stage *Define the system* involves establishing a set of desirable features. The criteria for prioritizing the features should be determined and use cases and actors ought to be identified. The stage *Manage the scope of the system* involves activities for ensuring that the system can be delivered on time and on budget and still meet the expectations of the stakeholders. The stage involves collecting important information from stakeholders and maintaining them as requirements attributes. The requirement attributes are used for prioritising and defining a suitable set of requirements.

During the stage *Refine the system definition*, the requirements for a system are detailed using a use case model and a supplementary requirements list. The stage includes activities for developing a user interface prototype and a storyboard. The stage *Manage changing requirements* is performed during the whole development process. During the stage requirement attributes are used to control changes in requirements.

Comparison of the Requirements workflows of the Unified Process and the RUP are summarized in TABLE 2. The evaluation is based on the information obtained from the Rational Software Corporation (2001, 12-20), Kruchten (2001, 163-165) and Jacobson et al. (1999, 114-117). Firstly, candidate requirements are listed during four stages of the Requirements workflow of the RUP. Understanding of the system context is obtained during the stage Analyse the problem.

Functional requirements are captured during all of the stages. Actors and use cases are found during three stages (Analyse the problem, Understand stakeholder needs and Define the system). Use cases are prioritized during the

stage Manage the scope of the system. Use cases are detailed and a user-interface is prototyped during the stage Refine system definition. A use case diagram is structured during the stage Manage changing requirements. Finally, functional requirements are captured in supplementary specifications during five stages (Analyse the problem, Understand stakeholder needs, Define the system, Manage the scope of the system and Refine the system definition).

TABLE 2. The correspondences between the Requirements workflows in the Unified Process (Jacobson et al. 1999) and in the Rational Unified Process (Kruchten 2001).

Requirements workflow in the Unified Process (Jacobson et al. 1999)	**Requirements workflow in the RUP (Kruchten 2001)**
List candidate requirements	Analyse the problem
Understand system context	Understand stakeholder needs
Capture functional requirements	Define the system
- Find actors and use cases	Manage the scope of the system
- Prioritize use cases	Refine the system definition
- Detail a use case	Manage changing requirements
- Prototype user-interface	
- Structure use case model	
Capture non-functional requirements	

All in all, the Requirements workflow of the RUP includes all the stages and activities of the Requirements workflow of the Unified Process. In addition, the RUP includes also other functions (see e.g. Rational Software Corporation 2001) and uses more artefacts than the Unified Process.

2.4 Summary

In this chapter we have investigated the requirements engineering on many levels. The requirements engineering has been given a different position and context in the different approaches and methods of systems development. In order to get a clear picture of requirements engineering, we investigated the process and artifacts of the RE of the two main approaches: the traditional

systems development (Sommerville 1998) and the UML. The later has been described in the forms of the Unified Process (Jacobson et al. 1999) and the Rational Unified Process (Kruchten 2001).

In the traditional systems development the analysis phase stands for requirements engineering. In the Unified Process and in the RUP the Requirements workflows represent the RE.

The use case technique consists of a use case model and a procedure to model use cases. In this chapter we have described the stepwise procedure. In the next chapter we will present the use case model in the UML and the Unified Process.

3 USE CASE TECHNIQUE

Ivar Jacobson introduced the use case technique in 1992 in conjunction with the Objectory method (Jacobson et al. 1992). Ever since, use case modelling has become a popular and widely used technique for capturing and describing the functional requirements for a system. The concept of a use case has been integrated in the UML, which has emerged as the dominant modelling language in systems development. Still, the use case technique has its fair share of critics.

The technique has two parts; a use case model and a procedure to model use cases. The use case model consists of use case diagrams and use case descriptions. In Chapter 2, we introduced the stepwise procedure to model use cases in the Unified Process and in the RUP. In this chapter, we will first introduce the use case model in the UML 1.5 and the Unified Process. After that, we will examine the pros and cons of the technique.

3.1 Use case model

In this section, we will present the use case model as it is defined in the UML 1.5 (OMG 2003; see also Booch, Rumbaugh & Jacobson 1999) and in the Unified Process (Jacobson et al. 1999). According to OMG (2003, 2-137) a *use case* is a kind of classifier, which represents a coherent unit of functionality offered by a system, and is defined as follows:

> The use case construct is used to define the behaviour of a system or other semantic entity without revealing the entity's internal structure. Each use case specifies a sequence of actions, including variants, which the entity can perform, interacting with actors of the entity.

The definition has several significant parts. First of all, use cases are used to capture the intended behaviour of a system without having to specify how the

behaviour is implemented. To be more precise, use cases are used both for specification of external requirements and for specification of the functionality offered by a system. In the first case, use cases are used to model the context of a system and involve drawing a line around the whole system. In the second case, use cases are used to specify the desired behavior of a system. In addition, use cases also indirectly state the requirements the specified system poses on its users, i.e. how they should interact so that the system will be able to perform its services.

Secondly, a use case may describe the whole system or only part of it, e.g. a subsystem, a class or a user-interface. Thirdly, a use case describes sequences of actions, where every sequence represents interaction between a system and the outside world. Fourthly, use cases may include different variants of sequences of actions, for example alternative sequences, exceptional behavior and error handling. A *use case instance* is the performance of a sequence of actions included in a use case. An explicitly described use case instance is called a *scenario*. Usually, there are *primary scenarios*, which define essential sequences of action, and *secondary scenarios*, which define alternative sequences.

Lastly, an actor describes the word outside the system. According to UML 1.5 (OMG 2003, 3-97) an actor is defined as follows. 'An *actor* defines a coherent set of roles that users of an entity can play when interacting with the entity. An actor may be considered to play a separate role with regard to each use case with which it communicates.' Actors may be human beings, devices or other systems. A use case does something that is valuable to an actor and must contain a tangible and coherent amount of work.

A use case represents a functional requirement and describes what a system does, but does not specify how it does it. Well-structured use cases denote only essential system or subsystem behaviour and are neither too specific nor overly general. In the Unified Process, besides defining the functionality of the system,

use cases are seen to provide a way to come to a common understanding about the system to be built. In addition, use cases are used in analysis and design phases to validate the system architecture and to verify the system as it evolves during the development. In the implementation phase the use cases are realized by collaborations. In the testing phase, the use cases used as the basis of the test cases.

Use cases are modeled in a *use case diagram*, which shows the relationships among use cases within a system or other semantic entity and their actors (FIGURE 12). Use cases may be related to other use cases by generalization, extend and include relationships. A *generalization* relationship means that the *child* use case inherits the behaviour and features of the *parent* use case, and may add new features and associations. An *include* relationship signifies that a *base* use case contains the behaviour of an *addition* use case, whereas an *extend* relationship implies that an *extension* use case may extend the behaviour described in a *base* use case under certain conditions. A use case may include an *extension point*, which is a reference to a location within a use case at which the action sequences from other use cases may be inserted.

An *association* between a use case and an actor states that the use case and the actor communicate with each other. A *generalization* from an actor to another actor indicates that the *child* actor can play the same roles as the *parent* actor and thus communicate with the same kinds of use cases. Realization of a use case may be specified by a set of *collaborations*, which define how use case instances interact to perform the sequences of action. In addition, use cases may be grouped into *packages,* which define a coherent set of functionality.

A use case diagram is presented as follows. A use case is rendered as an ellipse, which will be given a name. An optional stereotype keyword may be placed above the name and a list of properties inserted below the name. Because a use case is a classifier, it may also have compartments displaying attributes and

operations. Extension points may be listed in a compartment with the heading 'Extension points'. Each extension point has a unique name within a use case, and a description of a condition and location within the behavior of the use case. The location of an extension point is given usually as text, but it can also be given in other forms, like as a name of a state in a state machine or as a pre- or post-condition.

An actor is rendered as a stick figure with the name below the figure. A name of an abstract actor, as well as of an abstract use case, may be shown in italics. Extend and include relationships between use cases are shown by dashed arrows with an open arrowhead from the use case providing the extension to the base use case, or from the base use case to the included use case. An arrow is labeled with a keyword, <<extend>> or <<include>>. A condition for an extend relationship may be presented close to the keyword. A generalization relationship between use cases is indicated by a generalization arrow, i.e. a solid line with a closed, hollow arrowhead pointing at the parent use case.

An association between an actor and a use case is rendered as a solid line that may have end adornments such as multiplicity. A generalization between actors is shown by a generalization arrow pointing at the more general actor. Use cases may be included in a named rectangle, which represents the system boundary. A package is rendered as rectangle with a name on and a box in the upper corner. Collaboration is modeled with an ellipse drawn with a dashed line. Lastly, an interface between a use case and its collaboration is presented by a realization dependency, i.e. by a dashed arrow with a closed, hollow arrowhead associated with the keyword <<realize>>.

As an example of a use case diagram we present Invoicing and Payment System in FIGURE 12 (cf. OMG 2003, 3-99-100; Jacobson et al. 1999, 146). The actors Salesperson and Customer have associations with the use cases Place order, Confirm order and Pay invoice. The use case Place order includes use cases

Supply customer data, Order Product and Arrange Payment. In addition, use case Request catalog extends the use case Place order if the condition 'the salesperson asks for the catalog' is valid. The extension point for the use case Request catalog is called 'additional requests' and is performed in the location 'after creation of the order'.

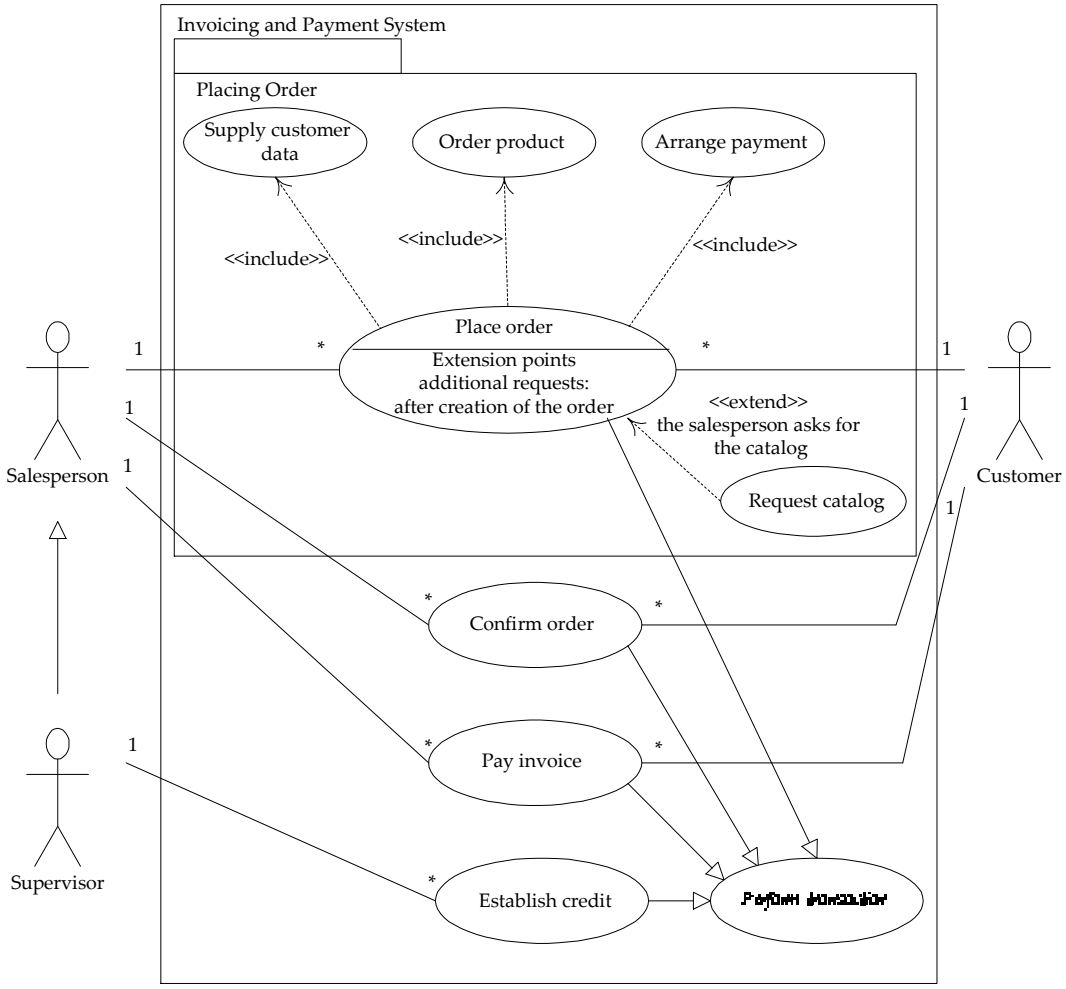

FIGURE 12. Use case diagram for an Invoicing and Payment System (cf. OMG 2003, 3-99-100; Jacobson et al. 1999, 146).

The uses cases are included in a package called Placing order. The actor Salesperson generalizes the Actor Supervisor and has an association with the use case Establish credit. The abstract use case Perform transaction generalizes

the use cases. The uses cases belong to a system called Invoicing and Payment System.

Besides the use case diagram, the use cases are described using other forms. These forms include textual representations, sequence diagrams, activity graphs, state machines and collaboration diagrams. Different textual description types are informal text, structured descriptions and pseudo code. Usually, use cases are detailed in a *use case description*. As an example of a use case description, we present the use case Pay invoice (see FIGURE 13). The description has four parts. First, a precondition is presented. After that, a basic path and alternative paths standing for the primary and secondary scenarios are described. Lastly, a post-condition is given.

Precondition: The customer has received the goods or services ordered and at least one invoice from the system. The customer now plans to schedule the invoice(s) for payment.
Flow of events
Basic Path
1. The customer invokes the use case by beginning to browse the invoices received by the system. The system checks that the content of each invoice is consistent with order confirmations received earlier (as part of the Confirm Order use case) and somehow indicates this to the customer. The order confirmation describes which items will be delivered, when, where, and at what price.
2. The customer decides to schedule an invoice for payment by the bank, and the system generates a payment request to transfer money to the salesperson's account. Note that a customer may not schedule an invoice for payment twice.
3. Later, if there is enough money in the customer's account, a payment transaction is made on the scheduled date. During the transaction, money is transferred from the customer's account to the salesperson's account, as described by the abstract use case Perform Transaction (which is used by Pay Invoice). The customer and the salesperson are notified of the result of the transaction. The bank collects a fee for the transaction, which is withdrawn from the customer's account by the system.
4. The use case instance terminates.
Alternative paths
In Step 2, the customer may instead ask the system to send an invoice rejection back to the salesperson.
In Step 3, if there is not enough money in the account, the use case will cancel the payment and notify the customer.
Postcondition: The use-case instance ends when the invoice has been paid or when the invoice payment was cancelled and no money was transferred.

FIGURE 13. Use case description for the use case Pay invoice (cf. Jacobson et al. 1999, 156-157).

3.2 Use case technique - pros and cons

The use case technique introduced by Jacobson et al. (1992) has become a popular and widely used technique for capturing and describing the functional requirements of a system. In the literature, a large variety of use case approaches have been proposed. Still, the use case technique has its fair share of critics. In this section, we will first investigate the pros and after that the cons of the technique.

In the literature, use cases have been introduced into several purposes. The use cases have been deployed in systems development, human-computer interaction and business process re-engineering. They have been applied into several phases of systems development (analysis, design, testing and maintenance). Moreover, use cases have been deployed e.g. for object identification, test case generation, project scheduling, user-interface design, and user manual documentation (e.g. Barnard 2001, 19; Jacobson & Christerson 1995; Rumbaugh 1994). Use cases have been introduced in several representation formats including use case diagrams, textual formats, sequence diagrams, state machines and Petri nets.

In the literature, a variety of benefits from applying the use case technique has been reported. As stated before, use cases are regarded as a powerful tool for identifying, capturing and validating the functional requirements for a system (e.g. Jacobson & Christerson 1995). In addition, use cases are easy to present and understand (Lee, Cha & Kwon 1998, 1115; Juric & Kuljis 1999, 69; Berard 1996, 3). Therefore, they provide the stakeholders a forum to communicate with each other (e.g. Lamsweerde & Willemet 1998, 1089; Siau, Lee & Korhonen 2001, VIII-2; Weidenhaupt, Pohl, Jarke & Haumer 1998).

The use case technique reduces complexity by offering scalability and focusing on one specific aspect at a time (e.g. Weidenhaupt et al. 1998; Lee & Xue 1999,

92; Lee et al. 1998, 1115). In addition, the technique supports requirements traceability throughout the design and implementation (e.g. Lee et al. 1998, 1115; Juric & Kuljis 1999, 69) and assist in managing change and evolution (Antón et al. 2001, 63). Use cases help to identify the objects of a system (Berard 1996, 3; Juric & Kuljis 1999, 69) and contribute to business process modeling (Juric & Kuljis 199, 69). Last but not least, richness and informality are the benefits gained from the textual use cases (Antón et al. 2001, 63; Lamsweerde & Willemet 1998, 1089).

In conclusion, it is justifiable to say that use cases act as bridges between stakeholders, between requirements and architectural and implementation components, between the phases of system development, between the structural and behavioral aspects of a system and, lastly, between RE and BPR (cf. Weidenhaupt et al. 1998).

Nevertheless, the use case technique has its weaknesses. The shortcomings of the technique, as well as the lack of a clear and complete explanation and insufficient examples of use cases in the Objectory method (Jacobson et al. 1992) and in the UML 1.1 documentation have resulted in different interpretations of use cases among object-oriented practitioners (e.g. Juric & Kuljis 1999, 69; Anda, Sjøberg & Jørgensen 2001, 403; Ratcliffe 2001, 366). Our survey of the literature shows extensive differences in how the use cases are defined and used. Moreover, Dobing and Parsons (2000, 29-31) find differences in the content, format, comprehensiveness, level of detail, and role of the use cases (see also Berard 1999, 1; Cockburn 1998a).

The lack of standardised and precise definitions and of formal semantics offering too free an interpretation of the use case concept is said to be one of the drawbacks of the use case driven requirements engineering (see e.g. Dano et al. 1997, 80). For instance, difficulties have been identified in determining what must be included in a use case and what must be part of another use case (see

also Lilly 2000; Rosenberg & Kendall 2001; Dano et al. 1997, 80; Vadaparty 2000). Moreover, like the well-known panel discussion on use cases at OOPSLA '98 (Cockburn & Fowler 1998) brings out, even the experts disagreed about the usage, content and relationships of the use cases.

The use case approaches introduced in the literature as well as the faced critics have contributed to the use case definition in the UML (see also Jacobson 1995). Initially, Jacobson et al. (1992, 154) defined a use case as 'a special sequence of related transactions performed by an actor and the system in dialogue' and used uses, extends and inherits relationships between the use cases.

According to Simons (1999), in the UML 1.1 the use case concept had two relationships, 'uses' and 'extends', which were formally defined as a kind of inheritance between use cases. Because of the conceptual problems that this caused and the fact that developers completely disregarded the semantics in how they deployed use cases, the uses and extends relationships were replaced by the include and extend relationships in the UML 1.3 (Fowler 1999, 211-212; Simons 1999, 197-198, 200). Simons (1999, 200) points out that use case modellers commonly used the extend relationship to indicate exceptions even though Jacobson et al. (1992) originally expected the extend relationship to be able to characterise insertions, exceptions and alternatives. Still, Simons (1999, 200) comments that it is clear that the extend relationship can only stand for the first of these.

Another problem with the use case technique is the fact that the use cases are functional oriented instead of object-oriented in nature (e.g. Roberts 1999b, 3-4; Juric & Kuljis 1999, 69; Siau et al. 2001, VIII-2; Berard 1996, 4). That is why the stakeholders must be careful when using use cases in the context of the object-oriented systems development. For example, Berard (1996, 5) argues that the use cases should not be used as a basis for the creation of an object-oriented architecture. Also Dobing & Parsons (2000, 29) discuss the lack of object-

orientation in the use case technique, and even present a theoretical argument that use cases may not, in fact, be necessary or valuable in the UML because of their functional nature (see also Siau et al. 2001).

Simons (1999, 199-200) states that the use case technique encourages wrong kind of conceptualisation leading to missed logical dependencies in analysis. Use cases hide an extraordinary complexity in the flow of control, which must be completely deconstructed in order to avoid disastrous logical program structures. Moreover, Mitchell & Lecoeuche (1997, 52) point out that the mapping of the use cases into the dynamic models in less clear. Because of their functional orientation, use cases are inherently task focused and partial, fostering a kind of tunnel vision in analysis and design (e.g. Lamsweerde & Willemet 1998, 1089; Simons 1999, 202). The partitioning of the structural and behavioral characteristics of a system into individual use cases leads to granularity (Dutoit & Peach 2002, 4).

Because of the single path nature of the use cases, it is hard to determine the completeness. This can lead to developers using their imagination to complete exception handling cases or rarely taken paths and causes the risk of overspecification. (Roberts 1999b, 3-4; Lamsweerde & Willemet 1998, 1089.) In addition, the relationships between use cases (Lee & Xue 1999, 92) and the control-flow semantics of use cases are not very well defined (McLeod 2002, 2; van den Berg & Simons 1999, 651). Thus, there are no systematic approaches for analysing dependencies among use cases and to detecting flaws resulting from the use case interactions (Lee et al. 1998, 1115-116).

The fragmentation of the object information across use cases leads to lack of encapsulation (Dobing & Parsons 2000, 33; Juric & Kuljis 1999, 69; Roberts 1999b, 3-4). A complete use case model may not offer a cohesive picture of the structural and behavioral characteristics of the objects in the domain. Instead, these characteristics may be spread over several use cases. This leads to the

fragmentation of information needed to construct class definitions across use cases and therefore to the lack of integration between the use cases and class models.

Also the temptation to violate information hiding results from the functional orientation of use cases (Berard 1996, 5-9; Dobing & Parsons 2000, 36). Information hiding is the process of making certain pieces of information inaccessible. Several authors (e.g. Roberts 1999a; Roberts 1999b, 3-4; Siau et al. 2001, VIII-2; Dobing & Parsons 2000, 36) state that use cases should not include design or implementation decisions, because that could be a possible barrier to effective design. Roberts (1999a) even argues that it is not possible to have a good user-interface interaction design and usability together with use cases.

Moreover, the textual format has its drawbacks. The textual use cases do facilitate the communication between the stakeholders, but increase the risk of ambiguity, inconsistency and incompleteness of the description. Many authors (e.g. Lee et al. 1998, 1115-116; Dano et al. 1997, 81) state that it is important to use a more structured and formal technique in order to avoid the problems with the natural language.

Moreover, McLeod (2002) and Arlow (1998) identify problems in using the use case technique for the BPR. McLeod (2002, 2) points out that in the use case technique there is no adequate means to show what is computerised or manual in a business process. In addition, there are no means to express timing or synchronisation and therefore, it is difficult to show parallelism explicitly. Arlow (1998, 151) stresses that the use case technique is good at capturing what is important to the actors, but use cases are not good at capturing requirements generated by the business context or business axioms arising from the context. Business axioms define the rules for interacting with a system and how the system reacts, but they are not interactions themselves.

Lastly, Arlow (1998, 151-152) emphasises that it is difficult to identify and prioritise specific, individual requirements. Although detailed functional requirements are captured in use cases, they remain embedded within the use case description rather that being explicitly stated. Typically, a relationship between requirements and use cases is many-to-many. One requirement may be part of several use cases, and one use case may cover several requirements.

