Research Topics

Wytse Jan Posthumus

August 3, 2011

Preface

This document is a background study on the topic of requirements engineering and the situated Cognitive Engineering methodology, which was developed at TNO.

The work was done to gain insight in the topic of my final master project at TNO Soesterberg. This document was written under supervision of Dr. Luís Ferreira Pires, Associate Professor of the Software Engineering group at the University of Twente.

Wytse Jan Posthumus

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

This chapter introduces this work by motivating this research and giving our main objective and research questions.

This chapter is structured as follows: Section 1.1 gives the motivation behind the research, Section 1.2 describes our main objective and research questions and finally Section 1.3 presents the outline of this report.

1.1 Motivation

With the increasing complexity of systems, more is demanded from users. Engineering methods, however, do not always consider the limitations of the psychological capabilities of users. To address these limitations, a new approach to create systems was developed in the 1980s, namely Cognitive Engineering. The Cognitive Engineering approach was developed to improve the design of systems that are oriented at effective human-computer interaction (Hollnagel and Woods, 1983).

To increase the knowledge base of systems which are designed in a certain domain, TNO developed the situated Cognitive Engineering methodology as an extension to Cognitive Engineering (Neerincx and Lindenberg, 2008).

Situated Cognitive Engineering (sCE) focuses on the situated theory of cognition. In this approach relevant human factor knowledge (i.e., theories) is selected by system designers and tailored to the specific operational demands and envisioned technologies, which is explicated in the design rationale. This design rationale holds design decisions for the requirements of the system. It consists of use cases, which are stories about users undertaking activities, and claims, which are hypothesis with positive and negative effects of a requirement.

Explicating the design rationale with use cases and claims results in a requirement baseline which is created from the perspective of the user. Because each requirement is justified by claims, it becomes apparent why the system would increase human performance by fulfilling the requirement.

Projects which apply the sCE methodology often consist of interdisciplinary groups who work together. For example, sometimes people with background in Psychology have to work with people from Software Engineering. Not all disciplines have the same way of thinking and they even may work on different locations. To enable cooperative working all those groups must have a common level of thinking. Sometimes definitions of terms can differ between groups, which makes communication quite difficult.

To enable cooperative working, a tool needs to be developed. However, knowledge of comparing the field of requirements engineering to the situated Cognitive Engineering methodology to get insight for designing a tool is limited.

1.2 Objectives

As stated above, there is lack of insight in comparing requirements engineering and the situated Cognitive Engineering methodology. This work aims to get more insight in requirements engineering and the situated Cognitive Engineering methodology and to compare them. This knowledge can be used in the execution of the master project, which aims at designing a tool. To gain more insight the following questions have to be answered:

1. What is requirements engineering?

The sCE methodology is a requirements engineering method. Before the sCE methodology can be understood, we need some background information about requirements engineering.

2. What is the situated Cognitive Engineering methodology?

The sCE methodology needs to be understood before a tool can be developed. It is important to know the purpose of sCE and the benefits of applying the methodology.

3. What are the key focus points of the sCE methodology?

What are the phases of the methodology and what are its main artefacts? How does the sCE methodology compare to other requirement engineering methods?

1.3 Document structure

The document is structured as follows: Chapter 2 gives the background of this work by defining requirements engineering, Chapter 3 describes the situated Cognitive Engineering Methodology and finally Chapter 4 gives a conclusion of this work.

Chapter 2

Background

This chapter gives background information about requirements engineering and its related methods and tools available in the literature. The purpose of this chapter is to present our insights in the current state-of-the-art in the requirements engineering field.

This chapter is structured as follows: Section 2.1 defines the concept of requirement and Section 2.2 explains requirements engineering.

2.1 Requirements

Requirements play a key role in requirements engineering, hence it is important to understand the concept of requirement.

2.1.1 Defining requirements

In the literature, there are dozens of definitions for the term *requirement*. All these definitions slightly differ from each other. These definitions mostly depend on the area of expertise from which the definition is given. However, in general, the essence of these definitions remain the same. For this thesis, the definition of Sommerville and Sawyer (Sommerville and Sawyer, 1997) is chosen, because fits our purpose. The definition is as follows:

> *"Requirements are a specification of what should be implemented. They are descriptions of how the system should behave, or of a system property or attribute. They may be a constraint on the development process of the system."* (Sommerville and Sawyer, 1997)

According to this definition, requirements describe what a system should do, or some property it should possess, after implementation, which is, ideally, the result of a project if that aims at building the system.

2.1.2 Requirements Characteristics

The definition, as given in the previous section, is still quite general. To further refine the definition, we give some additional characteristics to requirements. Many different characteristics of requirements are defined within the literature. A list of generally accepted characteristics has been described by Davis (Davis, 1993). Davis describes ten desirable characteristics with corresponding explanation as shown in Table 2.1.

In this work, the focus has been on the two characteristics which are indicated in bold font in Table 2.1, namely *verification* and *traceability*. These characteristics were chosen because they are of importance for the sCE methodology. Both characteristics are explained in more detail below.

