Creating and Evaluating a Framework for HCI-aspects of Design in Use Case Driven Systems Development at Land Systems Hägglunds

Andreas Anundi & Anders Holm

November 16, 2005
Master's Thesis in Computing Science, 2*20 credits
Supervisor at Land Systems Hägglunds: Anders Mannelqvist
Supervisor at CS-UmU: Magnus Eriksson
Examiner: Per Lindström

UMEÅ UNIVERSITY
DEPARTMENT OF COMPUTING SCIENCE
SE-901 87 UMEÅ
SWEDEN

Abstract

HCI-related issues in systems development are important so that focus is not only on functional requirements, but also on user-centered issues as well. In this paper a framework for HCI-related use case driven design and recording of design rationale in systems development at Land Systems Hägglunds is proposed. The framework is based on methods within the areas of Task Analysis and Design Space Analysis. A case study is performed using the framework on a subsystem of the grenade launcher turret AMOS called MIS. The results of the evaluation show flaws in the framework and points out shortcomings in the use cases when used as input for the framework, but also showed that Design Space Analysis is very suitable for the recording of design rationale.

Contents

1 Introduction **1**

2 Problem and Goal Description **3**
2.1 Purposes and Goals . . . 3
2.2 Methods . . . 3
2.2.1 Preparations . . . 3
2.2.2 In-depth Study . . . 4
2.2.3 Designing the Framework . . . 4
2.2.4 Evaluating the Framework in a Case Study . . . 5

3 In-depth Study of Task Analysis and Design Space Analysis **7**
3.1 Task Analysis . . . 8
3.1.1 Background . . . 8
3.1.2 Task Analysis in Detail . . . 9
3.1.3 Hierarchical Task Analysis . . . 10
3.1.4 GOMS . . . 13
3.1.5 Cognitive Task Analysis . . . 15
3.1.6 Sub-Goal Template . . . 18
3.1.7 Conclusions . . . 26
3.2 Design Space Analysis . . . 28
3.2.1 Design Rationale in General . . . 28
3.2.2 Background . . . 29
3.2.3 DSA in Detail . . . 29
3.2.4 Questions . . . 34
3.2.5 Options . . . 36
3.2.6 Criteria . . . 36
3.2.7 Conclusions . . . 38

4 A Framework for HCI-design in Systems Development at Land Systems Hägglunds **41**
4.1 Introduction . . . 42
4.2 The framework in detail . . . 42

4.2.1 System Knowledge 44
4.2.2 Task Analysis: SGT 44
4.2.3 Design Space Analysis 48
4.3 Summary – Using the Framework 52

5 Evaluating the Framework in a Case Study 53
5.1 Study Context 54
5.2 Methods 54
5.3 Applying the Framework 56
5.3.1 Acquiring System Knowledge 56
5.3.2 Creating a Hierarchical Structure of Tasks 56
5.3.3 Classifying Sub-tasks as SGTs 58
5.3.4 Combining User-centered Knowledge and System Knowledge 58
5.3.5 Brainstorming and Discussions Concerning the Design 58
5.3.6 Creating the Design Space Representation 59
5.4 Generating a Design Space Using DREAM 59
5.5 Results and Observations 62
5.6 Summary of Results 65

6 Accomplishments and Conclusions 69

7 Future Work 71

8 Acknowledgments 73

References 75

List of Figures

3.1 Schematic description of the course of action of the SGT-method [24] 21
3.2 The QOC-notation (Bellotti and MacLean, [6]) 30

4.1 A schematic overview of the framework. 43
4.2 The waterfall characteristics of the framework. 43
4.3 Flowchart of the Task Analysis method for information requirements specification. 45
4.4 A schematic overview of the work flow in DSA. 48

5.1 The goal/task hierarchy based on use cases and scenarios 57
5.2 Screenshot of the DREAM application 59
5.3 The use of factors in DREAM. 60
5.4 The use of arguments in DREAM. 61
5.5 Representation of the main problem. 65

List of Tables

3.1 The SGT-scheme 24
3.2 SGT-sequence notation [10] 25
3.3 Styleguide for creating the QOC-representation [28]. 32
3.4 Characteristics of the QOC-notation (MacLean, Young, Bellotti and Moran [20]. 35
3.5 The sequential model of DSA (MacLean, Bellotti and Shum [19]) 38
3.6 The declarative model of DSA (MacLean, Bellotti and Shum [19]) 39

4.1 Standard representations of the hierarchical task decomposition. 46
4.2 Summary of the workflow in the framework 52

Chapter 1

Introduction

The functional requirements stated in a specification of a product describe what the system must be able to do. Customers also often have usability requirements, whether they are implicit or not. In systems used in, for example, military or nuclear power plants, the usability requirements may be more salient than in other contexts. In order to fulfil the usability requirements, informational requirements must be posed on the system. These informational requirements state what a user must know, for example how a task is performed, or where certain menus are placed. Both the functional and informational requirements form the basis for making design choices, that lead to an artefact meeting the specified requirements.

When designing systems, it is important to document and motivate design choices, the so called design rationale. According to MacLean et al [21], the final product of a design process should not only be the artefact itself, but also design rationale, motivating why the design is the way it is, and how it could have been otherwise. This is to say that design rationale should be a co-product of the design process, and not just a recording of the design process. By having this explicit structured design rationale, they also point out, the possibilities for ending up with a good design will increase.

In system development, it is good to have a formal and structured design process within the development process, that captures the recording of design rationale as well as fulfils the requirements, both functional and informational, stated in the product specification and those that arise during development [13].

Land Systems Hägglunds is a defense and all terrain vehicles manufacturer. The company also develops systems for the components in these vehicles. The system development at Land Systems Hägglunds has gone through a restructuring towards working with a company specific, use case-driven method called FAR (Functional Architecture by use case Realizations). The systems developed by the company are very long-lived systems with life times of 30 years or more. The company creates systems that are suited for the customer and their needs and requirements. Often the systems are slight alterations of earlier versions . These factors make it important to have design rationale, making it easy to motivate and explain how and why the design is the way it is when having design deliberations with new customers.

The goal of this paper is to develop a framework for design in the system development at Land Systems Hägglunds, that focus on the interaction issues (so called Human Computer Interaction-

related issues) between users and the system, but also include the recording of design rationale.

The company works with a scenario based systems development and expressed a desire for a framework with concerns for HCI-aspects in design of systems. The choice of HCI-methods for the framework are two common and relatively well-known techniques for developing and analyzing HCI-aspects of a system. The two methods Task Analysis and Design Space Analysis were proposed by Land Systems Hägglunds for making up the foundation in the framework, to investigate their suitability in the current development of systems.

The framework is created based on the benefits and strengths from each of the two methods. A case study is performed to evaluate the frameworks feasibility and suitability in systems development at Land Systems Hägglunds.

Chapter 2

Problem and Goal Description

This chapter defines the goals and purposes of the paper, and describes the methods used for accomplishing the goals.

2.1 Purposes and Goals

The purposes and goals of this paper are:

- to develop a framework for HCI-related issues in design processes in use case driven systems development
- to evaluate the framework in a case study, to evaluate the frameworks feasibility in the systems development processes in the company.

