

# THE SITUATED COGNITIVE ENGINEERING TOOL

Wytse Jan Posthumus

FACULTY OF ELECTRICAL ENGINEERING, MATHEMATICS AND COMPUTER  
SCIENCE  
SOFTWARE ENGINEERING GROUP

**EXAMINATION COMMITTEE**

Dr. Luís Ferreira Pires  
Dr. Ing. Christoph Bockisch  
Prof. Dr. Mark Neerincx  
Dr. Jurriaan van Diggelen

**DOCUMENT NUMBER**  
EWI/SE - 2011-005



# The situated Cognitive Engineering Tool

*Thesis of a master project in context of the study Computer Science  
at the University of Twente. This internship is performed at the  
TNO institute at Soesterberg.*

*Author:*

Wytse Jan POSTHUMUS

*Study:*

Computer Science

*Institute:*

University of Twente  
TNO Soesterberg

*University Supervisors:*

Dr. Luís Ferreira PIRES  
Dr. Ing. Christoph BOCKISCH

*TNO Supervisors:*

Prof. Dr. Mark NEERINCX  
Dr. Jurriaan van DIGGELEN

*Date:*

October 16, 2011



# Abstract

With the increasing complexity of systems, more effort is required of users to control them. Engineering methods, however, do not always consider the limitations of the psychological capabilities of users. To address these limitations, an approach to create systems was developed in the 1980s, namely Cognitive Engineering (CE). 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 has developed the situated Cognitive Engineering methodology as an extension to Cognitive Engineering (Neerincx and Lindenberg, 2008).

Currently, the sCE methodology is applied using textual documents. However, this approach has three main limitations, namely on the areas of *collaboration*, *soundness and completeness* and *reusability*.

To address the above mentioned limitations, a software tool could be used instead. Several tools exist, but they do not meet the requirements for usage with the sCE methodology. This thesis proposes a design for a Cognitive Engineering tool to create requirements following the situated Cognitive Engineering Methodology.

Knowledge of requirements engineering and the sCE methodology was applied to develop the situated Cognitive Engineering Tool (sCET) to support the developers of joint cognitive systems. sCET was developed according to the four phases of the sCE methodology, namely *derive*, *specify*, *test* and *refine*.

A first version of sCET has been used to further specify its requirements baseline and to test its claims, as the methodology prescribes. This has been done by performing an evaluation on the usage of sCET. The results of the sCET evaluation have been used to refine the methodology and the tool. From these results, improvements for the sCE methodology and the tool have been derived.

The results of the evaluation show that sCET addresses the limitations on the areas of *collaboration*, *soundness and completeness* and *reusability*. However, users may dislike learning a new tool if the design and usability of the user-interface are not attractive.

Currently, the prototype version of sCET tool is being used in several projects. In the near future, this version will be used to get more experience in the use of a sCE tool. This will give the opportunity to create a new version of sCET which incorporates all the new ideas which have been gathered using the current version of sCET.

# Preface

This thesis is written for the master project of Computer Science at the University of Twente in the Netherlands. The project is carried out at the Perceptual and Cognitive Systems department of TNO, located at Soesterberg.

The goal of this master project is to design a tool that supports multidisciplinary project groups in designing complex cognitive systems. This thesis is written under supervision of Dr. Luís Ferreira Pires, Associate Professor of the Software Engineering group at the University of Twente.

First, I would like to thank Luís and Christoph Bockisch for supervising on behalf of the university and Mark Neerincx and Jurriaan van Diggelen for supervising on behalf of TNO. Their input and great ideas made this project a great success, thanks!

Second, I would like to thank my fellow roommates at TNO, especially Youssef for encouraging me at moments when I got stuck.

Third, I would like to thank my family and friends for supporting me and to be there when I needed some distraction from my work.

Finally, thanks to my girlfriend Ellen for supporting me throughout the process of graduating. Throughout the process of my graduation research, she has been loving and supportive.

October 16, 2011  
Wytse Jan Posthumus

# Table of Contents

<b>Abstract</b>	<b>i</b>
<b>Preface</b>	<b>ii</b>
<b>Table of Contents</b>	<b>iii</b>
<b>List of Figures</b>	<b>vi</b>
<b>List of Tables</b>	<b>vii</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Motivation . . . . .	1
1.2 Objectives . . . . .	3
1.3 Approach . . . . .	4
1.4 Outline . . . . .	4
<b>2 Background</b>	<b>5</b>
2.1 Requirements . . . . .	5
2.1.1 Definition . . . . .	5
2.1.2 Requirements Characteristics . . . . .	6
2.1.3 Types of Requirements . . . . .	8
2.1.4 Use Cases . . . . .	9
2.2 Requirements Engineering . . . . .	11
2.2.1 Definition . . . . .	11
2.2.2 Requirements Engineering Phases . . . . .	11
2.2.3 Requirements Engineering Methods . . . . .	12
2.2.4 Requirements Engineering Tools . . . . .	13
2.2.5 Requirements Design Rationale . . . . .	14
<b>3 The situated Cognitive Engineering Methodology</b>	<b>16</b>
3.1 Cognitive Systems Engineering . . . . .	16
3.2 Situated Cognitive Engineering . . . . .	17
3.2.1 Basic principles . . . . .	17
3.2.2 sCE Phases . . . . .	18

3.2.3	sCE Use Cases . . . . .	19
3.2.4	sCE Requirements and Claims . . . . .	20
3.3	sCE comparison . . . . .	21
<b>4</b>	<b>The situated Cognitive Engineering Tool (sCET)</b>	<b>23</b>
4.1	sCET Motivation . . . . .	23
4.2	sCET Design . . . . .	24
4.2.1	Derive . . . . .	24
4.2.2	Specify . . . . .	25
4.3	sCET Prototype . . . . .	30
4.3.1	sCET Data Model . . . . .	30
4.3.2	Architecture . . . . .	30
4.3.3	Interface . . . . .	31
<b>5</b>	<b>sCET evaluation</b>	<b>34</b>
5.1	Experiment Set-up . . . . .	34
5.2	Experiment Results . . . . .	36
5.2.1	Informal Evaluation . . . . .	36
5.2.2	Formal Evaluation . . . . .	37
<b>6</b>	<b>Discussion</b>	<b>41</b>
6.1	Tools Comparison . . . . .	41
6.2	sCET Improvements . . . . .	45
6.2.1	Guidance . . . . .	45
6.2.2	Reporting . . . . .	46
6.2.3	Security . . . . .	46
6.2.4	Visual Design and Usability . . . . .	47
6.2.5	Validation . . . . .	47
6.2.6	Reusability . . . . .	47
6.3	sCET Refinement . . . . .	48
6.3.1	Refined Requirements . . . . .	48
6.3.2	Added Requirements . . . . .	50
<b>7</b>	<b>Conclusion</b>	<b>52</b>
7.1	General conclusions . . . . .	52
7.2	Recommendations . . . . .	53
7.3	Future Research . . . . .	54
	<b>Bibliography</b>	<b>56</b>
<b>A</b>	<b>sCET Use Cases</b>	<b>59</b>
<b>B</b>	<b>sCET Requirements</b>	<b>63</b>
<b>C</b>	<b>sCET Workshop Questionnaire</b>	<b>69</b>



<b>D sCET Questionnaire Results</b>	<b>73</b>
<b>E sCET BBCode Reference</b>	<b>74</b>

# List of Figures

2.1	Connections among several types of requirements information (Wiegiers, 2003) . . . . .	9
2.2	Use case diagram example (Windle and Abreo, 2003) . . . . .	10
2.3	Phases of requirements engineering (Wiegiers, 2005) . . . . .	12
2.4	Using the design rationale to create alternative requirements . . . . .	15
3.1	The relations between artefacts of the sCE methodology . . . . .	18
3.2	The situated Cognitive Engineering process (UX = User Experience, HitL = Human in the Loop, Sim = Simulation) (Mioch et al., 2010) . . . . .	19
4.1	sCET data model . . . . .	31
4.2	sCET simplified page structure . . . . .	32
4.3	sCET requirement screen . . . . .	33
4.4	sCET requirement edit screen . . . . .	33
5.1	Mean scores of the questions before (1, 2, 3) and after (39, 29, 37/41) the workshop. Note: For the sCE learning questions only participants with no experience have been considered. . . . .	40
E.1	sCET BBCode tags (1/2) . . . . .	74
E.2	sCET BBCode tags (2/2) . . . . .	75
E.3	sCET BBCode short tags . . . . .	75

# List of Tables

2.1	Characteristics of desirable requirements (Davis, 1993) . . . . .	7
3.1	Use case format . . . . .	20
3.2	Requirement format . . . . .	21
4.1	UC.01 . . . . .	26
4.2	RQ.1 . . . . .	26
4.3	RQ.2 . . . . .	27
4.4	RQ.3 . . . . .	27
4.5	RQ.4 . . . . .	28
4.6	RQ.5 . . . . .	28
4.7	RQ.6 . . . . .	29
4.8	RQ.7 . . . . .	29
4.9	RQ.8 . . . . .	29
5.1	Claim 1.2 questionnaire results . . . . .	37
5.2	Claim 2.1 questionnaire results . . . . .	38
5.3	Claim 3.1 questionnaire results . . . . .	38
5.4	Claim 4.1 questionnaire results . . . . .	38
5.5	Claim 4.2 questionnaire results . . . . .	38
5.6	Claim 5.1 questionnaire results . . . . .	39
5.7	Claim 6.1 questionnaire results . . . . .	39
5.8	Claim 8.1 questionnaire results . . . . .	39
6.1	Tools Comparison. An empty cell indicates that no functionality of that subject is present. A ‘-’ indicates that limited functionality, ‘+’ indicates full functionality and ‘.’ indicates that the functionality is not imple- mented in the prototype yet. . . . .	44
6.2	RQ.3 . . . . .	48
6.3	RQ.5 . . . . .	49
6.4	RQ.6 . . . . .	49
6.5	RQ.9 . . . . .	50
6.6	RQ.10 . . . . .	50
6.7	RQ.11 . . . . .	50

6.8	RQ_12	51
6.9	RQ_13	51
A.1	UC_01	59
A.2	UC_02	59
A.3	UC_03	60
A.4	UC_04	60
A.5	UC_05	60
A.6	UC_06	61
A.7	UC_07	61
A.8	UC_08	61
A.9	UC_09	62
B.1	RQ_1	63
B.2	RQ_2	64
B.3	RQ_3	64
B.4	RQ_9	64
B.5	RQ_4	65
B.6	RQ_5	65
B.7	RQ_6	66
B.8	RQ_7	66
B.9	RQ_8	66
B.10	RQ_10	67
B.11	RQ_11	67
B.12	RQ_12	67
B.13	RQ_13	68
D.1	sCET questionnaire results	73

# Chapter 1

## Introduction

The aim of this chapter is to introduce this work by giving a motivation on why this research was performed followed, by the main objective and research questions.

This chapter is organised as follows: Section 1.1 describes the motivation behind this work, Section 1.2 describes the main objective and the research questions, Section 1.3 describes the approach of this thesis and finally Section 1.4 presents the outline of this thesis.

### 1.1 Motivation

With the increasing complexity of systems, more effort is required of users to control them. Engineering methods, however, do not always consider the limitations of the psychological capabilities of users. To address these limitations, an approach to create systems was developed in the 1980s, namely Cognitive Engineering (CE). 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 has 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 (1) use cases, which are stories about users undertaking activities, and (2) 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.

Currently, the sCE methodology is applied using textual documents. However, this approach has three main limitations, namely on the areas of *collaboration*, *soundness and completeness* and *reusability*. Each limitation is discussed below.

*Collaboration.* Projects which follow CE approaches are often performed by cooperating interdisciplinary groups. For example, in a project where people with a background in psychology may have to work with people from software engineering. Not all disciplines have the same way of thinking and they even may work on different physical locations. To enable cooperative working, all of these groups must have a common level of thinking. At the bare minimum level, project members should follow the same vocabulary, otherwise these differences between word definitions may result in less accurate communication.

Another problem which hinders cooperative working is version management. It is often hard to ‘keep current’ with changes of documents. Textual documents have to be exchanged between project members, and because it is not possible to see if users work on the latest document, they need to be merged which can cause errors.

*Soundness and Completeness.* Each requirement in a requirements baseline should be sound and complete. This means that a requirements is valid (sound) and that no information is missing (complete). A sound and complete requirement can be achieved when all conditions of the used methodology are met.

A sound and complete requirements baseline makes sure the design specification can be verified. In the current situation it can become difficult to achieve a sound and complete design specification, because textual documents do not force users to fill in data in any specific way.

In order to create a sound and complete requirements baseline, understanding the methodology used is necessary. Often, some project members may not be familiar with the used methodology. This can create problems since they probably do not know how they should apply this methodology.