Requirements Verification

Verification was chosen because in essence it implies four other characteristics. For example, if a requirement is incomplete, inconsistent, infeasible, or ambiguous, the requirement is also unverifiable (Drabick, 1999).

The aim of requirements verification is to determine whether a product properly implements a requirement. Several methods are proposed in the literature to carry out verification. Davis names four methods: inspection, demonstration, test, and analysis (Davis, 1993).

In this work, requirements verification plays a key role. If a requirement is not verifiable, determining whether the requirement was correctly implemented becomes more of an opinion since no objective verification is possible.

Requirements Traceability

Traceability of requirements consists of documenting the life of a requirement. This means that one should describe how the requirement propagates from the beginning of the project to the implementation of the system.

The aim of making requirements traceable, is to link the business needs and wishes of stakeholders to each requirement. This enables the designers to create alternative requirements if some requirements may not be feasible.

Traceability of requirements within design documents can be achieved by uniquely labelling these requirements (Wiegers, 2003). A minimal approach to achieve traceable requirements is to define (1) a unique identifier for the requirement, (2) a requirement description, (3) original motivation, and (4) future design consequences.

Characteristic	**Explanation**
Unitary (Cohesive)	A requirement addresses one and only one aspect of the system.
Complete	A requirement is fully stated in one place with no missing information.
Consistent	A requirement does not contradict any other requirement and is fully consistent with all authoritative external documentation.
Non-Conjugated (Atomic)	A requirement is atomic, i.e., it does not contain conjunctions. E.g., "The postal code field must validate American and Canadian postal codes" should be written as two separate requirements: (1) "The postal code field must validate American postal codes" and (2) "The postal code field must validate Canadian postal codes".
Traceable	The requirement meets all or part of a business need as stated by stakeholders and authoritatively documented.
Current	A requirement is still valid after passage of time, otherwise it should be removed or replaced.
Feasible	It should be possible to fulfil a requirement within the constraints of the project.
Unambiguous	A requirement is concisely stated without recourse to technical jargon, acronyms (unless defined elsewhere in the Requirements document), or other esoteric verbiage. It expresses objective facts, not subjective opinions. It is subject to one and only one interpretation. Vague subjects, adjectives, prepositions, verbs and subjective phrases are avoided. Negative statements and compound statements are prohibited.
Mandatory	A requirement represents a stakeholder-defined characteristic the absence of which will result in a deficiency that cannot be ameliorated. An optional requirement is a contradiction in terms.
Verifiable	The implementation of a requirement can be determined through one of four possible methods: inspection, demonstration, test or analysis.

Table 2.1: Characteristics of desirable requirements (Davis, 1993)

2.1.3 Types of Requirements

Various types of requirements exist. Some types focus on the behaviour of a system, whereas other requirement types focus on how the technical implementation should look like.

For each situation a specific method for classifying requirements may be appropriate. Classifying requirements can be useful to identify similar requirements which may focus on the same area of the system. Also classifying requirements fosters interoperability and reuse of requirements. Below two generally known classification methods are explained, namely the method from Kulak (Kulak and Guiney, 2004) and the method from Wiegers (Wiegers, 2003).

Method 1 - Kulak

The first classification method divides requirements in two types:

1. Functional requirements

   Functional requirements can be directly implemented in the software of the system. They describe functions and features of the system to be implemented. An example of a functional requirement is: 'The system shall store the data from users in a database'.

2. Non-functional requirements

   Non-functional requirements, on the other hand, are more 'hidden' requirements which describe qualitative aspects of the system. They are hidden in the sense that, although they are important, users may not realize their existence because they do not deal with the functionality of a system. A lot of these requirements can be expressed with -ility words, for example: scalability, accessibility, maintainability, testability or reliability. An example of non-functional requirement is: 'The system should response in 20 seconds.'

This method of classification plays an important role in Software Engineering, because it separates the implementation requirements from the quality requirements. This can be useful, because the functional requirements are mostly about functions of the software, and non-functional requirements can be influenced by external variables as well (e.g. hardware).

Method 2 - Wiegers

The second method for classification of requirements divides all requirements in three levels and two dimensions. Figure 2.1 illustrates the model from Wiegers for the relationships between the requirements. This model only

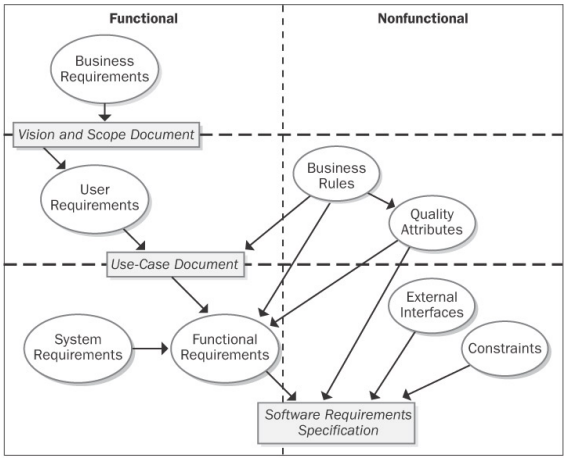

Figure 2.1: Connections among several types of requirements information (Wiegers, 2003)

covers the requirements for products, and does not include other requirements such as staffing, scheduling, etc. because they do not fall within the scope of this work (Wiegers, 2003).