2.2 Methods

This section describes the methods used for developing the framework, and for the thesis as a whole, beginning with the preparations, continuing with a brief description of the in-depth studies and ending with the design of the framework and the evaluation of the same.

2.2.1 Preparations

In the beginning, it was important to get a feel for how system development at Land System Hägglunds was done, what kind of products are developed and how they work. Therefore, the first task was to gather information, and to learn about the company and the system development at Land Systems Hägglunds. This was done in a number of ways, such as talking to the staff, studying product specifications, participating in meetings with system developers and end users, as well as observing a system in action, while visiting a military training facility.

The information gathering is actually not only an initial state, but is quite understandable an ongoing process. It has been done throughout the work, as more and more information was discovered and needed.

2.2.2 In-depth Study

When studying the literature surrounding the areas of HCI-issues, the focus lay on areas of Task Analysis and Design Space Analysis. Being the methods of choice for the company, and their wide areas of use, no other areas were investigated. The source of information in the in-depth study was books and articles in the area found when searching information about the methods, and recommended by the company.

When searching for related work, a paper by Martijn der Welie [31] was found. It involves Task Analysis, Design Space Analysis and the use of scenarios. However, it has a different approach and the resulting design method/framework has a different scope.

Searches were made to find tools for use when creating the methods resulting representations.

2.2.3 Designing the Framework

When the development of the framework was in an early stage, information relevant for the system development was gathered and examined together with the conclusions from the in-depth studies to find out how to use the information and conclusions to get a suitable way of working with design issues. The conclusions of the in-depth studies were investigated and the discovered strengths and weaknesses were examined. The strengths of Task Analysis include giving an overview of goals and tasks in systems, and finding and focusing on the users and their constraints and limitations. Design Space Analysis gives support for creating design spaces and recording design rationale.

The methods were studied in the context of the system development, i.e. using use cases as input, which resulted in proposals for using the two methods together in the system development with focus on HCI-related issues.

The information concerning the systems being produced was gathered from FAD-documents, functional requirements, manuals and from meetings with users and system developers (see Chapter 5.1 for a description of these information sources). The FAD consists of documents that describe the system in scenarios, by describing them in use cases. By studying these documents, it is decided that these documents can be a base for finding out what tasks and goals are involved in the system. This makes it possible to have traceable flow of information and since the FAD is central in the system development, it also gives support for understanding and communication between the members in different areas of development. The functional requirements are derived from the set of functions, described as a tree structure. However functions that the system should have and the functional requirements are not the same, they can be viewed as such, to get a feel of what the system is supposed to provide with in terms of functionality.

The manuals for the system are used to learn how the systems are used, also described in an almost functional way. In meetings with end users and system developers, it is easier to detect problems and flaws in the system that are more directed towards the user, and to get a feel of what the users find hard to manage and operate in the system, usually prototypes.

The use of the framework in an ideal sense assumes and postulates that the information that makes up the system knowledge is based on

- goal driven use case scenarios [1].
- functional requirements [17].
- other system information and specifications (e.g. manuals, meeting protocols, user comments etc.).

Not only are the use cases a precondition, they need to be goal driven in order to fit into the proposed framework. The use cases should also be descriptions of the main tasks/goals in the system, with a complete description of the sub tasks necessary to perform in order to achieve the goal state.

The functional requirements and the other system information are used to understand the system as a whole.

2.2.4 Evaluating the Framework in a Case Study

The AMOS-system (**A**dvanced **MO**rtar **S**ystem) is a vehicle-mounted grenade launcher system produced by Land Systems Hägglunds. Today, the system is controlled through a graphical user interface called WCS (Weapon Control System). The WCS is a prototype, and the next step in the development of AMOS is to produce a new prototype for the user interface. The new prototype will be called MIS (Mortar Information System).

The framework was applied to MIS. The input to the case study was use cases describing tasks/scenarios in MIS. The results from the Task Analysis-phase of the framework provided a set of information requirements for the design. These information requirements, along with functional requirements from other sources, where then used in the Design Space Analysis, to create a design space, consisting of design rationale and design choices.

Chapter 3

In-depth Study of Task Analysis and Design Space Analysis

In this chapter, the two analysis methods Task Analysis [10] and Design Space Analysis [21] are examined, to find their strengths and weaknesses, and in what way they could be combined, to address HCI-related design issues in system development.

The first part of this chapter focuses on Task Analysis, giving the background and history of the area. The next sections describe the main branches within the Task Analysis field in detail. The section about Task Analysis ends with conclusions about the pros and cons of the method.

The second part of this chapter is about Design Space Analysis, starting with giving some background of the area, and describing some general terms important in the method. After that comes a detailed description of the method and the elements in it. The part about Design Space Analysis ends with conclusions about the method, and also with a brief description of an application used for generating and representing design spaces.

3.1 Task Analysis

The interest of trying to match tasks that are being performed against human capabilities has grown steadily in the last 100 years. This will hopefully make human work more productive and safe in the future [26]. Performing this matching, requires deep understanding of both the human who performs the task and the domain in which the task is being performed. This endeavor has lead to hundreds of methods that go under the name Task Analysis [10].

3.1.1 Background

The first attempt to analyze tasks on the basis of science was made already in early 1900 in the work of F. W. Taylor and John and Lillian Gilbreth [2]. They were trying to optimize work in terms of time and effort which were critical aspects in this time when the industrialization had begun. They realized that it was not only the materials and tools that were important in the production. The actual workers and in what way they used these resources were just as important for the profit in the production. The Gilbreths, a constructor and a psychologist, thought that a method was needed so that work could be described in a better way. To be able to describe this, they used behavior entities which were named ”Therbligs” after their creators. This set of behavior elements could be used to describe a wide range of activities such as bricklaying and laundering. With this method, the number of steps (movements) the task consisted of could be reduced, making the work more efficient and easier to do.

In the time before and during the First World War, scientists started to perform studies on how fatigue, work situation and the surroundings affected people who perform tasks [2]. Before this, the production development in the analysis phase always came first and the psychological aspects in the performed task were secondary. The use of psychological methods in the analysis became more and more useful, especially when the aviation industry started to bloom, where these aspects became an important factor.

The development that took place in the 1950s laid the foundation of the modern cognitive psychology. That was when they could describe human operators as a transmitter of information with a fixed maximum capacity of about 10-15 bits/second. W. E Hick and Paul Fitts then showed that it is really the information processing and not the physical procedure that decides how long time it takes to perform a task [11]. The fact that humans have limited capacity on both sensory information and decision making created the new concept ”Mental Workload” which is often used in modern task analysis. One of the first organizations that first used engineering psychologists in system development was in fact the military. In this period of time the behaviourism was the leading part in psychology and these new cognitive concepts were not accepted by the behaviorists. Despite this antagonism, the cognitive part of psychology got more and more popular.

In the end of the 1960s, Annett and Duncan came up with a method known as Hierarchical Task Analysis (HTA) [3]. The emphasis of this method is to do a systematic decomposition of tasks into sub-tasks in more than one level. The central unit of this analysis of tasks is the operations, statements that briefly describe the operators current objective. To know when to stop the decomposition a pragmatic stop rule is needed. The purpose of HTA is to be able to find where potential errors and critical faults may emerge in complex tasks. These faults can be based either in the physical action or in the information processing and the system must be redesigned to prevent these faults from happening again. The hierarchical way of looking at

tasks fit well into how human behavior is considered. Human behavior is a fundamentally goal driven process which can be seen as a hierarchical goal structure, in which the primary goals are fulfilled through sub-goals.