In the literature, some other drawbacks of the use case technique have also been introduced. For instance, Lee and Xue (1999, 92) underline problems in handling the non-functional requirements.

3.3 Summary

In this chapter we have discussed the use case technique. First, we defined the key concepts of the use case model in the UML 1.5 and the Unified Process and described the notations with an illustrative example. After that, we discussed the pros and cons of the technique based on the literature. The multiplicity of the use case approaches results from the shortcomings and insufficient definitions of the use cases. In addition, the variety arises from the evolution of the use case technique and the employment of the use cases into new application domains and purposes. The use case technique consists of two parts, the use case model and the procedure to model use cases. The stepwise procedure was presented in Chapter 2 in conjunction with the requirements workflows.

The use case technique has been greatly influenced by the philosophy and advancements of its originator, Jacobson, who welcomes all use case approaches and attempts to formalize the technique (see Jacobson 1995).

According to Jacobson (1995, 10-11) use cases can be formalized as far as possible if not violating three rules, which define what cannot be expressed in a

use case diagram. Firstly, a use case diagram cannot express internal communication among instances inside the system. That is because use case instances are the only types of objects inside a system and therefore, they cannot communicate with one another. Secondly, conflicts among use case instances cannot be modeled because relationships between use cases represent internal details and should be expressed in object models. Thirdly, concurrency cannot be modeled, because use cases are atomic, serialized slices of a system.

Moreover, according to Jacobson (1995, 10) a use case diagram answers the questions 'what' and 'how'. That is, a use case diagram is used as an outside view of a system e.g. in RE and maintenance. In addition, it is regarded as an inside view of a system during analysis, design and test phases.

In the next chapter we will present a framework for analysing the use case approaches.

4 USE CASE CLASSIFICATION FRAMEWORK

In systems development literature, a variety of use case classification frameworks have been introduced. The multitude of the frameworks includes wider and more compact frameworks (e.g. Cockburn 1997a). In this chapter, we will describe three significant use case classification frameworks. They are a framework for classifying approaches for describing and formalizing use cases by Hurlbut (1997), a scenario classification framework by Rolland et al. (1998) and a representational framework for scenarios of system use by Anton & Potts (1998).

We will compare the frameworks and decide, which of them is the most suitable one for emphasising the characteristics and concepts of and the differences and similarities between the use case approaches from the different viewpoints of requirements engineering. We will modify the chosen framework to get it more appropriate for analysing the use case approaches. In our discussions, we consider the terms use case and scenario as synonyms and use the initial terms presented in the articles.

4.1 Framework for classifying approaches for describing and formalizing use cases

Hurlbut (1997) presents a framework for classifying approaches for describing and formalizing use cases. That is, for classifying use case approaches and related techniques. The elements of each approach are classified from two perspectives. The first perspective is the use case formalism focus, and the second perspective is the format of the use case representation. The formalism focus categories are static, dynamic, policy and process, and the presentation format categories are textual, graphical and dynamic. The categories are further divided into properties. Comparison of the use case approaches is presented in

tables, which denote the presence of some measure of meaningful coverage by each approach. Relative merits are not indicated.

The *focus* perspective has been partitioned into four categories of properties: static, dynamic, policy, and process. The *static* properties are concerned with the static aspects of the use case model elements. Inheritance, aggregation, dependency and refinement are familiar object-oriented concepts and discuss the relationships between the elements of a use case model. Data is concerned with the structure of information that is exchanged between a stakeholder and a system. Context relates to the world outside the system.

Most of the *dynamic* properties correspond to the UML 1.1 semantics. State, event and action are model elements of the UML 1.1 that relate to state machines. Contracts/patterns and roles are concepts that relate to the UML 1.1 collaboration model. On the other hand, transactions and agents are not well addressed in the UML 1.1 specification. Transactions are larger aggregations of interactions that may provide a basis for the decomposition of a use case. Agents add mentalist concepts such as beliefs and goals, which are usually associated with workflows and business processes.

Policy properties relate to business rules and are subdivided into three types. Behavioural rules deal with invariants and pre- and post-conditions. Exception handling makes a distinction between alternative courses of action and exceptional courses of action, including failure. Finally, composition relates to a functionality that is included. The last general category, *process*, is concerned with the development process and its participants. Evolution is the key basis for the process comparison. It involves e.g. history of the evolution and metrics for measuring the impact of change. Parameterization is closely related to evolution, since its intent is to provide future variability. Users addresses those approaches that elevate the importance of the user's perspective. Finally, tools refers to the approaches that include automated tool support.

The formalism *formats* perspective has several types of representations for use cases: textual, graphical and dynamic. Although each of the use case approaches can be fitted into one or more of these categories, the rich variety of representational schemes merits special attention. To start with, there are five *textual* formats. Unstructured text narratives and semi-structured scripts are purely textual formats. Structured descriptions refer to a form that is used as a template. Tabular formats are similar to a database table, such as for state-event matrices. Finally, formal language expressions are regular expressions in a well-defined syntax.

The *graphical* formats category includes structure diagrams, state diagrams, interaction diagrams and implementation diagrams similar to those defined in the UML 1.1 notation guide. One additional type of diagram, rule diagram, is included for graphical representation of rules. Finally, there are two kinds of *dynamic* properties. Assembly, visualization, and navigation of use case model elements are properties that require automated tool support. On the other hand, role playing, referring e.g. to CRC cards, and storyboards properties do not need automated tool support.

4.2 Scenario classification framework

Rolland et al. (1998) propose a four-dimensional scenario classification framework that includes form, content, purpose and life cycle views (see FIGURE 14). The form view represents the expression mode of a scenario. The contents view describes the knowledge expressed in a scenario. The purpose view captures the role that a scenario is aiming to play in the requirements engineering process. Lastly, the life cycle view sees scenarios as artefacts existing and evolving in time and discusses the issue of scenario manipulation.

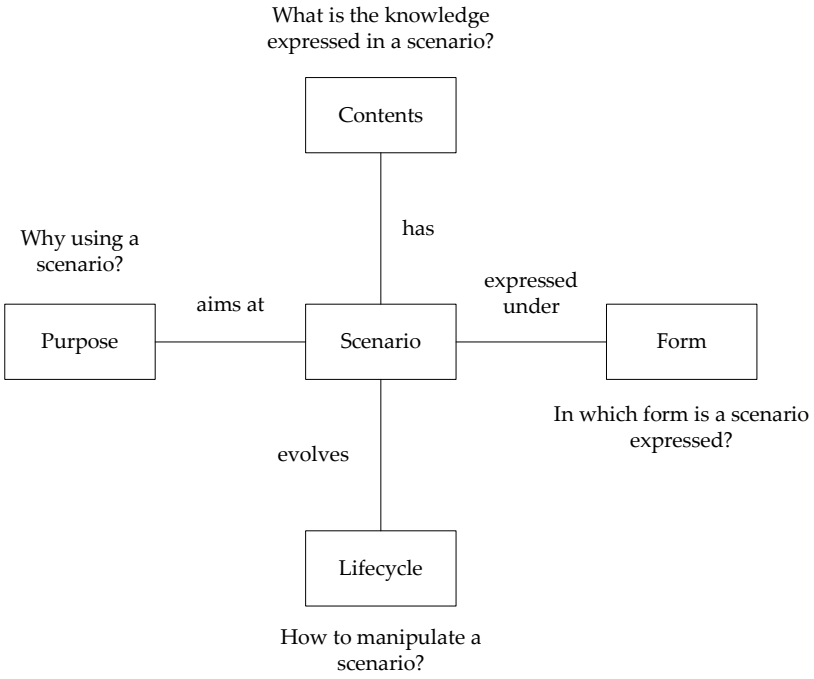

FIGURE 14. The four views on scenarios (cf. Rolland et al. 1998, 24).

The dimensions have multiple facets, which are viewpoints suitable to characterize and classify a scenario. Each facet is measured by a set of relevant attributes. The attribute values are defined within a domain. A domain may be a predefined type (Integer, Boolean ..) , an enumerated type (ENUM {a, b, c}) or a structured type (SET or TUPLE).

For instance, the description facet of the form view is measured with two attributes, medium and notations. The notations attribute is defined on the enumerated type {formal, semi-formal, informal} whereas the medium attribute is defined on a set type, whose elements belong to an enumerated type {text, graphics, image, video, software prototype}. Thus, a scenario approach might be positioned within the description faced with the (attribute: value) pairs: medium: {text, image} and notations: informal. Next, we will take a closer look at the four views.

Firstly, the *form view* describes the form in which the scenario is expressed. It includes description and presentation facets. The *description facet* consists of two attributes, medium and notations. The former defines the medium used for the description of the scenarios whereas the latter captures the formality level of the notations used for the description. Text is a very common medium, as many times scenarios are associated with narrative stories. There are also variants of text, i.e. tables, structured text, and scripts. Other media include graphics, image, video and software prototypes. In addition, scenarios are described using more or less formally defined notations. In many cases a modelling language providing semi-formal and formal notations is used to capture descriptions. Thus, the attributes of the description facet are:

- Medium: Set (Enum {Text, Graphics, Image, Video, Software prototype})
- Notations: Enum {Formal, Semi formal, Informal}

The *presentation facet* of the form view describes the interactivity the scenarios provide and the presence of animation in the scenarios. Thus, it has two attributes, animation and interactivity. A scenario may be visualized in an animated mode or it may be a static scenario displayed in texts or diagrams. Animated presentations benefit from the animation techniques to highlight the expected behaviour of a system and meet validation requirements as the animation enhances the communication and understanding between different stakeholders. In contrast, static presentations are used in manual methods. Interactivity means the capability offered to a user to control the way the scenario progresses through time. Basic interactivity includes different functions such as stopping the animation. In addition, hypertext links are another means of planning interactivity. The presentation facet is as follows:

- Animation: Boolean
- Interactivity: Enum {None, Hypertext-like, Advanced}

The *contents view* discusses the content of a scenario and includes coverage, abstraction, context and argumentation facets. The *abstraction facet* deals with

the content of a scenario, which may be concrete, abstract or a mix of different degrees of abstraction or concreteness. Concreteness helps in eliciting requirements, because people react to 'real things'. Therefore, many current scenario-based approaches define scenario contents at the instance level, which may be costly. This is done with instance scenarios, also called concrete scenarios.

Type scenarios, which are also called abstract scenarios, are more abstract than instance scenarios: the entities are not entity instances, such as customers' names, but entity types, such as 'customer'. It is still probable, that an individual scenario will contain different levels of abstraction, and thus have mixed abstraction. In conclusion, each scenario is classified according to the abstraction facet using three attributes, which may be true, if the approach accepts scenario instances, scenario types or scenarios, which contain both the instance and type information. The abstraction facet is defined as follows:

- Instance: Boolean
- Type: Boolean
- Mixed: Boolean

The *context facet* classifies scenarios according to the amount of the contextual information they capture. It has four attributes: system internal, system interaction, organisational context, and organisational environment. A scenario may describe merely internal behaviour and have a glass box view of a system. Sometimes scenarios see a system as a black box and therefore focus on a description of a system interaction with its environment.

Descriptions of the organisational context in scenarios aim to describe the application domain knowledge. That may include the knowledge on the stakeholders, such as goals, but also deal with the structure of an organisation and relations to business processes and resources. Furthermore, a scenario may involve organisational environment, but that is not commonplace.

Organisational environment includes external factors such as legislation, economics and history that influence the organisation itself. Thus, the following attributes characterise the scenario-based approaches:

- System internal: Boolean
- System interaction: Boolean
- Organisational context: Boolean
- Organisational environment: Boolean

The *argumentation facet* discusses argumentation about system features, agents, roles or activities that can be expressed within a scenario. The facet has four attributes, positions, arguments, issues and decisions, which are adapted from the well-known IBIS model (Conklin & Bagemann 1988; Conklin, Burges & Yakemovic 1991). Positions mean descriptions of alternative solutions to a problem, arguments involve opinions for objecting or supporting a given position, issues means descriptions of problems or conflicts, and finally, decisions expresses choices of a particular position. Thus, the attributes are as follows:

- Positions: Boolean
- Arguments: Boolean
- Issues: Boolean
- Decisions: Boolean

The *coverage facet* classifies scenarios according to the type of the information they capture. It has three attributes: functional, intentional and non-functional. The facet must be regarded as orthogonal to the context facet. That means that the coverage may be applied either to the context or to the internal of a system or both. The functional attribute is further decomposed into structure, function and behavior according to the standard classification of the functional aspects in the systems development community (e.g. Iivari 1991). The intentional attribute captures goals, problems, responsibilities, opportunities, causes and

goal dependencies of the stakeholders. These issues may be elements of scenario content in rich scenarios.

Some scenarios include non-functional extensions. The non-functional attribute is further decomposed into e.g. performance, time constraints, cost constraints, user support, documentation, and backup/recovery. The types are adapted from the requirements specification model (RSM), which subsumes the non-functional requirements stated in 21 international standards and guidelines (Pohl 1996). Thus, the attributes of the coverage facet are associated with the following domains:

- Functional: Set (Enum {Structure, Function, Behaviour})
- Intentional: Set (Enum {Goal, Problem, Responsibility, Opportunity, Cause, Goal dependency})
- Non-functional: Set (Enum {Performance, Time constrains, Cost constraints, User support, Documentation, Examples, Backup/Recovery, Maintainability, Flexibility, Portability, Security/Safety, Design constraints, Error handling})

The *purpose view* captures the role a scenario plays in the requirements engineering process. The authors identify three roles: descriptive, exploratory and explanatory. Descriptive scenarios are mainly used for capturing requirements. Exploratory scenarios serve the purpose of exploring and evaluating different solutions for satisfying a given system requirement. These scenarios make explicit the link between the solutions and requirements. Lastly, explanatory scenarios provide detailed illustrations of situations and their rationale. In conclusion, the Purpose view classifies scenarios with three attributes:

- Descriptive: Boolean
- Exploratory: Boolean
- Explanatory: Boolean

The *life cycle view* considers scenarios as artefacts evolving and existing in time. Thus, it looks upon scenarios from a dynamic perspective instead of static. It is concerned about how scenarios are captured, evolve and are improved. The authors agree that the most existing scenario-based approaches deal with the static aspects, but some approaches are also concerned with the dynamic aspects. The life cycle view has lifespan and operation facets.

The *lifespan facet* distinguishes between transient scenarios, which are used as temporary items, and persistent scenarios, which persist over time. Persistent scenarios exist as long as the documentation of the project they belong to. That means that scenarios are a part of a requirement specification, or project documentation keeps track of scenarios. Transient scenarios are thrown away after being used and are meant to support some RE or design activities. The scenarios could function e.g. as temporary support for requirements validation. In conclusion, the facet is described with one attribute:

- Lifespan: Enum {Transient, Persistent}

The *operation facet* classifies scenarios according to the kinds of operations they have been carried out. Thus, this facet is concerned with how scenarios are captured, evolve and are transformed during the requirements engineering process. The operations are classified into capture, integration, refinement, expansion, and delete operations. The capture operations discuss the generation of scenarios. Usually scenarios are generated from scratch, but also capture by reuse has been proposed. The integration operations aim at integrating scenarios, which are initially produced as fragmentary pieces of details. For instance, narrative story types of scenarios are fragmentary in nature.

The refinement operations restructure scenarios without increasing their content. They aim at transforming scenarios to make them more reusable or easier to understand. The expansion operations add new knowledge in

scenarios. The delete operations terminate a scenario's lifespan. If a scenario is just supporting the discovery of the requirements, deletion may occur immediately after use. If a scenario is a part of the requirements specification, it has the lifespan of that document. Thus, the operation facet has the following attributes:

- Capture: Enum {Fromscratch, Byreuse}
- Integration: Boolean
- Refinement: Boolean
- Expansion: Boolean
- Deletion: Boolean

4.3 Representational framework for scenarios of system use

Antón and Potts (1998) propose a representational framework for scenarios. The framework is developed for understanding what different scenario approaches are and what they are for. The framework lays out a space of representational properties for scenarios and enumerates a set of criteria that are important for different uses of scenarios. The authors agree that there are no good locations per se, there are only good locations for different purposes. Thus, the discussion of criteria amounts to a discussion of purpose.

Next, we will investigate the proposed framework in detail. FIGURE 15 illustrates a graphical summary of the framework. The representational criteria are arrayed around the figure, and the terms inside the figure represent the dimensions. The locations of the dimension terms indicate approximately the relevance of the criteria to the dimensions. Next, we will shortly discuss the criteria. After that, we will go on with the representational dimensions, which are of our primary interest.

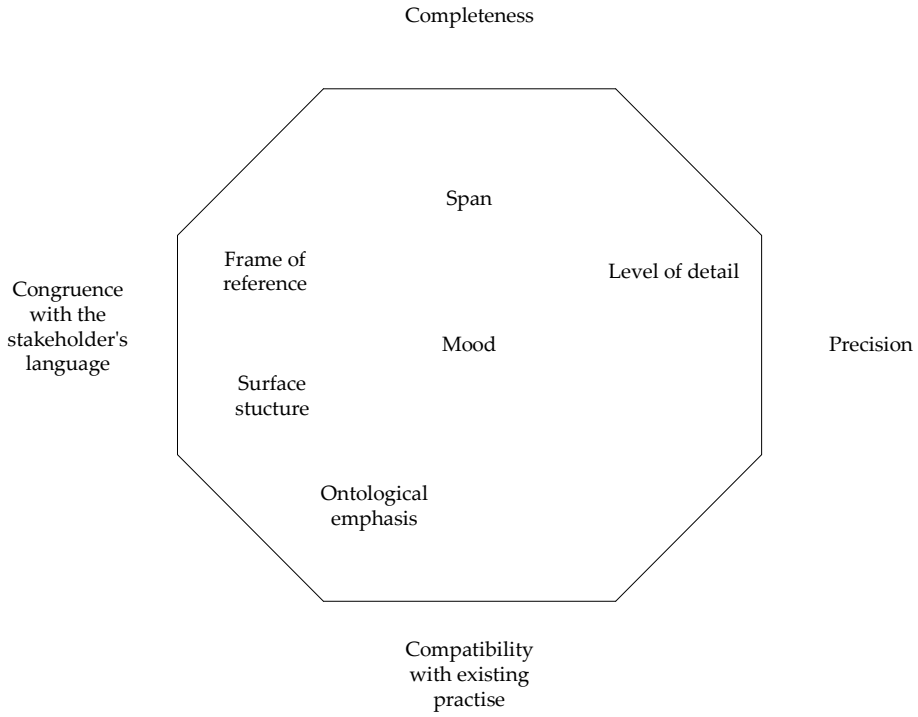

FIGURE 15. The representational framework for scenarios of system use (Antón & Potts 1998, 221).

The framework consists of four criteria, according to which the effectiveness of scenario representations is evaluated. The criteria are congruence with the stakeholder's language, completeness, precision and compatibility with existing practise. One of the goals of the scenario usage is to clarify the implications of a system design for different stakeholders so that they can comment on the adequacy of the design. Thus, a scenario must be understandable and have *congruence with the stakeholder's language*. The stakeholder's language has multiple layers. Lexical layer refers to terminology, syntax stands for principal grammatical categories, semantics signifies the meanings of the scenario content, and finally, pragmatics refers to how language is used in concrete communicative situations.

Secondly, c*ompleteness* is a relative measure of salience or representativeness. Scenarios are inherently incomplete and illustrate example behaviours. Some scenarios seek to achieve greater completeness than others. That is done

through precision and association with systematic methods. Sometimes, the completeness is not critical. Thirdly, *precision* is the absence of ambiguity in the semantics of a representation. Precision is necessary for some purposes, whereas for others imprecision may be desirable. Finally, it is advantageous to have *compatibility with existing practise*. There is pressure to use familiar and well-established representations. Sometimes organisations can easily mix and match scenario representations, but in other cases scenario representations have grown together with other approaches. Next, we will go on with the representational dimensions.

The representational dimensions form a multidimensional space of features that together define what scenario representations are. The dimensions are ontological emphasis, surface structure, span, level of detail, frame of reference, and mood. The ontological emphasis dimension is divided into four categories; two of the categories are further divided into sub-categories. The dimensions, categories or sub-categories are described in text. Next, we will shortly describe each dimension. After that, we will discuss the dimensions in more detail.

The ontological emphasis dimension discusses the assumed nature of the reality being modelled in a scenario. The surface structure dimension handles the appearance of a scenario and the span describes the temporal continuation. The level of detail dimension discusses the detail of the behavior and the frame of reference deals with the perspective from which the scenario is viewed. Finally, the mood dimension handles the type of behavior described. For example, a scenario could be described to have 'natural language descriptions and directed graphs' as surface structure and 'end-to-end from user initiation' as span.

The *ontological emphasis* is divided into four categories: temporality, agency, teleology, and normativeness. The first one, *temporality*, presents different assumptions about the nature of time in the scenarios and is divided into sub-categories. The sub-categories are duration, linearity, determinacy and

triggering. *Duration* handles the questions, if the temporal constituents of the scenario, e.g. events and actions, occur during time intervals or at precise moments. The constituents may also form a total or partial ordering and may overlap. *Linearity* discusses the time line of the scenario, which may be branching or linear. In the first case, alternative courses of action or possible futures are allowed. The alternative courses may recombine in a directed graph or continue to diverge in a tree structure.

Determinacy discusses the branching model. If the conditions leading to the choice of alternative paths are fully specified, the model is deterministic. If the conditions are specified in terms of probabilities, the model is stochastic. Finally, if the conditions are unspecified, the model is non-deterministic. The last temporality category is *triggering*. It handles the question, if the temporal constituents of the scenario (events, actions etc.) are initiated as the result of some other specified factors. The factors may be conditions that hold at the time, events that occurred immediately beforehand or the passage of time.

The second category of the ontological emphasis is *agency*. It involves the interaction of entities that are being referred as agents and is divided into openness and identity sub-categories. The *openness* sub-category discusses the agents in a scenario. In a closed-world model, the agents are all that exist. In an open-world model, there are two sets of agents: those that are deterministic and those that are not. The deterministic agents are usually those that are to be implemented. The non-deterministic agents are given by the system's environment. The system boundary is found by drawing a border around the deterministic agents.

The agents may represent structural or functional entities. This subject is discussed in the *identity* sub-category. In the real world, one functional entity often maps onto several structural entities and vice versa. Separate individuals

are examples of structural entities, and organisational and work process models represent functional entities.

The third category of the ontological emphasis dimension is *teleology*. Scenarios are classified as mechanistic, purposive, or mixed-purposive depending on whether goals are ascribed to none, all or some of the agents that interact in it. Lastly, a scenario may describe behaviours that are desirable or undesirable. This subject is discussed in the *normativeness* category of the ontological emphasis dimension. The label of the desirable or undesirable behaviours may be present in the scenario representation itself or normative and exceptional scenarios may be presented similar, and the differing in practise may be in the names or labels attached to them. In the first case erroneous and incomplete actions and undesirable or exceptional situations are lexically or syntactically distinguished.

Surface structure means the way the ontology presents itself to the scenario reader. Different surface structure categories are natural language text, tabular representation, directed graphs and storyboards. Many scenario representations employ natural language text. The differentiating feature is the role that the natural language plays. On one extreme, a scenario may be a natural language narrative prose document. In systems development, scenarios generally have some formalism and natural language is used to provide informal specification.