In Figure 2.1 the three levels are shown as horizontal rows, namely the *business requirements*, *user requirements* and *functional requirements*. The columns show the dimensions, namely *functional* and *non-functional*. Each rectangular block represents a deliverable document which contains requirements information, which is represented as ovals.

Business requirements describe the reason why the project is started. They should include the benefits that justify the execution of the project. They are mostly used to communicate how the business process should work. These requirements are contained in a vision and scope document.

User requirements describe the benefits of the product from the view of the user. They mostly incorporate what the user will be able to do, such as goals or tasks that the user should be able to perform. User requirements can be visualized by means of scenarios and use cases (see Section 2.1.4).

Functional requirements describe what the developers is supposed to build. They are like user requirements in the sense that they describe the 'what' of the system. However, these requirements describe from the system's point of view. Most of these requirements contain the word 'shall', like 'the system shall do ...'.

Figure 2.1 illustrates that the lowest level also includes *System requirements*. These requirements describe the top-level requirements of a product that contains multiple subsystems. They can be seen as a platform or the context on which the product has to be build.

2.1.4 Use Cases

A *use case* is a description of the usage of a system from one or more users' point of view (Windle and Abreo, 2003). These users are often denoted as *actors*, because they can be either a person, a system or some part of a machine (e.g. a timer).

Each use case describes the interaction between actors and the system. The use cases are specified using the information from scenarios. Scenarios are stories of actors undertaking activities, which are often described using storyboards. A use case should at least contain the following information:

1. Motivation of the use case
2. State of the system at the start of the use case
3. State of the system at the end of the use case
4. Normal sequence of events describing the interaction between the actor and the system
5. Any alternative courses to the normal sequence of events
6. Any system reactions to exceptions the system encounters

Use case diagrams

Use cases can also be described using figures. One of the most well known techniques for modelling use cases is the Unified Modelling Language (UML). UML use case diagrams give a graphical representation of the actors and their use cases.

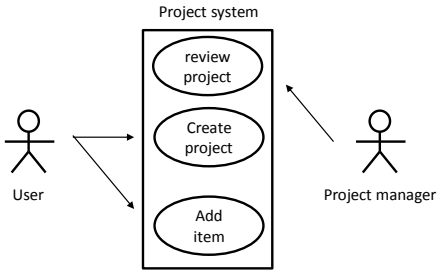

Figure 2.2: Use case diagram example (Windle and Abreo, 2003)

Figure 2.2 shows a simple example of a use case diagram. The sticky man on the left side represents an actor, in this case a user, who wants to accomplish the use case, in this case to create a project or to add an item. Multiple actors can be added to a use case who all interact with a system.

2.2 Requirements Engineering

This section explains how requirements can be captured. This is important for this study, because the main purpose of the situated Cognitive Engineering methodology is the process of defining requirements.

2.2.1 Defining Requirements Engineering

Requirements engineering, or requirements capturing, is the process of acquiring requirements. Many different definitions of requirements engineering exist in the literature. For this work the definition of software requirements engineering from Nuseibeh and Easterbrook (Nuseibeh and Easterbrook, 2000) was chosen, because it matches the view of the situated Cognitive Engineering methodology. The definition is as follows:

> *"The primary measure of success of a software system is the degree to which it meets the purpose for which it was intended. Broadly speaking, software systems requirements engineering (RE) is the process of discovering that purpose, by identifying stakeholders and their needs, and documenting these in a form that is amenable to analysis, communication, and subsequent implementation"* (Nuseibeh and Easterbrook, 2000)

The definition stated above describes RE as a process in which you need to find 'that purpose'. This purpose is the description of what a system should do, and why it should be build, hence the *intention* of the system.

2.2.2 Requirements engineering phases

Requirements engineering can be understood by considering the two main requirements engineering phases, namely *requirements development* and *requirements management* (Wiegers, 2003). While the first phase is about defining requirements, the second phase is about managing the requirements, and their changes, during implementation. Although both phases share similar concepts and they often overlap in time, in this work they will be treated as separate phases for simplicity. This study focuses only on the requirements development phase, because the key point of this study is capturing the requirements, instead of actually managing them. Therefore requirements management is out of scope of this study.

The requirements development phase consists of four sub-phases: (1) *Elicitation*, (2) *Analysis*, (3) *Specification*, and (4) *Validation*. Figure 2.3 shows the phases of requirements engineering. The lower part shows the four sub-phases of requirements development.

The order in which the requirements development phases are executed is of vital importance to understand and implement the business needs and wishes of the users.