When computers and their programs became more complex in the 1970s, new methods were needed to be able to examine and measure the tasks that they perform. This became "Software engineering" [29]. With the use of this, more formal examinations could be performed on such things as logical consistency and the efficiency on system software. There are counterparts to Software Engineering in Human Computer Interaction. One of these is GOMS (Goals, Operators, Methods and Selection rules) [15] which has a lot in common with HTA. A goal in GOMS and HTA consists of a number of states that has to be reached. Operators are equal to the physical and psychological actions that are being executed to reach the goal. Within GOMS there is a control structure so that the behaviors get a meaning. This structure consists of methods and selection rules. Methods are collections of elementary operators and the selection rules specify when an operator should be used.

The term Cognitive Task Analysis (CTA) first appeared in articles in the end of the 70s [26]. The aim with this method is to use concepts from cognitive psychology when analyzing complex tasks. By only observing the user, it does not mean that the whole picture of what the user does (and why he or she does it) is captured. Even when interviewing expert users it can be hard to notice the underlying cognitive structure. They can have problems explaining how they perform tasks. To be able to predict potential errors, a detailed and correct cognitive model must be created that specifies the necessary procedural and declarative knowledge a user must have to be able to successfully perform a task. To get the procedural knowledge and the skill of expert users the use of subtle methods may be needed. This new way of thinking has increased the awareness about the cognitive activities that underlie even pretty simple tasks.

In these modern times, tasks become more and more automated to facilitate humans in their work. When observing these tasks, focus tend to switch from the physical aspects over to the cognitive parts, that is not directly observable. For example, the physical pressing of a button in a military command center is not that interesting compared to the mental process that take place when a decision to execute such an action have to take place. These mental processes give the physical actions a meaning so they can be understood and explained.

3.1.2 Task Analysis in Detail

A task can be defined as an activity needed to fulfill a goal using certain equipment and resources [27]. Diaper [10] states that Task Analysis is a term that consists of several different methods. The purpose and goals of the methods are to

- receive descriptions and representations about the tasks that users perform when interacting with a system,
- to find were potential problems could arise in performing the tasks, and to
- measure the usability of a system and how easy it is to learn.

The main activity in achieving the goals is to break down a primary task into sub-parts and revise these parts critically. It is important to know which resources could be used and which constraints that could exist in the surroundings/domain where the task is being carried out. If task analysis is being performed, there must be some kind of information gathering. This can

be done by talking to and interviewing users of the system, observing of usage of the system or by reading manuals.

Task analysis is often used when analysing existing systems, rather when creating new systems. The emphasis in TA is to analyze what humans do and why they do it. The purpose of Task Analysis is to fully understand users activity in the context of the complete Human-Machine (-Computer) system, either in the present or the future.

Diaper[10] claims that

> "*Task Analysis is potentially the most powerful method available to those working in HCI and has applications at all stages of system development, from early requirements specification through to final system evaluation.*"

But there are people who say that Task Analysis can be hard to get a grip of, that there are confusion in the TA-society and that, as Diaper [10] puts it, *"designers and human factors specialists fumble with the concept"*.

3.1.3 Hierarchical Task Analysis

Hierarchical Task Analysis (HTA) was developed at the Hull University in the late 60s by John Annett and Keith Duncan [4]. This was done because there was a need for a rational foundation to stand on to be able to describe and fully understand what a worker has to do when working with complex non-repetitive tasks, e.g. process supervision tasks. Before HTA, people used simple notions as "choose", "grip" and so on to describe how simple repetitive tasks were performed manually. But as the tasks became more and more complex and the cognitive proportions of the tasks grew, these simple descriptions of the work were not enough. To better describe the mental workload connected to the task more advanced notations were needed. These tasks can for example be supervision or controlling or decision making tasks.

HTA did not have the same approach to classification of tasks as the older methods [3]. Instead of basing it on Skinners behavioristic approach, a more functional way to classify the underlying terms of tasks was used. In HTA these underlying terms are operations (brief statements of the operators current objectives). One important thing to think of when an analysis of a task is to be performed is to first define what goals that exist in the task before considering what actions that could be used to perform the task. By analyzing tasks in this way, Annett [3] states that the taxonomies of the behaviorism disappear. Now the attention can be put where it really matters, that is, whether a task has been successfully accomplished or not, and if not why. Hierarchical Task Analysis is seen as a systematic method for investigating problems of human performance of tasks [27].

HTA differs radically from earlier Task Analysis methods. For example, HTA does not start with a list of activities. Instead there is an endeavour to locate the goals in the task that is going to be analyzed. The goal of HTA is to provide a method for a more functional analysis rather than just a description of behavior [3]. Annette states that complex systems that for example consist of both humans and computers are designed with goals in mind, and to be able to understand how goals in these systems are being achieved, or at least trying to be achieved, is the primary goal of this analysis method.

HTA is one of the most well-known methods among human factors specialists. According to Annett, it is not so much of a strict procedure, than a generic approach to find out where and why humans have problems within complex, goal oriented systems involving computers [3]. HTA is a method for structuring tasks involved in achieving a certain goal, in a hierarchical structure. One of the main aspects with HTA is also to gather information and to do a representation of the task that is being analyzed.

A goal can best be described as a specific state of affairs, a goal-state. Annett [3] states that a goal can be either active or latent. Latent goals can only be reached under certain circumstances. Goals can be complex, meaning that they can be defined by more than one event or values consisting of multiple variables. When a set of goals has been identified, the analysis should specify their sub-goals through decomposition. Within HTA there are two types of decompositions. The first tries to identify those goal states specified by multiple criteria. The second one comprises the identification of all the sub-goals that can be found in all routes that leads to fulfilling the overall goal. The goals can then be unpacked successively and through this unravel the nested hierarchy of goals and their sub-goals. This process is called redescription [27].

Operations

An operation is a statement that briefly describes the operators current objective. This can be done at different levels giving the analysts the benefit of being able to adjust the level of description at different parts of the task under analysis, pointed out by Annett [3]. An operation is stated in a neutral way towards its solution, nothing is said about how to make the operation to a success. The operational behavior that enables the operations to be carried out is goal directed. The operator selects an action from the available actions in the interface of the task in order to attain the goal state.

An operation can be seen as the foundation of the analysis which is performed. The operation is defined by its goals. An operation is further specified at three different levels:

- Input
  - ◇ Circumstances surrounding the operation.
  - ◇ Received information to act upon.
- Action
  - ◇ The activities that help to attain the goal
  - ◇ An instruction what to do according to the given information under specified circumstances.
- Feedback
  - ◇ How to notice that the goal has been achieved.

Another name for operations is in fact IAF-units (Input-Action-Feedback) described in Shepherd [27]. Operations can also be decomposed into a nested hierarchy just as task and goals can be. Super ordinate operations consist of sub-operations.

Plans

Plans are important parts of HTA that capture things that can not be found merely in the operations themselves.

When an operation has been decomposed into sub-operations with a hierarchical structure, a system to organize these is needed. The organizational system should indicate in what order the component operations are carried out to complete the super ordinate operations.