*Reusability.* Textual documents give a huge freedom on how the data is represented. Each project could have its own format for the requirement documents. This creates inconsistencies when you want to compare and reuse the data in new projects.

Textual documents also have a problem that it is more difficult to keep track of old documents. Documents from previous projects can become scattered when there is no standard way for archiving project data.

To address the above mentioned limitations of text documents in the application of the sCE methodology, we decided to develop a software tool. Several Cognitive Engineering tools exists, but they do not focus on requirements engineering. Several requirements engineering tools exist as well, but these tools focus only on requirements development, and do not consider the human factor knowledge of the Cognitive Engineering methodologies.

Because the current existing requirements engineering tools do not meet the requirements of CE projects, a new tool had to be designed. In this tool the knowledge of requirements engineering and Cognitive Engineering could be combined to create a more complete Cognitive Engineering requirements tool in which human factor knowledge takes a central place. This thesis proposes the design for a Cognitive Engineering tool to engineer requirements following the situated Cognitive Engineering Methodology.

## 1.2 Objectives

The three limitations of using textual documents, as mentioned in Section 1.1, should be addressed in our work. Since we can achieve this by means of a tool, therefore the main objective of this thesis is:

*Improve the support of Cognitive Engineering to enable cooperative working in multidisciplinary research projects, to stimulate soundness and completeness of the design specification and to support the reuse of earlier work.*

To achieve the main objective, we will develop a tool which supports Cognitive Engineering projects. During the development, the following four research questions will be considered:

1. *How can a CE tool enable cooperative working between project members with different background?*

The key to a successful project is that project members cooperate with each other. However, in a multidisciplinary project this may be difficult because each person may have different definitions for certain words. In this work the aspects which can enable this cooperation have been studied.

2. *How can a CE tool stimulate users to create a sound and complete design specification?*

When designing a system it is important that all data is entered correctly and that no data is missing. In this work the aspects which can stimulate entering sound and complete data have been studied.

3. *How can a CE tool support the reuse of earlier work?*

Reuse of data for old projects can save time when performing new projects. In this work the aspects which can support the reuse of data have been studied.

4. *What are potential disadvantages of using a tool for CE projects?*

The effect of using a tool have been studied. From these results the tool has been evaluated.

## 1.3 Approach

To answer the research questions three main steps have been taken: a background study was performed, a tool to support the sCE methodology has been designed and a prototype of this tool has been built.

A background study has been performed to get insight in the history and current state of requirements engineering. Next, Cognitive Engineering and the situated Cognitive Engineering methodology, which is developed by TNO, have been studied. Finally, the key focus points of the sCE methodology was compared to requirements engineering.

The knowledge gathered in the background study have been applied to develop a tool to support the developers of joint cognitive systems which follows the sCE methodology. This situated Cognitive Engineering Tool (sCET) has been designed by applying the situated Cognitive Engineering methodology to the tool. This was done according to the four phases of the sCE methodology, namely *derive*, *specify*, *test* and *refine*.

A first version of sCET was used to further specify its requirements baseline and to test its claims, as the methodology prescribes. This was done by performing an evaluation on the usage of sCET.

The results of the sCET evaluation have been used to refine the methodology and the tool. From these results, improvements for the sCE methodology and the tool have been identified.

## 1.4 Outline

The remainder of this thesis is organised as follows. Chapter 2 describes the background of this work by discussing requirements engineering. Chapter 3 describes Cognitive Engineering and the situated Cognitive Engineering Methodology. Chapter 4 describes how the tool has been designed. It reports on the first two phases of the sCE design process, namely the *derive* and *specify* phases. Chapter 5 describes how sCET has been evaluated. The chapter reports on the test phase of the sCE methodology. Chapter 6 discusses the sCE methodology and the tool give recommendations for the tool. The chapter reports on the final phase of the sCE methodology, namely the *refine* phase. Finally, Chapter 7 gives the conclusions 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 Definition

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 it 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 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 in 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-cycle 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 (Wieggers, 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.

<b>Characteristic</b>	<b>Explanation</b>
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”.
<b>Traceable</b>	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.
<b>Verifiable</b>	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. Classifying requirements also 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 directly 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, while non-functional requirements can be influenced by external variables as well (e.g. hardware).

#### Method 2 - Wiegers

The second method for the 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 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).

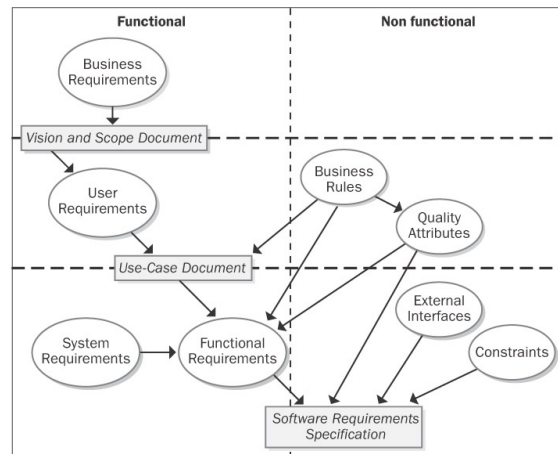


Figure 2.1: Connections among several types of requirements information (Wiegers, 2003)

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 the requirements information 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 are described 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 correspond to 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 built.

#### 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) use case diagrams which allows designers to graphically represent actors and their use cases.

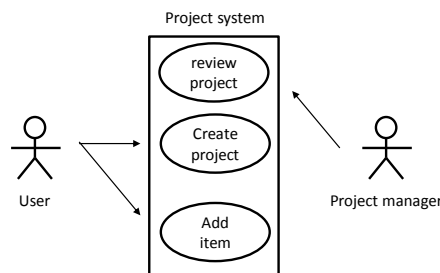


Figure 2.2: Use case diagram example (Windle and Abreo, 2003)

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 capturing requirements.

### 2.2.1 Definition

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 the purpose of the software system being considered is discovered. This purpose is the description of what the 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* (Wieggers, 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 are 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.

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.

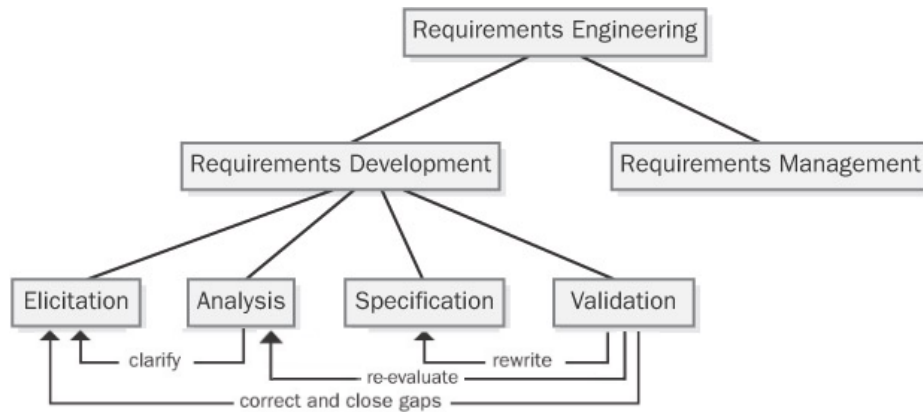


Figure 2.3: Phases of requirements engineering (Wiegiers, 2005)

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 (Wiegiers, 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 (Wiegiers, 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 consists of decomposing the target domain in which the system will be used, 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

The goal-oriented approach focuses 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

The scenario-based approach focuses 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. Wiegiers states the following four important difficulties (Wiegiers, 1999):

- Difficult to keep a textual document current, especially when requirements change rapidly.
- Difficult to communicate changes to the affected team members.
- Difficult to store supplementary information about each requirement.
- Difficult 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. Wiegiers states seven reasons why one should use a tool to manage requirements (Wiegiers, 1999):

1. **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 to older versions.
2. **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.

3. **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.
4. **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.
5. **View requirement subsets.** Sorting, filtering and searching through the requirements baseline increases the insight in the requirements specification.
6. **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.
7. **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 Caliber-RM (Borland Software Corporation, 2011), IBM's DOORS (IBM, 2011a), IBM's Rational RequisitePro (IBM, 2011b) and Sparx Systems Enterprise Architect (Sparx Systems, 2011).

### 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 should be, in its simplest form: explicitly listing the decisions made during the 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, allowing alternative requirements to be defined.

Figure 2.4 illustrates the possible process of creating alternative requirements using design rationale. Multiple requirements could be related to each other, making the

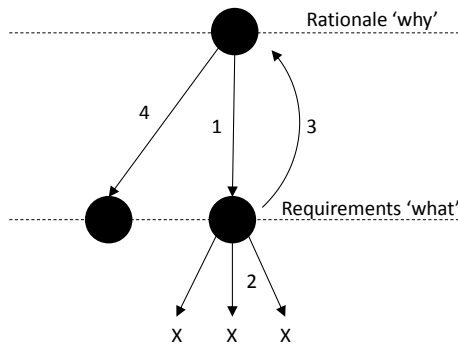


Figure 2.4: Using the design rationale to create alternative requirements

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 its 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 raise the question whether they should be implemented at all. This reduces the risk of implementing improper 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 normally goes 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 perform a process, so that the user moves 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, 2008), 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).

The area of Cognitive Engineering is a very large area of expertise, and incorporates many disciplines, like psychology, neuroscience, communication and many others. For this work, we mainly focus on the systems engineering area.

## 3.2 Situated Cognitive Engineering

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

### 3.2.1 Basic principles

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.

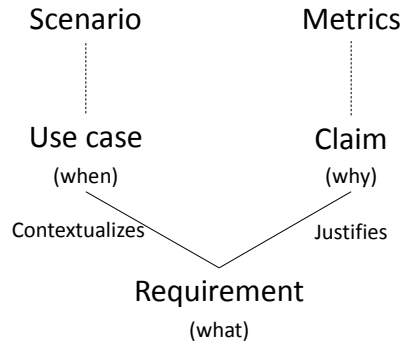


Figure 3.1: The relations between artefacts of the sCE methodology

### 3.2.2 sCE 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-loop evaluations. The

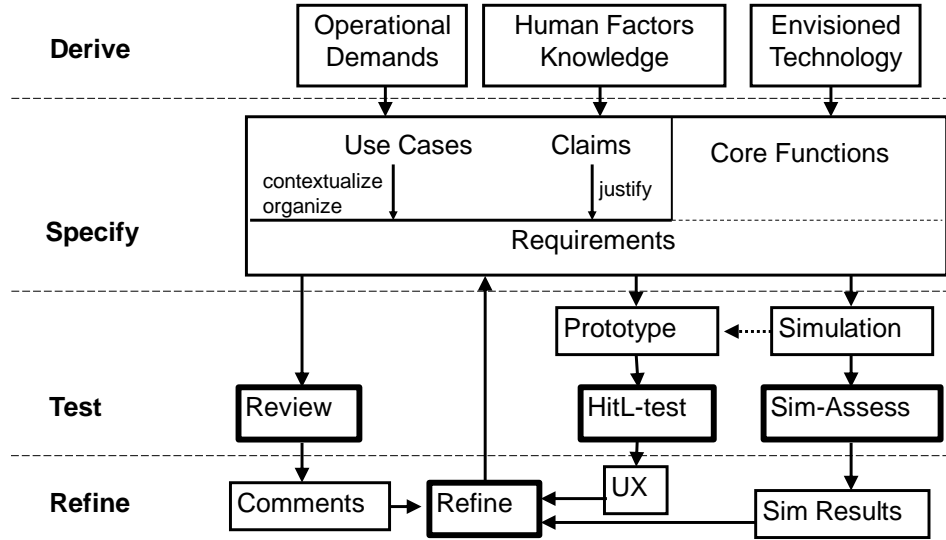


Figure 3.2: The situated Cognitive Engineering process (UX = User Experience, HitL = Human in the Loop, Sim = Simulation) (Mioch et al., 2010)

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.

Use case #	<<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 that triggers the use case (i.e., time, alarm).
Requirements:	List of requirements that relate to this use case, referred to by their identifier.
Main Action	1 Step 1.
Sequence:	2 Step 2.
Alternative	1 Step 1.
Action Sequence:	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.



Requirement #	<<identifier>>
Description:	Description
Claim	<i>Hypothesis (e.g., fast victim finding in areas that are inaccessible for humans)</i> + 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 cases that link to this requirement, referred to by their identifier.

Table 3.2: Requirement format

### 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 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 built.

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 requirements capturing using scenarios and use cases. This enables the requirements capturing process as a dynamic activity. Requirements are elicited from the domain of interest and can be identified 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 each requirement make its impact on 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. By validating the claims one can make sure each requirement is useful. This makes the sCE methodology extremely suitable 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

# The situated Cognitive Engineering Tool (sCET)

We designed a tool called the situated Cognitive Engineering Tool (sCET) to support the process of creating project specifications using Cognitive Engineering. In this chapter we describe the design and prototype implementation of this tool.