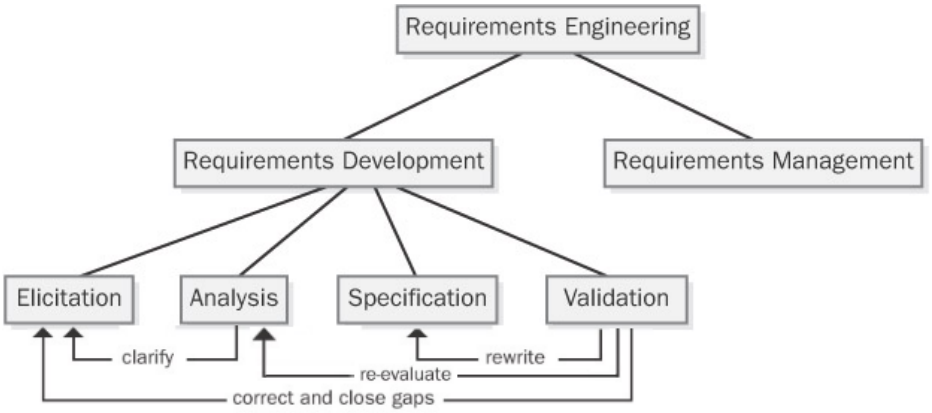

Figure 2.3: Phases of requirements engineering (Wiegers, 2005)

The *Elicitation* phase focuses on understanding the users and discovering their needs. An important sub-step within this phase is the discovery of the 'stakeholders' related to the software system. Identifying these stakeholders in an early stage helps construct the use cases at a later stage.

The *Analysis* phase has the main goal of clarifying the results of the elicitation phase. The analysis sub-phase aims to derive more detailed requirements from higher-level requirements (Wiegers, 2005). This sub-phase also aims at creating prototypes, graphical analysis models and performing tests. Eventually, the analysis sub-phase provides better understanding of the information gathered during the elicitation sub-phase.

The *Specification* phase aims at capturing requirements information in such a way that it facilitates communication with various system stakeholders. Capturing requirements information usually means documenting them in text documents, although the use of graphical models and tables is advisable (Wiegers, 2005).

The *Validation* phase aims at ensuring the correctness of the captured requirements information, in such a way that these requirements satisfy the customer. In practice, the validation sub-phase implies the modification and rewriting of the earlier defined requirements. The requirements development process is an ongoing process, thus iteration between the four sub-phases is required in order to obtain proper requirements.

2.2.3 Requirements Engineering Methods

Several requirements engineering methods have been discussed in the literature. Tsumaki and Tamai give an overview of four requirement elicitation approaches (Tsumaki and Tamai, 2005).

1. Domain decomposition

   The domain decomposition approach is to decompose the target domain into sub-domains. Step by step the sub-domains are decomposed until they are of a manageable size so that they can be translated into requirements e.g., (Prieto-Díaz, 1990).

2. Goal-oriented

   Goal-oriented approaches focus on the goals that need to be achieved by a project, and translates them into requirements. For example, each use case could be translated into a requirement e.g., KAOS (Lapouchnian, 2005).

3. Scenario-based

   Scenario-based approaches focus on the creation of scenarios and their integrated set of use cases. This approach is the most relevant for this work, because the situated Cognitive Engineering method can be placed in that category.

4. Brainstorming

   The brainstorming approach focuses on the creation of new products in areas where there is not much experience. It focuses on generating new ideas and creating an orderly system from chaos e.g., KJ method (Takeda et al., 1993)

Besides the requirements elicitation methods described above, one could use Agile development approaches, such as in (Beck et al., 2001). However, these approaches focus more on the software development process, which is out of scope for this work.

2.2.4 Requirements Engineering Tools

A requirements design specification can be represented as a textual document, however this has some limitations when projects become complex and interrelated requirements change often. Wiegers states the following four important limitations (Wiegers, 1999):

- It is difficult to keep this textual document current, especially when requirements change rapidly.
- It is hard to communicate changes to the affected team members.
- It is difficult to store supplementary information about each requirement
- It is hard to define links between functional requirements and corresponding use cases, designs, code, tests, and project tasks.

Some tools have been developed specifically to address these problems. Wiegers states seven reasons why one should use a tool to manage requirements (Wiegers, 1999).

Manage versions and changes. Tools can track the changes which are made in the requirements specification. Keeping a history of changes makes it possible to revert back to older versions.

Store requirements attributes. The attributes of a requirement should be stored and open to view, and edit, for every project member. Several pre-defined attributes can be generated automatically, and custom attributes can be added for extra information.

Link requirements to other system artefacts. Defining links between requirements and other artefacts can help gain overview in the design specification. When a change is proposed, it is possible to trace the impact of the changes on other requirements.

Track Status. When using a tool it is possible to track the current status of the project. For each requirement status information can be added. This makes it possible to assert if the project is running on schedule.

View requirement subsets. Sorting, filtering and searching through the requirements baseline increases the insight in the requirements specification.

Access control. Setting permissions for each user makes sure the right information is shared by the right people. Remote access makes cooperation with geographically separated group members possible.

Communicate with stakeholders. Communication among stakeholders is important when developing a system. With the use of tools it is possible to keep the stakeholders up-to-date, which should reduce the risk of miscommunication.

Examples of popular requirements management tools are: Borland CaliberRM (Borland Software Corporation, 2011), IBM's DOORS (IBM, 2011a), IBM's Rational RequisitePro (IBM, 2011b) and Sparx Systems Enterprise Architect (Sparx Systems, 2011).