This specification of rules concerning the order in which sub-operations should be executed is called a ”plan”. Plans can be of different types of sequences:

- Fixed order
  - ◇ Operations are being processed in a fixed order, e.g. ”do this, then this and then this”.
- Selective Rule
  - ◇ What operation to perform is chosen according to the circumstances within the surroundings, e.g. ”If x is the case, do this. If y, do that”.
- Parallel order
  - ◇ Two or more operations are pursued at the same time (parallel). The super ordinate goal can not be reached unless a number of its sub-goals are attained at the same time.

The fixed and Selective plans imply that the operator has knowledge of the context. This can be simple procedural knowledge or extensive declarative knowledge. This is an essential part of Task Analysis which is referred to as *Cognitive Task Analysis*.

The third type of plan is also known as a time-sharing or dual task plan, and has also several cognitive implications, mainly for the division of attention.

Stopping Rules

The decomposition of goal hierarchies and the redescription of operations and their sub operations could go on forever, if the decomposition stop criteria were not found. These stopping rules decide when a certain level of detail has been reached and the redescription can stop.

An ultimate stop rule would be: ”Stop when all information needed to attain the goal of the analysis has been gathered”. But since HTA often is about locating problems in the human performance, the most common used stopping rule is: ”*Stop when **probability of fault** x **cost of the fault** is acceptable*”. This rule is more known as the P x C criteria. The most obvious reason to stop is when the source of the fault has been located and the analyst can propose a possible cure for the problem. This can be done with a new system design, or by changing how the system is controlled or by better training of the operators of the system.

3.1.4 GOMS

When analysts are interested in capturing and characterizing the procedural knowledge of users of the system that are being analyzed, a well established analyze method within Task Analysis is commonly used. This method decomposes a task into Goals, Operators, Methods and Selection Rules and is therefore called GOMS [15].

- Goals
  - ◇ Certain states that the user wants to reach.
- Operators
  - ◇ The cognitive and physical processes being executed in order to reach the Goals. The difference between a goal and an operator is that a goal is reached and an operator is executed.
- Methods
  - ◇ Stored procedures used to reach the goals. A method is an exact sequence of steps that are needed in a task, e.g. move the mouse-pointer, press left button.
- Selection Rules
  - ◇ Used to choose what method is suitable to use when there are more than one method to choose from at a stage in the task.

GOMS contribute with a quantitative way to predict the human learning and performance within designing interfaces for describing in a qualitative way how users will use computers to perform certain tasks. This method was developed in the beginning of the 80s by Stu Card, Tom Moran and Alan Newell [8], to be able to model users knowledge and the cognitive processes that are used when the users interact with a system. This is a so called predictive method, which means that users performance when performing tasks can be measured without any testing of the actual users [15].

To fully understand human activity is in reality a goal within the psychological community and other social sciences. When Task Analysis is performed for system design, it is often done in an informal way. The analyst should know in which situation and context users find themselves carrying out tasks. This helps the analyst to consider the limitations on time and other resources when designing the system. Even though Task Analysis is quite informal in its nature, there have emerged propositions for more formal and quasi-formal methods. GOMS is one of these formal representations for predicting human performance well enough to replace almost all empirical user testing that is needed to reach a level of design of a system that is both functional and usable [15].

The GOMS-model is made out of descriptions of the Methods that are needed to reach the specified Goals. A Method is a sequence of steps consisting of Operators that a user executes. These methods can have a hierarchical structure similar to the Hierarchical Task Analysis. If there is more than one method that can be used to reach a certain goal, Selection Rules are used to determine the most suitable method to use in the context where the task is being performed.

One of the simplest models of GOMS is called ”The Keystroke Model” which was first described by Card, Moran and Newell in the 1980s [7]. In this model the time it takes to carry out a task is predicted. This is done through adding the estimated times for the elementary actions that are needed to perform the task.

A much more advanced model of GOMS is the CPM-GOMS developed by Gray, John and Atwood in 1993 [14]. The essence of this more advanced model is to try to find the critical path that predicts the execution time of a task.

Strengths and Weaknesses

One of the greatest benefits of GOMS is, according to Kieras [15] that it is easy to perform comparative analyses on many different systems and interfaces. What limits the use of these models are the limited range. GOMS models are supposed to be used to predict the performance of subject-matter experts (SMEs) and it does not allow any fault modeling. This makes it harder to predict how a ”average” user would manage a system. There are also many unpredictable factors to take into consideration, e.g. mental workload, effective learning and other social aspects. A dilemma when using GOMS is consequently that these predictive models only can make predictions that are *easy to predict*, while humans (often) are quite unpredictable in their behavior and the way they perform tasks, Kieras states.

It is important to know what can be done and what can not be done with the GOMS-models. Each one has it own strengths and limitations, but Kieras [15] describes some traits they all have in common:

- GOMS is often preceded by a basic Task Analysis
  To be able to apply the GOMS-technique, the analyst first has to decide what goals the user tries to achieve.
- GOMS can only represent the procedural aspects of a task
  The procedural aspects of usability are the exact steps users take. Analysts using GOMS-models are free to decide how many procedures that should be conducted by the user and what level of consistency and efficiency these procedures should have. Usability of systems is often defined by the simplicity and efficiency of its procedures. This is why GOMS is a good guide for design of interfaces.
- GOMS models are practical and effective
  There is a belief saying that the making and using of GOMS models takes too much time to be practical. But Kieras states that the work of John and Kieras [15] has shown that many of the models in the GOMS family, when applied to practical situations have managed really well both in cost and time efficiency.

Kieras [15] points out that a GOMS analysis can be done in many different ways and in different phases of the system development:

- Existing systems
  To create and use a GOMS model for analysis of an existing system is the easiest way to do it, because much of the information needed in the analysis can be reached from the finished system. The main problem for the analyst is to determine whether or not the users actually do what the designers intended, which seldom is true.
- After the design specification
  A GOMS-analysis does not have to be done on an already existing system. The only thing necessary is that it must be possible to specify the components for a GOMS model. If the design has been specified at an adequate level, the analyst will be able to identify the users goals and describe suited methods to use, just as it would be done in an existing system.

- During development of the design
  Instead of analyzing an existing system or a specified one, a GOMS model can be elaborated at the same time as the interface is being designed. Starting by looking at the top-level tasks and defining the top-level methods for these goals and then look at the lower-level goals and their methods, decisions can be made about the design of the user interface directly in the correct context where it is supposed to be used. Bad design decisions can lead directly to the methods that are inconsistent and complex. The designer then can directly consider other possible, perhaps better alternatives.

3.1.5 Cognitive Task Analysis

Much of the work that every day people perform these days focus on the mental parts of the task. So when trying to simplify this kind of work it is the cognitive aspects that are most interesting to look at. But these could sometimes be quite hard to see and are not very accessible for investigation [9].

This branch of Task Analysis started in the 1960s and has broadened the awareness of the cognitive activities that exist even in quite simple tasks. The term Cognitive Task Analysis was first used to describe another area of applied psychology. This is in other words a rather new way of thinking and the interest has grown steadily, especially within the military community, e.g NATO. NATO conducted a group study to catch the essence and the most prominent properties of CTA. The purpose of the study was to make this kind of analysis easier to perform so the analysts would not need a doctor's degree in the area. The book "Cognitive Task Analysis" [26] describes the fundamental aspects and parts of CTA, many of which being results from the NATO study.