Section 4.1 introduces sCET, Section 4.2 describes how the tool is designed, and Section 4.3 describes how the prototype is implemented.

### 4.1 sCET Motivation

Currently, in projects which apply the CE methodology, textual documents are used to capture the design specification. However, this has some limitations as stated by Wiegers (see Section 2.2.4). We designed a tool to provide a solution to these limitations.

Wiegers (1999) states four limitations of using textual documents. From these limitations, three main core functions have been derived, namely *Collaboration*, *Soundness and Completeness* and *Reusability*, which are described in Section 1.1. These core functions currently cause the most problems when performing a CE project.

To address these limitations a tool needs to be used. Several Cognitive Engineering tools exist, but they do not focus on requirements development. Several requirements engineering tools exist as well, but these tools focus only on requirement development, and do not consider the human factor knowledge of the Cognitive Engineering methodologies.

Because the current existing requirements engineering tools do not meet the requirements for use with CE projects, a new tool had to be designed. In this tool the knowledge of requirements engineering and Cognitive Engineering can be combined, resulting in the Cognitive Engineering requirements tool.

The situated Cognitive Engineering methodology is chosen as method which the tool has to support. The tool has been designed by applying the sCE methodology to itself. Hence the design of the tool can be applied to itself.

## 4.2 sCET Design

To design the tool, we followed the sCE methodology. Since the sCE methodology consists of four phases, the same phases were followed, namely the derive, specify, test and refine phases. In this section the derive and specify phases are described. The test and refine phases are reported in Chapters 5 and 6, respectively.

### 4.2.1 Derive

In the derive phase, the operational demands, human factor knowledge and envisioned technology is tailored in a scenario. As an example project for designing sCET, the Mutual Empowerment project was chosen because it reflects a project in which people with different disciplines have to cooperate.

#### Scenario

TNO is a non-profit research organization which aims at developing and applying knowledge. Currently, TNO has about 3500 employees (TNO, 2010).

TNO is organized by a matrix structure in which projects are performed within *themes*. There are in total seven themes, namely:

- Healthy Living
- Industrial Innovation
- Defence, Safety and Security
- Energy
- Transport and Mobility
- Built Environment
- Information Society

Besides these themes, TNO has three *expertise centres*. These centres house the scientists who perform projects. Each expertise centre consists of several expertise groups with a total of 67 groups. The expertise centres are:

- Technical Sciences (36 expertise groups)
- Earth, Environmental and Life Sciences (13 expertise groups)
- Behavioural and Societal Sciences (18 expertise groups)

One of the projects within TNO is the Mutual Empowerment (ME) project. ME is a defence, safety and security project which aims at designing a mobile platform for soldiers in war situations. The goal is to create a system in which a soldier and some machine can work together to optimise the soldier's performance in social patrols. Hence, mutual empowerment means that human and machine make each other more powerful.

ME is a TNO-wide project, which means that multiple expertise groups are participating in this project. Each of these expertise groups specialized in a certain area of knowledge. Eleven groups participate on this project such as, for example, *Business Information Systems*, *Human Performance*, *Perceptual and Cognitive Systems* and *Weapon Systems*. These groups come from different expertise centres, differ significantly from each other and are often located in different cities.

All these groups are involved in the project because many aspects need to be taken in consideration when designing a complex cognitive system where human and machine have to work together. The system should not only be designed, but also the needs of the soldiers have to be taken into account. In the end the soldiers have to use the system and if the system does not conform to their needs they will most likely not use the system.

Because the members of the different groups have different disciplinary backgrounds, it is often difficult for them to cooperate. Different disciplines require a different way of thinking, and combining those is not an easy task. Furthermore, because TNO is located at multiple locations it is not always possible to meet in a single physical place when questions arise.

A way to improve communication so that project members know what each member is doing has become necessary. This should facilitate the alignment of the work of the groups so that working designs can be delivered in cooperation.

#### 4.2.2 Specify

In the specify phase, the scenario is formalised into use cases, and requirements are derived from these use cases.

##### Use cases

Each scenario is formalised in several use cases. These use cases are written in the format described in Section 3.2.3. Table 4.1 describes one use case of sCET. It gives an example use case from which Requirement 1 has been derived. All remaining use cases for sCET are shown in Appendix A.

Use case 1	UC_01
Description:	Multiple users work on one requirement
Goals:	Get better requirements
Actors:	multiple users
Pre-Conditions:	A sCET project is created by a user
Post-Conditions:	The requirement is updated by another user.
Requirements:	RQ_1
Main action sequence:	1 User 1 creates a requirement 2 User 2 looks at the requirement 3 User 2 makes changes to the requirement

Table 4.1: UC\_01

## Requirements

Below, the requirements are given in the format described in Section 3.2.4. They are categorised by the three main core functions, namely *collaboration*, *soundness and completeness* and *reusability*.

**Collaboration.** Collaboration is the key focus point of sCET. The goal is to improve the cooperation between people who work on a project and the stakeholders.

Requirement 1	RQ_1
Description:	sCET shall improve collaboration when users work on dependent parts.
Claim 1.1	<i>Because sCET supports designing requirements in an iterative way, better requirements will be created.</i> + More input per requirement is generated, resulting in a more thought-out requirement. - More discussions can arise, which takes more time to develop the system.
Claim 1.2	<i>Because users can see the work of others, they can make their work align to each other.</i> + Involvement of distributed users in the design process is increased. - It can be difficult for the user to determine relevant data.
Use Cases:	UC_01, UC_04

Table 4.2: RQ\_1

Requirement 2	RQ_2
Description:	sCE(T) shall support parallel collaboration on different parts.
Claim 2.1	<i>Because sCE(T) is modular, users can work on different parts at the same time.</i> <ul style="list-style-type: none"> <li>+ Users do not have to wait for each other to finish working on their part.</li> <li>- A network connection is required to access the tool.</li> </ul>
Use Cases:	UC_02

Table 4.3: RQ\_2

Requirement 3	RQ_3
Description:	sCET shall improve communication with outsiders.
Claim 3.1	<i>Because sCET can export data, outsiders can easily understand the design specification.</i> <ul style="list-style-type: none"> <li>+ Less time is needed to explain the design specification.</li> <li>- If they still do not understand it, more effort is required for explaining.</li> </ul>
Claim 3.2	<i>Because users can create ontologies, the design specification will become more understandable for outsiders.</i> <ul style="list-style-type: none"> <li>+ If the design specification is more easy to understand it will prevent miscommunication.</li> <li>- Creating an ontology takes time.</li> </ul>
Use Cases:	UC_03, UC_04

Table 4.4: RQ\_3

**Soundness and completeness.** The tool should make sure that the design specification is sound and complete. This means that all the relevant data should be entered in a correct way. A sound and complete requirements baseline makes the design robust and understandable.

Requirement 4	RQ_4
Description:	sCET shall enforce users to create an complete design specification.
Claim 4.1	<i>Because sCE(T) has defined its data fields, it discourages entering irrelevant data.</i> + Less irrelevant data results in better understandable design specifications. - Sometimes additional information might be hard to enter.
Claim 4.2	<i>Because sCE(T) has defined its data fields, it is possible to see what data is missing.</i> + Missing data can be identified and added easily. - If some fields are irrelevant for a specific case, they cannot be removed.
Use Cases:	UC_05

Table 4.5: RQ\_4

Requirement 5	RQ_5
Description:	sCE(T) shall enforce the specification of a verifiable design specification.
Claim 5.1	<i>Because claims are defined by metrics, claims can be tested on their validity.</i> + If the claims are refuted, they can be refined to be replaced by better claims. - Defining adequate metrics takes time.
Use Cases:	UC_06

Table 4.6: RQ\_5



Requirement 6	RQ_6
Description:	sCET shall facilitate the learning of the sCE methodology.
Claim 6.1	<i>Because sCET is easy to learn the learning curve of the sCE methodology is decreased.</i> <ul style="list-style-type: none"> <li>+ New user can deliver results more quickly.</li> <li>- Users could proceed in a disorganized way and not really learn the methodology.</li> <li>- Users could dislike learning a new tool.</li> </ul>
Use Cases:	UC_07

Table 4.7: RQ\_6

**Reusability.** The tool should make data reusable. Making the data reusable avoids situations of ‘inventing the wheel again’. It makes sure that important data is not lost and avoids making the same mistakes twice.

Requirement 7	RQ_7
Description:	sCET shall make the iterative process insightful.
Claim 7.1	<i>Because users can see the history of requirements better evaluation choices will be made.</i> <ul style="list-style-type: none"> <li>+ Users do not repeat previously made mistakes.</li> <li>- Users can lose the overview because of too much information.</li> <li>- Navigation across the design specification can become more difficult when more information is present.</li> </ul>
Use Cases:	UC_08

Table 4.8: RQ\_7

Requirement 8	RQ_8
Description:	sCET shall make design specifications reusable.
Claim 8.1	<i>Because requirements from previous projects can be imported, new projects can start more quickly.</i> <ul style="list-style-type: none"> <li>+ Importing old requirements results in less work for users.</li> <li>- Errors which were made previously can persist in related projects.</li> </ul>
Use Cases:	UC_09

Table 4.9: RQ\_8

## 4.3 sCET Prototype

We built a prototype of the tool to test its suitability. This tool was developed as a web-application with web-forms as graphical user interface. This prototype was deployed on a web server which was accessible through the Internet.

The tool has been built using PHP5 to create dynamic HTML pages with JavaScript which interact with a MySQL database. The interaction with the database was implemented using an Active Record library.

### 4.3.1 sCET Data Model

The data structure of sCET is displayed in Figure 4.1. This data model was derived from the schematic overview of the relations between the sCE artefacts as shown in Figure 3.1.

*Project* is the main object. sCET supports multiple projects simultaneously in the tool. The Project object makes sure that the data of each project is grouped together.

A project can have several *scenarios*. Each scenario tells a story about the product being designed. A scenario can be formalised by specifying a *use case*. A use case describes the specific actions that can be taken in terms of *action sequences*. Action sequences give a step by step description of how the functionality is used. Each use case can have several action sequences.

From a use case, one or more *requirements* can be derived. A requirement describes specific information about what the system should be able to achieve. *Claims* give positive and negative effects of each requirement. Claims should justify the existence of each requirement.

As the specification process progresses, several requirements may be refined. Each requirement can be refined into multiple new requirements. The old requirement then becomes deprecated. Requirements can be grouped into *functional modules*. A functional module is a group of requirements with some common characteristics, for example, a module with requirements for a user interface. They are the core functions of the product being designed.

### 4.3.2 Architecture

The tool is designed as an interface to the database scheme as shown in Figure 4.1. This data model was mapped onto several PHP pages, which interface directly with the database. A simple schematic overview of the pages is shown in Figure 4.2.

The figure shows that *project.php* is the centre part of the application. From that page it is possible to navigate to the artefacts of the design rationale.

The scenario, use case, module and requirement pages contain a list of the artefacts which are specified in the system and are connected through links. The use case and requirement pages also contain sequences and claims, respectively.

An example of a page is shown in Figure 4.3. The figure shows the *requirement.php* page with one selected requirement. The data from the requirement corresponds to the

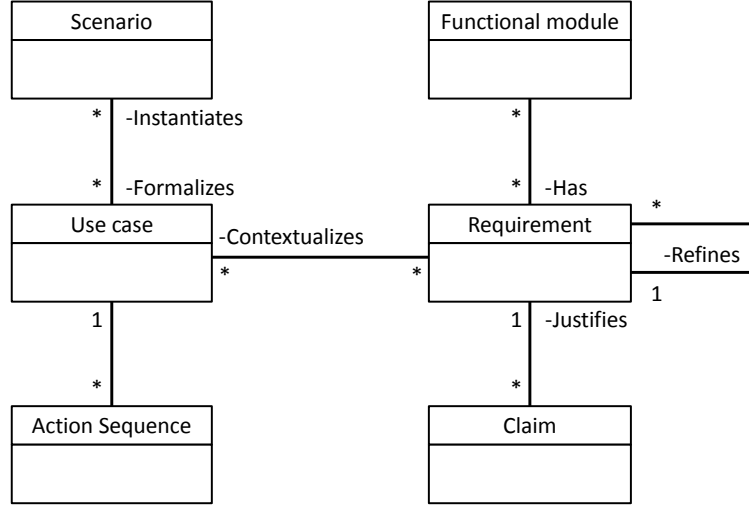


Figure 4.1: sCET data model

data as shown in Table 3.2.

### 4.3.3 Interface

The interface was created using *HTML*, to define the building blocks of the pages, *CSS*, to add visual style information and *JavaScript*, to make the interface dynamic. The interface elements are generated by the PHP pages.

A screenshot of the current version of sCET is shown in Figure 4.3. This figure shows navigation buttons below the sCET logo. From each page the user can navigate to the *project overview*, *scenarios*, *use cases*, *modules*, *requirements*, *ontology* and *tools* pages.