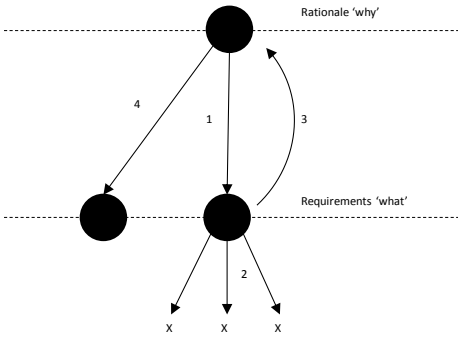

Figure 2.4: Using the design rationale to create alternative requirements

2.2.5 Requirements Design Rationale

Requirements engineering answers the questions about 'what' the system should do. With software projects though, answering the 'why' question is evenly important because it opens the option to create alternative requirements when they become infeasible after verification. Answering the 'why' question is formally known as explicitly documenting the reasons behind decisions made when designing a system or artefact, hence stating the *requirements design rationale*. A design rationale is in its simplest form: explicitly listing decisions made during a design process and the reasons why those decisions were made, as stated by Jarczyk (Jarczyk et al., 1992).

To create a suitable design rationale, a requirement should at least include the following concepts: the reasons behind a design decision, its justification, alternatives considered, the trade-offs evaluated, and the argumentation that led to the decision (Lee, 1997).

The important aspect of a design rationale is that it opens the option to create alternative requirements. Sometimes a requirement becomes infeasible after verification. The design rationale can then be used to trace the requirement back to its origin. This origin should contain the reason why the requirement was created, and therefore a alternative requirement can be created.

Figure 2.4 illustrates the possible process of creating alternative requirements using the design rationale. Note that multiple requirements could be related to each other, making the process more complicated. However, in the simplest case, the four steps in the process could be as follows:

1. A requirement is captured.
2. Verification shows that the implementation of the requirement is infeasible.
3. The designers trace the requirement back to the design rationale.
4. An alternative requirement is defined.

In this work, the design rationale takes a prominent place, due to the focus on system design of the situated Cognitive Engineering methodology. During system design, justification of requirements plays a key role. Requirements which are not properly justified will raise the question whether they should be implemented at all. This will reduce the risk of implementing the wrong functions.

Chapter 3

The situated Cognitive Engineering Methodology

This chapter describes the situated Cognitive Engineering (sCE) methodology. The purpose of this chapter is to get insight in the sCE methodology and its relation to other requirements engineering methods.

The outline of this chapter is as follows: Section 3.1 introduces the concept of cognitive systems engineering, Section 3.2 discusses the sCE methodology and Section 3.3 compares the sCE methodology with other requirement engineering methods.

3.1 Cognitive Systems Engineering

When machines are introduced, they are mainly created as an extension to the humans physical functions. They are designed to enhance the humans' physical skills and to compensate for their limitations. Because the function of those machines are so closely related to the activities without them, not much effort will go into the design of the human interface.

However, with the capabilities of machines growing, more and more functionality comes into these machines making them systems that performs a process. This means that the user will move away from the production floor to the control room. Instead of controlling a machine, the user has to control and/or monitor a process (Hollnagel and Woods, 1983).

The machine no longer has simple actions and indicators, but becomes an information processing system that can perform complex activities and communicate in a seemingly intelligent way. Designing systems like these requires knowledge of the process of the mind, because humans have certain psychological limitations. This knowledge of the process of the mind is also known as human cognition.

To address these changes a new way of engineering was developed in the 1980s to increase the insight in the cognitive factors of human-machine

interaction. Instead of looking at the physical limits of human performance, cognitive engineering focuses on the user's psychological limits.

The main idea of cognitive engineering is that a human-machine system needs to be seen as a cognitive system. A human and a machine working together can be seen as a single entity that interacts with an external environment. It is not merely a sum of its parts (human and machine), but a system that includes the psychological sides as well (Hollnagel and Woods, 1983). For example, if a machine does not consider the maximal workload of a user and it becomes too high, then the whole system may fail.

In 2005, an extended method of the cognitive engineering was proposed, namely the CE+ method. This method adds the technology as an input to achieve a better focus in the generation of ideas and the reciprocal effects of technology. Human factors are also made explicit and are integrated in the development process (van Maanen et al., 2005).

In addition to the CE+ method, a method that is situated in the domain of interest was proposed by Neerincx and Lindenberg, namely situated Cognitive Engineering. This method uses the previously described assessments to establish a common design knowledge base, and to develop a kind of design guide for cognitive systems. (Neerincx and Lindenberg, 2008).

3.2 Situated Cognitive Engineering

This section introduces the situated cognitive engineering (sCE) methodology. First the general concept of the methodology is given, then its four phases. Finally its main entities are described.

3.2.1 Introducing the sCE Methodology

The situated cognitive engineering methodology is an extension to the CE+ method. The sCE methodology has been specifically designed to create cognitive systems that are situated in a domain.

The sCE methodology can be used to create a sound requirements baseline for joint cognitive systems for research and development projects, which are often multi-party projects with distributed teams. Use cases and claims make sure the design rationale is described properly (Neerincx and Lindenberg, 2008).