CTA is an extension of the wider concept of Task Analysis. A main aspect of this extension is to elicit information about knowledge, process of thought and goal structures that could exist in systems where tasks are conducted [9]. There has been an antagonism towards this newer kind of thinking of mental modeling, especially in the USA.

The concept of CTA is a broad area consisting of tools and techniques for describing the knowledge and strategies required for task performance. A common factor of these techniques is seeking to model mental activity of a task operator. The increasing usage of design of systems and the fact that most of the design teams are fairly small has led to a demand of more flexible techniques.

Chipman et. al [9] points out that when this kind of analysis is performed it is important to think of what kind of tasks that are being analyzed, why the analysis is being performed and what resources are available. The first thing that should be done is to investigate the task under consideration, trying to find the critical parts that need to be examined from a cognitive view. Depending on how much information and material there are available concerning the task, the first step for the analysts could be to read through this material to get more familiar with the task at hand and get better knowledge of the vocabulary in its context.

Another way to go about with CTA is to conduct unstructured interviews with subject-matter experts and the users of the system, together with the use of questionnaires [9]. A study of this kind can lead to finding of parts in the task that are critical to perform. One of the main goals with these expert interviews is to identify the knowledge involved within the task that is being

analyzed, the knowledge representation that has to be used. When the knowledge representation has been identified, a knowledge elicitation technique has to be determined. This technique is used for incorporation of knowledge of persons that are interviewed.

All persons are not suitable to become a cognitive task analyst according to Chipman et. al [9], who also states that to perform a good CTA, the analyst must have great social skills to be able to communicate with the hand picked so called experts in a successive way. The analyst must manage different kinds of people with different social, cultural and economical backgrounds. They have to be intelligent and fast learners since a lot has to be learned about the task in a short period of time to be able to perform an effective analysis [9].

When CTA is performed, it is important to specify all the users of the system and what they are doing. In CTA, systems are viewed on the basis of their users and identification of aspects of the design that demands much of the users cognitive resources. This makes it easier for the designer to concentrate and focus on the parts of the system that users seem to have trouble learning and areas where faults most probably would occur. The designer then tries to locate ambiguities in the system to iterate new design specifications that are more cognitively ”straight forward”. The goal for the designer is to design so that users of the system do not have to focus on the actual interface and instead focus completely on the task [9].

CTA can also be looked upon in a more formal view. The basis for this formal view on Cognitive Task Analysis is called ”Interacting Cognitive Subsystem” or ICS [5]. This architecture represents a complete sequence of information processing at three levels of cognition:

- Goal Information
  - ◇ Realize what goals have to be achieved.
- Action Specification
  - ◇ Determine how to achieve the goals.
- Action Execution
  - ◇ Execute the actions needed to attain the goals.

This modeling provides an estimation of how easy each phase of the interaction can be carried out and where faults may arise and additional support is needed and how repeated usage of the system will affect the users behavior. This architecture (ICS), described by Barnard and May [5], describes human mental activities in 9 independent and parallel cognitive subsystems [5].

Sensory

- Visual
  - ◇ Information from the eyes, e.g. light and contrast.
- Acoustic
  - ◇ Information from the ears, e.g. rhythm and pitch.
- Body
  - ◇ Information from the body, e.g. arousal and feeling.

Central

- Object
  - ◇ Create mental images, e.g. motion pattern and shapes.
- Morphonolexical
  - ◇ Words and other lexical forms, e.g. commands.
- Propositional
  - ◇ Semantic affinity between different entities e.g. task.
- Implicational
  - ◇ Meaning and understanding, e.g. schematic models.

Effectory

- Articulatory
  - ◇ Tacit learning and speech.
- Limbs
  - ◇ Moving of limbs, eyes etc.

Each subsystem handles the representation it receives and produces another representation which other subsystems subsequently can handle afterwards, e.g. the visual representation created by the eyes transform into an object representation (this is only one step in the chain of representational transformations).

The representations received by a subsystem can be copied and stored in an image record, which works as a local memory for the subsystem. There are a number of different types of records available [5]:

- ATR - Active Task Records
  - ◇ The latest representation that is processed by the subsystem.
  - ◇ Can be reused instantaneous.
- ETR - Experiential Task Records
  - ◇ Representations that have been experienced before.
  - ◇ Can be revived if the subsystem gets a sufficiently similar input representation.
- CTR - Common Task Records
  - ◇ Abstractions of the commonality of a number of ETRs that are formed by their simultaneous revival.
- EPR - Entity Property Records
  - ◇ All input representations put together.
  - ◇ By combining ETRs and CTRs, entities within a subsystems representational domain could get information that is not present in the original input representation.

Within formal Cognitive Task Analysis, focus lay on the information flow between these subsystems and on reducing level of ambiguity. If a structural approach is used for the description of the representations, parts of the design can then be analyzed without a complete specification of the interface, an approximate description is often enough.

The best time to perform a CTA, according to Barnard and May [5], is in the beginning of the design phase, prior to any prototype work, while it is being fleshed out. It is especially valuable

to use CTA if alternate design decisions will be evaluated. Another good thing with CTA is that it is easy to use together with other design practices and can give quick answers to usability questions and because of that, the iterative design cycle will be speeded up by reducing the need for early prototypes and evaluation of the interface.

3.1.6 Sub-Goal Template

Within the design community there has been an increased interest in using user centered methods to specify systems. To increase the usage of this kind of Task Analysis methods, Ormerod [24] states that they have to be integrated with the system development and become a primary part within the design process [23]. Many of the articles written about Task Analysis today starts by stating the problem that system developers overlook the human users when the system is designed in an early phase. Instead focus lay on the functional requirements specification of the system. Ormerod and Shepherd [23] points out that by neglecting in which context and what people that will be using the system and the resources that they have, the usability of the system will probably not be optimal [24].

Many of the methods used today support only the specification of the functional requirements and when much of the system already has been designed the thought of designing the interface between the user and the system comes up. This is, according to Ormerod, too late but better late than as in many cases where then informational requirements is totally overseen. If the information requirements are neglected it will lead to systems that are difficult to operate and can maybe even result in a system not being capable of its intended purpose.

It is important to specify the number of people that will use the system at the same time to know the communication and work distribution in the system. This information could lead to limitations in the design. Finding limitations on design is also one of the main aspects of Task Analysis. Though this is important for the designer to know, Ormerod and Shepherd [24] points out that Task Analysis should not be viewed upon as an optimal and universal tool that tells the designer exactly what to do. The designer has every right to be a little suspicious towards the Task Analysis methods that focus upon HCI-related aspects at the expense of system engineering. HCI-related issues are important aspects of system engineering.

There are issues concerning the users associated with the context where the system will be used. The issues that need to be addressed are such as training, instructions, workload and feedback. Physical issues, such as regulatory and temporal, also need to get attention. The designer has to notice and understand the limitations on both the usage of the system and the human factors that are involved in the interaction with the system. Ormerod and Shepherd [24] states that finding these limitations and factors, should be the main aspect whenever a Task Analysis is being performed in the design of interactive systems.