The project overview screen shows general information about the particular selected project. It also lists news messages which can be added by the project manager. The *Scenarios*, *Use cases*, *Modules* and *Requirements* are the artefacts from the sCE methodology. The *Ontology* screen shows the page where ontologies can be defined. In the current version, only a taxonomy is supported, which means that terms can be defined in a hierarchical way. Creating instances of terms and relations between terms, which would make it an ontology, is currently not supported. The *Tools* screen shows options for exporting sCET data to *L<sup>A</sup>T<sub>E</sub>X*, XML or Microsoft Word's DOC format.

In Figure 4.3 the requirement screen is selected. At the left side a list of requirements is shown. If the user clicks a requirement, the information of that requirement is shown in the central part of the screen. This information consists of the requirement attributes according to the format of the sCE methodology. At the top of the central part of the page there are tabs to navigate to the *Document*, *Discussion*, *Edit*, *New* and *Delete* screens. By default the *Document* screen is selected, as shown in the figure. The *Discussion* tab shows user replies to the document. Each user can add comments to

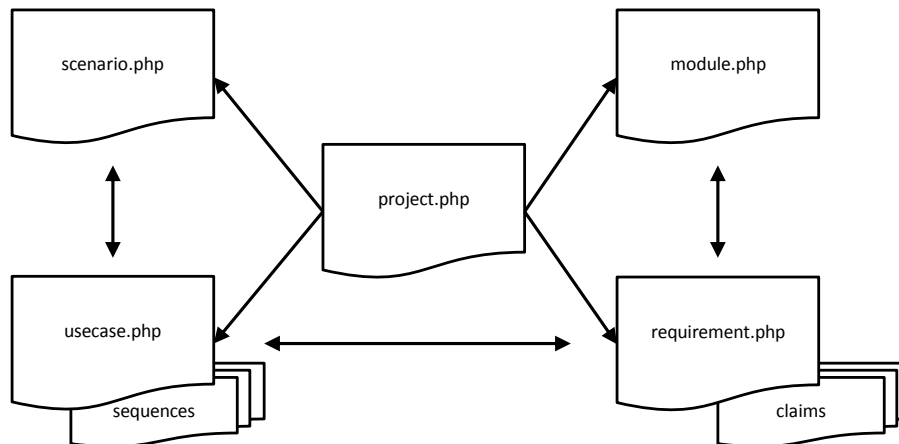


Figure 4.2: sCET simplified page structure

a document by adding a reply. The *Edit* tab shows the page to change the information of the document, and is described below. The *New* tab can be selected to create a new document, and the *Delete* tab can be selected to delete the current document.

The *Edit* tab of a document shows the screen in which it is possible to change the information concerning this document. The input can be inserted through HTML forms. An example is shown in Figure 4.4, where a part of the edit screen of a requirement is shown. This figure also shows how the relations between artefacts can be selected. This is achieved by two boxes where the left box displays, for example, the available modules which can be selected by clicking the button with the ‘>’ arrow. The module then moves to the right box, which shows the selected related modules.

The text of the document being manipulated can be formatted by using Bulletin Board Code (BBCode). With BBCode it is possible to add a tag around some text to create text-markup. The complete table with all BBCode options can be found in Appendix E.

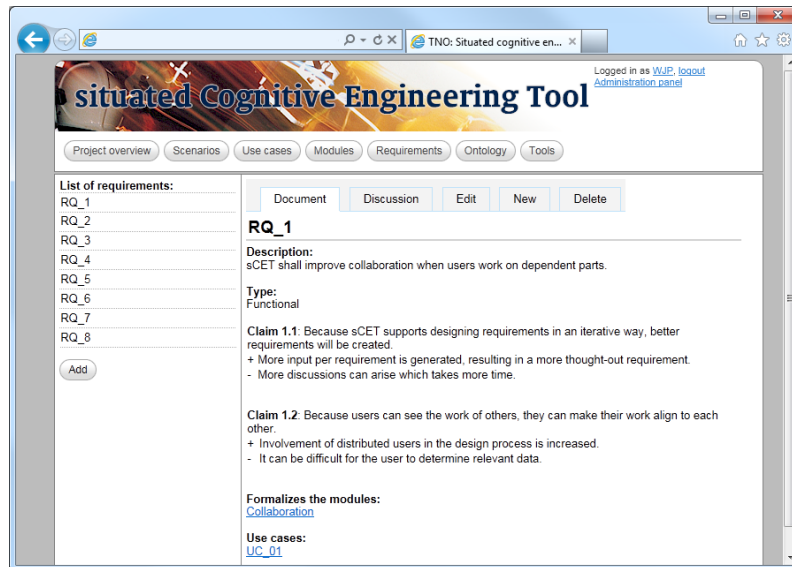


Figure 4.3: sCET requirement screen

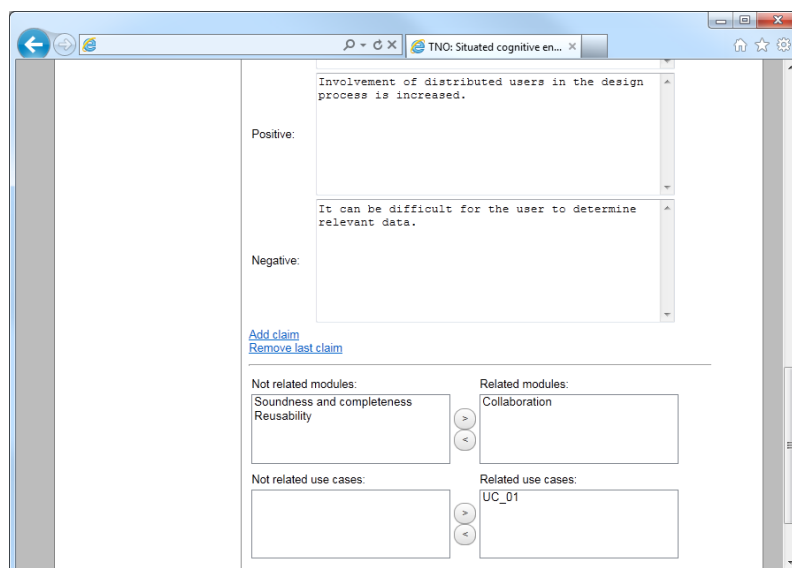


Figure 4.4: sCET requirement edit screen

## Chapter 5

# sCET evaluation

This chapter describes how the sCET tool has been evaluated. This evaluation has been done to check the effectiveness of using our tool to support the sCE methodology. The claims described in Section 4.2.2 are verified by user experience.

Section 5.1 describes how the evaluation has been performed and Section 5.2 describes the results of the experiment.

### 5.1 Experiment Set-up

An experiment has been carried out to verify the effectiveness of sCET. This is done by deploying a first version of sCET on a server and organising a workshop where users have the opportunity to learn and use sCET.

#### Workshop

We organised a workshop to test sCET in two ways, namely with an informal and a formal evaluation. The formal evaluation consisted of a questionnaire and the informal evaluation has been performed using the input from discussions. The group that attended the workshop consisted of members of the Mutual Empowerment project, as described in Section 1.1.

The course of the workshop was as follows: first the participants filled in the first part of a questionnaire about their expectations of using sCET. Second, the participants received a short oral introduction of the sCE methodology. Third, they went to a room where each participant got their own laptop on which they could work. They had one hour to define use cases, requirements and claims for their part of the project using sCET. If the participants had any questions they could ask the supervisors for help.

After the sCET session, participants had to present their results to each other. For each artefact each participant had created, a short explanation had to be given. During these presentations, the other participants had the opportunity to ask questions and discuss the work. After the presentations were given, participants had the opportunity

to share their thoughts on the workshop and the use of sCET in their project group. Finally, all participants filled in the second part of the questionnaire.

### User Questionnaire

A user questionnaire has been filled in by the participants to verify the user experience with the usage of sCET. The questionnaire was split into two parts, namely a part that is filled in at the beginning, and a part that is filled in at the end of the workshop.

The part that is filled in at the beginning of the workshop consists of two parts, namely questions about general information about the participant, like gender and age, among others, and questions focusing on the expectations of using sCET. This second part consists of eight multiple-choice questions and two open questions where participants can put down any other thoughts before starting with sCET.

The part that filled in at the end of the workshop consisted of seven parts, as described below:

1. *The situated Cognitive Engineering Methodology*. This section focuses on questions about the sCE methodology, whether the participants like it and think if it is useful for their work.
2. *sCET collaboration*. This section focused on questions about whether the participants think sCET helps improve cooperation between project members.
3. *sCET soundness and completeness*. This section focused on questions about whether the users think sCET will help them achieve a sound and complete design specification.
4. *sCET reusability*. This section focused on questions about whether the user expects that sCET data can be reused in other projects.
5. *sCET (with sCE experience)*. This section needs to be filled in by users who have sCE experience. It focused on questions about whether the users think that using sCET is better than textual documents.
6. *sCET (without sCE experience)*. This section needs to be filled in by users without sCE experience. This focused on questions about the experience the users had learning sCET.
7. *General remarks*. This section focused on open questions where participants can put down any other thoughts they have about using sCET.

The complete questionnaire can be found in Appendix C.

## Logging

The activity on the usage of sCET was logged using Google Analytics. This application keeps track of how the users navigate in the tool. A few features that are supported are *visitors count*, *page view count* and *average time on site* (Google, 2011). The information from the logging was used to improve the usability of the tool.

Besides the use of Google Analytics, logging information was stored in a local database. This was done to get additional information on areas which are not supported by Google Analytics, such as which data was entered by the users of the tool.

To fully take advantage of the logging features, a lot of usage of the tool and information about this usage is necessary. Because of the small user-base, the results of the logging features are not addressed in this thesis. However, the features have already been implemented so that they can be used directly when the usage increases.

## 5.2 Experiment Results

This section presents the results of the evaluation of sCET. Two types of evaluation were done, namely an informal and formal evaluation.

### 5.2.1 Informal Evaluation

During the workshop we obtained a lot of responses with practical tips on how the tool should be improved. Besides these practical tips, some ideas were also given on the general idea of using sCET.

*Guidance.* The participants mentioned that they required quite some guidance on the definition of use cases and requirements. Some participants had little or no experience with the sCE methodology, which made it hard to figure out how to write down the information. The level of depth in which they had to create requirements was also difficult to figure out.

*Reporting.* The participants had many questions about the reporting capabilities of sCET. They found it very important to be able to export the data to other programs, to further use it when creating reports. Although all information is accessible through sCET, it remains important to be able to collect all data in a central document which can be adjusted and printed out.

*Usability.* Participants gave a huge importance to usability, in the sense that the program should not have any quirks which could result in users getting irritated. For example, if users press the wrong button, data should not be lost.

*Security.* There were some concerns about the security of sCET. TNO is often working with sensitive data from the Dutch government and some of these pieces of information may not be accessed by unauthorized people.



*Visual design.* The visual design of the program is important because bad design dis-encourages new users from using it. The design should be in such a way that data is shown in a clear and evident way.

An interesting side result from the workshop came from a project member who joined the project after it was started. For other members it was not really apparent what this new member was going to do. At the workshop, he entered some simple requirements with just a few lines of information. At the presentations of the results he presented his work. Although he was not certain of his own work, other people started to understand what he wanted to do in the project. With those few lines he could communicate with the other project members by showing what he was doing. Hence, he brought the project members to the same level of thinking as himself.

### 5.2.2 Formal Evaluation

The questionnaire was filled in by five participants. These participants came from different disciplines and work on different parts of the ME project.

We statistically evaluated the results from the questionnaires with descriptive statistics. The central tendency and dispersion of the data is calculated and shown in Appendix D.

Three values are calculated, namely mean, median and range. For each claim, the results are evaluated. Some claims are difficult to verify, because it requires sCET to be used for a longer period. Therefore, they have not been included in the evaluation.

Since the questionnaire was only filled in by five participants, and no control group was used, no statistical significance over the mean values can be measured. Therefore it is also important to look at the median and range values, which tell something about how the participants agreed with each other. These values, in addition to the informal evaluation can still say something about the user experience if all participants agreed with each other. Below the results of the questionnaire are shown. They are grouped by the three core functions, namely *Collaboration*, *Soundness and Completeness*, and *Reusability*.

#### Collaboration

Claim 1.2	Because users can see the work of others, they can make their work align to each other.	
Questions:	17	mean: 4.4; median: 4; range: 1
	18	mean: 1.8; median: 3; range: 2
Evaluation:	Users think that sCET gets them involved in the work of others and do not fear that they will be distracted by the work of other users.	

Table 5.1: Claim 1.2 questionnaire results

Claim 2.1	Because sCE(T) is modular, users can work on different parts at the same time.	
Questions:	19	mean: 2.0; median: 3; range: 2
Evaluation:	Users do not really fear that using an internet connection will hinder them in using sCET.	