In tabular representations, the series of rows determines the temporal sequence of the narrative. The columns may represent different facets of the global state or simultaneous behaviour of several entities. Tables cannot show partial ordering of events, although annotations or superimposed graphics can group potentially overlapping event sequences. To go on with, directed graphs are a famous representation formats. In the graphs, arcs represent precedence relations between events or states, and branching usually stands for alternative courses of action. Lastly, storyboards segment behaviour into several discrete

frames, where each frame uses a mixture of text and pictures. They are widely used in human computer interaction.

A scenario may describe a short or a long *span* of behaviours. Sometimes scenarios encapsulate conceptually complete behaviours and describe end-to-end transactions. In the first case a scenario represents a logically complete behavioural sequence starting with some external event or input, and ending with system outputs that are final response to that event. In other cases, a scenario may describe fragments of end-to-end transactions that are useful when composing longer scenarios of reusable components.

The *level of detail* dimension refers to the degrees of behavioural complexity and agent refinement in a scenario. Usually, detailed scenarios may not span as much as the higher-level scenarios. The *frame of reference* dimension refers to the behavioural view of the scenario. In the Cartesian view, a scenario is viewed from a frame of reference outside system as a whole. A directed graph is an example of a Cartesian view. In a turtle view the behaviour is viewed by an entity participating in the scenario. The turtle view may be expressed concretely, e.g. as storyboard, but it could equally be presented as more abstract narrative. In practise, scenario representations mix frames of reference.

Lastly, following Jackson (1995) the concept of *mood* is used to reflect distinctions between actual, required or possible behaviour. Thus, scenarios may describe existing (indicative), required (optative), or proposed (subjunctive) behaviour. Any scenario representation may be used to express all of the three moods. Still, different representation formats are more suitable to describe some moods than others.

4.4 Comparison of the frameworks

To find the most appropriate framework for categorizing and evaluating the use case approaches in this study, we will evaluate the suitability and effectiveness of the proposed frameworks. Because our intent is to help understand the achievements gained from recently developed use case approaches and especially indicate where and how use cases are deployed, we need a framework, which emphasises the differences and similarities between the use case approaches. Additionally, it is necessary that the framework brings out essential aspects of the requirements engineering and covers broadly the concepts of the development world and system world (cf. Chapter 2).

Next, we will first compare the categorizing concepts of the frameworks. Secondly, we will assess the coverage of the three dimensions of the requirements engineering framework. Thirdly, we will discuss the requirements criteria manifested in the frameworks. Fourthly, we will evaluate the frameworks through the concepts of the development world and system world. Lastly, we will select the most suitable framework.

4.4.1 Categorizing concepts

Each of the frameworks is established with a pre-defined set of categorizing concepts (FIGURE 16). To start with, Hurlbut (1997) has two perspectives, format and formalism focus, which are divided into three categories. The formalism focus perspective discusses mainly different object-oriented concepts in the categories static, dynamic, policy and process. The categories have properties, which are measured with Boolean values.

Secondly, Rolland et al. (1998) have four views that are divided into facets. The facets have attributes, which are measured within a domain. There is one exception to the rule. The purpose view does not have facets, but it does have

attributes. A domain may be a predefined type, an enumerated type or a structured type. Lastly, Antón and Potts (1998) have six dimensions that relate to the four representational criteria. Some of the dimensions are divided into categories and even sub-categories. The values are represented in text; usually there is no a fixed domain of values. Based on the article, it seems like the normativeness and agency categories as well as the surface structure, frame of reference and mood dimensions have a fixed domain of values.

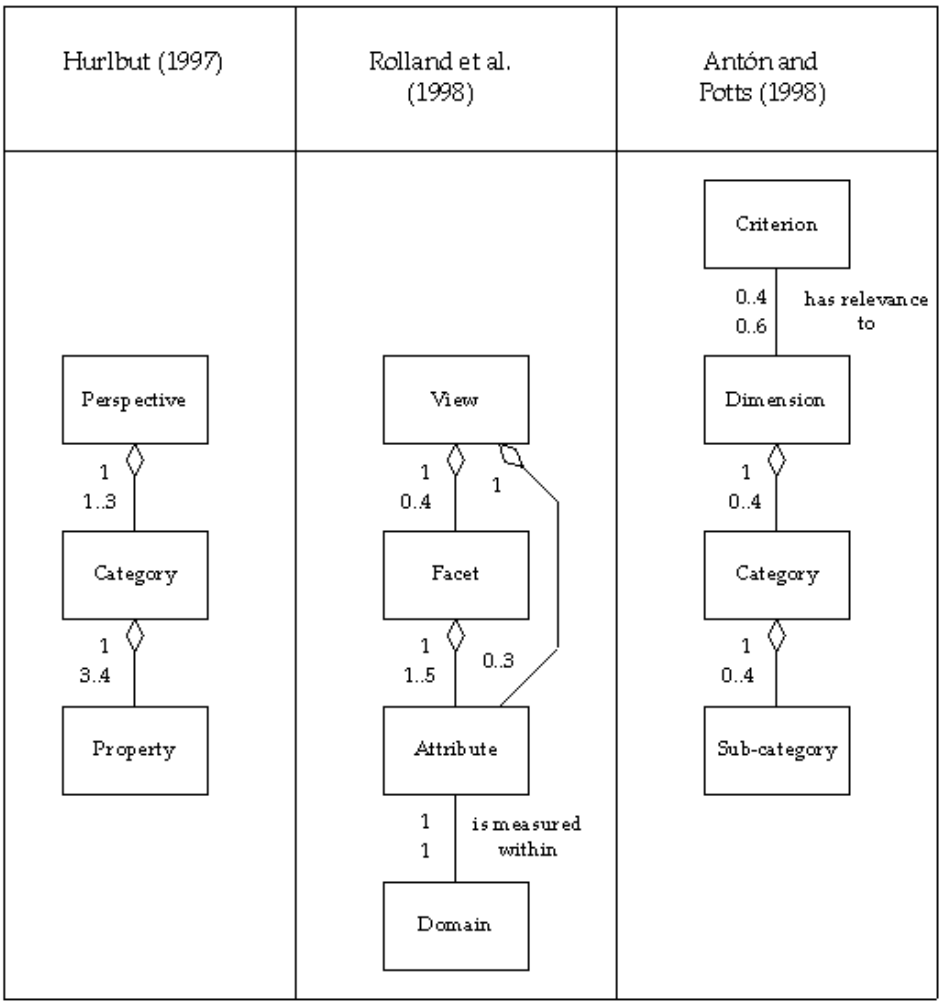

FIGURE 16. The categorizing concepts in the frameworks.

Antón and Potts (1998) have representational criteria that do not have correspondence in the other frameworks. On the other hand, perspective (Hurlbut 1997), view (Rolland et al. 1998) and dimension (Antón & Potts 1998)

are corresponding concepts. Also category in Hurlbut (1997) and facet in Rolland et al. (1998) have a correspondence. The same applies to the property (Hurlbut 1997) and attribute (Rolland et al. 1998) concepts. The difference between them is that the former gets Boolean values while the second is measured within a domain.

The framework of Antón and Potts (1998) is slightly different. Each categorizing concept is presented in text; usually there is no predefined domain. There is a minor correspondence between a category in Antón and Potts (1998) and a category in Hurlbut (1997) as well as facet in Rolland et al. (1998). The same applies to a sub-category in Antón and Potts (1998) and a property in Hurlbut (1997) as well as an attribute in Rolland et al. (1998).

4.4.2 Coverage of the three dimensions of the RE framework and the requirements criteria

The frameworks may be investigated through the three dimensions of the requirements engineering framework (see TABLE 3). The categorizing concepts that correspond to the dimensions are listed; a full coverage is marked with italics. As discussed in Chapter 2, Pohl (1994) describes requirements engineering through three dimensions, specification, representation and agreement, and a trace of the RE process. The specification dimension represents the completeness of the requirements description. The representation dimension concerns the formality level of the representation. The agreement dimension represents the difference between a common agreement and personal views. Lastly, the requirements engineering process is a trace in the space formed by these dimensions.

Firstly, Hurlbut (1997) discusses representational issues by the format perspective. The perspective deals with different representation formats, but does not divide use cases according to the formality level as the three

dimensions of the RE framework does. The evolution and parameterization properties of the focus perspective represent the RE process. The issues of the specification and agreement dimensions are not covered.

TABLE 3. The frameworks investigated though the three dimensions of the requirements engineering framework.

Pohl (1994)	**Hurlbut (1997)**	**Rolland et al. (1998)**	**Antón & Potts (1998)**
Specification	-	Contents view	*Span & Level of detail dimensions, Completeness & Precision criteria*
Representation	Format perspective	Form view: Description facet: *Notations attribute, M*edium attribute, Presentation facet	Surface structure & Frame of reference dimensions
Agreement	-	-	-
RE process	Focus perspective: Dynamic category: Evolution & Parameterization properties	Life cycle view	-

Secondly, the framework of Rolland et al. (1998) is, in the most visible way, based on the three dimensions of the requirements engineering framework. The framework has four views, form, contents, purpose and life cycle, relating to the representation, specification, and agreement dimensions and the requirements engineering process. The specification dimension does not have a direct correspondence in the framework, but the issue is dealt with in the contents view. The representation dimension corresponds to the notations attribute of the description facet, but has also a linkage to the medium attribute and the presentation facet. The purpose dimension and the agreement view do not have a straight correspondence. The requirements engineering process is represented by the life cycle view.

Finally, in Antón and Potts (1998) the specification dimension is covered by the span and level of detail dimensions, which relate to the completeness and precision criteria. The formality level of the use case representation is not

discussed, but the framework deals with other representational issues through the surface structure and level of detail dimensions. The agreement dimension and congruence with the stakeholders' language criteria have a weak correspondence. In order to have agreement, it is good to have congruence with the stakeholders' language. The connection is so weak that we do not include it in the table. The requirements engineering process is not at all covered.

In conclusion, the framework of Rolland et al. (1998) has the broadest coverage for the three dimensions of the requirements engineering framework (Pohl 1994). The framework has a linkage to three of the four dimensions of the RE framework. The other frameworks cover merely two dimensions of the RE framework.

The frameworks can also be examined and compared by relating them to the eight requirements criteria presented in the Chapter 2. The question is, if a framework brings out the criteria, which are to be fulfilled by a use case. As discussed before, the requirements criteria are validity, consistency, completeness, realism, verifiability, comprehensibility, traceability, and adaptability.

The framework of Antón and Potts (1998) involves four representational criteria. Three of the representational criteria relate to the requirements criteria. The completeness criterion is included in the both frameworks. Secondly, the precision criterion has a linkage to the completeness criterion, but also has a slight connection to the verifiability. Lastly, the congruence with the stakeholders' language criterion is related to the comprehensibility criterion. Merely the compatibility with existing practise criterion does not have a correspondence in the requirements criteria.

In the other frameworks, the correspondence to the requirements criteria is more difficult to define. That is because in these frameworks there are no

corresponding criteria or concepts to be found. Nevertheless, there is at least one notable connection. In Hurlbut (1997), the format perspective has a linkage to the comprehensibility, and in Rolland et al. (1998), comprehensibility is an issue of the form view.

Thus, the framework introduced by Antón and Potts (1998) has the most visible coverage for the requirements criteria. Still, it covers only three of the eight criteria. We may also ask, if the other requirements criteria could have been presented in the framework. That question is not an issue in this study and thus we leave the question unanswered.

4.4.3 Concepts of the development and system worlds

As discussed in Chapter 2, NATURE Team (1996) divides systems development into four worlds: usage world, subject world, system world and development world. The system world represents the system and models the subject and usage worlds. The development world stands for the systems development process. It has become commonplace in the systems development community to structure the worlds though a systems architecture. Sowa and Zachman (1992) extend and formalize the framework for systems architecture introduced by Zachman (1987). The framework provides taxonomy for relating concepts that describe a system and its implementation. We apply this framework architecture both at the system world and at the development world, and use it for comparing and evaluating the three use case classification frameworks.

The framework architecture is divided into entities, functions, locations, people, times, and motivations corresponding to a classification scheme based on the English interrogatives what, how, where, who, when, and why. The authors use the terms data, function, network, people, time, and motivation for these concepts. In this study, we use the terms object (O), action (An), location (L), actor (Ar), time (Ti) and purpose (P). In addition, we add one concept, tool (To),

which is often included in the systems architecture and responds to the question 'with what'. Next, we will describe the concepts of the development world and system world from the viewpoint of this study.

The object of systems development is the system itself. In this study, the object is a part of a system represented by use cases. The action concept refers to use case modelling. The location is the place the modelling occurs, and actor stands for a modeller. The time concept deals with the temporal dimension of the modelling, and the purpose concept stands for the rationale. Lastly, the tool concept refers to devices used in the modelling.

In the system world, i.e. in the world a use case refers to, the concepts have the following meanings. The object concept stands for the object of the behaviour modelled in a use case. The action describes the behaviour, and location is the spatial context of the behaviour. The actor concept corresponds to the actor concept at the system world. The time concept denotes the time dimension of the behaviour. The purpose describes the reasons for the action and finally, the tool concept refers to devices used in the action.

We present the results of the analysis of the frameworks in tables to indicate counterparts among the concepts of the frameworks. The first columns represent the development world, i.e. the development of a use case, and the latter columns describe the system world, i.e. a use case. The notation '- - - X - - -' indicates that the whole world is concerned, although on a general level.

To start with, the framework introduced by Hurlbut (1997) is analysed according to the concepts of the development world and system world (TABLE 4). The associations concept includes the inheritance, aggregation, dependency, and refinement properties. All of these properties concern the whole system world. The textual and graphical format properties are not listed separately because all of them deal with the object concept of the development world.

TABLE 4. The framework introduced by Hurlbut (1997) analysed according to concepts of the development and system worlds.

Hurlbut (1997)	**Development world**							**System world**						
	O	An	L	Ar	Ti	P	To	O	An	L	Ar	Ti	P	To
Focus														
Static														
Associations								-	-	-	X	-	-	-
Data								X						
Context								-	-	-	X	-	-	-
Dynamic														
State								X	X					
Event									X			X		
Action									X					
Contracts/patterns								-	-	-	X	-	-	-
Roles											X			
Transactions								-	-	-	X	-	-	-
Agents											X		X	
Policy														
Behavioral rules									X			X		
Exception handling									X			X		
Composition									X					
Process														
Evolution	X	X			X									
Parameterization	X	X			X									
Users											X			
Tools							X							
Format														
Textual	X													
Graphical	X													
Dynamic														
Assembly	X						X							
Visualization	X						X							
Navigation	X						X							
Role playing	X													
Storyboards	X													

The static, dynamic and policy categories of the focus perspective discuss the concepts of the system world. In other words, they deal with the contents of a use case. The format perspective and the process category of the focus perspective deal with the development of a use case. There is one exception to the rule. The property user covers the actor concept at the system level. The coverage shows that the framework is grouped into categories unevenly. The system world is covered better than the development world.

The framework introduced by Rolland et al. (1998) is analyzed according to the concepts of the development world and system world (TABLE 5). The views and facets are listed; the attributes and domains are not included. That is because the view and facet concepts are sufficient to represent the coverage. Again, there is one exception to the rule. The coverage facet could be divided into the functional, intentional and non-functional attributes. The functional attribute represents the action and object concepts, and the intentional attribute covers the purpose concept. Lastly, the non-functional attribute deals with the object, location, actor, time, purpose, and tool concepts.

TABLE 5. The framework introduced by Rolland et al. (1998) analysed according to the concepts of the development and system worlds.

Rolland et al. (1998)	**Development world**							**System world**						
	O	An	L	Ac	Ti	P	To	O	An	L	Ac	Ti	P	To
Form														
Description	X													
Presentation	X						X							
Contents														
Abstraction								-	-	-	X	-	-	-
Context								-	-	-	X	-	-	-
Argumentation													X	
Coverage								-	-	-	X	-	-	-
Purpose	X					X								
Life cycle														
Lifespan	X				X									
Operation	X	X												

The form, purpose, and life cycle views deal with the development world whereas the coverage facet covers the system world. Thus, the framework has a clear structure. The development world is nearly totally covered; merely the actor concept, i.e. the use case modeller, and the location concept, i.e. the place of the use case modelling, are not discussed. The system world is concerned in a general level.

The framework of Antón and Potts (1998) is analyzed according to the concepts of the development and system worlds (TABLE 6). As we can see, the

framework concentrates on describing the system world. Merely the surface structure dimension deals with the development world; it represents the object concept. Even though the name of the framework, 'a representational framework for scenarios of system use', refers to representational issues, it is obvious that the framework covers also other issues.

TABLE 6. The framework introduced by Antón and Potts (1998) analysed according to the concepts of the development and system worlds.

Antón & Potts (1998)	**Development world**							**System world**						
	O	An	L	Ar	Ti	P	To	O	An	L	Ar	Ti	P	To
Ontological emphasis														
Temporality														
Duration												X		
Linearity												X		
Determinacy									X			X		
Triggering									X			X		
Agency														
Openness								-	-	-	X	-	-	-
Identity								-	-	-	X	-	-	-
Teleology													X	
Normativeness									X					
Surface structure	X													
Span									X					
Level of detail								-	-	-	X	-	-	-
Frame of reference								-	-	-	X	-	-	-
Mood									X					

Results from the analysis of the coverage of the concepts of the development world and system world are summarized in TABLE 7. The values are derived from the columns of the tables presented above. The sign X refers to a strong, and the sign * for a weak coverage. Coverage is regarded to be strong, if a concept is covered individually by more than two classification concepts.

The concepts are covered as follows. Hurlbut (1997) covers almost every concept of the development world and every concept of the system world. So does also Rolland et al. (1998), but the coverage for some concepts, especially in

the system world, is a bit weaker. Lastly, Antón and Potts (1998) investigate every concept of the system world, but merely the object concept in the development world.

TABLE 7. Summary of the analysis of the concepts of the development world and system world.

	Hurlbut (1997)	**Rolland et al. (1998)**	**Antón & Potts (1998)**
Development world			
Object	X	X	*
Action	*	*	
Location			
Actor	*		
Time	*	*	
Purpose		*	
Tool	X	*	
System world			
Object	*	*	*
Action	X	*	X
Location	*	*	*
Actor	X	*	X
Time	X	*	X
Purpose	*	*	*
Tool	*	*	*

4.4.4 Selection

In this section, we have compared three use case classification frameworks in order to investigate the differences and similarities between them and to choose the most suitable one for analyzing use case approaches. In this study, our intent is to help understand the achievements gained from the recently developed use case approaches and investigate in which way the large variety of the use case approaches cover the requirements engineering. Moreover, we will explore what kind of characteristics, concepts, differences and similarities the approaches have. Particularly, we want to emphasise the purposes for and forms in which the use cases have been proposed.

Because of these requirements, we need a framework, which emphasizes the characteristics and concepts of and the differences and similarities between the

approaches. In addition, it is necessary that the framework brings out the different aspects of requirements engineering and covers broadly the concepts of the development and system worlds.

To start with, we inspected the coverage for the three dimensions of the RE framework, which describes the different aspects of requirements engineering. We noticed that the framework of Rolland et al. (1998) has the broadest coverage for the framework. After that, we found out that Antón and Potts (1998) emphasize the requirements criteria.

Last but not least, we investigated the coverage of the concepts of the development world and system world. We found out that both Hurlbut (1997) and Rolland et al. (1998) have a broad coverage for the concepts of the development and system worlds. Rolland et al. (1998) deals with the concepts on a general level. Hurlbut (1997) discusses the concepts in more details and actually in many cases on the same level as we do when modelling meta-models of the use case approaches in Chapter 5. Because we want to carry out an overall analysis of the approaches, it is favourable to choose a framework that covers the concepts on a more general level.

During our survey, we noticed that the framework of Rolland et al. (1998) has a clear structure. The contents view covers the concepts of the system world, whereas the other views deal with the development world. The purpose view discusses the purpose aspect, and the form view the deals with the formats of the use cases. In addition, the framework of Rolland et al. (1998) is widely recognized in the field of RE. It has been used, for instance, in a well-known survey of scenarios in industrial projects (Weidenhaupt et al. 1998; see also Ralyté & Achour 1997).

Based on these arguments, we choose the framework of Rolland et al. (1998) to be used in analyzing the use case approaches. The framework fulfils two of the

three objectives. It brings out different aspects of requirements engineering and covers broadly the concepts of the development and system world. In addition, the framework emphasises the purposes and forms of the approaches. To investigate the third objective of the capability to highlight the characteristics and concepts of and the differences and similarities between the approaches, it is necessary to test the framework. In the next section we make some modifications to the framework.

4.5 Modified framework

In this section, we will modify the scenario classification framework (Rolland et al. 1998) to provide a more suitable foundation for analysing use case approaches presented in the literature. The modifications are based on the analysis presented above as well as on our survey of current use case approaches in the literature. The modifications are made to the form, contents and purpose views. Next, we will firstly present our modifications to the scenario classification framework. After that, we will shortly discuss the process aspect.

In the literature, a variety of representation formats for use cases has been proposed. The multitude of the formats includes e.g. different diagrams and textual representations. In the framework, the medium attribute distinguishes between text, graphics, image, video, and software prototype. Based on our literature survey and the two other frameworks, we sub-divide the text and graphics to highlight the differences between the use case approaches.

We divide text into narrative text, script, structured description, table, and formal language expression. Narrative text is a story-like narration and used e.g. in the UML 1.5 (OMG 2003). Scripts describe a collection of system entities, each of which provides a set of well-defined services that may be used by other entities (e.g. Rubin & Goldberg 1992). Structured descriptions refer to a form

that is used as a template (e.g. Cockburn 1997b; Regnell, Kimbler & Wesslen 1995). Tabular formats are similar to a database table (e.g. Potts, Takashi & Antón 1994) and finally, formal language expressions are regular expressions in a well-defined syntax (e.g. Feijs 2000). The last one represents a formal notation, whereas scripts, structured descriptions and tables are semi-formal descriptions. Narrative text is an informal notation.

Following the terminology of the UML 1.5 specification (OMG 2003), graphics are further divided into use case diagram, collaboration diagram, sequence diagram, state machine, and activity graph. In addition, we add Petri net, and storyboard. A Petri net represents a formal notation, whereas the other diagram types and storyboard are semi-formal descriptions. Lastly, image, video and software prototype are representation formats that are preserved in the framework even though they are rarely deployed in the literature. They represent informal notations. The medium attribute is defined as follows:

- Medium: Set (Enum {Narrative text, Script, Structured description, Table, Formal language expression, Use case diagram, Collaboration diagram, Sequence diagram, State machine, Activity graph, Petri net, Storyboard, Image, Video, Software prototype}).