According to Ormerod [24], the usage of the various Task Analysis methods for design has been sparse until today. This can be explained by stating problems that often come up when Task Analysis is being discussed.

- Task Analysis does not provide both operational and interactive requirements.
- It might inhibit innovative design.
- It is difficult to introduce relatively informal methods in the design process.
- It could be difficult for a non-specialist to perform. Demands a lot of learning overhead.
- It has to be done on an existing system.

Many of the Task Analysis methods and other methods that put the human operator in focus have problems to deliver a usable product for the designer to use. They may help the designer to focus upon the important questions of the system. But they do not give much clues on how to make design decisions. There is also a fear within the design community that Task Analysis methods could generate too much limitations and constraints on the system, which also could inhibit the designer to come up with a detailed design space [24].

According to Ormerod [23] it is within the specification of the information requirements (that users must have to be able to carry out a task) that Task Analysis can be used as a method for design. To be able to specify the information that users need to carry out a certain task, the tasks that users must undertake on the way must be specified. But as stated earlier, the design community has been very slow to begin to use Task Analysis methods and therefore the importance of the specification of the information requirements has been forgotten.

To bridge some of these stated problems a new approach to Task Analysis was presented by Ormerod [23] and a revised by Ormerod and Sheperd [24]. This method is called the Sub-Goal Template (SGT). An outcome of this method is a specification of the required information that users will need to carry out the analyzed task.

Purpose of the SGT-method

The philosophy of this new approach to Task Analysis, the SGT-method, was to find a way to capture how to handle the control of the system and to find out what information is crucial for the users of the system. At the same there was a desire to minimize the jargon and need for learning for the designer, creating a readable, unambiguous and usable output for the designer and the users of the system. The criticism of Task Analysis that it provides the designer with too many limitations for design solutions lead to that Ormerod wanted to leave the control of the design process to the designers of the system [23].

It is usually said that if a Task Analysis is going to be performed, then it must be done on an existing user-system interaction, which Ormerod and Shepherd [24] find incorrect. If analysists always were to analyze only existing user-system interaction, it will be difficult to generate new types of interaction methods, and future designs always will be based on previous versions. If the operations executed in the system could be described in a neutral way, these unnecessary limitations would disappear. It is important that the analyst differentiates between the functional requirements that exist in a task and the method that is used to perform the task in order to find other possibilities on how the task can be performed.

Ormerod and Shepherd [24] concur with the view that Task Analysis should be an iterative

process. This process should consist of information gathering, organization, negotiation and revision [24]. Something that must exist in these iterative processes is rules that say when to stop. These rules contribute with two important features:

1. They tell the analyst when a certain level is reached when there are nothing more interesting to describe associated with the design.
2. They force the analyst to stop the analysis before any description on how the task should be implemented. The neutrality is retained.

To completely separate between Task Analysis and the design is quite hard according to Ormerod, but this is one of the main motives for the construction of the SGT-method.

The SGT-method

Shepherd and Ormerod started to develop the SGT-method in the first half of the 1990s [24]. Their goal was to use a Hierarchical Task Analysis as the basis of a method that could give a complete requirements specification of information needed for users performing the task. Their concern was to create a task analysis method that could work as an integrated approach to specification of interfaces. The main elements of this approach are the hierarchical task decomposition, the SGT-scheme (see Table 3.1), SGT-sequencing elements (see Table 3.2) and finally the provision of information requirements.

The task of specifying the information requirements involves the specification of sending and receiving of information that the operator requires to operate a certain system. The scheme used in the SGT-method provides means for classifying and organizing of stereotypical tasks that users perform in all sorts of environments. The information necessary for conducting these stereotypical tasks is captured throughout the analysis process and when the analysis is finished, an information requirements specification can be handed over to the designers. This analysis is being performed in two steps:

1. Task Decomposition
   - Tasks are broken down into sub-tasks
2. Task Redescription
   - Redescription using the notation of the SGT-method.

The schematic description for the process in the SGT-method can be found in Figure 3.1 (as found in Ormerod and Shepherd [24])

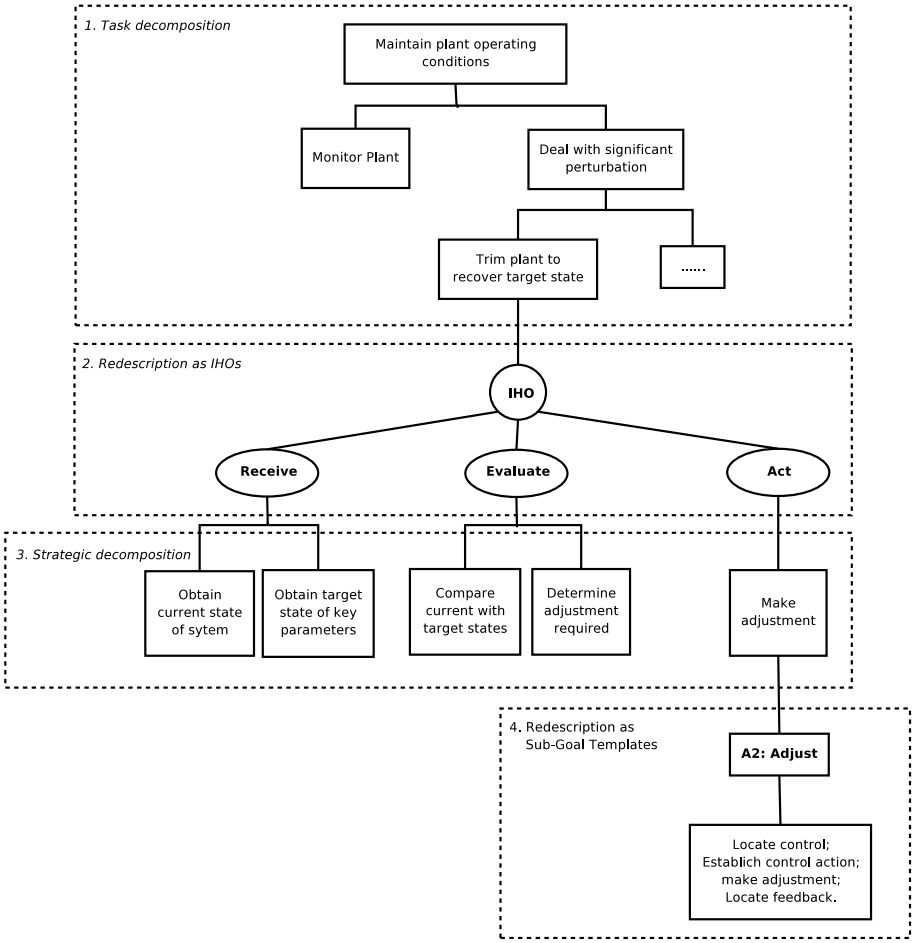


Figure 3.1: Schematic description of the course of action of the SGT-method [24]

Heritage from Hierarchical Task Analysis

A common way to analyze a system is in terms of the goals and sub-goals that exists when users carries out tasks. Hierarchical Task Analysis is a well established method that utilizes this way of thinking (see chapter 3.1.3). Ormerod and Shepherd chose this technique as a basis for the SGT-method. This choice has been made for two reasons:

1. It is a relatively easy approach, and
2. no assumptions are made about the users knowledge structure.

When addressing the issue of interface design, Ormerod and Shepherd [24] wants to create a method that preserves the benefits of HTA concerning thoroughness, flexibility, utility and transparency [24]. When some kind of interface technology is being used, communication between the system and the operators is an essential part. It is often the operator who elicits information about the state of the system in order to decide what to do next and how to gain control over the system so that the required changes could be performed. The choice of which stopping-rule

to use in the SGT-method has to be reviewed, when tasks are decomposed. There should be a distinction between redesign of an existing system and when a new system is developed where the interfacing method yet has to be decided.