Table 5.2: Claim 2.1 questionnaire results

Claim 3.1	Because sCET can export data, outsiders can easily understand the design specification.	
Questions:	20	mean: 3.5; median: 3.5; range: 3
Evaluation:	Users are not convinced that sCET will help them with creating reports.	

Table 5.3: Claim 3.1 questionnaire results

### Soundness and Completeness

Claim 4.1	Because sCE(T) has defined its data fields, it discourages entering irrelevant data.	
Questions:	22	mean: 3; median: 2; range: 3
	23	mean: 3.2; median: 3; range: 2
Evaluation:	Users are a bit concerned that they cannot enter everything they want. They want a way to be able to enter additional information with their requirements.	

Table 5.4: Claim 4.1 questionnaire results

Claim 4.2	Because sCE(T) has defined its data fields, it is possible to see what data is missing.	
Questions:	24	mean: 3.6; median: 4; range: 1
Evaluation:	Users sometimes find data fields irrelevant, but maybe they do not really know what to enter there, or what is desired in that field.	

Table 5.5: Claim 4.2 questionnaire results

Claim 5.1	Because claims are defined by metrics, claims can be tested on their validity.	
Questions:	25	mean: 3.0; median: 3; range: 2
	26	mean: 3.6; median: 4; range: 1
Evaluation:	Users are not sure they can validate the requirements and do think that it takes too much time to create adequate claims.	

Table 5.6: Claim 5.1 questionnaire results

Claim 6.1	Because sCET is easy to learn, the learning curve of the sCE methodology decreases.	
Questions:	27	mean: 2.2; median: 2; range: 1
	28	mean: 3.8; median: 3; range: 2
	29	mean: 1.8; median: 1; range: 3
Evaluation:	Users were not convinced that they could deliver quick results. They had average convincing that they had learned the sCE methodology. But they did not dislike to learn a new tool.	

Table 5.7: Claim 6.1 questionnaire results

## Reusability

Claim 8.1	Because requirements from previous projects can be imported, new projects can start more quickly.	
Questions:	30	mean: 1.6; median: 2; range: 1
	31	mean: 2.4; median: 3; range: 2
Evaluation:	Users are not very convinced that they can reuse project data. Users do not think that they can find old project results better when using sCET either.	

Table 5.8: Claim 8.1 questionnaire results

## Before and after

Since one part of the questionnaire was answered before the workshop and the other was answered after the workshop, conclusions can be taken about the expectations and the real experience of using sCET.

Figure 5.1 shows the results of the questionnaire. Three main topics were chosen to be evaluated, namely whether the participants thought that sCET helps learn the sCE methodology, whether they like to learn a new tool and whether learning sCET is easy.

Before the start of the workshop, the participants of the workshop were convinced that sCET would help them learn the sCE methodology. After the workshop the average

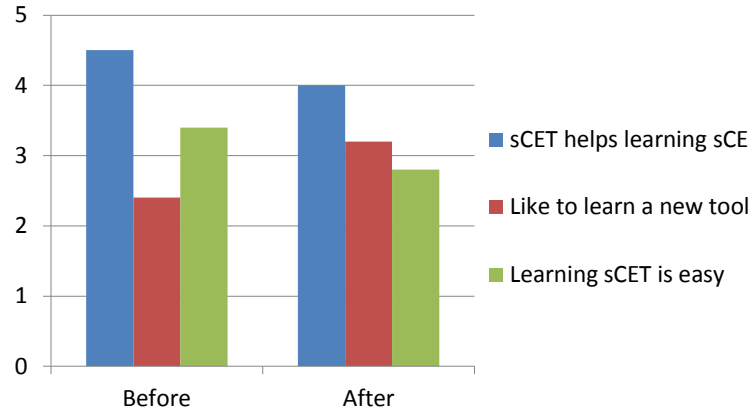


Figure 5.1: Mean scores of the questions before (1, 2, 3) and after (39, 29, 37/41) the workshop. Note: For the sCE learning questions only participants with no experience have been considered.

score dropped 0.5 points. This may be because users also found sCET harder to learn (see below).

Before the workshop, the participants thought they would not like to learn the tool. However, after the workshop the score of users who liked to learn the tool was higher. This means that using sCET was not as bad as they first thought.

After using sCET, participants found sCET harder to learn than they thought before the workshop. This was mainly on the aspect of how they should carry out the sCE methodology aspects. One reason for this could be that they did not really know the level of abstraction they needed to follow. Another difficulty was that some participants did not have in mind what they were going to do in the project. Using sCET forced them to think about their purpose in the project and to make their goals explicit.

# Chapter 6

## Discussion

The aim of this chapter is to reflect on the design and prototype of sCET and to give ideas to improve the design and implementation of the tool.

This discussion can be seen as the refine phase of the sCE methodology. First, sCET is compared to the other tools existing in the literature. Second, the results from the workshop evaluation, which are described in Chapter 5, are addressed. Third, the input of the evaluation is translated to refined requirements for sCET.

### 6.1 Tools Comparison

Several tools exist in the literature. Tools from requirements engineering and cognitive engineering have been selected to be compared with sCET.

From requirements engineering, the following tools are chosen: Borland CaliberRM (Borland Software Corporation, 2011), IBM's Rational DOORS (IBM, 2011a), IBM's Rational RequisitePro (IBM, 2011b) and Sparx Systems Enterprise Architect (Sparx Systems, 2011). From cognitive engineering, COGENT (Cooper and Fox, 1998) and MacSHAPA (Sanderson et al., 1994) are chosen. The characteristics of each tool are described below.

**Borland CaliberRM** is a requirements management tool which focuses on capturing requirements from the users needs and communicating them to project members and the stakeholders throughout the application life-cycle.

In CaliberRM, requirements are created in a hierarchical way. Requirement details are accessible through several tabs which contain: detailed information, traceability, validation, discussions, history, use case data, responsibilities and references. This information can be accessed through a web interface.

Several integrations with development tools are available, for example with Visual Studio and Eclipse. With these integrations it is possible to follow the requirement from the initial phases to the implementation phase. CaliberRM also contains a 'document factory' which allows requirements to be exported to a Microsoft Word template. These documents can be used for purposes like communicating with stakeholders and archiving.

As an extension to CaliberRM, CaliberRDM can be used as a visual and interactive way to define and manage application requirements. In CaliberRDM it is possible to visualise the design specification through interactive scenarios and simulations with the goal to achieve a complete design specification.

**IBM's Rational DOORS** is a requirements management tool which focuses on requirements. The tool has a desktop and web interface, like CaliberRM. The default view for requirements is a display in a document format. This means that the requirements are placed into chapters, sections and sub-sections. The advantage view is that it immediately looks like a report. The disadvantage of this is that the attributes of the requirement are accessed through pop-up windows, making a lot of mouse clicks necessary.

Each requirement can be discussed by adding a discussion topic. Changes to a requirement are stored in the history of the requirement. Requirements can be linked to each other by creating traceability links. This can be achieved by dragging and dropping requirements onto each other.

Requirements can also be dragged into Microsoft Word. This creates a hyper-link to the requirement on the web interface. In this way it is possible to create a document and link to each requirement.

Doors also supports integration with Microsoft Visual Studio and several other tools to create and maintain traceability between requirements in DOORS and Visual Studio.

**IBM's Rational RequisitePro** is a document-based approach to requirements management, in contrast to database-based approaches. To achieve this, a tight integration with Microsoft Word is in place.

In RequisitePro, a word document can be created with several types: glossary, requirements management plan, stakeholders request, supplementary requirements specification, use case specification and vision. Newly created document already contains a title page, a revision history table, a table of contents and several chapters with descriptions of what should be contained in that chapter. The content of these chapters can then be filled in by the project members.

In Word it is possible to add a requirement by clicking a button, which makes a dialogue-box appear. In this dialogue-box it is possible to see the revision history, attributes, traceability, hierarchy, and discussions of the requirement. These requirements also show in a list in the main screen of RequisitePro, where they can be edited.

The integration with Word has several limitations. To enable collaboration, the documents need to be stored on a shared network disk. Because the data is stored in a Word document, it is not easy to work on the same document at the same time. Although there is an interface to edit the requirements, changes do not show in the Word document until they are edited there. When one is working on the document, Word only shows the requirement type and description. To show more information, a pop-up dialogue has to be used.

Creating reports, however, is very easy. Since the working document is already a document, it can be easily modified to be used as a report to the stakeholders.

**Sparx Systems Enterprise Architect** is a product centered on the Unified Modelling Language (UML), which are visual diagrams to create models of systems. The tool consists of documents which are divided into several categories, for example, Testing, Maintenance, Database Engineering and Software Engineering, among others. Each document can contain text boxes and visual diagrams. These visual diagrams can be created in different forms, from mind mapping models to class diagrams.

Enterprise Architect also supports the creation of requirements, however, this is not a focus point of the tool. Integration is possible with DOORS, which can be used as the main requirements editing tool. In this case Enterprise Architect can be used to create traceability links in the diagrams.

Integration with development tools are also possible, for example, with Visual Studio and Eclipse. Enterprise Architect can even generate source code from the models, to facilitate the beginning of a project and generate reports in HTML or Word format.

**COGENT** (Cognitive Objects within a Graphical Environment) is an application to design models for cognitive processes and systems. It can be used to construct and test information processing models of cognition. Models are constructed through a graphical model editor and consist of box and arrow diagrams. Once some models are constructed, they can be analysed to check their behaviour.

Models are created by dragging and dropping graphical components to the screen. By clicking on these items it is possible to set the rules and conditions of these components. This is done by a scripting language where it is possible to define rules in an if-then-else construct. The history of a model is contained and shown in a diagram, where it is possible to see the evolution of the model, and allowing different branches to be created.

COGENT is very different from the other tools, because it does not contain any requirements information. Although the tool is useful, it is not as extensive as the tools described so far.

**MacSHAPA** is a cognitive engineering software tool for observational data analysis. This means that it is used to collect and analyse data about the usage of systems. The results of this analysis can identify problems or strengths of existing systems and are useful to test prototype designs.

The testing of these systems is done by analysing video recordings of real-time tasks performance. All the collected data can then be stored in a spreadsheet, which can be manipulated through a Prolog-based Query Language.

Although MacSHAPA does not support creating requirements like most requirements engineering tools, it still can be used for gathering user needs, performing requirement analysis and producing systems specification. The knowledge gathered through MacSHAPA can then be used to refine the design of the system.

	Tool name	IBM's DOORS	Enterprise Architect	Caliber-RM	Rational RequisitePro	COGENT	MacSHAPA	sCET
General	Web interface	+		-	-			+
	Database-centric	+	+	+	-			+
	Document-centric				+	+	+	
	Graphical-centric		+			+	-	
	Text-centric	+		+	+		-	+
	Shared repository	+	+	+	-			+
	History	+	+	+	+	+		.
	Report creation	+	+	+	+			-
Requirements eng.	Requirements	+	-	+	+			+
	Use cases	-	+	-	+			+
	Scenarios		-	-	-	+	+	+
	Functional modules	+	-	-	+			+
	Claims							+
	Metrics					+	+	-
	Traceability	+	+	+	+			+
	Discussions	+	+	+	+			+
Modelling	Mind mapping		+					
	Flow charts		+			+		
	Cognitive model					+	-	
	Use case diagrams		+					
	Design specification overview	+	-	-	+			
Impl.	Import/generate code							
	Test results	+	+	+	+		+	.
	Development integration	+	+	+				

Table 6.1: Tools Comparison. An empty cell indicates that no functionality of that subject is present. A '-' indicates that limited functionality, '+' indicates full functionality and '.' indicates that the functionality is not implemented in the prototype yet.



Table 6.1 shows an overview of the functionality of each tool discussed here. Tool functionality is divided into the following topics: general, requirement engineering, modelling and implementation aspects.

The tools from the requirements engineering area focus on different topics than the cognitive engineering tools. RE tools focus on defining the technical architecture of a product, and the CE tools cover the processes of the human mind when using the product. Most RE tools come from the software engineering industry and are often sophisticated products with a lot of functionality. The CE tools are more specialized on one topic of the CE process. CE tools also seem to be mostly coming from research institutes, as tools to support theories in contrast with RE tools, which often come from commercial companies. This is also visible in the prices, as CE tools tempt to be free of charge, and RE tools may cost up to \$2,500.-, as, for example, RequisitePro.

The current version of the sCET prototype focuses on the requirements engineering topic. To increase the support of CE projects, the requirements of sCET need to be extended to the CE areas.

## 6.2 sCET Improvements

The aim of this section is to translate the suggestions given in the workshop into concrete improvement to future versions of sCET. Six main improvements have been derived from the workshop, namely *guidance*, *reporting*, *security*, *visual design and usability*, *validation* and *reusability*.