A use case approach may use several representation formats, in other words, it may gain several values in the domain of the medium attribute. The notations attribute represents the formality level of the medium attribute, and is defined as enum {Formal, Semi formal, Informal}. Because an approach may represent several formats, we modify the notations attribute to be able to include several values. Thus, the notations attribute is as follows:

- Notations: Set (Enum {Formal, Semi formal, Informal})

As discussed above, a use case may describe a desirable or a undesirable behaviour (Antón and Potts 1998). The label of the desirable or undesirable behaviour may be present in the scenario representation itself. On the other

hand, normative and exceptional use cases may be presented in the similar way with the only difference being the practise in the names or labels attached to them. During our survey, we noticed that it is commonplace that use cases include both the desirable and undesirable behaviour. Still, there are use case approaches that describe merely desirable behaviour. Thus, we add the normativeness facet to the coverage view to distinguish between the desirable and undesirable behaviour. The domain is derived from Antón and Potts (1998). The *normativeness facet* is:

- Set (Enum {Desirable, Undesirable})

To go on with, the coverage facet of the contents view distinguishes between the functional, intentional and non-functional attributes. The domain of the non-functional attribute is divided into several values including e.g. performance and time constraints. However, even Rolland et al. (1998) agree that only a few use case approaches give explicit and extended guidelines about what kind of non-functional requirements should be expressed in a use case. Our survey of use case approaches supports the claim. We did not find any approach that explicitly states what kind of non-functional aspects should or should not be included in a use case. However, the question of whether to include non-functional extensions in a use case or not is clearly expressed in the approaches. Thus, we modify the non-functional attribute of the coverage facet. The attribute is as follows:

- Non-functional: Boolean

The use case approaches are used in different application domains (also called domains or problem domains). An application domain means an organisational and technical context to which an approach, method or a technique applies. An application domain may include, for instance, human artifacts, organizational functions and information structures. Use cases are commonly used in systems development and business process re-engineering application domains (cf. Antón and Potts 1998). The systems development distinguishes between object-

oriented analysis and design, requirements engineering, human computer interaction, and test case generation (cf. phases (traditional RE) and workflows (Unified Process & RUP)) domains. During our survey, we noticed that use cases are also intended for project management purposes. Following the terminology of the RUP (see Chapter 2), we include project management to the systems development application domain.

An application domain can be seen to be related to the purpose concept of the development world, and therefore, be conceptually related to the purpose view. The purpose view is coved by the descriptive, exploratory and explanatory attributes, which denote the role a use case approach represents in RE. As we want to include the application domains into the purpose view, we divide the view into two facets, application domain facet and *role facet*.

Even though we concentrate on RE, the use cases may be used in multiple application domains. Thus, we include all relevant application domains into the framework. The application domains could be further refined. For instance, the requirements engineering domain could be sub-divided into the phases of the RE process described in Chapter 2. The *application domain facet* is defined as follows:

- Systems development: Set (Enum {Object-oriented analysis and design, Requirements engineering, Human computer interaction, Test case generation, Project management})
- Business process re-engineering: Boolean

The use case approaches have different interpretations of the relationship between a use case and a scenario. In some approaches, a scenario is just another name for a use case. In other approaches, a use case consists of scenarios. Based on our survey, it looks like both of the alternatives are applied in the literature. To go on with, in some approaches, either the term use case or the term scenario is used and the other term is not at all employed.

Cockburn (1997a) introduces a minor use case classification framework that has four dimensions: purpose, content, structure, and multiplicity. Three dimensions of the framework are, in one way or another, covered in Rolland et al. (1998). The multiplicity dimension, i.e. the relationships of a use case and scenario, is not at all dealt with. Hence, we include it in the framework. The *multiplicity facet* gets the value true if a use case contains several scenarios. The facet is defined as follows:

- Multiplicity: Boolean

The lifespan facet distinguishes between transient use cases, which are used as temporary items, and persistent use cases, which exist as long as the documentation of the project they belong to. That means that the use cases are either a part of the requirement specification or the project documentation keeps track of the use cases. Transient use cases are thrown away after being used and are meant to support some RE or design activities. In most of the cases, the use case approaches that we investigated during our survey represent persistent use cases. Thus, we remove the redundant lifespan facet from the framework.

As a summary of the discussions above, we present the framework modified from the classification framework proposed by Rolland et al. (1998) in TABLE 8. The modifications are marked with italics.

The framework of Rolland et al. (1998) is product-oriented and intended to classify use case models. Rolland et al. (1998) agree that the systems development community has pointed out the importance of the process aspect in addition to the product aspect (see also Weidenhaupt et al. 1998). However, the process aspect is not included in the framework, because it is seldom discussed in the literature. Rolland et al. (1998) state that even some guidance

TABLE 8. The modified framework from Rolland et al. (1998). The modifications are marked with italics.

Form view
Description
Medium: Set (Enum {*Narrative text, Script, Structured description, Table, Formal language expression, Use case diagram, Collaboration diagram, Sequence diagram, State machine, Activity graph, Petri net, Storyboard*, Image, Video, Software prototype})
Notations: Set (*Enum* {Formal, Semi formal, Informal})
Presentation
Animation: Boolean
Interactivity: Enum {None, Hypertext-like, Advanced}
Contents view
Abstraction
Instance: Boolean
Type: Boolean
Mixed: Boolean
Multiplicity
Multiplicity: Boolean
Context
System internal: Boolean
System interaction: Boolean
Organisational context: Boolean
Organisational environment: Boolean
Argumentation
Positions: Boolean
Arguments: Boolean
Issues: Boolean
Decisions: Boolean
Coverage
Functional: Set (Enum {Structure, Function, Behaviour})
Intentional: Set (Enum {Goal, Problem, Responsibility, Opportunity, Cause, Goal dependency})
Non-functional: Boolean
Normativeness
Normativeness: Set (Enum {Desirable, Undesirable})
Purpose view
Application domain
Systems development: Set (Enum {Object-oriented analysis and design, Requirements engineering, Human computer interaction, Software test case generation, Project management})
Business process re-engineering: Boolean
Role
Descriptive: Boolean
Exploratory: Boolean
Explanatory: Boolean
Life cycle view
Operation
Capture: Enum {Fromscratch, Byreuse}
Integration: Boolean
Refinement: Boolean
Expansion: Boolean
Deletion: Boolean

for the use case development process is provided many methodological concerns such as motivations, situations, and alternative ways of working are not covered. Moreover, they discuss the lack of formalization of the use case model and state that only a few use case approaches provide a meta-model.

In our study, we investigate basically the static aspects of the use cases. Thus, we need a product-oriented framework; the process aspect should not be highlighted. However, it is necessary to emphasize the importance of the process aspect e.g. when surveying use case usage in the industry (see Weidenhaupt et al. 1998, 36). Usually process guidance is presented as process guidelines. Anda et al. (2001a) divide the guidelines into three sets: minor, template and style guidelines. Minor guidelines describe how to identify actors and use cases, but give little direction on how to construct them. Template guidelines include a template for describing actors and a template for describing use cases. Style guidelines are recommendations on how to structure the description of the flow of events.

Thus, if we would survey the use case usage in practice, we would add a guidance facet to the life cycle view to denote the process guidance provided in the approaches. Based on our survey of the use case approaches, it seems like a meta-model is rarely deployed, so it is not reasonable to include a meta-model attribute. The same applies to the alternative ways of working and situations attributes. In conclusion, the process guidance facet could be as follows:

- Guidelines: Set (Enum {Minor, Template, Style})
- Motivations: Boolean

4.6 Summary

In this chapter, we have made a comparative evaluation of three use case classification frameworks. We compared the categorizing concepts of the frameworks and assessed the coverage of the three dimensions of RE

framework. We discussed the requirements criteria manifested in the frameworks and, finally, evaluated the frameworks through the concepts of the development world and system world. Based on the comparison, we selected the framework of Rolland et al. (1998) to be used as a basis for the use case analysis. We modified the framework to get it more suitable for our needs. The modifications were based on the information obtained from our survey of the current use case approaches and the two other frameworks. In the next chapter we will analyze use case approaches with the modified framework and by metamodeling.

5 ANALYSIS

In this chapter, we will analyze use case approaches on two levels. We will first carry out an overall analysis to get a general picture of the use case approaches and to help understand the achievements gained from recently developed use case approaches in the literature. In the analysis, we will apply the classification framework (Chapter 4) as a tool. The overall analysis deals with the concepts of the development and system worlds (cf. Chapter 3). After the overall analysis, we will carry out an in-depths analysis in terms of metamodeling. The aim of the in-depth analysis is to gain a more detailed picture of the recently developed use case approaches. The in-depth analysis discusses the concepts of the system world.

5.1 Overall analysis

In this section, we will carry out an overall analysis of the use case approaches. The analysis is accomplished by analysing fifteen use case approaches according to the framework presented in the previous chapter. First, we will shortly introduce the selected approaches. After that, we will analyse the approaches.

5.1.1 Introduction

During our survey, we have explored and grouped use case approaches presented in the literature. All in all, at least eighty use case approaches have been introduced in the literature (see TABLE 9 for some references to use case approaches and related techniques). In this study, our intent is to help understand the achievements gained from the recently developed use case approaches. Especially, we want to emphasise the purposes for and the forms in which the use cases are have been deployed in RE. Therefore, we have chosen for the analysis use case approaches that represent different purposes and

forms in the field of RE. Moreover, in the next section we analyse use case diagrams in more details. Therefore, we concentrate especially on use case approaches that deploy use case diagrams. The analyzed approaches are marked with italics in the TABLE 9. Next, we will discuss the selection criteria in more details.

TABLE 9. References to use case approaches and related techniques. Selected use case approaches are marked with italics.

Alexander (2002a)	Kösters et al. (2001)
Allenby & Kelly (2001)	van Lamsweerde & Willemet (1998)
Anda et al. (2001a)	Lang & Duggan (2001)
Anda et al. (2001b)	Lee et al. (2001a)
Antón et al. (2001)	*Lee et al. (2001b)*
Barnard (2001)	Lee et al. (1999)
Berg van den & Simons (1999)	Lee et al. (1998)
Berkem (1999)	Leite et al. (1997)
Buhr (1998)	McLeod (2000)
Chou & Chen (2000a)	Leonardi & Leite (2002)
Chou & Chen (2000b)	Li (1999)
Cockburn (1997a)	Lunn & Lindsay (2002)
Cockburn (1997b)	*Lycett (2001)*
Cockburn (2000)	Mitchell & Lecoeuche (1997)
Dano et al. (1997)	Mylopoulos et al. (2001)
Dutoit & Paech (2002)	Phillips et al. (2001)
Feijs (2000)	Potts et al. (1994)
Glintz (1995)	Ratcliffe & Budgen (2001)
Gough et al. (1995)	*Regnell et al. (1995)*
Gross & Yu (2001)	*Regnell et al. (1996)*
Haumer et al. (1998)	Rolland & Achour (1997)
Henderson-Sellers et al. (2002)	Rolland et al. (1998)
Hsia et al. (1994)	Rowlett (1998)
Insfrán et al. (2002)	Rubin & Goldberg (1992)
Jaaksi (1997)	Santander & Castro (2002)
Jaaksi (1998)	Sindre & Obdahl (2001)
Jacobson et al. (1999)	Sutcliffe et al. (1998)
Juric & Kuljis (1999)	*UML 1.5 (OMG 2003)*
Kotonya & Sommerville (1996)	Wade & Hopkins (2002)
Kruchten (2001)	

In this study, we concentrate on RE. Therefore, as the primary selection criterion, we chose use case approaches that have bee crafted for RE. As the secondary selection criterion, every use case approach that has introduced a use case diagram differing from the use case diagram of the UML 1.5 (OMG 2003) or Jacobson et al. (1992) was selected for the analysis. As the third selection

criterion, the approaches were chosen so that they would represent different purposes of use in the field of RE. The fourth selection criterion was the representation format. In other words, the approaches were selected so that they would represent several representation formats. While previewing the use case approaches, we came into a conclusion that fifteen use case approaches discuss these aspects sufficiently.

We have excluded use case approaches that are mainly used in some other application domain or other phase of systems development. For instance, we excluded approaches that deploy use cases in business process re-engineering (e.g. McLeod 2000; Berkem 1999; Lunn & Lindsay 2002; Juric & Kuljis 1999; Santander & Castro 2002) and design phase (see e.g. Rubin & Goldberg 1992; Ratcliffe & Budgen 2001; Rowlett 1998).

A large variety of approaches with weak resemblances to use cases have also been proposed. These approaches are conceptually close to the notion of the use case and deployed for the requirements capture. Also these approaches have been excluded from the analysis. The multitude of these approaches includes e.g. action cases (Chou & Chen 2000a & 2000b), use case maps (Buhr 1998) and models for goal-oriented analysis (Gross & Yu 2001; Mylopoulos, Chung, Liao, Wang & Yu 2001).

The results of the overall analysis are summarized in TABLES 11, 12, 13, 14 and 15. During the analysis, the whole approach is taken into account. In other words, there may be issues, which are not dealt with by the use case representation itself, but other aspects of the approach. For instance, in the RUP the non-functional attribute is covered by a supplementary requirements list. During the classification, references to use case approaches that the analysed approach represents are given. Abbreviations for the attribute domain values that are used in classification are listed in TABLE 10.

TABLE 10. Abbreviations for the attribute domain values.

Attribute	Abbreviation	Domain value
Medium		
	NT	Narrative text
	S	Script
	STD	Structured description
	T	Table
	FLE	Formal language expression
	UCD	Use case diagram
	CD	Collaboration diagram
	SD	Sequence diagram
	SM	State machine
	AG	Activity graph
	PN	Petri net
	SB	Storyboard
	I	Image
	V	Video
	SP	Software prototype
Notations		
	I	Informal
	SF	Semi formal
	F	Formal
Interactivity		
	Hypertext	Hypertext-like
	Adv.	Advanced
Multiplicity		
	?	Indeterminate value
Functional		
	S	Structure
	F	Function
	B	Behavior
Intentional		
	G	Goal
	P	Problem
	R	Responsibility
	O	Opportunity
	C	Cause
	GD	Goal dependency
Normativeness		
	D	Desirable
	U	Undesirable
Systems development		
	A & D	Object-oriented analysis and design
	RE	Requirements engineering
	HCI	Human computer interaction
	TEST	Software test case generation
	PM	Project management
General		
	T	True
	F	False

The analysis is organised as follows. We will first analyse the use case approach of the UML 1.5 and the Unified Process. After that, we will describe and analyse all the other use case approaches in the alphabetical order according to the author information. The section ends with conclusions.

5.1.2 UML 1.5 and the Unified Process (OMG 2003 & Jacobson et al. 1999)

The use case technique of the UML 1.5 and the Unified Process (OMG (2003); see also Jacobson et al. (1999) and Booch et al. (1998) for the UML and the Unified Process) has been described in Chapter 3. Here we present a short summary. In the Unified Process, use cases are used for business process re-engineering, requirements engineering, analysis, design, and test case generation (TABLE 11). In RE, use cases are used to define the behaviour of a system or other semantic entity without revealing the entity's internal structure and to describe interaction with actors. Still, business use cases deployed in business process re-engineering capture also organisational context information.

The use cases are presented as use case diagrams, different textual formats including narrative text, structured descriptions and formal language expressions, activity graphs and state machines. In addition, use case realizations may be specified by collaboration diagrams, and user-interface prototype describes the user interface. Use cases are described at the type level, whereas scenarios representing the sequences of actions are regarded as use case instances. The use cases contain neither intentional aspects nor non-functional requirements. The relationships between use cases are used to sort the use cases.

TABLE 11. Analysisof the use case approaches of the UML 1.5 and the Unified Process (OMG 2003 & Jacobson et al. 1999), Alexander (2002a) and Allenby and Kelly (2001).

	UML 1.5 & Unified Process (OMG 2003 & Jacobson et al. 1999)	**Alexander (2002a)**	**Allenby & Kelly (2001)**
Form view			
Medium	{NT, STD, FLE, UCD, CD, AG, SM, SP}	{T, UCD}	{T, UCD}
Notations	{I, SF, F}	{SF}	{SF}
Animation	F	F	F
Interactivity	None	Hypertext	None
Contents view			
Instance	T	T	T
Type	T	T	T
Mixed	F	F	F
Multiplicity	T	T	T
System internal	F	F	T
System interaction	T	T	T
Organisational context	T	T	F
Organisational environment	F	F	F
Positions	F	T	F
Arguments	F	F	T
Issues	F	T	T
Decisions	F	F	F
Functional	{S, F, B}	{S, F}	{S, F, B}
Intentional	{}	{G, P, C, GD}	{}
Non-functional	F	T	F
Normativeness	{D, U}	{D, U}	{D, U}
Purpose view			
Systems development	{A & D, RE, HCI, TEST}	{A & D, RE}	{RE}
Business process re-engineering	T	F	F
Descriptive	T	T	T
Exploratory	F	T	T
Explanatory	F	F	F
Life cycle view			
Capture	Fromscratch	Byreuse	Fromscratch
Integration	F	F	T
Refinement	T	F	T
Expansion	F	F	F
Deletion	F	F	F

5.1.3 Alexander (2002a)

Alexander (2002a) applies the concept of a misuse case in a trade-off analysis. A misuse case is a use case from the point of view of an actor hostile to a system. Trade-off analysis is a systematic examination of the advantages and disadvantages of each proposed requirement or design approach. The misuse cases may also be deployed e.g. for eliciting functional and non-functional requirements for security, safety and usability (see e.g. Sindre & Obdahl (2001) for misuse cases and security).

The approach is supported by a tool that provides all scenarios for a given use case diagram in a single table and offers links between use cases and related misuse cases (TABLE 11). A use case diagram is constructed showing use cases for goals held by actors, and misuse cases for goals of hostile actors. The special feature of the technique is the attention given to relationships between positive and negative use cases (i.e. use cases and misuse cases), which are regarded as goals in the approach. The article introduces four new relationships between use cases: threatens, mitigates, aggravates and conflicts with relationships. In addition, also the more common includes and has exception relationships are used. The use cases are used as a temporary support for a decision-making. The article proposes requirements reuse through use cases.

5.1.4 Allenby and Kelly (2001)

Allenby and Kelly (2001) propose an approach to conduct a hazard analysis on use case requirements specifications (TABLE 11). The hazard analysis provides a mechanism for associating safety related functional requirements into a use case in order to improve the safety of a system. Use cases are first identified and corresponding scenarios are documented in tables. After that, use case diagrams are decomposed from the scenarios and defined at the system level. The system level use case diagrams include the use case decomposition, their relationships and associations to actors and subsystems. The results from a

failure identification and interpretation of failures, i.e. arguments and issues, are recorded into an analysis table.

5.1.5 van den Berg and Simons (1999)

Van den Berg and Simons (1999) present an approach to analyze control-flow semantics of use cases (see also McLeod (2000) for control flow of use cases). The flow of control within each use case is derived from a sequence diagram or corresponding collaboration diagram, and mapped onto state machines (TABLE 12). The approach distinguishes between five types of use cases: common, variant, component, specialized and ordered use cases. A new relation, precedes-relation, between use cases is introduced. The approach models merely system internal behaviour. The term scenario is not at all deployed.

5.1.6 Cockburn (1997a, 1997b)

Cockburn (1997a, 1997b, see also 2000) introduces a goal-based requirements analysis technique (TABLE 12). The goal information is valuable e.g. in identifying, organizing, and justifying functional and non-functional requirements. (see also Antón, Carter, Dagnino, Dempter, & Siege (2001); Rolland, Souveyet & Achour (1998) and Lamsweerde & Willemet (1998) for goal-oriented approaches). Cockburn (1997a, 1997b) emphasizes the role of goals in project management, project tracking, staffing, and business process re-engineering. The approach offers two use case templates, i.e. structured descriptions, and a general form for a use case sentence (see also Leonardi, Sampaio & Leite (2002) and Wade & Hopkins (2002) for structured descriptions).

Three dimensions of goal refinement are distinguished. The first dimension involves the system scope or boundary and distinguishes between system function use cases and business use cases. Thus, use cases may be describe

system interaction and/or organisational context. The second dimension of refinement is goal specificity. It consists of three types of goals; summary goals, user goals and sub-functions, which have relationships with each other. The third type of goal refinement is the interaction detail, which distinguishes between the dialog and semantic interfaces.

TABLE 12. Analysis of the use case approaches in van den Berg and Simons (1999), Cockburn (1997a, 1997b) and Dano et al. (1997).

	van den Berg & Simons (1999)	**Cockburn (1997a, 1997b)**	**Dano et al. (1997)**
Form view			
Medium	{CD, SM, SD}	{STD}	{T, UCD, SM, PN}
Notations	{SF}	{SF}	{SF, F}
Animation	F	F	F
Interactivity	None	None	None
Contents view			
Instance	F	T	T
Type	T	T	T
Mixed	F	F	F
Multiplicity	?	T	F
System internal	T	F	T
System interaction	F	T	T
Organisational context	F	T	F
Organisational environment	F	F	F
Positions	F	F	F
Arguments	F	F	F
Issues	F	F	F
Decisions	F	F	F
Functional	{S, F, B}	{S, F, B}	{S, F, B}
Intentional	{}	{G, GD}	{}
Non-functional	F	T	F
Normativeness	{D, U}	{D, U}	{D, U}
Purpose view			
Systems development	{RE}	{RE, PM}	{RE}
Business process re-engineering	F	T	F
Descriptive	T	T	T
Exploratory	F	F	F
Explanatory	F	F	F
Life cycle view			
Capture	Fromscratch	Fromscratch	Fromscratch
Integration	F	F	F
Refinement	T	F	T
Expansion	F	F	T
Deletion	F	F	F

5.1.7 Dano et al. (1997)

Dano et al. (1997) introduce a use case driven requirements engineering process to assist in the requirements collection and conceptualization activities, i.e. in requirements analysis (TABLE 12). The final goal of the approach is to produce object-oriented specifications, i.e. system models. During the collection activity, the approach first applies a use case tree consisting of tables to present a use case. The root table models the basic course of actions, whereas the other tables cover the variant and exception courses.

After that, seven types of scheduling links between use cases are introduced and used to form Petri nets from the use case trees (see also Lee et al. (1998) and Lee et al. (2001a) for using Petri nets in use cases). During the conceptualization activity, the Petri nets are transformed into state machines (see also Glintz (1995); Ratcliffe & Budgen (2001); Mitchell & Lecoeuche (1997); Barnard (2001) and Kösters, Six & Winters (2001) for using state machines in use cases).

5.1.8 Dutoit and Paech (2002)

Dutoit and Paech (2002) propose a detailed process of requirements engineering for capturing requirements and their rationale (TABLE 13). Rationale methods aim at capturing, representing, and maintaining records about why developers have made the decisions they have (see also Potts, Takahashi & Antón 1994). That involves e.g. justification for selected requirements, trade-offs negotiated by stakeholders and alternative requirements that were discarded. An issue model, which includes questions, options, criteria, assessments, arguments, and decision elements, is used to represent the rationale.

The approach uses tool support with hypertext navigation to link the requirements and rationale elements. Usually, hypertext and multimedia are used to enhance comprehensibility (Gough, Fodemski, Higgins & Ray 1995)

and traceability (Lang & Duggan (2001) and Haumer, Pohl & Weidenhaupt (1998)). A use case realises user tasks and is composed of system services. A user task is a unit of work that is meaningful to an actor. A use case describes how a user task can be achieved by sequence of interactions. Lastly, a system service describes the input and output of a system function. The approach relates non-functional requirements into a use case.

5.1.9 Feijs (2000)

Feijs (2000) proposes a formal approach to RE (see also Hsia, Samuel, Gao, Kung, Toyoshima & Chen (1994); Rolland & Achour (1997); Leite, Rossi, Balaquer, Maiorana, Kaplan, Hadad & Oliveros (1997) and Li (1999) for formal approaches). The formal approaches are commonly used in validation, and sometimes even in verification, of use cases. In other words, the purpose of these approaches is to model complete, correct and verifiable use cases. The approach of Feijs (2000) emphasises the usage of use cases in requirements engineering, test case generation and as a basis for user manuals (TABLE 13).