The methodology guides a system designer through three questions, namely *what*, *when* and *why*. Answering these three questions should give a solid design rationale for the designed system. These three questions are answered through the three main artefacts. These questions and their relations are displayed in Figure 3.1.

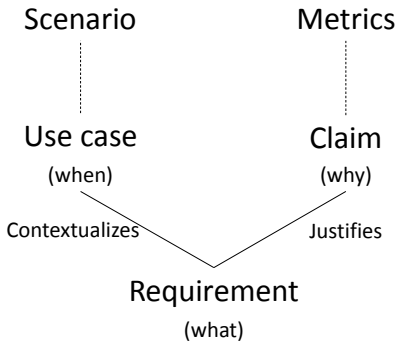

Figure 3.1: The relations between artefacts of the sCE methodology

3.2.2 Phases

The sCE methodology consists of four phases, namely: *derive*, *specify*, *test* and *refine*. Figure 3.2 displays the elements in each of these phases, which are discussed below.

Derive

The *derive* phase is the starting point of the whole requirements design process. This phase has as input operational demands, human factors knowledge and the envisioned technology which are all used to create the scenarios. These scenarios are used in the specify phase. Scenarios are stories about actors undertaking activities using technologies in a certain context. This phase can be compared to the *elicitation* phase as described in Section 2.2.2.

Specify

The *specify* phase is about capturing requirements from scenarios. The requirements are created together with their design rationale (i.e., the use cases and claims). Each requirement should be justified by claims and each claim should be truthful and exclusive, otherwise they serve no purpose and should be removed. The core functions can be seen as modules that group requirements by certain functionality. For example, in a design project for an aeroplane, you could have a core function that defines the functionality for the autopilot. This phase can be seen as a combination of both the *analysis* and *specification* phase as described in Section 2.2.2.

Test

The *test* phase forms, together with the refine phase, an iterative process which tests the requirements by using review, simulation and human-in-the-

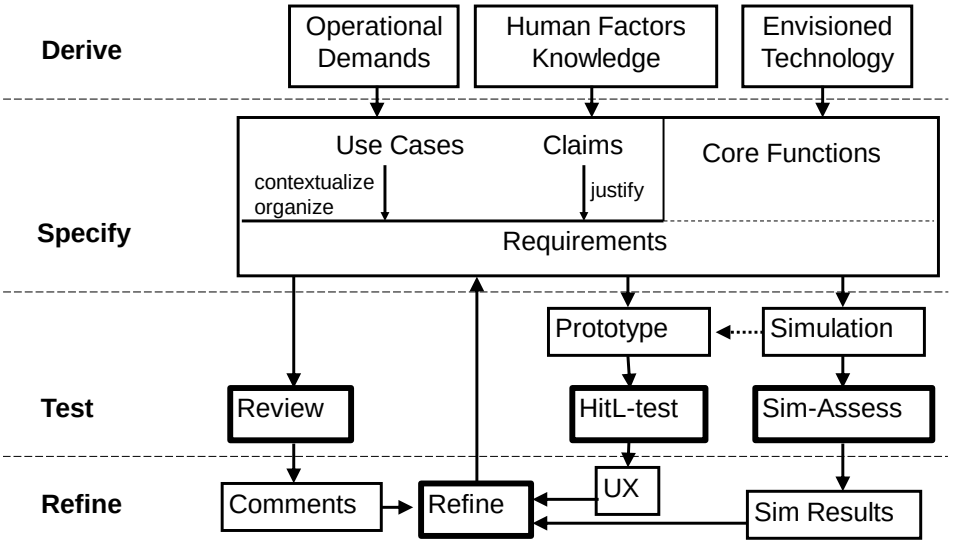

Figure 3.2: The situated Cognitive Engineering process (UX = User Experience, HitL = Human in the Loop, Sim = Simulation) (Mioch et al., 2010)

loop evaluations. The results from the tests are evaluated and used to verify if the requirements meet the claims. If some requirements do not trigger the desired effect, then they should be refined and tested again. This phase, together with the refine phase, can be compared to the *validation* phase as described in Section 2.2.2.

Refine

The *refine* phase is executed after the test phase. The test results and comments from the test phase are processed to improve the requirements and claims in order to suit the needs of the stakeholders.

3.2.3 sCE Use Cases

Use cases are one of the artefacts of the sCE methodology. Use cases are derived from the scenarios which are defined in the derive phase. A scenario can be seen as an instance of one or more use cases. Use cases should describe the general behaviour requirements and should have a specific specification format. Each use case should refer to one or more requirements.

The format of use cases as used in the sCE methodology is defined as shown in Table 3.1.

UC 1	<<identifier>>
Description:	Description of the use case.
Goals:	What is achieved by carrying out the use case.
Actors:	Main human and/or machine actors.
Pre-Conditions:	The state of the system or user just before using the function.
Post-Conditions:	The state of the system or user just after the function was used.
Trigger:	Defines the event (e.g., time, alarm) when a user needs the functionality or how the system knows that the function needs to be carried out.
Requirements:	List of requirement-numbers that relate to this use case
Main Action Sequence:	1 Step 1. 2 Step 2.
Alternative Action Sequence:	1 Step 1. 2 Step 2.