### 6.2.1 Guidance

The participants had some difficulties entering the use cases and requirements. It was mentioned that it was hard to determine the level of depth and what should be entered in each input field. Although this was mainly caused by lack of experience of the participants with the sCE methodology, the results of the questionnaire also showed that the participants found sCET more difficult to use than they had expected before the workshop. To decrease the difficulty of using sCET, the users need to be better guided.

Several adjustments to the prototype can improve how users are guided. First, it is important that every input field has some examples with possible values that can be entered. This can be done by adding information buttons next to the text field, which enable the user to quickly see some example text. One example is the requirement description input field. The information added to the field can be: ‘The requirement description field often starts with “The program shall...”’.

Second, sCET can help users add field information by giving concrete suggestions for the input fields. One could think of the possibility to create terms in the ontology which inherit from an Actor superclass. In the use cases actor fields, a list of available actors could be shown. In this way the users can be prompted with possibilities that can be entered in each input field.

Finally, one can think of a setup assistant (e.g., a wizard) that helps users set up

a project. When a new project is created, the wizard would ask questions about the domain in which the project is situated, which actors are involved, which core functions are defined, among others. At the end of the wizard, the data is analysed and use cases and requirements examples can be generated and added to sCET.

### 6.2.2 Reporting

During the workshop, the participants mentioned that reporting is important. However, the results of the questionnaire show that participants are not convinced that they can use sCET to create reports which can be shown to stakeholders. Since the participants work at TNO, where reporting to the stakeholders is a key factor for the success of a project, they have to create reports at certain moments. These reports are used to check if the execution of the project is still justifiable. Therefore, the reports have to be created in such a way that results are made explicit.

The current version of sCET has several exporting capabilities. It is possible to export the use cases and requirements to the textual document in L<sup>A</sup>T<sub>E</sub>X and Microsoft Word formats. However, this functionality only compiles all available use cases and requirements and puts them straight into these documents.

To create usable reports, the wishes of both the sCET users and the stakeholders needs to be investigated. Interviews with users and stakeholders can generate ideas on desirable formats for reports. The tool should then be able to create the documents in these specific formats.

Ultimately, sCET can have an extensive reporting capability which can be applied by going through a report setup which then generates a report template that only needs some minor adjustments to create a ready-made report for the stakeholders.

### 6.2.3 Security

Security is an important aspect when working with sensitive data. The participants mentioned that they had some concerns about the security of using a web application to create design specifications. Stakeholders could also have problems with their innovative ideas being accessible through the Internet. TNO sets high standards for security. However, in a lot of projects, users from different companies have to work together, so that it is not possible to deny all access to documents from outside the company.

Security needs to be tight, however, awareness of users that sensitive data should not be entered into sCET is something the tool cannot enforce. Although, sCET needs to have a decent security to prevent unauthorised access, no 100% security can be guaranteed.

Ideally, there would be two versions of sCET: a local version which is not accessible through the Internet that could be used for high security projects, for example, for the Ministry of Defence, and a version which would be accessible through the Internet. The second version could be used for projects where confidentiality is less of an issue. In this way, the benefits of access through the internet would be preserved, while high security projects can also benefit from using sCET.

#### **6.2.4 Visual Design and Usability**

The sCET prototype has a fairly simple interface. Although all functionality to use sCET was implemented, the participants indicated that an attractive design of the interface would motivate users to continue using sCET. New users would also be compelled to use sCET. If the design of a tool is attractive, users most likely experience the tool as easy and fun to use (Sutcliffe, 2002).

Usability, in general, is important. This applies specifically for sCET, because project leaders can force project members to use sCET. The level of annoying errors and bugs need to be kept to a minimum, otherwise they discourage users from using sCET. At the workshop we had some problems with users pressing the wrong buttons in the browser which directed them back to the previous page. This sometimes resulted in loss of data, which made some users quite frustrated. An auto-save function would help in reducing the risk of losing data when working with sCET.

#### **6.2.5 Validation**

The results of the questionnaire show that the participants were not certain about using sCET to validate claims in a structured way. They also stated that creating adequate claims takes too much time. However, claims are essential, because they justify the existence of the requirements. These justification make sure that the stakeholders know that the project is useful for them. Users somehow need to be stimulated to create the claims.

The current version of sCET does not support any validating options, therefore validation of claims must be done manually. To improve the validating capabilities, sCET should be able to show a list of claims. This list should contain information about the validation process of each claim. When a requirement or claim is modified, sCET should tell the user that the validation of the claim is out-of-date. Creating a report of claims where the validation is not correct should facilitate for checking of the validity of the requirements.

#### **6.2.6 Reusability**

The results of the questionnaire show that the participants were not convinced that sCET would enable easy access to old project results. The current version of sCET shows a list of projects to which the users have access. By selecting each project they can browse through the project's data. However, this is a time consuming activity. Currently, there is no search function with which users can search in existing project for certain keywords. By adding a search function, users could easily retrieve the data they are looking for. However, to use some search functionality users should know what they are looking for, but this is not always the case.

The participants also were not very sure about the reuse of older data into new projects. Currently, there is no support for copying old requirements into new projects. If users want to reuse data, they have to copy all information manually. A export/import

function that allows data (e.g., requirements) to be copied into new projects would save time on the time-consuming process of starting-up a new project.

To test reusability, sCET should be used for a longer period of time. Once there is enough old project data available, the reusability of sCET could then be properly evaluated.

## 6.3 sCET Refinement

Some ideas described in the previous sections are already in the requirements of sCET, but need to be implemented differently. Other ideas are not specified in requirements yet. This section describes the refined and new requirements which correspond to some ideas mentioned in Section 6.2.

First, we describe the requirements that have been refined. Second, we describe the additional requirements. The complete list of all requirements is shown in Appendix B.

### 6.3.1 Refined Requirements

Requirement 3	RQ_3
Description:	sCET shall improve communication with outsiders.
Claim 3.1	<i>Because users can create reports with sCET, stakeholders can easily understand the design specification.</i> <ul style="list-style-type: none"> <li>+ Less time is needed to explain the design specification.</li> <li>+ Stakeholders can identify the progress of the project.</li> <li>- Time is needed to create reports.</li> </ul>
Claim 3.2	<i>Because users can create ontologies, the design specification will become more understandable for outsiders.</i> <ul style="list-style-type: none"> <li>+ If the design specification is easier to understand miscommunication shall be prevented.</li> <li>- Creating an ontology takes time.</li> </ul>
Use Cases:	UC_03, UC_04

Table 6.2: RQ\_3

In Requirement 3, the exporting capabilities of sCET is revised. Claim 3.1 now explicitly describes that sCET should have the possibility to create reports that improve the communication with other people.

Requirement 5	RQ_5
Description:	sCE(T) shall enforce the specification of verifiable design specifications.
Claim 5.1	<i>Because claims are defined by metrics, claims can be tested for their validity.</i> + If the claims are refuted, they can be refined to be replaced by better claims. - Defining adequate metrics takes time.
Claim 5.2	<i>Because sCET can give an overview of claims and their validation status, users can easily check which requirements need validation.</i> + Claims can be validated easily. - Users may be fixated on keeping each claim verified, while they may still be changed. This can result in unnecessary work being done.
Use Cases:	UC_06

Table 6.3: RQ\_5

In Requirement 5, a new claim is added. This claim adds the functionality that sCET should give an overview of claims and their validation status. This functionality should give the users the possibility to review the status of their progress in creating the claims.

Requirement 6	RQ_6
Description:	sCET shall facilitate the learning of the sCE methodology.
Claim 6.1	<i>Because sCET guides the users in creating the design specification, the learning curve decreases.</i> + New user can deliver results more quickly. - Users could proceed in a disorganized way and not really learn the methodology. - Users could dislike learning a new tool in general.
Use Cases:	UC_07

Table 6.4: RQ\_6

In Requirement 6, the claim is changed in the sense that sCET should guide users when creating a design specification, instead of just being easy to use. This should help users to learn how to apply the sCE methodology in sCET.

### 6.3.2 Added Requirements

Requirement 9	RQ_9
Description:	sCET shall ensure the security of sensitive data.
Claim 9.1	<i>Because sCET ensures security, no unauthorized users can access sensitive data.</i> + Sensitive data is secured. - No 100% security can be guaranteed. Therefore, users need to be aware that security can be compromised.
Use Cases:	UC_01, UC_02

Table 6.5: RQ\_9

Requirement 9 is added, because the participants of the sCET workshop indicated that sCET should have an adequate security which prevents unauthorised users from accessing data.

Requirement 10	RQ_10
Description:	sCET shall show sCE artefacts in a graphical way.
Claim 10.1	<i>Because sCET shows use cases in the form of use case diagrams, it is easy to see the relations between actors and functions.</i> + More insight is given in the use cases - Users need to learn the notation for use case diagrams how to use it.
Use Cases:	

Table 6.6: RQ\_10

Requirement 10 is added, because sCET should be able to display the sCE artefacts in a way that they are easy to understand. A graphical representation can help to make the design specification easier to understand.

Requirement 11	RQ_11
Description:	sCET shall incorporate test results.
Claim 11.1	<i>Because sCET incorporates test results, design decisions can be made more easily.</i> + Looking at test results facilitates the refinement of requirements. - Entering test results takes time.
Use Cases:	

Table 6.7: RQ\_11

Requirement 11 is added, because sCET should support the entire duration of sCE projects. Since the testing phase is an important aspect of the sCE methodology, its support should be incorporated in sCET.

Requirement 12	RQ_12
Description:	sCET shall have integration with development tools such as Eclipse and Visual Studio.
Claim 12.1	<i>Because sCET has integration with development tools such as Eclipse and Visual Studio, it is possible to link requirements to the actual code.</i> + Code can be traced back to its requirements. - The design and implementation phases fade into one phase.
Use Cases:	

Table 6.8: RQ\_12

Requirement 12 is added, because design specification are often tested by means of creating prototypes. Linking requirements to actual code makes it easier to see which requirements are implemented and which requirements still need to be incorporated in the prototype.

Requirement 13	RQ_13
Description:	sCET shall support cognitive modelling diagrams.
Claim 13.1	<i>Because sCET supports cognitive modelling diagrams, it is possible to get a better understanding of the human-machine interaction.</i> + More insight is gained in the human-machine interaction. - sCET becomes complicated for new users.
Use Cases:	

Table 6.9: RQ\_13

Requirement 13 is added, because the area of CE is much broader then creating a requirements document. The other aspects of CE should be incorporated in sCET as well, because it increases the insight in the human factor knowledge when designing human-machine interaction systems.

## Chapter 7

# Conclusion

This chapter gives the final remarks of this thesis. It gives a conclusion to this work and some recommendations for future work.

Section 7.1 gives the general conclusion of this work, Section 7.2 gives recommendations on how to proceed with the tool and, finally, Section 7.3 discusses future work.

### 7.1 General conclusions

Using a tool to create complex cognitive systems has several advantages, such as keeping documents current, linking artefacts, among others. Currently, there are no tools that combine the knowledge of Requirements Engineering and Cognitive Engineering to create design specifications.

In this thesis a design for a Cognitive Engineering tool that follows the situated Cognitive Engineering methodology is proposed. The goal of our situated Cognitive Engineering Tool is to address the three limitations of using textual documents, namely in the areas of *collaboration*, *soundness and completeness* and *reusability*.

After we designed the tool and created a prototype, the tool was evaluated. From the evaluation, we showed that the tool indeed addresses the limitations. Therefore we can conclude that the questions formulated in Section 1.2 have been answered.

Collaboration between project members with different disciplinary backgrounds can be achieved by using sCET. The workshop showed that capturing requirements and their claims can help a project member explain to other project members the goals he/she tries to achieve in the project.

Soundness and completeness of the design specification can also be achieved by using sCET. This can be done by guiding the users through the creation of the design specification. The tool should inform to the users what they need to write down in each input field and give suggestions with sample statements.

Currently, it is difficult to evaluate how sCET can support the reuse of earlier work, because the tool has not been used for a sufficiently long period. Still, a couple of concluding remarks can be made. The tool should support search functionality, which



can be used to search through older project data. sCET should also support importing data from other projects into new projects.

There are some disadvantages of using a tool for CE projects. Some users do not want to learn a new tool, so it is important that users are attracted to the tools layout, interface and functionality. The tool could be designed in such a way that it is pleasant to work with and the tool should not have any quirks that may frustrate the users.

Using a prototype of sCET to evaluate the usage of a tool gave a huge insight in its effectiveness. The gathered experience can be used to improve the tool in the future. From the results of the workshop, several recommendations for improvements were identified.

One of the most important aspects of using a tool is that it forces project members upfront to think about their goals. Making these goals explicit does not only help themselves know what they want to achieve, but also helps them communicate with other project members.

Currently, the prototype version of sCET tool is being used in several projects. There are a lot of positive responses about using sCET instead of textual documents. In the near future, this version will be used to get more experience on the use of a sCE tool. This experience will be used to express the usefulness of using a sCE tool and to convince other people in using sCET. This should enable the development of a new version of sCET that incorporates all the new ideas gathered during the usage of the sCET prototype.