Feijs (2000) deploys sentences or formal language expressions and sequence diagrams, called message sequence charts (MSC), for the representation of use cases. The sequence diagrams are derived from the sentences. In the sequence diagrams, object classes and object identities are represented in a mixed manner. The terms use case and scenario are regarded as synonyms. Ingredients of natural language sentences are selected from the object models. After that, these atomic sentences are concatenated to composite sentences, which are converted to sequence diagrams.

5.1.10 Henderson-Sellers et al. (2002)

Henderson-Sellers et al. (2002) discuss the usage of use case metrics for estimating the size of a system to be built (see also Anda, Dreiem, Sjøberg & Jø

rgensen 2001). Use cases are used as an estimate for the effort needed to realize the functional requirements that involve an interaction with an actor (TABLE 13).

TABLE 13. Analysis of the use case approaches in Dutoit and Paech (2002), Feijs (2000) and Henderson-Sellers et al. (2002).

	Dutoit & Paech (2002)	**Feijs (2000)**	**Henderson-Sellers et al. (2002)**
Form view			
Medium	{STD}	{FLE, UCD, SD}	{STD}
Notations	{SF}	{SF, F}	{SF}
Animation	F	F	F
Interactivity	Hypertext	None	None
Contents view			
Instance	T	T	F
Type	T	T	T
Mixed	F	T	F
Multiplicity	?	F	?
System internal	T	T	F
System interaction	T	T	T
Organisational context	F	F	T
Organisational environment	F	F	F
Positions	T	F	F
Arguments	T	F	F
Issues	T	F	F
Decisions	T	F	F
Functional	{S, F, B}	{S, F, B}	{S, F}
Intentional	{}	{}	{G}
Non-functional	T	F	F
Normativeness	{D, U}	{D, U}	{D, U}
Purpose view			
Systems development	{RE}	{RE, TEST}	{RE, PM}
Business process re-engineering	F	F	F
Descriptive	T	T	F
Exploratory	F	F	T
Explanatory	T	F	F
Life cycle view			
Capture	Fromscratch	Fromscratch	Fromscratch
Integration	F	T	F
Refinement	F	F	F
Expansion	T	F	F
Deletion	F	F	F

The authors state that the use case modelling needs to be standardized before reliable and repeatable metrics can be obtained. Therefore, they have set out requirements for such a standardisation and propose the basic metrics for it. The metrics include size metrics, environment factors metrics and composite metrics. A use case is provided as a structured description, which is derived from the use case models of Regnell, Anderson and Bergstrand (1996) and Cockburn (2000). Just like in Cockburn (2000), the description includes goal information.

5.1.11 Insfrán et al. (2002)

Insfrán et al. (2002) introduce a requirements engineering-based conceptual modelling approach (TABLE 14). The aim of the approach is to provide a way to move from the requirements to a conceptual schema modelled in analysis and design phases in a traceable way. In other words, the purpose is to link RE smoothly to analysis and design phases. For this purpose, use cases act as a bridge (see also Berard (2001) and Ratcliffe & Budgen (2001) for use cases in analysis and design). Also validation and verification are key issues in the approach.

The approach combines a framework for requirements engineering, including a requirements model, and a graphical object-oriented method for conceptual modelling and code generation. The requirements model includes a mission statement, a function refinement tree to partition the external interaction according to different business areas or business objectives, and a use case model that includes use case specifications and a use case diagram. The use case specification is presented in a table (see also Rubin & Goldberg (1992); Potts et al. (1994) and Phillips et al. (2001) for tables). Furthermore, the approach produces at least one sequence diagram per use case, one for the basic course of action, and one for each alternative course of action.

TABLE 14. Analysis of the use case approaches in Insfrán et al. (2002), Lee et al. (2001) and Lycett (2001).

	Insfrán et al. (2002)	**Lee et al. (2001)**	**Lycett (2001)**
Form view			
Medium	{T, UCD, SD}	{STD, UCD}	{STD, UCD}
Notations	{SF}	{SF}	{SF}
Animation	F	F	F
Interactivity	None	None	None
Contents view			
Instance	T	F	F
Type	T	T	T
Mixed	F	F	F
Multiplicity	F	?	?
System internal	T	F	F
System interaction	T	T	T
Organisational context	T	T	T
Organisational environment	F	F	F
Positions	F	F	F
Arguments	F	F	F
Issues	F	F	F
Decisions	F	F	F
Functional	{S, F, B}	{S, F, B}	{S, F}
Intentional	{}	{G, GD}	{}
Non-functional	F	T	F
Normativeness	{D, U}	{D, U}	{D, U}
Purpose view			
Systems development	{A & D, RE}	{RE}	{RE}
Business process re-engineering	T	F	T
Descriptive	T	T	T
Exploratory	F	F	F
Explanatory	F	F	F
Life cycle view			
Capture	Fromscratch	Fromscratch	Byreuse
Integration	F	F	F
Refinement	F	T	T
Expansion	F	F	F
Deletion	F	F	F

5.1.12 Lee et al. (2001b)

The use case approach proposed by Lee et al. (2001b) presents a goal-oriented modelling technique (TABLE 14). The approach extends use cases with goals. A use case is viewed as a process that can be associated with a goal to be achieved, optimized, maintained or impaired by a use case. Requirement specifications

are structured with goals by establishing a goals hierarchy. In addition, the approach reveals and analyzes interactions and conflicts among non-functional requirements. The article extends the earlier works of Lee and Xue (1999) and Cockburn (1997a, 1997b) by distinguishing between several different types of goals. The goal types are rigid, soft, actor-specific, system-specific, functional, and non-functional. The approach basically concentrates on the systems interaction level. Additionally, the organisational context is partly covered by the actor-specific goals.

5.1.13 Lycett (2001)

Lycett (2001) emphasises the importance of capturing variation in a component-based development (TABLE 14). The component-based development aims at the dynamic 'plug-and-play' composition of systems from heterogeneous components. In ideal terms, a component-based architecture provides a flexible environment in which to cater for variation of components. The approach proposes use case production by reuse (see also Sutcliffe, Maiden, Minocha & Manuel (1998) and Woo & Robinson (2002) for use case reuse).

The forms of variation recorded in textual use cases are business rules, logic flows, types, and parameters. In a use case diagram, variation is proposed to be controlled by uses, extends, inheritance, parameterisation and configuration relationships. A difference to a standard a use case modelling is the further refinement of detail, as well as the concentration on modelling the structural and functional aspects.

5.1.14 Rational Unified Process (Kruchten 2001)

As the Unified Process (Jacobson et al. 1999), the Rational Unified Process (Kruchten 2001, see Section 2.3) applies a use-case driven approach (TABLE 15). That means that the use cases are the basis for the entire systems development

process. In RE use cases act as a common language for the communication between stakeholders and structured text descriptions are deployed in conjunction with use the case diagrams to facilitate the communication (see also Jaaksi (1997, 1998) for use cases as a media for communication). In addition, a user-interface prototype and a storyboard can be used to illustrate a user interface.

In the analysis and design phases, use cases are the bridge that unites the requirements and the design. During testing, use cases constitute the basis for identifying test cases and procedures. In addition, use cases are used in business process re-engineering, writing user manuals, deployment, and project management. Non-functional requirements are recorded in a supplementary requirements list. Use cases are organized by relationships.

5.1.15 Regnell et al. (1996)

Regnell et al. (1996) introduce a hierarchical use case model for managing a large and complex set of use cases (see also Juric & Kuljis (1999) for use case models in complex systems in BPR) (TABLE 15). Use cases are described at environment, structure and event levels. At the environment level, a use case diagram is deployed. In structure level, use cases are described as graphs of episodes, which have similarities to activity diagrams (see also Leite et al. (1997) and Leonardi & Leite (2002) for episodes).

Finally, at the event level, the episodes are modelled with sequence diagrams. Episodes are used to divide use cases into coherent parts. They support reuse of use case fractions, because a same episode can occur in many use cases. Despite the fact that the detail of a use case is refined from one level to another, are use cases deployed merely at the system interaction level.

TABLE 15. Analysis of the use case approaches in the RUP (Kruchten 2001), Regnell et al. (1996) and Regnell et al. (1995).

	RUP (Kruchten 2001)	**Regnell et al. (1996)**	**Regnell et al. (1995)**
Form view			
Medium	{STD, UCD, SB, SP}	{UCD, AG*, SD}	{STD, AG, SD}
Notations	{SF}	{SF}	{SF}
Animation	F	F	F
Interactivity	None	None	None
Contents view			
Instance	T	T	T
Type	T	T	T
Mixed	F	F	F
Multiplicity	T	T	F
System internal	F	F	T
System interaction	T	T	T
Organisational context	T	F	F
Organisational environment	F	F	F
Positions	F	F	F
Arguments	F	F	F
Issues	F	F	F
Decisions	F	F	F
Functional	{S, F, B}	{S, F, B}	{S, F, B}
Intentional	{}	{}	{}
Non-functional	T	F	F
Normativeness	{D, U}	{D, U}	{D, U}
Purpose view			
Systems development	{A & D, RE, HCI, TEST, PM}	{RE}	{RE, TEST}
Business process re-engineering	T	F	F
Descriptive	T	T	T
Exploratory	F	F	F
Explanatory	F	F	F
Life cycle view			
Capture	Fromscratch	Fromscratch	Fromscratch
Integration	F	T	T
Refinement	T	T	F
Expansion	F	F	T
Deletion	F	F	F

* variation of activity graph

5.1.16 Regnell et al. (1995)

Regnell et al. (1995) introduce a usage-oriented requirements engineering approach (TABLE 15). A use case describes behaviour of a system as seen by

one actor only and is modelled as a use case description and a use case specification (see also Kotonya & Sommerville (1996) and Glintz (1995) for actor-oriented approaches). A use case description is given as a structured description and formalized through a use case specification, which is represented as a sequence diagram. After that, the use case specification is transformed into an abstract usage scenario represented as an activity diagram. The abstract scenarios are integrated into a synthesized usage model, one for each actor. Finally, the system can be verified against the synthesized usage model. System internal behaviour is modelled in a use case specification, whereas use case descriptions and scenarios cover the system interaction.

5.1.17 Conclusions

In this section, we have described and analyzed fifteen use case approaches. Our aim has been twofold: (a) to get an overall picture of the use case approaches in the literature and (b) to test the use case classification framework by applying it. Next, we will first go through the framework from the beginning to the end and highlight the characteristics of and the differences and similarities between the analysed use case approaches. After that, we will examine the inter-dependencies between the classification concepts. The inter-dependencies signify that the characteristics of the use case approaches have relationships. For instance, if the multiplicity facet gets the value true, the abstraction facet involves both the type and instance level use cases. Lastly, we will analyse the pros and cons of the framework.

Conclusions from the overall analysis

The use case approaches are frequently represented with semiformal notations. Some approaches involve also formal (Dano et al. 1997; Feijs 2000) or informal notations (OMG 2003; Jacobson et al. 1999). Common representation formats are structured descriptions, tables, use case diagrams and sequence diagrams.

However, rich use cases including e.g. images, videos, software prototypes, animation or interactivity are rarely deployed (e.g. Alexander 2002a; Dutoit & Paech 2002) even though they could be valuable e.g. in the collection and validation of the requirements (see Alexander (2002b) for further details). Usually, an approach deploys several representation formats and, moreover, typically at least one textual format is used. Merely two approaches do not include textual descriptions (Regnell et al. 1996; van den Berg & Simons 1999).

Both the instance level and the type level use cases are common among the approaches (e.g. Insfrán et al. 2002), but the mixed level use cases are rarely used (Feijs 2000). Usually, a scenario is regarded as an instance of a use case (e.g. RUP). Still, there are approaches that use the terms use case and scenario as synonyms (e.g. Dano et al. 1997), or do not employ one or the other of the two terms (e.g. Lee et al. 2001b). In addition, one approach (Regnell et al. 1995) considers scenarios as abstract use cases.

Almost every use case approach (excluding van den Berg & Simons 1999) covers the system interaction attribute of the context facet. That is not surprising, as the notion of a use case is initially created to describe the communication between actors and a system (Jacobson et al. 1992). In many cases, also the system internal (e.g. Dano et al. 1997; Regnell et al. 1995) and the organisational context (e.g. Lycett 2001) attributes are covered, but coverage for the organisational environment is not mentioned in any of the approaches. That would suggest that the organisational environment attribute is not, perhaps, necessary to describe the information that a use case covers (see also Kösters, Six & Winters 2001, 4). The argumentation facet, or part of it, is covered in three approaches (e.g. Dutoit & Paech 2002).

The coverage facet is discussed as follows. Each of the three aspects of the functional attribute, the structure aspect, the function aspect and the behaviour aspect, are frequently dealt with (e.g. Cockburn 1997a, 1997b). Usually a use

case approach deploys several representation formats that together involve all these artefacts. For instance, textual use cases typically cover at least the structure and function aspects, and in many cases, also the behaviour aspect (e.g. Cockburn 1997a, 1997b).

Especially sequence diagrams, state machines and Petri nets are widely used for modelling a system's behaviour, for validation and verification of requirements and for seamless transformation from BPR to RE or from RE to analysis and design phases (e.g. Insfrán et al. 2002). Only three approaches involve the structure and function aspects, but not the behavioural issues (e.g. Henderson-Sellers et al. 2002).

The intentional attribute is dealt within four goal-oriented approaches. The attribute distinguishes between six domain values, but usually merely the goal and maybe the goal dependency domain values are involved (Cockburn 1997a, 1997b; Lee et al. 2001b). Moreover, one of the approaches deals with the problem and cause domain values (Alexander 2002a). Thus, all the domain values might not be necessary to describe the information that a use case covers. Non-functional requirements are included into several approaches (Lee et al. 2001b). Nevertheless, it is justifiable to state that often use cases concentrate on the functional requirements (e.g. OMG 2003; Feijs 2000)

To proceed, each of the analysed use case approaches covers both the desirable and undesirable behaviour, but they differ from each another in how the undesirable behaviour is dealt with. The desirable and undesirable behaviour may be discussed similarly (Alexander 2002a) or the undesirable behaviour may be presented as variations or exceptional flows of control in a textual use case (e.g. Lee et al. 2001b). However, a few approaches exists that do not include undesirable behaviour (see e.g. Rubin & Goldberg 1992; Potts et al. 1994) even thought they are not common.

The use case approaches are deployed for several purposes even in the context of RE, including e.g. validation and verification of the requirements, eliciting safety requirements and capturing the rationale. Moreover, it seems like use cases have been applied into new purposes especially during the recent years. The fact that most of the analysed use case approaches have been introduced quite recently supports the claim.

In addition to RE, the use case approaches are also applied in other application domains (analysis and design, human computer interaction, software test case generation, project management and business process re-engineering). BPR (e.g. Cockburn 1997a; 1997b) is the secondly regular and analysis and design (e.g. RUP; Insfrán et al. 2002) is the third frequent application domain. In RE, use cases are usually used for descriptive purposes, but also exploratory (e.g. Henderson-Sellers et al. 2002) and explanatory (Dutoit & Paech 2002) use cases are deployed. The purpose of the approach and the reasons for applying it are usually clearly stated.

Use cases are typically captured from scratch, but also the use case reuse has been proposed (Alexander 2002a; Lycett 2001). The refinement operations are most frequently covered (e.g. Dano et al. 1997; Lee et al. 2001). Also the integration (e.g. Regnell et al. 1995; Feijs 2000) and expansion (Dutoit & Paech 2002) operations have been applied. On the other hand, the deletion operations are rare and have not at all been deployed in the approaches classified in this study. That would suggest that the deletion attribute is not, perhaps, necessary. Generally speaking, the approaches provide merely a little guidance on the operations that are carried out during the use case modeling.

Following findings of inter-dependencies between the classification concepts are derived from the analysis of the use case approaches (TABLES 11, 12, 13, 14 and 15). The inter-dependencies do not have causalities to one direction or another, but emerge from the facts of how the classification concepts appear

concurrently. To start with, the context and application domain facets are slightly related. As expected, the system internal coverage is especially important in test case generation (Feijs 2000; Regnell et al. 1995). Likewise, especially BPR requires information of the organisational context (e.g. Lycett 2001) and system interaction is emphasized in each of the application domains.

The context facet has some inter-dependencies with other facets, too. For instance, a formal notation is used in modelling the systems internal behaviour (Feijs 2000; Dano et al. 1997). Moreover, if an approach deploys merely type level use cases, it does not usually involve system internal information (e.g. Lycett et al. 2001; Lee et al. 1997). There is one exception to the rule. Van den Berg and Simons (1999) analyse the control-flow semantics of use cases, i.e. temporal relations between use cases inside a system, and apply type level use cases because they are suitable to describe the necessary information. On the other hand, instance level use cases are used for modelling the system internal, system interaction and organisational context information (e.g. Dutoit & Paech 2002; Regnell et al. 1995).

The coverage facet has inter-dependencies with other classification concepts. In many cases, if the intentional attribute is covered, also the non-functional attribute is dealt with and vice versa (e.g. Alexander 2002a; Cockburn 1997a, 1997b). Moreover, an approach deploying a formal notation does not usually include intentional or non-functional information (OMG 2003; Dano et al. 1997; Feijs 2000). In other words, formal notations are used for describing behavioural aspects.

The intentional and non-functional aspects are usually discussed by approaches involving system interaction or organisational context information, but not by approaches concerning the system internal attribute (e.g. Insfrán et al. 2002; Feijs 2000). There is one exception to the rule. Dutoit & Paech (2002) involve the non-functional attribute. Furthermore, the approach is exceptional in that

manner that it describes system internal information in a textual form and therefore facilitates the inclusion of the non-functional requirements.

Each of those analyzed approaches, excluding RUP, that deploys use cases in the project management, covers also the intentional attribute (Henderson-Sellers et al. 2002; Cockburn 1997a, 1997b). Moreover, the approaches applied in the project management deploy structured descriptions and concentrate on describing the system interaction and organisational context information.

The argumentation facet shares some features with other facets. If an approach covers the argumentation facet or part of it, the role of a use case approach seems to be either explanatory or exploratory in addition to the usual descriptive role (Alexander 2002a; Allenby & Kelly 2001; Dutoit & Paech 2002). There is one exception to the rule. In Henderson-Sellers et al. (2002) the role of the use cases is exploratory, but argumentation is not discussed. The reason for that seems to be that the use cases are applied for estimating a size for a systems development project. In addition, the argumentation facet is usually discussed in the use case approaches that are applied in the RE application domain (e.g. Dutoit & Paech 2002; Allenby & Kelly 2001). That is, the argumentation information is recorded during RE.

Finally, the multiplicity and abstraction facets have inter-dependencies. If the multiplicity gets the value true, are use cases considered as type level use cases, and scenarios are regarded as instance level use cases. On the other hand, both the type level and the instance level use cases are also used in approaches that give the multiplicity facet the value false (e.g. Dano et al. 1997) or a indeterminate value (Dutoit & Paech 2002). In conclusion, the instance level use cases are not always called scenarios.

The use case approaches have been motivated by the shortcomings (van den Berg & Simons 1999; Cockburn 1997a, 1997b; Dano et al. 1997; Insfrán et al.

2002; Lee et al. 2001b; Regnell et al. 1996 and Regnell et al. 1995) and insufficient definitions of the use cases. Moreover, the large variety of the approaches arises from the evolution of the use case technique (OMG 2003; Kruchten 2001) and the employment of the use cases into new application domains and purposes (Alexander 2002a; Allenby & Kelly 2001; Dutoit & Paech 2002; Feijs 2000; Henderson-Sellers et al. 2002 and Lycett 2001).

Conclusions from the applicability of the framework

In this study we laid out three objectives for a use case classification framework. The framework must (a) bring out the essential aspects of requirements engineering and (b) cover broadly the concepts of the development and system worlds. Additionally, the framework should (c) provide a suitable foundation for analysing the use case approaches so that it would highlight the characteristics and concepts of and the differences and similarities between the approaches.

In Chapter 2 we noticed that the framework fulfils two of the objectives. The original framework of Rolland et al. (1998) has quite a broad coverage of the development world and system world concepts. It is also in congruence with the three dimensions of the RE framework (Pohl 1994), which describes the essential aspects of the RE. In this chapter we used the framework by analysing use case approaches. The usage acted as a test of the third objective.

Basically, the framework lays out a solid foundation for analysing the approaches in the RE. Just like the analysis of the approaches and the conclusions from it testifies, the framework proved to be effective in emphasizing the differences and similarities between the approaches and assisted in understanding the achievements gained from recently developed approaches. The analysis highlighted the purposes and forms the use cases may take. Especially the description, abstraction, multiplicity, context,

argumentation, coverage, application domain, and role facets support efficiently the analysis of the approaches. On the other hand, the framework contains some concepts that appear not to be so effective in categorizing the approaches. Next, we will shortly discuss these aspects.

The multiplicity attribute is measured with Boolean values and the sign '?', which stands for the indeterminate value. However, it might be more reasonable to measure the attribute with several domain values to exclude the indeterminate value and to include the case of multiple use cases for one scenario. The organisational environment attribute of the context facet is not covered in any of the approaches. That would suggest that the attribute would not, perhaps, be necessary in analysing the use case approaches. Although, when analysing use case approaches that are mainly applied in some other application domain, e.g. in BPR, also the organisational environment attribute may be covered.

A use case approach frequently covers each of the domain values of the functional attribute. Thus, in order to gain a more precise picture of the coverage of the systems world concepts in the approaches, it would be reasonable to deploy some other technique. In fact, for this reason we will carry out an in-depth analysis in the next section. To proceed, three of the six domain values of the intentional attribute are not covered in the classification, which would suggest that maybe it would be reasonable to modify the attribute to include fewer domain values. Nevertheless, especially when investigating goal-based approaches, each of the domain values could be valuable.

Typically, a use case approach covers both the desirable and undesirable behaviour, but differ in how the undesirable behaviour is death with. Thus, to highlight the differences and similarities between the approaches, it would be better to modify the normativeness facet to describe, how the undesirable behaviour is covered. The use cases are used in application domains that are not

included in the framework. For instance, use cases are applied in deployment of marginal value (Kruchten 2001). However, every significant application domain is included in the framework and, thus, it would not be reasonable to add new domains.

Lastly, like discussed before, the approaches provide merely a little guidance on the operations that are carried out during the use case modeling. Therefore, the operations facet is not so effective and maybe even not so necessary to analyse the approaches. Consequently, it would be reasonable to state that the form, contents and purpose views are valuable in emphasizing the differences and similarities, and to help understand the achievements gained from recently developed use case approaches. Moreover, the operations for manipulating use cases are usually not so exactly described, so the life cycle view is not so useful in the classification.

In conclusion, all the three objectives for a use case classification framework have been fulfilled. The framework brings out the different aspects of requirements engineering and covers broadly the concepts of the development world and the system world. In addition, the framework provides a substantial foundation for analysing use case approaches and therefore helps to understand the achievements gained from recently developed use case approaches.

5.2 In-depth analysis

In this section, we will analyse the use case approaches in more details. The in-depth analysis is made in terms of metamodeling use case diagrams. The aim of the analysis is to find the essential concepts and concept constructions. During the overall analysis, the concepts of the system world were discusses quite superficially and basically by the functional facet of the coverage view

(structural, functional and behavioural aspects). In this section we will concentrate analysing the concepts of the system world.

The section is organized as follows. Firstly, we will shortly discuss about metamodeling and present the analysed approaches. Secondly, we will carry out the in-dept analysis. We will first analyze the use case diagram of the UML 1.5. After that, we will carry out an in-depth analysis of the other approaches in the alphabetical order according to the author information. In the end of the section we will draw conclusions.