When an existing system is analyzed, the decision to change the design of the interface can be based on an already completed Task Analysis. The main objective for the analyst is to establish and understand how the existing interface is being used by its operators. The decision to make changes to the interface is then based upon the risk that exists in using the present interface technology and the need for improved training and other organizational factors. If a redesign seems to be a good idea, the first step is to try to find the place in the hierarchical structure of the main task, where the functional requirements of the interaction are described, before any assumptions of what interface technology to use are made so the neutrality could be preserved. From this point new technology or approaches for the interface design can be devised.

If a new system is being developed without any given interface technology, the analysis should proceed until a level where the functional specifications are being specified. It is at this point where the SGT-method could be utilized as a method for interface design.

The neutrality in terms of design hypothesis is retained within HTA by two features:

1. The analyst retains operational neutrality in the descriptions of the tasks so different kinds of solutions can be chosen.

2. HTA moves from general operational statements towards specific methods where solutions are assumed.

Information Handling Operations

All systems that involve humans in some way will at some time reach a point when information has to be handled so that the system can proceed to another state. If there were not any information to handle, humans would be a useless part of the system. As stated by Ormerod and Shepherd [24] these operations are called Information Handling Operations, or IHOs. A task can be redescribed as an IHO when an information-based intervention is required.

The activity of an IHO can be understood through three different classes [24]:

1. **Receiving information**
   To receive information could be to obtain the system status, receive a communication (e.g. actions to perform adjustments of current operating goals) or to retrieve data from the system. The received information does not have to originate from the actual system; it can be retrieved from another user or even from within the operators themselves.

2. **Evaluating information**
   The evaluation of the information can be some sort of monitoring of the system, planning of activity or maybe managing and overseeing changes to the system.

3. **Acting on information**
   The procedure of acting on information can be to perform some kind of action that changes the state of the system, start a communication event with another user or with the system itself or to record data in the system for further use.

To reach the level where these information handling operations can be found, the analyst starts by looking at the goal of the task being analyzed. This goal is then decomposed into its sub-goals. If these sub-goals could be recognized as IHOs the decomposition should be ended, for now at least. This is done because this is the first time when the neutrality of the design of the task is being questioned. The ways that information can be processed and how to use this information is almost infinite. This decision should not be done in this part of the Task Analysis; this is a strategic decision that should be made together with the end users (clients), system developers and other people that are part of the project.

The analyst can not do any further decompositions before the negotiation has taken place. Finally, when a strategy for achieving an IHO has been identified, the decomposition could continue. But in this phase the decomposition is a little more defined, that is finding and decomposing each Receive, Evaluate and Act components into their element tasks.

Sub-goals and Task Elements

To be able to specify the information requirements within the decomposition of a task, Ormerod and Shepherd states that

- the decomposition has to be rigorous to ensure that the analysis ceases at a suitable level of description,
- the required information associated to each task must be detailed, and that
- standardized description of the emerging tasks is needed.

To unambiguously decide where the decomposition should cease so that the results from the analysis could be handed over to the design team, the lowest level of tasks that operators perform must be specified. This is described in the SGT-scheme (table 3.1). This scheme consists of what could be described as the essence of the SGT-model, namely the Sub-Goal Templates. These templates are a set of standard task elements that capture the tasks that users encounter when using any interface.

The SGT-scheme contributes with a vocabulary and a nomenclature for redescription of the output of a Task Analysis. It also facilitates the process through identification of tasks and a higher level of systematicity to the naming and organization of tasks. When the lowest levels of the Hierarchical Task Analysis are reached, each task is assigned with a so called task element. These elements can subsequently be divided into four different classes of sub-goals: Act, Exchange, Navigate and Monitor. Theses classes have been revised over the years to try to make the method as simple as possible. The Navigate and Monitor classes were added so that all kinds of interactive systems could be analyzed. The task elements within each class are specific variants of the sub-goal. One of the main aspects with these SGT-task-elements is that they act like stopping points for the analysis where the independence and neutrality for design and technique is preserved [24].

Table 3.1: The SGT-scheme

SGT	Task Element	Context	Information Requirements
Act		Perform as a part of a procedure or subsequent to a decision made about changing the system.	Action points and order;Current, alternative and target states; pre-conditions, outcomes, dependencies; halting and recovery indicators
	A1: Activate	Make sub-unit operational - switch from ”off” to ”on”.	Temporal/stage progression, outcome activation level
	A2: Adjust	Regulate the rate of operation of a unit maintaining ”on” state.	Rate of state change
	A3: Deactivate	Make sub-unit non-operational - switch from ”on” to ”off”.	Cessation descriptor
Exchange		To fulfill a recording requirement. To obtain or deliver operating value.	Indication of item to be exchanged; channel for confirmation
	E1: Enter	Record a value in a specified location.	Information range (continuous, discrete)
	E2: Extract	Obtain a value of a specified parameter	Location of record for storage and retrieval; prompt for operator
Navigate		To move to an informational state for exchange, action or monitoring.	System/state structure, current relative location
	N1: Locate	Find the location of a target value or control.	Target information, end location relative to start
	N2: Move	Go to a given location and search it.	Target location, directional descriptor
	N3: Explore	Browse through a set of locations and values.	Current/next/previous item categories
Monitor		To be aware of system states that determine need for navigation, exchange and action.	Relevant items to monitor; record of when actions were taken; elapsed time from action to present
	M1: Monitor to detect deviance	Routinely compare system state against target state to determine need for action.	Normal parameters for comparison
	M2: Monitor to anticipate cue	Compare system state against target state to determine readiness for know action.	Anticipated level
	M3: Monitor the transition	Routinely compare rate of change during state transition.	Template against which to compare observed parameters

Sequencing Elements

Another thing that is good to know when analyzing tasks is in what possible orders the sub-tasks can be performed to fulfill the super ordinate goal. This order are part of the Hierarchical Task Analysis method and is called plans. But to be able to better fit the design interfaces the notion of these plans have been further developed in the SGT-model. Ormerod and Shepherd found this choice to be a little bit controversial because it mixes data and control flow in a single specification [23]. But Ormerod states that the sequence in which operations are carried out gives as much information requirements as the operations themselves. This information could for example be about in which order information must be presented to the operator.

SGT-sequencing elements are used to provide a notion for describing the task plans in terms of four different dependencies between the tasks (table 3.2):

- Fixed
  - Operations must be carried out in a strict order. Outcomes of earlier operations are necessary for the operation that follows.
- Contingent
  - An operation is carried out depending on the outcome of earlier or other contextual factors. The correct pre-conditions have to exist.
- Parallel
  - Operations have to occur simultaneously.
- Free
  - There is no predetermined order for the operations to take place.