This project has pioneered the possibilities of a tool for designing complex human-machine systems. The possibilities to broaden the scope of the tool are great. Currently sCET only covers one part of the design process, namely requirements capturing. Future research is needed to explore these possibilities. In the end, a complete system could be created which supports all the activities of research projects.

## 7.2 Recommendations

Based on the findings and conclusions drawn we recommended to keep using the current version, and only make some minor adjustments in the near future. In this way, it is possible to extensively document all the wishes from the users and the stakeholders. An external company can be hired later to create a new version of sCET which incorporates all these wishes. The following points should be addressed in a new version:

- *Guidance.* sCET should help in guiding users through the sCE methodology. A way to achieve this is by adding a setup guide which guides users to specify use cases and requirements when a new project is started.
- *Reporting.* The ability to export data to reports is important for communication with stakeholders. Wishes of users and stakeholders should be investigated and applied to the tool.

- *Security.* Security should be adequate in such a way that confidential information is not compromised. However, users should also be aware that no critical data should be entered in sCET.
- *Visual design and usability.* An appealing interface layout and functionality will help in such a way that new users will be compelled to use sCET, and current users will keep using sCET.
- *Validation.* The validation of requirements should be implemented in a way that users can create oversights which show the validation progress.
- *Reusability.* The users should be able to reuse older project data in an easy way. This will reduce the amount of duplicate work.

### 7.3 Future Research

In this thesis, a tool was designed to improve the sCE process. However, the possibilities of using a tool for creating design specifications are much greater. Based on ideas that were gathered during the development of sCET, the following recommendations for future research are given.

- Teamforge integration.

The use of sCET could be integrated with TeamForce, which is used at TNO to manage source code of projects (CollabNet, 2011).

Design specification are often tested by means of creating prototypes. Linking requirements to actual code makes it easier to see which requirements are implemented and which requirements still need to be incorporated in the prototype. Integrating the design of a product with the implementation could help align both phases in such a way that information is easily accessible.

- Designing using images.

The process of generating ideas is important when designing a new product. McCrickard has studied ways to improve the process of generating ideas by using touch tables (McCrickard et al., 2011). This could be incorporated in sCET in such a way that the ideas created with touch tables can be imported into sCET. This would make it attractive for users to start using sCET, as visual designing is an easy way to get users interest. It can also be used to get stakeholders involved into the design process, as no real knowledge of the sCE methodology should be required to generate the ideas for use cases and requirements by means of visual design.

- Design space visualisations. Work has been done on the visualization of requirements, claims and their relations (Westera et al., 2010). Incorporating this into sCET can enable the user to see the status of sCE artefacts and which requirements and claims are affected by change.

By creating overviews using visual representations it makes it easy to see the current progress of the design specification. Requirements who are not yet linked to other artefacts, like use cases and claims, can be spotted easily. These overviews can make sure that the design specification is as complete as possible.

- Use case diagrams. Use cases can be visualised through use case diagrams. These diagrams allow users to have a graphical overview of the relations between actors and actions. To be able to incorporate this in sCET, the relation between the use cases of sCET and the use case diagrams have to be studied.

# 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/>.
- Inc. CollabNet. Teamforce, 2011. URL <http://www.collab.net/products/ctf/>.
- R. Cooper and J. Fox. Cogent: A visual design environment for cognitive modeling. *Behavior Research Methods*, 30(4):553–564, 1998.
- 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.
- Google. Analytics, 2011. URL <http://www.google.com/analytics/>.
- 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.
- D.S. McCrickard, S. Wahid, S.M. Branham, and S. Harrison. Achieving both creativity and rationale: Reuse in design with images and claims. *Human Technology*, 7(1), 2011.
- 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.
- P. Sanderson, J. Scott, T. Johnston, J. Mainzer, L. Watanabe, and J. James. Macshapa and the enterprise of exploratory sequential data analysis (esda). *International Journal of Human-Computer Studies*, 41(5):633–681, 1994.
- 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](http://www.sparxsystems.eu/Sparx).
- A. Sutcliffe. Assessing the reliability of heuristic evaluation for web site attractiveness and usability. In *System Sciences, 2002. HICSS. Proceedings of the 35th Annual Hawaii International Conference on*, pages 1838–1847. IEEE, 2002.
- 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.
- TNO. Tno jaarverslag 2010, 2010. URL <http://www.rijksoverheid.nl/bestanden/documenten-en-publicaties/jaarverslagen/2011/06/20/jaarverslag-2010-tno/jaarverslag-2010-tno.pdf>.
- 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.

## Appendix A

### sCET Use Cases

Use case 1	UC_01
Description:	Different [[scet—users]] work on one requirement
Goals:	Get better requirements
Actors:	2x [[scet—user]]
Pre-Conditions:	A sCET project is created
Post-Conditions:	The requirement is updated by [[scet—user]] 2.
Requirements:	RQ_1, RQ_9
Main action sequence:	1 User 1 enters a requirement 2 User 2 looks at the requirement 3 User 2 makes changes to the requirement

Table A.1: UC\_01

Use case 2	UC_02
Description:	Users work on different parts of the design specification at the same time.
Goals:	Parallel cooperation
Actors:	2x [[scet:user]]
Pre-Conditions:	Defined workload for both users
Post-Conditions:	A design specification
Requirements:	RQ_2, RQ_9
Main action sequence:	1 User 1 logs into sCET. 2 User 2 logs into sCET. 3 User 1 and User 2 both work on their part of the design specification

Table A.2: UC\_02

Use case 3	UC_03
Description:	Exporting data
Goals:	Communicate with stakeholders
Actors:	1x User 1x Stakeholder
Pre-Conditions:	A design specification which is inserted into sCET.
Post-Conditions:	An up-to-date Stakeholder
Requirements:	RQ_3
Main action sequence:	1 A user performs an export of the sCET design specification 2 The user shows the export to the stakeholder 3 The stakeholder understands the export

Table A.3: UC\_03

Use case 4	UC_04
Description:	Create ontologies
Goals:	Communicate with users
Actors:	2x [[scet—User]]
Pre-Conditions:	A term which is ambiguous
Post-Conditions:	A term with definition
Requirements:	RQ_1, RQ_3
Main action sequence:	1 User 1 creates a requirement. 2 User 2 looks at the requirement, but does not understand a term. 3 User 1 adds the term to the ontology. 4 User 2 now understands the requirement.

Table A.4: UC\_04

Use case 5	UC_05
Description:	Create a complete requirement
Goals:	A complete requirement
Actors:	[[scet—User]]
Pre-Conditions:	An incomplete requirement
Post-Conditions:	A complete requirement
Requirements:	RQ_4
Main action sequence:	1 User looks at a requirement. 2 User spots an empty field. 3 User adds information for that field.

Table A.5: UC\_05



Use case 6	UC_06
Description:	Verifiable design space
Goals:	Create a verifiable design space
Actors:	User
Pre-Conditions:	A requirement without metrics
Post-Conditions:	A verified requirement
Requirements:	RQ_5
Main action sequence:	1 The user looks at a requirement 2 The user sees that there are no metrics defined for the claims of the requirement 3 The user adds metrics for the requirement 4 The requirement can be verified

Table A.6: UC\_06

Use case 7	UC_07
Description:	User learns the sCE methodology
Goals:	learn the sCE methodology
Actors:	User
Pre-Conditions:	User without knowledge of the sCE methodology
Post-Conditions:	User with knowledge of the sCE methodology
Requirements:	RQ_6
Main action sequence:	1 The User knows nothing about the sCE methodology 2 The User uses sCET 3 The User now knows something about the sCE methodology

Table A.7: UC\_07

Use case 8	UC_08
Description:	Iterative process
Goals:	Develop the design specification through an iterative process.
Actors:	User
Pre-Conditions:	An existing requirement
Post-Conditions:	A updated requirement
Trigger:	The requirement becomes outdated
Requirements:	RQ_7
Main action sequence:	1 A user spots an outdated requirement. 2 The user looks at the previous versions of the requirement. 3 The user updates the requirement

Table A.8: UC\_08

Use case 9	UC_09
Description:	Reuse the project data to reduce duplicate work
Goals:	Reuse of project data
Actors:	User
Pre-Conditions:	An existing project
Post-Conditions:	A new project with data
Trigger:	A new project is started
Requirements:	RQ_8
Main action sequence:	1 A new project is started 2 A user exports data from a previous project 3 The user imports this data into the new project

Table A.9: UC\_09

## Appendix B

# sCET Requirements

### Collaboration

Requirement 1	RQ_1
Description:	sCET shall improve collaboration when users work on dependent parts.
Claim 1.1	<i>Because sCET supports designing requirements in an iterative way, better requirements will be created.</i> + More input per requirement is generated, resulting in a more thought-out requirement. - More discussions can arise, which takes more time to develop the system.
Claim 1.2	<i>Because users can see the work of others, they can make their work align to each other.</i> + Involvement of distributed users in the design process is increased. - It can be difficult for the user to determine relevant data.
Use Cases:	UC_01, UC_04

Table B.1: RQ\_1

Requirement 2	RQ_2
Description:	sCE(T) shall support parallel collaboration on different parts.
Claim 2.1	<i>Because sCE(T) is modular, users can work on different parts at the same time.</i> <ul style="list-style-type: none"> <li>+ Users do not have to wait for each other to finish working on their part.</li> <li>- A network connection is required to access the tool.</li> </ul>
Use Cases:	UC_02

Table B.2: RQ\_2

Requirement 3	RQ_3
Description:	sCET shall improve communication with outsiders.
Claim 3.1	<i>Because users can create reports with sCET, stakeholders can easily understand the design specification.</i> <ul style="list-style-type: none"> <li>+ Less time is needed to explain the design specification.</li> <li>+ Stakeholders can identify the progress of the project.</li> <li>- Time is needed to create reports.</li> </ul>
Claim 3.2	<i>Because users can create ontologies, the design specification will become more understandable for outsiders.</i> <ul style="list-style-type: none"> <li>+ If the design specification is easier to understand miscommunication shall be prevented.</li> <li>- Creating an ontology takes time.</li> </ul>
Use Cases:	UC_03, UC_04

Table B.3: RQ\_3

Requirement 9	RQ_9
Description:	sCET shall ensure the security of sensitive data.
Claim 9.1	<i>Because sCET ensures security, no unauthorized users can access sensitive data.</i> <ul style="list-style-type: none"> <li>+ Sensitive data is secured.</li> <li>- No 100% security can be guaranteed. Therefore, users need to be aware that security can be compromised.</li> </ul>
Use Cases:	UC_01, UC_02

Table B.4: RQ\_9

## Soundness and Completeness

Requirement 4	RQ_4
Description:	sCET shall enforce users to create an complete design specification.
Claim 4.1	<p><i>Because sCE(T) has defined its data fields, it discourages entering irrelevant data.</i></p> <ul style="list-style-type: none"> <li>+ Less irrelevant data results in better understandable design specifications.</li> <li>- Sometimes additional information might be hard to enter.</li> </ul>
Claim 4.2	<p><i>Because sCE(T) has defined its data fields, it is possible to see what data is missing.</i></p> <ul style="list-style-type: none"> <li>+ Missing data can be identified and added easily.</li> <li>- If some fields are irrelevant for a specific case, they cannot be removed.</li> </ul>
Use Cases:	UC_05

Table B.5: RQ\_4

Requirement 5	RQ_5
Description:	sCE(T) shall enforce the specification of verifiable design specifications.
Claim 5.1	<p><i>Because claims are defined by metrics, claims can be tested for their validity.</i></p> <ul style="list-style-type: none"> <li>+ If the claims are refuted, they can be refined to be replaced by better claims.</li> <li>- Defining adequate metrics takes time.</li> </ul>
Claim 5.2	<p><i>Because sCET can give an overview of claims and their validation status, users can easily check which requirements need validation.</i></p> <ul style="list-style-type: none"> <li>+ Claims can be validated easily.</li> <li>- Users may be fixated on keeping each claim verified, while they may still be changed. This can result in unnecessary work being done.</li> </ul>
Use Cases:	UC_06

Table B.6: RQ\_5

Requirement 6	RQ_6
Description:	sCET shall facilitate the learning of the sCE methodology.
Claim 6.1	<i>Because sCET guides the users in creating the design specification, the learning curve decreases.</i> <ul style="list-style-type: none"> <li>+ New user can deliver results more quickly.</li> <li>- Users could proceed in a disorganized way and not really learn the methodology.</li> <li>- Users could dislike learning a new tool in general.</li> </ul>
Use Cases:	UC_07

Table B.7: RQ\_6

## Reusability

Requirement 7	RQ_7
Description:	sCET shall make the iterative process insightful.
Claim 7.1	<i>Because users can see the history of requirements better evaluation choices will be made.</i> <ul style="list-style-type: none"> <li>+ Users do not repeat previously made mistakes.</li> <li>- Users can lose the overview because of too much information.</li> <li>- Navigation across the design specification can become more difficult when more information is present.</li> </ul>
Use Cases:	UC_08