5.2.1 Introduction

In this section, we will carry out an in-depth analysis of the use case approaches by metamodeling use case diagrams. Next, we will shortly discuss metamodeling according to Tolvanen (1998). *Metamodeling* is a modelling process, which takes place one level of abstraction and logic higher than the standard modelling process (Gigch 1991, 230). Metamodeling develops a *metamodel*, which is a conceptual model of a systems development technique or model (Brinkkemper 1990, 29). Metamodels can be further divided into different types. In this study, we use the term metamodel to refer to a meta-data model, which describes the static aspects of a use case diagram. A metamodel instantiates to a model of the system world. In this study, the model is a use case diagram. In the metamodeling, we deploy the Meta Object Facility (MOF, OMG 2002) as a metamodeling language.

We have chosen five use case approaches from those analysed in the previous section for the in-depth analysis. In addition to the UML 1.5 use case diagram we will analyze four use case approaches that have introduced a use case diagram differing from the use case diagram of the UML 1.5 (see e.g. OMG 2003, Jacobson et al. 1999; Booch et al. 1998) or Jacobson et al. (1992). These approaches are Alexander (2002a), Allenby and Kelly (2001), Lee et al. (2001b)

and Regnell et al. (1996). On the other hand, we have excluded approaches that deploy the notion of a use case diagram introduced in the UML (Kruchten 2001) or in Jacobson et al. (1992, Dano et al. 1997; Insfrán et al. 2002; Feijs 2000). In addition, we have excluded an approach that deploys use case diagrams, but does not provide a sufficient example to be analyzed (Lycett 2001).

In the metamodels we will present the essential concepts as classes and the relationships between these concepts as line symbols. Essential concepts are the concepts that are modelled in the use case diagrams as graphical symbols. The notation is expressive to emphasise the differences and similarities between the approaches. Based on the metamodels, we will investigate the correspondences between these concepts and the concepts of the system world (object, action, location, actor, time, purpose, and tool).

5.2.2 UML 1.5 (OMG 2003)

The UML 1.5 (OMG 2003) use case model is described in detail in Section 3.1. Here we present the essentials of the model. In the UML 1.5, a use case is used to define a behavior of a *system* or other semantic entity, such as a subsystem, a class or a user-interface, without revealing the internal structure of the entity (FIGURE 17). A system is regarded as that part of a computer system that does not include actors. Usually, there is merely one system in a use case diagram. Still, in the case of use case reuse, a use case may be included in one or more systems. Use cases may be grouped into *packages*. A use case interacts with one or more *actors* that describe the word outside the system and define a set of roles that users can take. An *association* between a use case and an actor states that the use case and the actor communicate with each other.

A use case may be related to other use cases by generalization, include and extend relationships. A *generalization* relationship means that a *child* use case inherits the behaviour and features of a *parent* use case and may add new

features and associations. An *include* relationship signifies that a *base* use case contains the behaviour of an *addition* use case, whereas an *extend* relationship implies that an *extension* use case may extend the behaviour described in a *base* use case under certain conditions. In addition, a use case may include an *extension point*, which is a reference to a location within a use case at which the action sequences from other use cases may be inserted. Lastly, actors may have *generalization* relationships indicating that a *child* actor can play the same roles as the *parent* actor and therefore communicate with the same kinds of use cases.

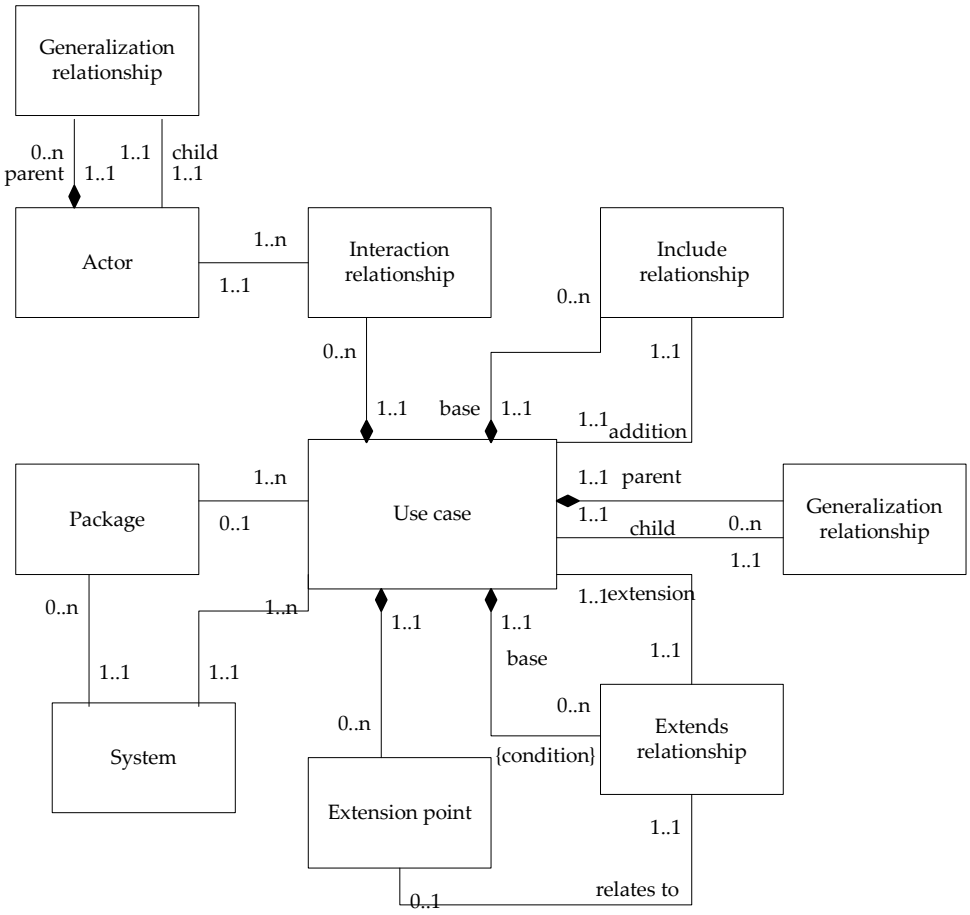

FIGURE 17. Metamodel of the use case diagram of the UML 1.5.

Two system world concepts are covered in the use case diagram. A use case defines a behavior of a system or other semantic entity and represents the action concept. Use cases may be grouped into packages that stand for the

action concept, as well. Furthermore, an actor defines a set of roles that users can take when interacting with a system and therefore an actor stands for the actor concept. In addition, a system stands for the object concept of the development world.

5.2.3 Alexander (2002a)

Alexander (2002a) applies the concept of a misuse case in trade-off analysis (see Section 5.1). A misuse case is a use case from the point of view of an actor hostile to a system. To be more precise, a *misuse case* describes a hostile goal, which if carried out would result in harm to a resource associated with one of the actors, one of the stakeholders or the system itself. A use case diagram is constructed showing *use cases* for goals held by *ordinary actors*, and misuse cases for goals of *hostile actors* (FIGURE 18). In other words, use cases have *interaction* with *actors*. In this study, we use the concept *ordinary use case* to denote a use case as opposite to a misuse case.

The approach introduces four new relationships between use cases. The proposed relationships are threatens, mitigates, aggravates, and conflicts with. Moreover, *includes* and *has exception* relationships are used. The former relationship corresponds to the include relationship and the latter the extends relationship of the UML 1.5.

The proposed relationships are defined as follows. A misuse case *threatens* a use case if achieving the goal of a misuse case reduces a system's ability to achieve the goal of the use case. A use case *mitigates* a misuse case if it reduces the effect of the misuse case on use cases that it threatens. A use or misuse case *aggravates* a misuse case if it increases either the probability of achievement or the seriousness of the damage that the misuse case threatens. Lastly, a use case *conflicts with* another use case if achieving its goal makes achieving a goal of the

second use case more difficult or impossible. The last relationship is mutual, bidirectional relationship.

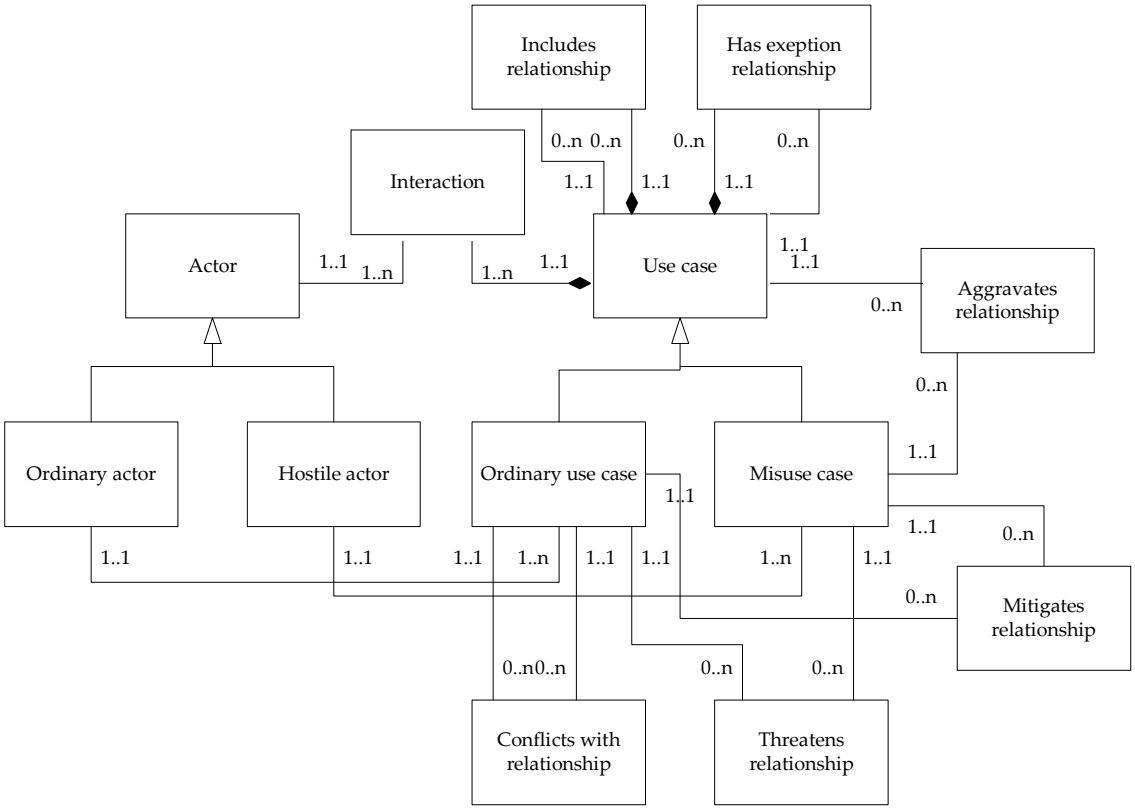

FIGURE 18. Metamodel of the use case diagram introduced by Alexander (2002a).

A use case diagram for a misuse case analysis of security requirements for a car is presented as follows (FIGURE 19, Alexander 2002a). The white ellipses are use cases for the actor Driver. The black ellipses represent misuse cases for the hostile actor Car Thief. The use case Drive the Car includes the use case Lock the Car, which in turn includes the use case Lock the Transmission. The misuse case Steal the Car threatens the use case Drive the Car and includes the misuse case Short the Ignition. On the other hand, the use case Lock the Car mitigates the misuse case Steal the Car. The misuse case Short the Ignition threatens the use case Lock the Car, but is mitigated by use case the Lock the Transmission.

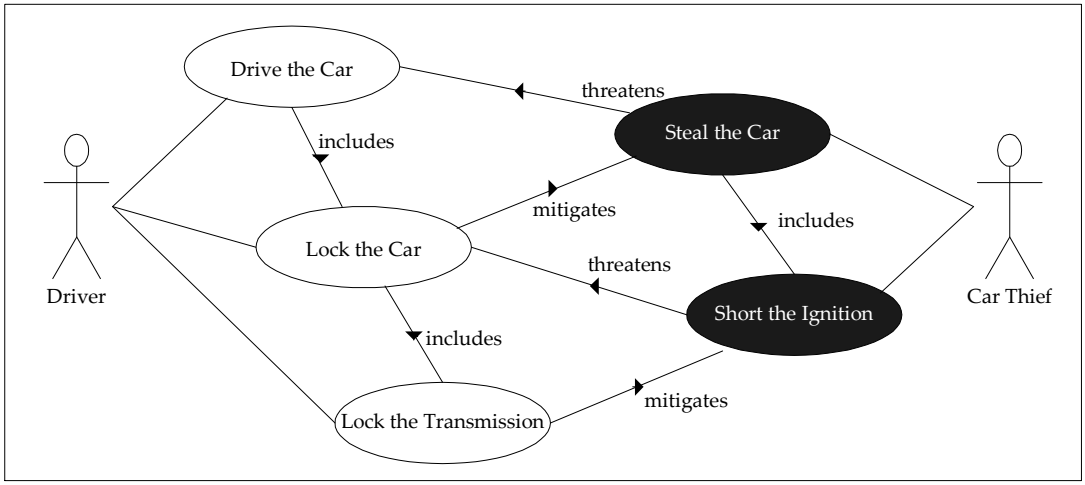

FIGURE 19. Use case diagram for security requirements for a car (Alexander 2002a).

Three system world concepts are covered in the use case diagram introduced by Alexander (2002a). Use cases and misuse cases are regarded as goals and as functional behaviour. In that sense, they represent the purpose as well as the action concepts. In addition, an actor refers to the actor concept in the system world.

5.2.4 Allenby and Kelly (2001)

Allenby and Kelly (2001) propose an approach to conduct hazard analysis on use cases (see Section 5.1). A hazard analysis provides the mechanism for relating safety related functional requirements to a use case approach to improve the safety of a system. A use case diagram is defined at the system or sub-system level. The diagram includes the decomposition of the use cases and allocation of the use cases to subsystems (FIGURE 20). Term function is also used to denote a use case.

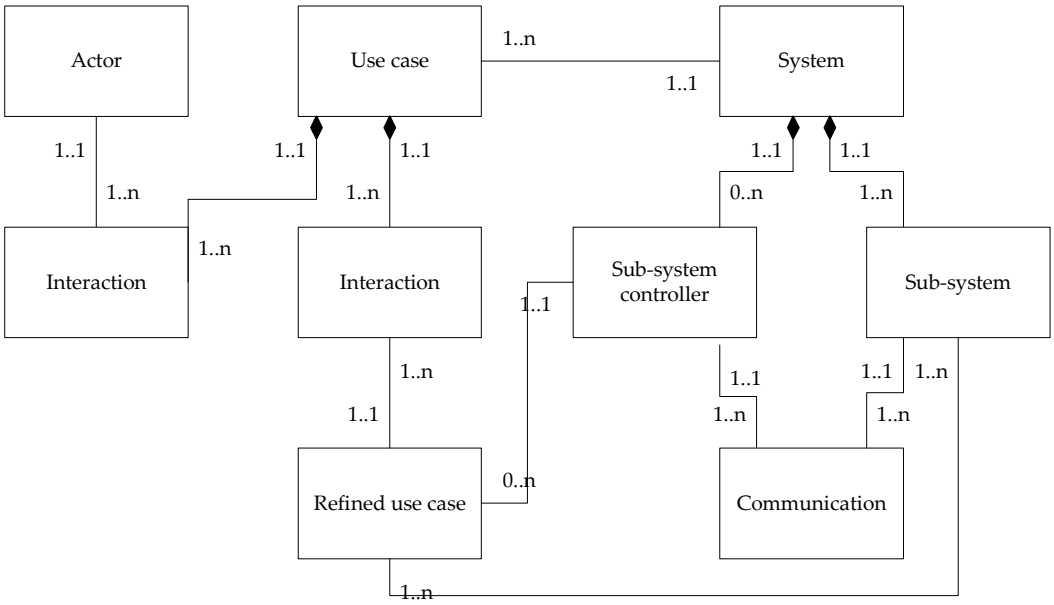

FIGURE 20. Metamodel of the use case diagram introduced by Allenby and Kelly (2001).

A *use case* has *interaction* with an *actor.* Moreover, a use case is composed of and has *interaction* with *refined use cases*, which in turn are allocated to sub-system controllers. A *sub-system controller communicates* with *sub-systems* in order to affect control and gather data. The sub-systems and controllers belong to a *system*. The approach refers to the usage of UML 1.5. generalization, extend and include relationships, but they are not modelled in the use case diagram example.

System level use case diagram for an aircraft represents a typical use case diagram for a two-layer system (FIGURE 21, Allenby & Kelly 2001, 233). The actor Pilot communicates with the use case Decelerate. The use case is decomposed into three refined use cases, Reduce Thrust, Reverse Thrust Direction and Apply Air Breaking. The refined use cases are allocated to the sub-system controllers Engine Controller and Airframe Controller. The controllers communicate with the sub-systems Engine, Thrust Reverser and Airframe.

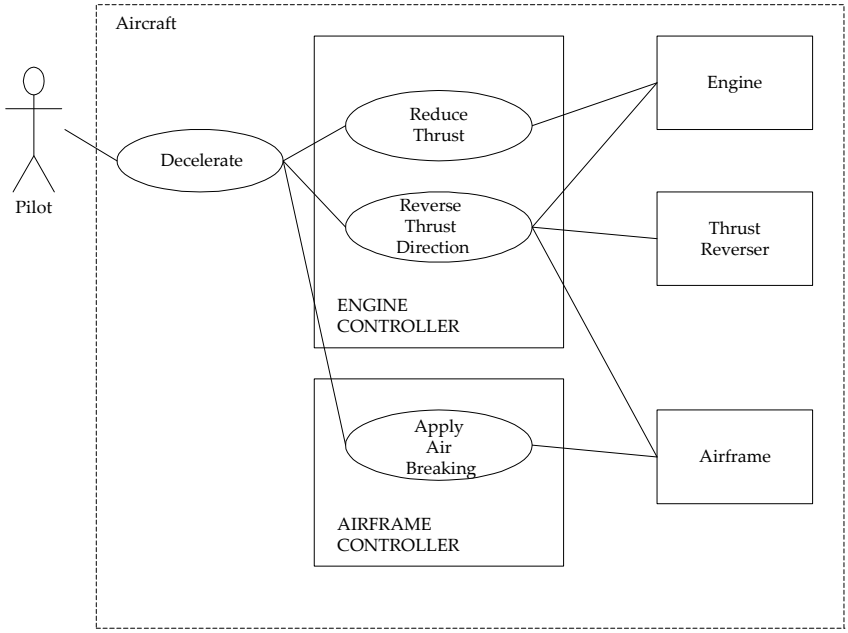

FIGURE 21. System level use case diagram for an aircraft in Allenby and Kelly (2001, 233).

In the use case diagram introduced by Allenby and Kelly (2001), three system world concepts are discussed. A use case represents a function of a system and therefore denotes the action concept. An actor stands for the actor concept and a sub-system controller refers to the tool concept. In addition, the system and sub-system concepts stand for the object concept in the development world.

5.2.5 Lee et al. (2001b)

Lee et al. (2001b; see also Lee & Xue 1999) propose a goal-driven approach (see Section 5.1). The approach uses goals to structure use cases and their extensions, and considers several different types of goals (FIGURE 22). A *use case* is viewed as a process that can be associated with a goal to be achieved, optimized or maintained by a use case. The approach distinguishes between *original use cases* and *extension use cases*, which extend original use cases. In addition, *abstract use cases* are applied.

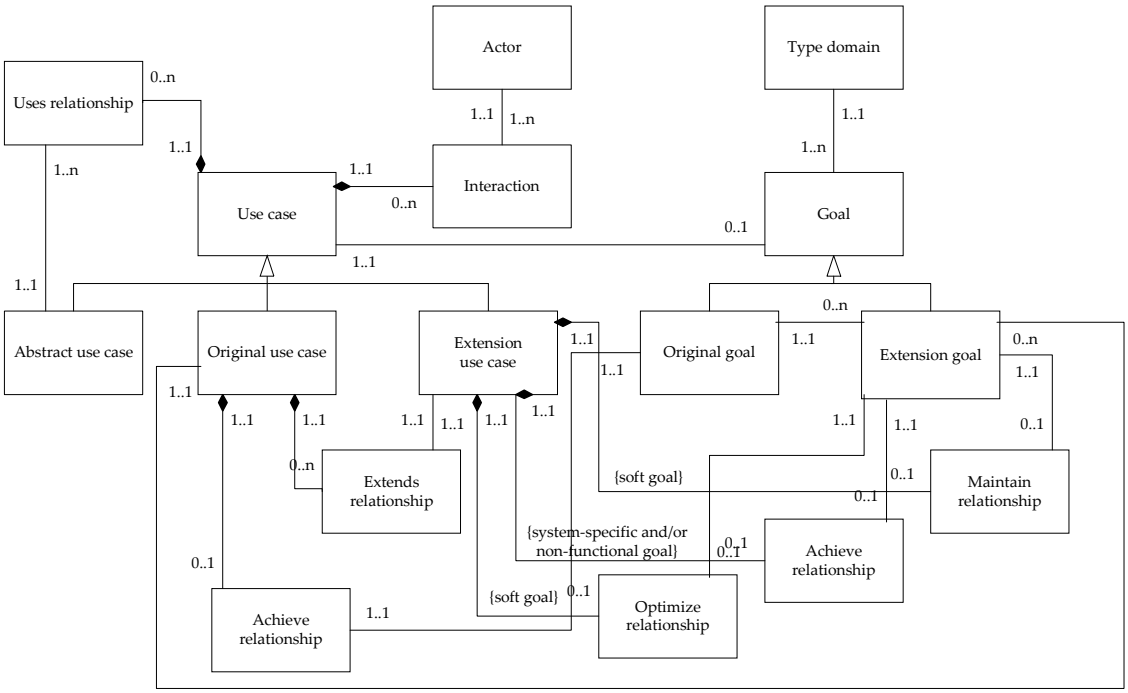

FIGURE 22. Metamodel of the use case diagram proposed by Lee et al. (2001b).

Two types of relationships between use cases are deployed. A *extends* relationship between use cases specifies, how a use case may be embedded into another use case and extend its functionality. The relationship is similar to the extend relationship of the UML 1.5. A *uses* relationship is used for refinement and extracts similar parts of two or more use cases. It is similar to the include relationship of the UML 1.5. In an extends relationship both of the use cases have their associated goals. On the other hand, in a uses relationship an abstract use case enhances reusability and does not have a goal associated with it.

The approach distinguishes between *original goals* that are *achieved* by original use cases and *extension goals* that are achieved, optimized or maintained by an extension use case. Goals are defined within a *type domain*. That is, they are classified into rigid or soft, actor-specific or system-specific and functional or non-functional. A *rigid goal* must be completely achieved, because it explains a minimum requirement. A *soft goal* describes a desirable property and can be partially achieved. *Actor-specific goals* are objectives of an actor and *system-*

specific goals are requirements on services that a system provides. A *functional goal* is achieved by performing a sequence of actions and, lastly, a *non-functional goal* is a constraint to qualify related functional goals.

An original goal is rigid, actor-specific and functional. An extension goal is weakly dependent on its original goals. By achieving an original, rigid goal, its related soft goals can be achieved to some extent. Extension use cases are created to *optimize* or *maintain* soft goals or to *achieve* system-specific or non-functional goals.