Table 3.1: Use case format

3.2.4 sCE Requirements and claims

Requirements are artefacts that describe desired facts about the system. Each use case leads to one or more requirements. A requirement is justified by claims, which point out the usefulness of the requirement. Each claim has its positive and negative effect. If the negative effects outweigh the positive ones, then the requirement should be removed (Westera et al., 2010).

Claims play an important role in the sCE methodology. They are hypotheses that tell something about the goal the requirement needs to achieve. This goal is important because it points out why it is useful to implement a requirement. Because the sCE methodology focuses on the design phase, and not on the implementation phase, a lot of time can be saved if a requirement does not need to be implemented at all. The positive and negative effects, which are described in the claims, play a key role in deciding whether to keep a requirement because they determine the usefulness of the requirement.

The format of the requirements, with their claims, as defined by the sCE methodology is shown in Table 3.2.

3.3 sCE comparison

This section compares the requirements engineering process with the situated Cognitive Engineering methodology.

The sCE methodology focuses on the design of the requirements baseline, which can be compared to the requirements development phase of the

Requirement 1	<<identifier>>
Description:	Description
Claim	*Hypothesis (e.g., fast victim finding in areas that are inaccessible for humans)* + Upsides: (e.g., fast navigation time), [measures, such as time] - Downsides: (e.g., no attention to areas the robot does not enter), [measures, such as number of misses]
Use Cases:	List of use case-numbers that link to this requirement

Table 3.2: Requirement format

requirements engineering phases. The methodology, however, does not treat the requirements management phase, as the sCE methodology only considers the design of the system, not the implementation.

The sub-phases of the requirements development phase are similar to the phases of the sCE methodology. The elicitation sub-phase from the requirements development phase is similar to the derive phase from the sCE methodology. Both phases focus on discovering the needs of the stakeholders and defining the scope of the project.

Both the analysis and specification sub-phase from the requirements development phase can be compared to the specify phase from the sCE methodology. They focus on translating the needs of the stakeholders into requirements. Defining the design rationale is also important in these phases as it gives an answer to why the system needs to be build.

The validation sub-phase of the requirements development phase can be compared to the test and refine phase of the sCE methodology. In these phases the requirements baseline is tested to assess if the requirements meet the claims. If not, the requirements need to be refined or removed.

Several different types of requirements engineering methods exist. The sCE methodology can be classified as a scenario-based approach because it focuses on the creation requirements using scenarios and use cases. This enables the requirements creation process as a dynamic activity. Requirements are elicited from the domain of interest and can be generated by stakeholders. Because the activity is dynamic it encourages inspiration and imagination (Tsumaki and Tamai, 2005).

An important aspect of the sCE methodology is the assignment of claims to requirements. These claims justify why the requirements exist in some specific design. The positive and negative effects of the requirement make the impact to the system explicit. The positive effects should outweigh the negative effects, otherwise the requirement does not properly serve its

purpose.

Together with the use cases, the claims define the design rationale of the system. Validating the claims makes sure each requirement is useful. This makes the sCE methodology extremely useful to apply when creating complex systems, because when the claims justify all the requirements, no unnecessary work is done when the system is implemented.

The complete design rationale explains why a system should be built in the first place. When all requirements are validated, The system implementation is justified. However, if the requirements are not validated, the usefulness of the whole system should be reconsidered.

Chapter 4

Conclusion

The problem statement of this report was to get more insight in both the field of requirements engineering and the situated Cognitive Engineering Methodology and to compare them to each other.

A requirement is defined as a description of the system. A desirable requirement has several characteristics, where *verification* and *traceability* have been chosen as the most important for this study. Two methods for requirements classification have been discussed (Kulak and Guiney, 2004; Wiegers, 2003).

Requirements engineering is defined as the process of acquiring requirements. Important is to find the *purpose* of the system. Requirements engineering two main phases, namely requirements development and management. Only the requirements development is considered, because requirements management is out of scope for this work.

The requirements development phase has four sub-phases, namely *elicitation*, *analysis*, *specification* and *validation*.

Several different methods exists for requirements engineering. They can be classified in four different approaches, namely domain decomposition, goal-oriented, scenario-based and brainstorming. Other methods exist, but are out of scope because they focus on the implementation phase.

Some tools exist to aid the requirements engineering process. Some advantages of using a tool are given, in contrast to only using textual documents.

Answering the 'why' question is important when designing a system. Creating a design rationale helps document the design decisions and find alternative requirements when requirements become infeasible.

The situated Cognitive Engineering methodology is defined as a method designed to create cognitive systems which are situated in the desired domain. The main goal of the method is to find an answer to the three questions 'what', 'when' and 'why'. These questions are answered by three artefacts, namely use cases, requirements and claims.

The sCE methodology consists of four phases, namely *derive*, *specify*, *test* and *refine*. In the derive phase the scenarios are defined, in the specify phase use cases, requirements and claims are captured in a specific format and in the test and refine phase the created requirements are verified and refined if necessary.