Table B.8: RQ\_7

Requirement 8	RQ_8
Description:	sCET shall make design specifications reusable.
Claim 8.1	<i>Because requirements from previous projects can be imported, new projects can start more quickly.</i> <ul style="list-style-type: none"> <li>+ Importing old requirements results in less work for users.</li> <li>- Errors which were made previously can persist in related projects.</li> </ul>
Use Cases:	UC_09

Table B.9: RQ\_8

Requirement 10	RQ_10
Description:	sCET shall show sCE artefacts in a graphical way.
Claim 10.1	<i>Because sCET shows use cases in the form of use case diagrams, it is easy to see the relations between actors and functions.</i> <ul style="list-style-type: none"> <li>+ More insight is given in the use cases</li> <li>- Users need to learn the notation for use case diagrams how to use it.</li> </ul>
Use Cases:	

Table B.10: RQ\_10

Requirement 11	RQ_11
Description:	sCET shall incorporate test results.
Claim 11.1	<i>Because sCET incorporates test results, design decisions can be made more easily.</i> <ul style="list-style-type: none"> <li>+ Looking at test results facilitates the refinement of requirements.</li> <li>- Entering test results takes time.</li> </ul>
Use Cases:	

Table B.11: RQ\_11

Requirement 12	RQ_12
Description:	sCET shall have integration with development tools such as Eclipse and Visual Studio.
Claim 12.1	<i>Because sCET has integration with development tools such as Eclipse and Visual Studio, it is possible to link requirements to the actual code.</i> <ul style="list-style-type: none"> <li>+ Code can be traced back to its requirements.</li> <li>- The design and implementation phases fade into one phase.</li> </ul>
Use Cases:	

Table B.12: RQ\_12

Requirement 13	RQ_13
Description:	sCET shall support cognitive modelling diagrams.
Claim 13.1	<i>Because sCET supports cognitive modelling diagrams, it is possible to get a better understanding of the human-machine interaction.</i> <ul style="list-style-type: none"> <li>+ More insight is gained in the human-machine interaction.</li> <li>- sCET becomes complicated for new users.</li> </ul>
Use Cases:	

Table B.13: RQ\_13



## Appendix C

# sCET Workshop Questionnaire

### General information

Gender: male / female

Age:

Level of education:

Education:

Company:

Company location:

Sector expertise:

### Questions before the workshop

Please indicate how much you agree with the following statements, by encircling number of a scale from 1 (disagree very much) to 5 (agree very much).

- |    |                                                                                        |   |   |   |   |   |
|----|----------------------------------------------------------------------------------------|---|---|---|---|---|
| 1. | I think that using sCET will help me learn the sCE methodology                         | 1 | 2 | 3 | 4 | 5 |
| 2. | I like it to learn to use sCET                                                         | 1 | 2 | 3 | 4 | 5 |
| 3. | I expect that learning to use sCET will be easy and fast                               | 1 | 2 | 3 | 4 | 5 |
| 4. | I think that I can improve communication between project members with sCET             | 1 | 2 | 3 | 4 | 5 |
| 5. | I think that with using sCET I can easily create and refine good design specifications | 1 | 2 | 3 | 4 | 5 |
| 6. | I think that with sCET I can easily share and maintain my design specification         | 1 | 2 | 3 | 4 | 5 |
| 7. | I have experience in requirement engineering                                           | 1 | 2 | 3 | 4 | 5 |
| 8. | I have worked with the situated Cognitive Engineering methodology before               | 1 | 2 | 3 | 4 | 5 |

Please answer the following questions in your own words:

9. What do you expect from sCET?

10. Do you have any other remarks?

## Questions after the workshop

### The situated Cognitive Engineering Methodology

Please indicate how much you agree with the following statements, by encircling number of a scale from 1 (disagree very much) to 5 (agree very much).

- |                                                                           |   |   |   |   |   |
|---------------------------------------------------------------------------|---|---|---|---|---|
| 11. I like the sCE methodology.                                           | 1 | 2 | 3 | 4 | 5 |
| 12. I think the sCE methodology is useful for my work.                    | 1 | 2 | 3 | 4 | 5 |
| 13. The sCE methodology helps me in defining a good design specification. | 1 | 2 | 3 | 4 | 5 |
| 14. I think I will use the sCE methodology in future projects.            | 1 | 2 | 3 | 4 | 5 |

### sCET collaboration

Please indicate how much you agree with the following statements, by encircling number of a scale from 1 (disagree very much) to 5 (agree very much).

- |                                                                                         |   |   |   |   |   |
|-----------------------------------------------------------------------------------------|---|---|---|---|---|
| 15. I like the idea that other people can edit my data.                                 | 1 | 2 | 3 | 4 | 5 |
| 16. Looking at the input of others helps me with creating my own.                       | 1 | 2 | 3 | 4 | 5 |
| 17. sCET gets me involved with the work of others.                                      | 1 | 2 | 3 | 4 | 5 |
| 18. The data of others distract me from my work.                                        | 1 | 2 | 3 | 4 | 5 |
| 19. The need of an internet connection hinders me from working with sCET.               | 1 | 2 | 3 | 4 | 5 |
| 20. Exporting the data of sCET would save me time in creating reports for other people. | 1 | 2 | 3 | 4 | 5 |

### sCET soundness and completeness

Please indicate how much you agree with the following statements, by encircling number of a scale from 1 (disagree very much) to 5 (agree very much).

- |     |                                                                                       |   |   |   |   |   |
|-----|---------------------------------------------------------------------------------------|---|---|---|---|---|
| 21. | The input fields of sCET help me in knowing what data to enter.                       | 1 | 2 | 3 | 4 | 5 |
| 22. | I dislike it that I cannot enter everything I want.                                   | 1 | 2 | 3 | 4 | 5 |
| 23. | The input fields of sCET prevent me from entering irrelevant data.                    | 1 | 2 | 3 | 4 | 5 |
| 24. | I sometimes leave fields empty because they are not relevant.                         | 1 | 2 | 3 | 4 | 5 |
| 25. | I think with using sCET I can validate the claims I have defined in a structured way. | 1 | 2 | 3 | 4 | 5 |
| 26. | Defining adequate claims takes too much time.                                         | 1 | 2 | 3 | 4 | 5 |
| 27. | With sCET I can deliver results quickly.                                              | 1 | 2 | 3 | 4 | 5 |
| 28. | I didn't really learn the sCE methodology that well when using sCET.                  | 1 | 2 | 3 | 4 | 5 |
| 29. | I disliked using a new tool                                                           | 1 | 2 | 3 | 4 | 5 |

### sCET reusability

Please indicate how much you agree with the following statements, by encircling number of a scale from 1 (disagree very much) to 5 (agree very much).

- |     |                                                                      |   |   |   |   |   |
|-----|----------------------------------------------------------------------|---|---|---|---|---|
| 30. | I think that I can reuse some of the data in sCET in other projects. | 1 | 2 | 3 | 4 | 5 |
| 31. | sCET will help me in finding results of old projects.                | 1 | 2 | 3 | 4 | 5 |
| 32. | I would like to use sCET in the future.                              | 1 | 2 | 3 | 4 | 5 |

### sCET (with sCE experience)

Please only answer the following questions if you have experience with the sCE methodology and indicate how much you agree with the following statements, by encircling number of a scale from 1 (disagree very much) to 5 (agree very much).

- |     |                                                                                 |   |   |   |   |   |
|-----|---------------------------------------------------------------------------------|---|---|---|---|---|
| 33. | Using sCET to apply the sCE methodology is easier than using textual documents. | 1 | 2 | 3 | 4 | 5 |
| 34. | With sCET I spend less time doing the same.                                     | 1 | 2 | 3 | 4 | 5 |
| 35. | I have a better view on what other people are doing than before.                | 1 | 2 | 3 | 4 | 5 |
| 36. | I think that using sCET will give me better results than before.                | 1 | 2 | 3 | 4 | 5 |
| 37. | Learning sCET was more difficult than I expected.                               | 1 | 2 | 3 | 4 | 5 |
| 38. | Using sCET gave me a better understanding of the sCE methodology.               | 1 | 2 | 3 | 4 | 5 |

### **sCET (without sCE experience)**

Please only answer the following questions if you do not have any experience with the sCE methodology and indicate how much you agree with the following statements, by encircling number of a scale from 1 (disagree very much) to 5 (agree very much).

39. Using sCET gave me a good understanding of the sCE methodology. 1 2 3 4 5
40. Learning the sCE methodology was more difficult than I expected. 1 2 3 4 5
41. Learning sCET was more difficult than I expected. 1 2 3 4 5

### **General remarks**

Please answer the following questions in your own words:

42. What was your overall experience with sCET?
43. Do you have any suggestions for sCET?
44. Do you have any other remarks?

## Appendix D

### sCET Questionnaire Results

Question	Mean	Median	Range	Question	Mean	Median	Range
1	4.4	4	1	23	3.2	3	2
2	2.4	3	2	24	3.6	4	1
3	3.4	3	2	25	3.0	3	2
4	3.2	3	1	26	3.6	4	1
5	3.0	3	0	27	2.2	2	1
6	2.6	2	2	28	2.8	3	2
7	2.8	3	1	29	1.8	1	3
8	1.8	2	2	30	1.6	2	1
11	3.0	3	0	31	2.4	3	2
12	3.4	3	1	32	3.0	3	0
13	3.4	3	1	33	4.0	4	0
14	3.2	3	2	34	4.0	4	0
15	3.0	3	3	35	4.0	4	0
16	4.2	4	1	36	4.0	4	0
17	4.4	4	1	37	4.0	4	0
18	1.8	3	2	38	5.0	5	0
19	2.0	3	2	39	4.0	4	2
20	3.5	3.5	3	40	2.25	2	3
21	2.4	2	1	41	1.75	2	1
22	3.0	2	3				

Table D.1: sCET questionnaire results

## Appendix E

### sCET BBCode Reference

BBCode	description	Effect
[b]Some Text[/b]	Bold Text	<b>Some Text</b>
[i]Some Text[/i]	Italic Text	<i>Some Text</i>
[u]Some Text[/u]	Underline Text	<u>Some Text</u>
[cool]Some Text[/cool]	Highlighted Text	<b>SOME TEXT</b>
[indent]Some Text[/indent]	Indent Text	Some Text
[lyrics]Some Text[/lyrics]	Speech or Lyrics Text	<i>Some Text</i>
[smallcaps]Some Text[/smallcaps]	Small Caps Decoration	SOME TEXT
[big]Some Text[/big]	Bigger Text	<b>Some Text</b>
[small]Some Text[/small]	Smaller Text	Some Text
[tt]Some Text[/tt]	Monospaced Text (Teletype Output)	Some Text
Some [sub]Text[/sub]	Subscript Text	Some <sub>Text</sub>
Some [sup]Text[/sup]	Superscript Text	Some <sup>Text</sup>
[url]http://www.tno.nl[/url]	Hyperlinks	<a href="http://www.tno.nl">http://www.tno.nl</a>
[url=http://www.tno.nl]TNO website[/url]	Hyperlinks	<a href="http://www.tno.nl">TNO website</a>

Figure E.1: sCET BBCode tags (1/2)


BBCode	description	Effect
[img]http://upload.wikimedia.org/wikipedia/commons/thumb/7/7c/Go-home.svg/100px-Go-home.svg.png[/img]	Images	
[email]e@mail.com[/email]	Email Links	<a href="mailto:e@mail.com">e@email.com</a>
[email=e@mail.com]Demo[/email]	Email Links	<a href="#">Demo</a>
[font=Verdana]Some Verdana Text[/font]	Text Font	Some Verdana Text
[color=red]Some Red Text[/color]	Text Color	Some Red Text
[color=#FF0000]Some Red Text[/color]	Text Color	Some Red Text
[php]some php code[/ php]	PHP Highlight	<pre>&lt;?php some php code ?&gt;</pre>
[code]some html or similar code[/code]	Code Type	<pre>some html or similar cc</pre>
[list]Some List Property[/list]	List	<ul style="list-style-type: none"> <li>■ Some</li> <li>■ List</li> <li>■ Property</li> </ul>
[list=dec]Some_ Ordered Text[/list]	List, with Order (Decimal numbers)	<ol style="list-style-type: none"> <li>1. Some</li> <li>2. Ordered</li> <li>3. Text</li> </ol>

Figure E.2: sCET BBCode tags (2/2)

Short tags	Description	Effect
[bull /]	Bull	•
[copyright /]	Copyright	©
[registered /]	Registered	®
[tm /]	Trademark	™
#TODO	To Do	#TODO
#TBD	To be defined	#TBD

Figure E.3: sCET BBCode short tags