Use case diagram for a Meeting Scheduler System is presented in FIGURE 22 (Lee et al. 2001b, 126). The actor Initiator interacts with the original use case Plan a meeting and the actor Participant interacts with the original use case Delegate to another participant.

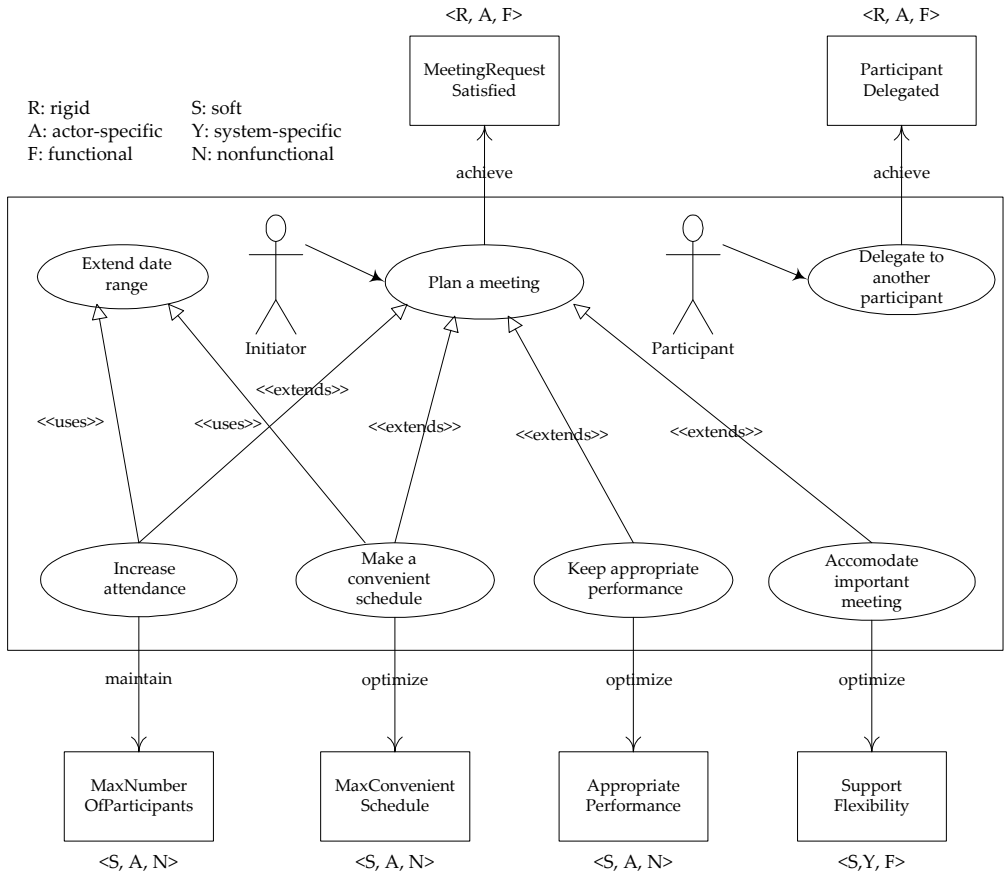

FIGURE 22. Use case diagram for a Meeting Scheduler System in Lee et al. (2001b, 126).

The use cases achieve original goals Meeting request satisfied and Participant delegated, respectively. The goals are rigid, actor-specific and functional. Several extension use cases extend the use case Plan a meeting. These use cases affect, i.e. maintain or optimize, extension goals, which are soft and either actor- or system-specific and either functional or non-functional. In addition, the use cases Increase Attendance and Make a convenient schedule use the abstract use case Extend date range.

In a use case diagram proposed by Lee et al. (2001b), three system world concepts are covered. A use case represents the action concept and an actor denotes the actor concept. In addition, a goal represents a purpose of the action, i.e. the purpose concept.

5.2.6 Regnell et al. (1996)

Regnell et al. (1996) introduce a hierarchical use case model for managing a large and complex set of use cases (see Section 5.1). Use cases are described at the environment, structure and event levels. Use case diagrams are deployed at the environment level. The notation is close to the one introduced by Jacobson et al. (1992) the main difference being that Regnell et al. (1996) have added the service concept. In addition, in Regnell et al. (1996) the relationships between the use cases and between the actors are not modelled.

A use case diagram is constructed showing use cases, actors, services, system and interaction between the use cases and the actors (FIGURE 23). A system is also called a *target system* and an environment in which a system will operate is called the *host system*. An *actor* belongs to a host system and represents a set of users that have some common characteristics with respect to why and how they use the target system. Inside the target system there are a number of services. A *service* is a package of functional entities offered to actors in order to satisfy goals that the actors have. A *use case* models a usage situation where one or

more services of the target system are used with the aim to accomplish the goals of an actor. A use case and an actor have an *interaction* relationship.

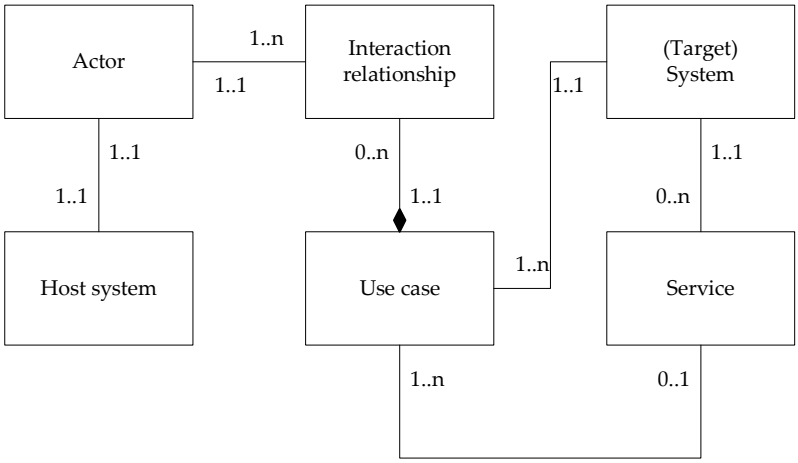

FIGURE 23. Metamodel of the use case diagram introduced by Regnell et al. (1996).

A use case diagram for an Access control System is presented in FIGURE 24. The actor Employee interacts with the use cases Open door and Register In/Out. The actor Visitor interacts with the use case Enter by Bell. The use cases belong to a service called Access. The actor System admin. communicates with the use cases Add Employee, Remove Employee and Produce log that are included to the service Admin. The use cases are included in a system called Access control.

In the use case diagram proposed by Regnell et al. (2001), two system world concepts are covered. The use case and service concepts represent the action concept and an actor denotes the actor concept. In addition, the object of the development world is discussed by the system concept.

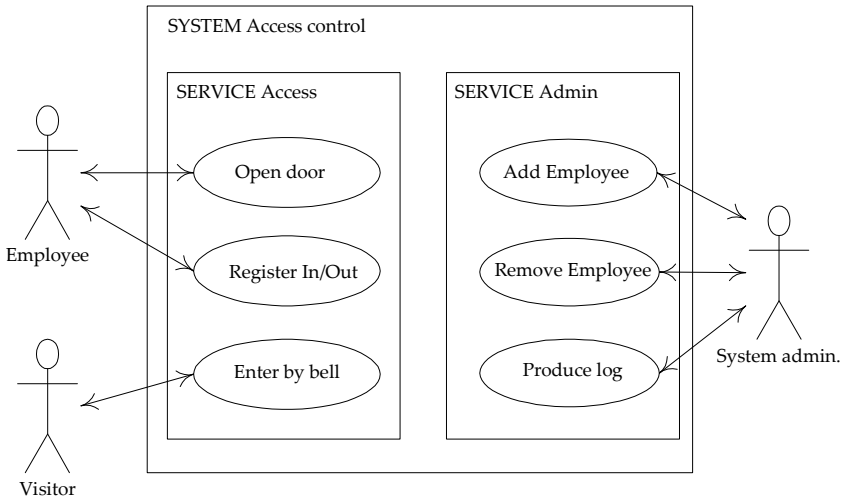

FIGURE 24. Use case diagram for an Access Control System in Regnell et al. (1996, 76).

5.2.7 Conclusions

In this section, we have analysed the use case diagrams of five use case approaches. Next, we will compare the concepts in these approaches. First, we will highlight the correspondences between the concepts of the UML 1.5 and the other approaches. After that, we will compare the concepts of the system world.

The concepts of the use case diagram of the UML 1.5 have many corresponding concepts in the other use case approaches (TABLE 16). In addition, the other approaches deploy concepts that are not included in the UML 1.5 (TABLE 17). Next, we will shortly discuss these issues. To start with, Alexander (2002) applies use cases and misuses cases in a trade-off analysis in order to examine the advantages and disadvantages of the requirement or design options. A use case diagram shows ordinary use cases for ordinary actors and misuses cases for hostile actors. The approach introduces four new relationships between use cases and misuse cases. In addition, it deploys the includes and has exception relationships.

TABLE 16. Summary of the concepts of the use case diagrams in the UML 1.5 and corresponding concepts in Alexander (2002a), Allenby and Kelly (2001), Lee et al. (2001b) and Regnell et al. (1996).

UML 1.5 (OMG 2002)	Alexander (2002a)	Allenby & Kelly (2001)	Lee et al. (2001b)	Regnell et al. (1996)
Use case	Use case, Ordinary use case, Misuse case	Use case, Refined use case	Use case, Original use case, Extension use case, Abstract use case	Use case
Extension point	-	-	-	-
Actor	Actor, Ordinary actor, Hostile actor	Actor	Actor	Actor
Package	-	-	-	Service
System	-	System, Sub-system	-	Target system
Include relationship	Includes relationship	-	Uses relationship	-
Extend relationship	Has exception relationship	-	Extends relationship	-
Generalization relationship between use cases	-	-	-	-
Interaction relationship	Unnamed	Unnamed	Unnamed	Interaction relationship
Generalization relationship between actors	-	-	-	-

TABLE 17. Summary of the concepts in the use case diagrams of Alexander (2002a), Allenby and Kelly (2001), Lee et al. (2001b) and Regnell et al. (1996) that do not have corresponding concepts in the UML 1.5 (OMG 2003).

Alexander (2002a)	Allenby & Kelly (2001)	Lee et al. (2001b)	Regnell et al. (1996)
Threatens, Mitigates, Aggravates and Conflicts with relationships between use cases and misuse cases	Sub-system controller	Goal, Original goal, Extension goal, Type domain, Achieve, Maintain and Optimize relationships between use cases and goals	Host system

Secondly, Allenby and Kelly (2001) deploy use cases in a hazard analysis and describe use cases at the system and sub-system level. A use case diagram includes decomposition of a use case into refined use cases and allocation to the sub-systems and sub-system controllers. Thirdly, Lee et al. (2001b) propose an approach that uses goals to structure use cases. The approach distinguishes between abstract use cases, original use cases and extension use cases as well as original goals and extension goals. In addition, the approach proposes three relationships between use cases and goals and deploys the uses and extends relationships between the use cases. Lastly, Regnell et al. (1996) introduce a hierarchical use case model for managing a large and complex set of use cases. A use case diagram is deployed at the environment level and modelled in a compact notation.

The use case, actor and interaction relationship concepts of the UML 1.5 have corresponding concepts in each of the other use case approaches. On the other hand, the system as well as the include and extend relationship concepts have similar concepts in two approaches. The extension point and generalization relationships between use cases and between actors do not have correspondences.

The system world concepts are covered as follows (TABLE 18). The several types of use cases (in each of the approaches) and the package concept (OMG 2003), including the service concept (Regnell et al. 1996), represent the action concept. The location concept denotes the spatial context and is weakly covered by the host system concept in one approach (Regnell et al. 1996). The actor concept is discussed by the actor concept in each of the approaches. The goal-oriented approaches (Alexander 2002a; Lee et al. 2001b) cover the purpose concept by use cases and goals. Lastly, the tool concept is discussed by one approach (Allenby & Kelly 2001).

TABLE 18. Summary of the system world concepts in the use case diagrams of UML 1.5 (OMG 2003), Alexander (2002a), Allenby and Kelly (2001), Lee et al. (2001b) and Regnell et al. (1996).

System world concept	UML 1.5 (OMG 2003)	Alexander (2002a)	Allenby & Kelly (2001)	Lee et al. (2001b)	Regnell et al. (1996)
Object	-	-	-	-	-
Action	Use case, Package	Use case, Ordinary use case, Misuse case	Use case	Use case, Original use case, Extension use case, Abstract use case	Use case, Service
Location	-	-	-	-	Host system
Actor	Actor	Actor, Ordinary actor, Hostile actor	Actor	Actor	Actor
Time	-	-	-	-	-
Purpose	-	Use case, Ordinary use case, Misuse case	-	Goal, Original goal, Extension goal	-
Tool	-	-	Sub-system controller	-	-

The object and time concepts are not discussed in any of the approaches. On the other hand, the time concept is usually embedded in other forms of the use cases. For instance, time is presented by pre- and post-conditions in textual use cases. The most common system world concepts that are covered in the approaches are the action and actor concepts. The object of the development, i.e. the system concept is dealt with in the use case diagrams of three approaches (TABLE 19; OMG 2003; Allenby & Kelly 2001 and Regnell et al. 1996).

TABLE 19. Summary of the system concepts in the use case diagrams of UML 1.5 (OMG 2003), Alexander (2002a), Allenby and Kelly (2001), Lee et al. (2001b) and Regnell et al. (1996).

UML 1.5 (OMG 2003)	Alexander (2002a)	Allenby & Kelly (2001)	Lee et al. (2001b)	Regnell et al. (1996)
System	-	System, Sub-system	-	(Target) System

As discussed before, the multiplicity of the use case approaches results from several reasons. The approaches analyzed in this section are created because of the shortcomings (Lee et al. 2001b; Regnell et al. 1996), insufficient definitions and the resulting evolution of the use case technique (OMG 2003) as well as the employment of the use cases into new application domains and purposes (Alexander 2002a; Allenby & Kelly 2001). Especially Lee et al. (2001b; see also Lee & Xue 1999) mention the problems resulting from the insufficient definitions of the relationships between the use cases and inclusion of the non-functional requirements.

5.3 Summary

In this chapter, we have analysed use case approaches on two levels. In Section 5.1 we carried out an overall analysis by classifying fifteen use case approaches according to the framework derived in Chapter 4. The analysis highlighted the characteristics of and the differences and similarities between the use case approaches. For instance, the analysis indicates that the use cases have been represented in several forms and applied for several purposes in the context of RE. The contents of the use cases vary remarkably, but the operations for manipulating use cases are not always described. The usage of the framework acted as a test of the third objective for the framework. We observed that the framework provides a suitable foundation for analysing use case approaches and therefore helps to understand the achievements gained from the recently developed use case approaches.

In Section 5.2 we analysed the concepts of five use case approaches in more details by metamodeling use case diagrams. We found out that the concepts of the use case diagram of the UML 1.5 have many correspondences in the other use case approaches. On the other hand, each of the analyzed use case approaches has introduced new concepts that do not have correspondences in the UML 1.5. Moreover, we discovered that all of the use case approaches deal

with the action and actor concepts of the system world. In addition, we noticed that also the location, purpose and tool concepts are discussed in the approaches. Therefore, it is justifiable to state that, on a general level, the concepts of the system world are covered quite averagely. In the next chapter we will draw conclusions.

6 CONCLUSIONS

In this study we have investigated use case approaches that have been suggested for requirements engineering. Requirements engineering is widely considered to be the most important and difficult part of systems development. The use case technique is widely used to capture functional requirements for a system. The technique has attracted considerable attention in systems development and business process re-engineering. Furthermore, the use cases have been applied e.g. in several phases of the systems development process.

The use case technique has been integrated in the UML (OMG 2003), which has emerged as the dominant modelling language in systems development. The use case concept of the UML is regarded as de facto standard. However, several problems and weaknesses concerning the technique have been reported in the literature. For instance, the lack of object-orientation as well as of the insufficient definitions and formal semantics concerning the technique are said to be the drawbacks of the technique. Motivated by the deficiencies, a large variety of use case approaches have been proposed.

Neither the use case technique of the UML nor the original use case technique (Jacobson et al. 1992) does cover the entire requirements engineering. In this study, we have inspected, in which way the large variety of the use case approaches covers requirements engineering and moreover, what kind of characteristics, concepts, differences and similarities the approaches have. Particularly, it was important to emphasise the divergent purposes for and the forms in which the use cases have been suggested for RE. In order to answer the questions, we needed a framework to investigate the role of the use case technique in requirements engineering. The study was theoretical and was based on the literature.

At the beginning of the study, we investigated the process and artifacts of the requirements engineering in the traditional systems development and in the UML. The latter was described in the forms of the Unified Process and the Rational Unified Process. Moreover, we discussed the use case technique. The technique consists of two parts, which are a use case model and a procedure to model use cases. The stepwise procedure was described in conjunction with the Requirements workflow of the Unified Process. The key concepts of the use case model of the UML 1.5 and the Unified Process were explained. In addition, we discussed the pros and cons of the technique.

In order to form out a solid foundation for the analysis, we made a comparative evaluation of three use case classification frameworks. We compared the categorizing concepts of the frameworks and assessed the coverage of the three dimensions of the RE framework. We discussed the requirements criteria manifested in the frameworks and, finally, evaluated the frameworks through the concepts of the development world and system world. Based on the analysis, we chose the framework of Rolland et al. (1998) to be used in the classification of the use case approaches. Furthermore, we modified the framework to make it even more suitable for our purposes.

We investigated the use case approaches on two levels. Firstly, we carried out an overall analysis by classifying fifteen use case approaches with the modified framework. The aim of the overall analysis was to get a general picture of the use case approaches. The analysis discussed the concepts of the development and system worlds. After the overall analysis, we carried out an in-depth analysis in order to gain a more detailed picture of the recently developed use case approaches. During the overall analysis, the concepts of the system world were discussed quite superficially. Therefore, in the in-depth analysis the concepts of the system world were inspected in a more detailed manner in terms of metamodeling use case diagrams of five approaches. The aim of the in-depth analysis was to find the essential concepts and concept constructions.

The overall analysis indicates that the use cases have been applied for several purposes and represented in several forms in the context of RE. The contents of the use case approaches vary remarkably, but typically the operations for manipulating use cases are not sufficiently described. Among other things, the overall analysis points out that almost every use case approach describes interaction between a system and the context of the system and, in many approaches, also the system internal and the organisational context aspect are covered. Furthermore, we have noticed e.g. that usually the use case approaches concentrate on describing functional requirements, but also non-functional requirements and intentional aspects are included into several approaches.

Moreover, we observed that the characteristics of the use case approaches have inter-dependencies. For instance, the intentional and non-functional aspects are usually discussed by approaches involving the system interaction or organisational context information, but not by approaches concerning the system internal aspect.

In the in-dept analysis, we came across the essential concepts and concept constructions of the approaches. We found out that the concepts of the use case diagram of the UML 1.5 have many corresponding concepts in the other use case approaches. Moreover, each of the analyzed use case approaches has introduced concepts that do not have correspondences in the UML 1.5. We inspected the system world concepts in the diagrams and noticed that, on a general level, the concepts of the system world are covered quite averagely.

We observed that the framework fulfils three objectives. Firstly, it has quite a broad coverage for the concepts of the development and system world. Secondly, the framework brings out the essential aspects of the RE, because it is in congruence with the three dimensions of the RE framework (Pohl 1994). The analysis of the use case approaches acted as a test of the third objective, which is

to emphasise the characteristics and concepts of and the differences and similarities between the use case approaches. Basically, the framework forms a solid foundation for comparing the use case approaches. However, some classification concepts proved to be less valuable than others (e.g. the multiplicity and normativeness facets).

In conclusion, the study has answered to the research problems. We carried out an overall analysis of the use case approaches with a framework, which is in congruence with the three dimensions of the RE framework. Therefore, we investigated the ways the use case approaches cover requirements engineering. The analysis answered the question, what kind of characteristics, concepts, differences and similarities the approaches have. Particularly, it was important to emphasise the divergent purposes for and the forms in which the use cases have been suggested for in RE. To fulfil the objective, we analysed use case approaches that represents different purposes and representation formats.

Use case classification frameworks have been used to analyse use cases in previous studies, as well (Rolland et al. 1998; Hurlbut 1997; Antón and Potts 1998). In comparison to the other examinations, the study has the following merits. Firstly, we analysed use case approaches that, on a general level, have been introduced quite recently. Secondly, we highlighted the common characteristics and concepts of and differences and similarities between the use case approaches, which is, perhaps, never done before. Thirdly, we analysed the inter-dependencies between the characteristics. Lastly, metamodeling was used to gain a more detailed picture of the use case approaches. In the literature, the metamodels of the use case approaches are rare and they have, according to our knowledge, never before presented in conjunction with an analysis of the use case approaches.

The study has implications for the practise, because the study helps to understand the achievements gained from the recently developed use case

approaches. The results may be used when choosing a use case approach for a systems development project. Understanding of the different forms the use cases may take and the benefits of applying these forms in practise are valuable.

Furthermore, the study has the following theoretical implications. A comparative evaluation of three use case classification frameworks and the concepts in them has been made. Therefore, the study situates the practise of the use case classification frameworks and assist researches to develop new ones. An existing use case classification framework is modified and used to analyse recently developed use case approaches. In addition, metamodeling was used to get a more detailed picture of the approaches. As a result, the study assists researchers to find gaps and deficiencies among the use case approaches and to develop new ones. For instance, the location concept could be valuable and therefore modelled in a use case diagram in the mobile systems development (see Kosiuczenko (2002) for location concept in sequence diagrams).

In this study we carried out an overall analysis of fifteen use case approaches and an in-depth analysis of five approaches. In order to get a more precise view of the recently introduced use case approaches, a larger number of use case approaches should be investigated. We used particular selection criteria when choosing the analysed approaches (see Chapter 5 for more details). Therefore, when using different selection criteria, diverse results would, perhaps, be gained. However, the criteria proved to be valuable because the study fulfils its objectives.

In a further study, it would be reasonable to concentrate on a certain type of use case approaches, such as the goal-oriented approaches. Moreover, it would be possible to focus on approaches that are mainly used in some other application domain, such as in business process re-engineering. In the in-depth analysis we

analysed five use case approaches that apply differing use case diagrams. The amount of the analyzed approaches could be expanded by metamodeling use cases applying formal representation formats, such as formal language expressions and Petri nets. In addition, e.g. structured descriptions, which are common among the approaches, could be metamodeled.

REFERENCES

Alexander, I. 2002a. Initial Industrial Experience of Misuse Cases in Trade-off Analysis. In D. C. Martin (Ed.) Proceedings of the IEEE Joint International Conference on Requirements Engineering (RE'02), Essen, Germany, September 9-13. Los Alamitos: IEEE Computer Society, 61-68.

Alexander, I. 2002b. On Abstraction in Scenarios. Requirements Engineering 6, 252-255.

Allenby, K. & Kelly, T. 2001. Deriving Safety Requirements Using Scenarios. In F. M. Titsworth (Ed.) Proceedings of the 7th International Workshop on Requirements Engineering (RE'01), Toronto, Canada, August 27-30. Los Alamitos: IEEE Computer Society, 228-235.

Anda B., Sjøberg D. & Jørgensen M. 2001a. Quality and Understandability of Use Case Models. In J. Lindskov Knudsen (Ed.) Proceedings of the 15th European Conference on Object-Oriented Programming (ECOOP 2001), Budapest, Hungary, June 18-22. LNCS 2072. Berlin: Springer-Verlag, 402-428.