The sCE methodology was compared to requirements engineering. The sCE methodology only focuses on the requirements development phase of requirements engineering since it does not handle the requirement management phase.

The sub-phases of requirements development could be matched to the sCE methodology phases, and the sCE methodology could be classified as a scenario-based requirements engineering approach. This gives the opportunity to create requirements with a dynamic and open view.

One important difference of the sCE methodology, in comparison with regular requirements engineering, is that claims are included in the description of requirements. These claims, together with the use cases, define the design rationale of the requirements baseline. The claims make the requirements verifiable and justify their existence.

The key focus point of the sCE methodology is finding the *purpose* of a system, namely *what* needs to be built, *when* it is used and *why* it needs to be built. Because of this focus point, the sCE methodology is extremely useful when designing a system. It gives a reason why the system should be built, and if the claims do not justify the requirements, the system should not be build at all.

This report has shown that it is possible to compare the situated Cognitive Engineering methodology to the field of requirements engineering. More insight is gained in both areas and this knowledge will be used in to design a tool to improve cooperation when applying the sCE methodology.

Bibliography

K. Beck, M. Beedle, A. Van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, et al. Manifesto for agile software development. *The Agile Alliance*, pages 2002–04, 2001.

Borland Software Corporation. CaliberRM, 2011. URL `http://www.borland.com/us/products/caliber/`.

Alan M. Davis. *Software Requirements: Objects, Functions, and States*. Prentice Hall, 2nd edition, 1993. ISBN 013805763X.

Rodger D. Drabick. On-track requirements. *Software Testing & Quality Engineering*, 1(3):54–60, 1999.

E. Hollnagel and D.D. Woods. Cognitive systems engineering: New wine in new bottles. *International Journal of Man-Machine Studies*, 18(6): 583–600, 1983.

IBM. Rational DOORS, 2011a. URL `http://www-01.ibm.com/software/awdtools/doors/`.

IBM. Rational RequisitePro, 2011b. URL `http://www-01.ibm.com/software/awdtools/reqpro/`.

A.P.J. Jarczyk, P. Loffler, and FM Shipmann III. Design rationale for software engineering: A survey. In *System Sciences, 1992. Proceedings of the Twenty-Fifth Hawaii International Conference on*, volume 2, pages 577–586. IEEE, 1992.

D. Kulak and E. Guiney. *Use cases: requirements in context*. Addison-Wesley Professional, 2004.

A. Lapouchnian. Goal-oriented requirements engineering: An overview of the current research. *University of Toronto*, 2005.

J. Lee. Design rationale systems: understanding the issues. *IEEE Expert*, 12(3):78–85, 1997.

T. Mioch, M.A. Neerincx, and N. Smets. sce methodology. EU FP7 NIFTi / ICT-247870, March 2010.

M.A. Neerincx and J. Lindenberg. Situated cognitive engineering for complex task environments. In J.M.C. Schraagen, editor, *Naturalistic Decision Making and Macrocognition*, pages 373–390. Ashgate Publishing Limited, Aldershot, 2008.

B. Nuseibeh and S. Easterbrook. Requirements engineering: a roadmap. In *Proceedings of the Conference on the Future of Software Engineering*, pages 35–46. ACM, 2000.

R. Prieto-Díaz. Domain analysis: an introduction. *ACM SIGSOFT Software Engineering Notes*, 15(2):47–54, 1990.

Ian Sommerville and Pete Sawyer. *Requirements Engineering: A Good Practice Guide*. John Wiley & Sons, Inc., New York, NY, USA, 1st edition, 1997. ISBN 0471974447.

Sparx Systems. Enterprise architect, 2011. URL `www.sparxsystems.eu/Sparx`.

N. Takeda, A. Shiomi, K. Kawai, and H. Ohiwa. Requirement analysis by the kj editor. In *Requirements Engineering, 1993., Proceedings of IEEE International Symposium on*, pages 98–101. IEEE, 1993.

T. Tsumaki and T. Tamai. A framework for matching requirements engineering techniques to project characteristics and situation changes. *Proceedings of Situational Requirements Engineering Processes (SREP), Paris, France*, 2005.

P.P. van Maanen, J. Lindenberg, and M.A. Neerincx. Integrating human factors and artificial intelligence in the development of human-machine cooperation. In *Proc. of the 2005 International Conference on Artificial Intelligence (ICAI05)*. Citeseer, 2005.

M. Westera, J. Boschloo, J. van Diggelen, L.S. Koelewijn, M.A. Neerincx, and N.J.J.M. Smets. Employing use-cases for piecewise evaluation of requirements and claims. In *Proceedings of the 28th Annual European Conference on Cognitive Ergonomics*, pages 279–286. ACM, 2010.

K.E. Wiegers. Automating requirements management. *Software Development*, 7(7):1–5, 1999.

K.E. Wiegers. *Software requirements*. Microsoft Press, 2003.

K.E. Wiegers. *More about software requirements: Thorny issues and practical advice*. Microsoft Press Redmond, WA, USA, 2005.

D.R. Windle and L.R. Abreo. *Software requirements using the unified process: a practical approach.* Prentice Hall PTR, 2003.