Anda, B., Dreiem, H., Sjøberg, D. & Jørgensen, M. 2001b. Estimating Software Development Effort based on Use Cases – Experiences from Industry [online]. In Fourth International Conference on the Unified Modelling Language – "Modelling Languages, Concepts and Tools" (UML 2001), Toronto, Canada, October 1-5 [Referred on 12.2.2003]. Available at URL: <http://www.idi.ntnu.no/emner/sif8080/docs/faglig/uml2001anda.pdf >.

Antón, A., Carter, R., Dagnino, A., Dempter, J. & Siege, D. 2001. Deriving Goals from a Use Case Based Requirements Specification. Requirements Engineering 6, 63-73.

Arlow, J. 1998. Use Cases, UML Visual Modelling and the Trivialisation of Business Requirements. Requirements Engineering 3, 150-152.

Barnard, J. 2001. Use Case Tree. Journal of Object-Oriented Programming 13 (10), 19-27.

Berard, E. V. 1996. Be Careful With "Use Cases" [online]. Object Agency [Referred on 15.8.1996]. Available at URL: <http://www.toa.com/pub/html/use_case.html>.

Berg van den, K. G. & Simons, A. J. H. 1999. Control-Flow Semantics of Use Cases in UML. Information and Software Technology 41, 651-659.

Berkem, B. 1999. Traceability Management from Business Processes to Use Cases with UML. A Proposal for Extensions to the UML's Activity Diagram Through Goal-Oriented Objects. Journal of Object-oriented Programming 12 (5) 29-34.

Berry, D. M. & Lawrence, B. 1998. Requirements Engineering. IEEE Software 15 (2), 26-29.

Booch, G., Rumbaugh, J. & Jacobson, I. 1999. The Unified Modelling Language User Guide. Reading, MA: Addison Wesley Longman.

Brinkkemper, S. 1990. Formalisation of Information Systems Modeling. University of Nijmegen, PhD Thesis.

Brooks, F. 1986. No Silver Bullet: Essence and Accidents of Software Engineering. In Proceedings of IFIP Congress 1986, Dublin, September 1-5. Information Processing 86. North Holland: Elsevier Science Publishers (BV), 1069-1076.

Buhr, R. J. A. 1998. Use Case Maps as Architectural Entities for Complex Systems. IEEE Transactions on Software Engineering 24 (12), 1131-1155.

Chou, S. - C. & Chen, J. - Y. 2000a. Process Program Development Based on UML and Action Cases - Part 1: The Model. Journal of Object-Oriented Programming 13 (2), 21-27.

Chou, S. - C. & Chen, J. - Y. 2000b. Process Program Development Based on UML and Action Cases - Part 2: The Method. Journal of Object-Oriented Programming 13 (3), 18-26.

Cockburn, A. 1997a. Goals and Use Cases. Journal of Object-Oriented Programming 10 (6), 35-40.

Cockburn, A. 1997b. Using Goal-Based Use Cases. Journal of Object-Oriented Programming 10 (7), 56-62.

Cockburn, A. 2000. Writing Effective Use Cases. Reading, MA: Addison Wesley.

Cockburn, A. & Fowler, M. 1998. Question Time! about Use Cases. In Proceedings of the 1998 ACM Sigplan Conference on Object-Oriented Programming, Systems, Languages and Applications (OOPSLA '98), Vancouver, USA, October 18-22. New York: ACM Press. ACM Sigplan Notices 33 (10), 226-229.

Conklin, J. & Bagemann, M. 1988. gIBIS: A Hypertext Tool for Exploratory Policy Discussion. ACMTOOIS 6 (4), 303-331.

Conklin, J., Burges, K. & Yakemovic K. 1991. A Process Oriented Approach to Design Rationale. Human Computer Interaction 6 (3-4), 357-391.

Dano, B., Briand, H. & Barber, F. 1997. A Use Case Driven Requirements Engineering Process. Requirements Engineering 2, 79-91.

Davis, A. M. & Hsia, P. 1994. Giving Voice to Requirements Engineering. IEEE Software 11 (2), 12-15.

Doping, B. & Parsons, J. 2000. Understanding the Role of Use Cases in UML: A Review and Research Agenda. Journal of Database Management [online] 11 (4), 28-36 [Referred on 24.9.2002]. Available at URL: <http://proquest.umi.com/pqdweb?Did=000000126518971&Fmt=4&Deli=1&Mtd=1&Idx=2&Sid=2&RQT=309>.

Dutoit, A. H. & Paech, B. 2002. Rationale-Based Use Case Specification. Requirements Engineering 7, 3 -19.

Feijs, L. M. G. 2000. Natural Language and Message Sequence Chart Representation of Use Cases. Information and Software Technology 42, 633-647.

Fowler, M. 1999. UML Distilled – A Brief Guide to the Standard Object Modeling Language [online]. 2nd edition. Reading, MA: Addison-Wesley [Referred on 26.5.2003]. Appendix B. Changes Between UML Versions. Available at URL: <http://www.martinfowler.com/umlsupp/changes.pdf>.

Gigch van, J. P. 1991. System Design Modeling and Metamodeling. New York: Plenum Press.

Glintz, M. 1995. An Integrated Formal Model of Scenarios Based on Statecharts [online]. In W. Schäfer, & P. Botella (Eds.) Proceedings of the Fifth European Software Engineering Conference (ESEC '95), Sitges, Spain, September. LNCS 989. Berlin: Springer-Verlag, 254-271 [Referred on 10.12.2002]. Available at URL: <http://www.ifi.unizh.ch/groups/req/ftp/papers/Modeling_scenarios.pdf>.

Gough, P. A., Fodemski, F. T., Higgins, S. A. & Ray, S. J. 1995. Scenarios – an Industrial Case Study and Hypermedia Enhancements. In M. Jackson (Ed.) Proceedings of Second IEEE International Symposium on Requirements Engineering (RE'95), York, UK, March 27-29. Los Alamitos: IEEE Computer Society Press, 10-17.

Gross, D. & Yu, E. 2001. From Non-Functional Requirements to Design through Patterns. Requirements Engineering 6, 18-36.

Haumer, P., Pohl, K. & Weidenhaupt, K. (1998) Requirements Elicitation and Validation with Real World Scenes. IEEE Transactions on Software Engineering 24 (12), 1036-1054.

Henderson-Sellers, B. Zowghi, D. Klemola, T. & Parasuram, S. 2002. Sizing Use Cases. How to Create a Standard Metrical Approach. In J. - M. Bruel & Z. Bellahsène (Eds.) Proceedings of 8ht International Conference on Object-Oriented Information Systems (OOOS´02), Montpellier, France, September 2. Lecture Notes in Computer Science 2426, 409-421.

Hsia, P., Samuel, J., Gao, J., Kung, D., Toyoshima, Y. & Chen, C. 1994. Formal Approach to Scenario Analysis. IEEE Software 11 (2), 33-41.

Hofmann, H. F. & Lehner, F. 2001 Requirements Engineering as a Success Factor in Software Projects. IEEE Software 18 (4), 58-66.

Hurlbut, R. 1997. A Survey of Approaches For Describing and Formalizing Use Cases [online]. [Referred on 1.11.2001]. Available at URL: <http://www.iit.edu/~rhurlbut/xpt-tr-97-03.html>.

IEEE. 1984. IEEE Recommended Practice for Software Requirements Specifications. ANSI/IEEE Standard 830-1984.

IEEE. 1990. IEEE Standard Glossary of Software Engineering Terminology. IEEE Std 610.12-1990.

IEEE. 1998a. IEEE Guide for Developing System Requirements Specifications. IEEE Standard 830-1998.

IEEE. 1998b. IEEE Guide for Developing System Requirements Specifications. IEEE Standard 1233-1998.

Iivari, J. 1991. Object-Oriented Design of Information Systems: the Design Process. In V. Van Assche, B. Moulin & C. Rolland (Eds.) Proceedings of the IFIP TC8/WG8.1 Working Conference on the Object Oriented Approach in Information Systems, Canada. Amsterdam: North-Holland.

Insfrán, E., Pastor, O. & Wieringa, R. 2002. Requirements Engineering-Based Conceptual Modelling. Requirements Engineering 7, 61-72.

Jaaksi, A. 1997. Object-Oriented Development of Interactive Systems. Tampere University of Technology, Publications 201, PhD Thesis.

Jaaksi, A. 1998. Our Cases with Use Cases. Journal of Object-Oriented Programming 10 (9), 58-65.

Jackson, M. 1995. Software Requirements and Specifications. A Lexicon of Practice, Principles and Prejudices. Reading, MA: Addison-Wesley.

Jacobson, I. & Christerson, P. 1995. A Growing Consensus on Use Cases. Journal of Object-Oriented Programming 8 (1), 15-19.

Jacobson, I., Christerson, P. & Overgaard G. 1992. Object-Oriented Software Engineering: A Use Case Driven Approach. Reading, MA: Addison Wesley Longman.

Jacobson, I. 1995. Formalizing Use-Case Modeling. Journal of Object-Oriented Programming 8 (3), 10-14.

Jacobson, I., Booch, G. & Rumbaugh, J. 1999. The Unified Software Development Process. Reading, MA: Addison Wesley Longman.

Juric, R. & Kuljis, J. 1999. Engineering Requirements Through Use Cases in Complex Business Environments. Requirements Engineering 4, 65-76.

Kavakli, E. 2002. Goal-Oriented Requirements Engineering: A Unifying Framework. Requirements Engineering 6, 237-251.

Kosiuczenko, P. 2002. Sequence Diagrams for Mobility. In J. Krogstie, K. Lyytinen & K. Siau (Eds.) Proceedings of the ER'02/IFIP8.1 Workshop on Conceptual Modelling Approaches to Mobile Information Systems Development (MobIMod'2002), Tampere, Finland, October 11, 25-36.

Kotonya, G & Sommerville, I. 1996. Requirements Engineering with Viewpoints. Software Engineering Journal 11 (1), 5-18.

Kruchten, P. 2001. The Rational Unified Process: An Introduction. 2nd ed., 4th pr. Reading, MA: Addison-Wesley.

Kösters, G., Six, H. - W. & Winter, M. 2001. Coupling Use Cases and Class Models as a Means for Validation and Verification of Requirements Specifications. Requirements Engineering 6, 3-17.

Lamsweerde van, A. & Willemet, L. 1998. Inferring Declarative Requirements Specifications from Operational Scenarios. IEEE Transactions on Software Engineering 24 (12), 1089-1113.

Lang, H. & Duggan, J. 2001. A Tool to Support Collaborative Software Requirements Management. Requirements Engineering 6, 161-172.

Lee, J. Pan, J. – I. & Kuo, J. - Y. 2001. Verifying Scenarios with Time Petri-nets. Information and Software Technology 43, 769-781.

Lee, J., Xue, N. - L. & Kuo, J. - Y. 2001. Structuring Requirement Specifications with Goals, Information and Software Technology 43, 121-135.

Lee, J. & Xue, N. - L. 1999. Analyzing User Requirements by Use Cases: A Goal-Driven Approach. IEEE Software 16 (4), 92-100.

Lee, W. J., Cha, S. D. & Kwon, Y. R. 1998. Integration and Analysis of Use Cases Using Modular Petri Nets in Requirements Engineering. IEEE Transactions on Software Engineering 24 (12), 1115-1129.

Leite, J. C. S. P., Rossi G., Balaquer, F., Maiorana, V., Kaplan, G., Hadad, G. & Oliveros, A. 1997. Enhancing a Requirements Baseline with Scenarios. In Proceedings of Third IEEE International Symposium on Requirements Engineering (RE'97), Annapolis, USA, January 6-10. Los Alamitos: IEEE Computer Society, 44-53.

Leonardi, M. C. & Leite, J. C. S. P. 2002. Using Business Rules in EXtreme Requirements. A. Banks Pidduck, J. Mylopoulos, C. C. Woo, M. Tamer Ozsu (Eds.) Proceedings of Fourteenth International Conference on Advanced Information Systems Engineering (CAISE'2002), Toronto, Canada, May 27-28. Lecture Notes in Computer Science LNCS 2348, 420-435.

Li, L. 1999. A Semi-Automatic Approach to Translating Use Cases to Sequence Diagrams. In R. Mitchell, A.C. Wills, J. Bosch, & B. Meyer (Eds.) Proceedings on Technology of Object-Oriented Languages and Systems (TOOLS 29), Nancy, France, June 7–10. Los Alamitos: IEEE Computer Society, 184-193.

Lilly, S. 2000. How to Avoid Use-Case Pitfalls. Software Development [online]. January 2000 [Referred on 14.6.2002]. Available at URL: <http://www.sdmagazine.com/print/documentID=11235>.

Lunn, K. & Lindsay, A. 2002. An Object Model to Support Requirements Development [online]. In T. Halpin, K. Siau & J. Krogstie (Eds.) Proceedings of 7th CaiSE/IFIP WG8.1 International Workshop on Evaluation of Modelling Methods in Systems Analysis and Design (EMMSAD'02), Toronto, Canada, May 27-28, 59-65. [Referred on 1.11.2001]. Available at URL: <http://scom.hud.ac.uk/scomkl2/prism/Papers/Requirements%20Modelling%20-%20Object%20Model.doc>.

Lycett, M. 2001. Understanding 'Variation' in Component-Based Development: Case Findings from Practice. Information and Software Technology 43, 203-213.

McLeod, G. 2000. Beyond Use Cases [online]. In K. Siau (Ed.) Proceedings of the Fifth CaiSe/IFIP8.1 International Workshop on Evaluation of Modelling Methods in Systems Analysis and Design (EMMSAD'00), Stockholm, Sweden, June 5-6 [Referred on 5.2.2003]. Available at URL: <http://www.inspired.org/EMMSAD2000PaperFinal.pdf>.

Mitchell, I. & Lecoeuche, H. 1997. On an Improved Approach to the Elicitation of 0-0 State Machines by Use-Cases. Journal of Object-Oriented Programming 9 (9), 52-55.

Mylopoulos, J., Chung, L., Liao, S., Wang, H. & Yu, E. 2001. Exploring Alternatives during Requirements Analysis. IEEE Software 18 (1), 92-96.

NATURE Team. 1996. Defining Visions in Context: Models, Processes and Tools for Requirements Engineering. Information Systems 21, 515-547.

OMG. 2002. Meta Object Facility (MOF) Specification [online]. Version 1.4 [Referred on 6.5.2003]. Available at URL: <http://www.omg.org/docs/formal/02-04-03.pdf>.

OMG. 2003. OMG Unified Modeling Language Specification [online]. Version 1.5 [Referred on 2.4.2003]. Available at URL: <http://www.omg.org/technology/documents/formal/uml.htm>.

Phillips, C., Kemp, E. & Kek, S. M. 2001. Extending UML Use Case Modelling to Support Graphical User Interface Design. In D. D. Grant & L. Sterling (Eds.) Proceedings of 2001 Australian Software Engineering Conference, Canberra, Australia, August 27–28. Los Alamitos: IEEE Computer Society, 48-57.

Pressman, R. 1997. Software Engineering – A Practitioner's Approach. 4th, International edition. New York: McGraw-Hill Companies.

Pressman, R. 2002. Software Engineering – A Practitioner's Approach. 5th, European edition. New York: McGraw-Hill Companies.

Pohl, K. 1994. The Three Dimensions of Requirements Engineering: A Framework and its Applications. Information Systems 19, 243-258.

Pohl, K. 1996. Process Centered Requirements Engineering. New York: Wiley.

Potts, C. Takahashi, K. & Antón, A. I. 1994. Inquiry-Based Requirements Analysis. IEEE Software 11 (2), 21-32

Ralyte, J. & Achour C. 1997. Scenario Integration into Requirements Engineering Methods [online]. In Proceedings of the Workshop on the Many Facets of Process Engineering (MFPE '97), Gammarth, Tunisia,

September 22-23. CREWS Report Series 97-08 [Referred on 23.9.2002]. Available at URL: <http://cui.unige.ch/~ralyte/publications/MFPE97.pdf>.

Ratcliffe, M. & Budgen, D. 2001. The Application of Use Case Definitions in System Design Specifications. Information and Software Technology 43, 365-386.

Rational Software Corporation. 1998. Rational Unified Process – Best Practices for Software Development Teams [online]. A Rational Software Corporation White Paper [Referred on 18.10.2002]. Available at URL: <http://www.rational.com/media/whitepapers/rup_bestpractices.pdf>.

Rational Software Corporation, Oberg, R., Probasco, L. & Ericsson, M. 2001. Applying Requirements Management with Use Cases [online]. A Rational Software Corporation White Paper TP505 [Referred on 25.10.2002]. Available at URL: <http://www.rational.com/media/whitepapers/apprmuc.pdf>.

Regnell, B., Kimbler, K. & Wesslén, A. 1995. Improving the Use Case Driven Approach to Requirements Engineering [online]. In M. Jackson (Ed.) Proceedings of Second IEEE International Symposium on Requirements Engineering (RE'95), York, UK, March 27-29. Los Alamitos: IEEE Computer Society Press, 40-47 [Referred on 10.3.2000]. Available at URL: <http://www.tts.lth.se/Personal/bjornr/thesis/thesis.pdf>.

Regnell, B., Anderson, M. & Bergstrand, J. 1996. A Hierarchical Use Case Model with Graphical Representation [online]. In Proceedings of IEEE International Symposium and Workshop on Engineering of Computer-Based Systems (ECBS'96), Friedrischshafen, Germany, March 11-15. Los Alamitos: IEEE Press, 270-277. [Referred on 3.10.2002]. Available at URL: <http://www.tts.lth.se/Personal/bjornr/thesis/thesis.pdf>.

Roberts, D. Are Use Cases the Death of Good UI Design? 1999a. Uidesign.net [online] February 1999 [Referred on 15.1.2003]. Available at URL: <http://www.uidesign.net/1999/imho/feb_imho.html>.

Roberts, D. Use Cases Sill Considered Dangerous! 1999b. Uidesign.net [online] October 1999 [Referred on 15.1.2003]. Available at URL: <http://www.uidesign.net/1999/imho/oct_imho.html>.

Rolland, C. & Achour, C. 1997. Guiding the Construction of Textual Use Case Specifications. Data & Knowledge Engineering 2, 79-91.

Rolland, C., Achour, B., Cauvet, C., Ralyté, J., Sutcliffe, A., Maiden, N., Jarke, M., Haumer, P., Pohl, K., Dubois, E. & Heymans, P. 1998. A Proposal for a Scenario Classification Framework. Requirements Engineering 3, 23-47.

Rolland, C., Souveyet, C. & Achour, C. 1998. Guiding Goal Modeling Using Scenarios. IEEE Transactions on Software Engineering 24 (12), 1055-1071.

Royce, W. W. 1970. Managing the Development of Large Software Systems: Concepts and Techniques. In Proceedings of IEEE WESCON Western Electronic Show and Conventum, Los Angeles, California, August 1–9.

Rosenberg, D. & Scott, K. 2001. Top Ten Use Case Mistakes [online]. February 2001 [Referred on 14.6.2002]. Available at URL: <http://www.sdmagazine.com/print/documentID=11094>.

Rowlett, T. 1998. Building an Object Process Around Use Cases. Journal of Object-Oriented Programming 11 (1), 53-58.

Rubin, K. S. & Goldberg, A. 1992. Object Behaviour Analysis. Communications of the ACM 35 (9), 48-62.

Santander, V. F. A. & Castro, J. F. B. 2002. Deriving Use Cases from Organizational Modelling. In D. C. Martin (Ed.) Proceedings of the IEEE Joint International Conference on Requirements Engineering (RE'02), Essen, Germany, September 9-13. Los Alamitos: IEEE Computer Society, 61-68.

Siau, K., Lee, L. & Korhonen, J. 2001. Use Case Diagram in Requirement Analysis: An Empirical Investigation. In Proceedings of the International Workshop on Evaluation of Modelling Methods in Systems Analysis and Design (EMMSAD'01), Interlaken, Switzerland, June 4-5, VIII-1-7.

Siddiqi, J. & Shekaran, M. C. 1996. Requirements Engineering: The Emerging Wisdom. IEEE Software 12 (6), 15-19.

Simons, A. J. H. 1999. Use Cases Considered Harmful. In R. Mitchell, A. C. Wills, J. Bosch & B. Meyer (Eds.) Proceedings on Technology of Object-Oriented Languages and Systems (TOOLS 29), Nancy, France, June 7-10. Los Alamitos: IEEE Computer Society, 194-203.

Sindre, G. & Opdahl, A. 2001. Templates for Misuse Case Description. In F. M. Titsworth (Ed.) Proceedings of the 7th International Workshop on Requirements Engineering (RE'01), Toronto, Canada, August 27–30. Los Alamitos: IEEE Computer Society, 228-235.

Sommerville, I. 1998. Software Engineering. 5th Edition. Reading, MA: Addison Wesley Longman.

Sowa, J. F. & Zachman, J. A. 1992. Extending and Formalizing the Framework for Information Systems Architecture. IBM Systems Journal 31 (3), 590-616.

Sutcliffe, A. 1996. A Conceptual Framework for Requirements Engineering. Requirements Engineering 1, 170-189.

Sutcliffe, A. G., Maiden, N. A. M., Minocha, S. & Manuel, D. 1998. Supporting Scenario-Based Requirements Engineering. IEEE Transactions on Software Engineering 24 (12), 1072-1088.

Tolvanen J. - P. 1998. Incremental Method Engineering with Modeling Tools: Theoretical Principles and Empirical Evidences [online]. University of Jyväskylä, Dissertation Thesis. Jyväskylä Studies in Computer Science, Economics and Statistics 47 [Referred on 14.05.2003]. Available at URL: <http://www.cs.jyu.fi/~jpt/Tolvanen_dissertation.pdf>.

Wade, S. & Hopkins, J. 2002. A Framework for Incorporating Systems Thinking into Object Oriented Design [online]. In Proceedings of Seventh CaiSE/IFIP WG8.1 International Workshop on Evaluation of Modelling Methods in Systems Analysis and Design (EMMSAD'02), Toronto, Canada, May 27-28, 49-58. [Referred on 14.05.2003]. Available at URL: <http://scom.hud.ac.uk/scomkl2/prism/Papers/A%20Framework%20for%20Incorporating%20Systems%20Thinking%20into%20Object%20Oriented%20Design.pdf>.

Wallnau, K. 2000. Wheels within Wheels. Model Problems in Practise. news@sei interactive [online] 3 (3), 6-14. Carnagie Melon Software Engineering Institute [Referred on 18.10.2002]. Available at URL: <http://interactive.sei.cmu.edu/news@sei/entire-issues/2000/entire-issue-sum-00.pdf>.

Weidenhaupt, K., Pohl, K., Jarke, M. & Haumer, P. 1998. Scenarios in System Development: Current Practice. IEEE Software 15 (2), 34-45.

Woo, H. & Robinson, W. 2002. Reuse of Scenario Specifications Using an Automated Relational Learner: A Lightweight Approach. In D. C. Martin (Ed.) Proceedings of the IEEE Joint International Conference on Requirements Engineering (RE'02), Essen, Germany, September 9-13. Los Alamitos: IEEE Computer Society, 173-180.

Vadaparty, K. 2000. Use Cases – Basics. Journal of Object-Oriented Programming 12 (9), 4-8.

Zachman, J.A. 1987. A Framework for Information Systems Architecture. IBM Systems Journal 26 (3), 276-292.