The purpose or function of these sequencing elements is to specify in what way and order information should be arranged when it should be presented for the users.

An interesting aspect of combining task elements together with sequencing elements is that tasks that earlier have been implicit in the informal plan from HTA, become explicit after the redescription [23].

Table 3.2: SGT-sequence notation [10]

Code	**Type**	**Syntax**
S1	Fixed Sequence	`S1 – Do X`
S2	Contingent Sequence	`S2 – If (c) then do X ...` `If not (c) then do Y ...`

S3	Parallel Sequence	`S3 – Do together`
		`X`
		`Y`

S4	Free Sequence	`S4 – In any order do`
		`X`
		`Y`

Information Requirements

The process of specifying the information requirements demands a certain amount of guidance and formalism according to Ormerod and Shepherd. And they also state that by using of formal or semi-formal methods it is easier to get a structured design process. The task of design can then be decomposed in more manageable units.

One of the things that the SGT-method provides is a taxonomy for classifying different kind of low level task types. But according to Ormerod and Shepherd [24] it is within the capture of the basic information requirements where the SGT-method could contribute the most, providing the designers with the correct input to develop a detailed specification for the design of the system [24].

Each of the SGT-task-elements has certain information requirements associated to themselves in the form of templates. Determining the information requirements for a SGT-task-element can be a little problematic at first, because the task element might itself be decomposed into a sequence of different activities. This set of activities is a prototype for the template of operations needed to execute the task element or sub-goal. This is where the method gets its name from. But as mentioned earlier, the decomposition should stop when a SGT-element can be associated with a sub-task. Otherwise each of the operations within the template could be said to be SGT-elements and a recursive decomposition of operations ad infinitum could be the result. Each of these stereotypical sequences of operations that are necessary to perform a SGT-element are collected into a list of the minimum information that the user would need to perform these operations.

To assign tasks with SGT-task elements gives an automatic specification of the information requirements that are needed for the user to carry out a task successfully without restricting the design decisions [24]. This gives that the SGT-method provides the interface designer with a complete requirements specification as an automatic outcome from HTA.

3.1.7 Conclusions

The use of Task Analysis as part of the framework for the design process in system development is a good idea. As Shepard [27] puts it:

> "*Applying methods of Task Analysis within a rational framework provides a thorough and economical basis for identifying performance problems and a foundation for specifying options.*"

The decision of what parts to use from this extensive set of different methods is not easy. It is not possible to investigate all areas and methods within Task Analysis. Some kind of decision had to be made about what kinds of methods should be examined in detail. Trying to get a variety of different aspects on Task Analysis, a decision was made to concentrate on well known methods that focuses on different things of Human Computer Interaction. The chosen methods are *Hierarchical Task Analysis*, *GOMS* and *Cognitive Task Analysis*.

By using Hierarchical Task Analysis, a functional way of work, considered the specification of goals within the system under development and the tasks that are required of the user to perform to attain these goal states, is maintained. It gives a clear overview of the different parts of the system and it is a good way of locating problems that could be found in tasks. Another positive

aspect of HTA is that it is relatively easy to perform [27].

GOMS is a predictive model which is used to capture procedural knowledge of users operating a system. The performance of the users could be examined without having to perform any exhaustive testing of the users. By using a GOMS-model the gathering of information through empirical testing is superfluous. This minimizes the use of resources as time, labour and money when performing an analysis, but the resulting design will still be functional and usable for the users. If a set of different designs for systems and/or interfaces are being compared, GOMS could be a suitable alternative to use.

But there are also some negative aspects of using GOMS. First of all, the range of usage for GOMS-models is quite low. It is preferably used for testing of SMEs (Subject Matter Experts). It is hard to predict the performance of users with less knowledge concerning the system. There is no possibility to handle faults and it can be hard to capture all the aspects of the users, e.g. mental workload and social aspects. A dilemma with GOMS is that predictions only can be made of things that are easy to predict.

Cognitive Task Analysis is an area which the military finds great interest in. NATO has for example conducted a study of this branch of Task Analysis which was presented at a workshop on CTA [26]. In CTA the main focus is upon the human users of systems, and the cognitive parts of the tasks that are performed. It captures user-centered information within tasks. To conduct a Cognitive Task Analysis demands highly skilled analysts. They need to be good at communicating, extensive social skills, intelligent and fast learners. An endeavour of this analysis method is that users should not have to focus upon the system or the interface and instead focus upon the tasks they are carrying out. When there are more than one proposition for the design of a system and they are to be evaluated, the use of a CTA-method could be suitable. Other positive aspects is that it is easy to use CTA together with other techniques for design and because of the fact that the need of prototypes is reduced, the design cycle will be speeded up.

Each of the different areas of Task Analysis has their own positive and negative aspects, and there is no method optimal method that always give the best results. The decision of which analysis method to use is dependant on things like: type of task to analyze, domain and context, user resources etc. But the most important issue when making the decision is to know why the task analysis is going to be performed.

In this paper, Task Analysis will be used as a part of a framework in the HCI-design process in the development of military systems. This was one of the reasons why Cognitive Task Analysis was examined. One important aspect of the chosen Task Analysis is that it should be a users-centered approach, so that information concerning the human users in the system could be captured. But Coginitive Task Analysis methods often demands a lot from the people performing the analysis and it often demands a lot of resources. A method which is a mix of Hierarchical Task Analysis and Cognitive Task Analysis would be a good method to get a functional way of working with goals and tasks and to focus on the user-centered aspects of the system. These features was found in a method called ”the Sub-Goal Template”.

The SGT-method has been developed to specify the information requirements so that the problems of using Task Analysis as a part of the design process could be addressed. One important aspect of this method is to maintain the neutrality in terms of device-independence so that the implementation and design can be left to the designers of the system. This is done by using

the task elements that are stated in the SGT-scheme. When the decomposition of the tasks is complete, a specification of the information requirements is attained. This specification is a very good source of information to use together with the functional requirements when a system is going to be designed.

3.2 Design Space Analysis

Design Space Analysis (DSA) is a method for recording design rationale. It is used for creating a representation of why the design is the way it is [6]. It provides a notation called QOC as an explicit representation of the design space, and the reasons for the design choices made within it [19].

Design Space Analysis can be defined as

> "*Exploring design options and their implications, by considering pros and cons and dependencies between design decisions.*"[30]

In contrast to traditional methods in design, this is not a method for creating a single artefact or prototype. DSA is about producing a design space [6], i.e. the "complete" (the relevant) set of design choices are included, and therefore it provides a good base for presenting the resulting design. MacLean, Bellotti and Shum state that the output of a DSA is to be considered not as a artefact, but as a co-product of the design process, along with the artefact itself[19].

Because of the notation used in DSA for representing the resulting design includes the choices that were not chosen (and why these were not chosen), DSA is well suited for arguing for a certain design. According to MacLean et. al. the use of DSA in design processes is supported by two highly relevant issues. First, the DSA is useful because it provides *"an explicit representation to aid reasoning about the design and about the consequences of changes to it"* [20]. Secondly, DSA serves as a "vehicle for communication" among members of the development team(s) and subsequently among those who will maintain the system.

DSA does not require any prior knowledge about the information used in the analysis and evaluation. It is a method suitable for both evaluating existing systems and for use in the development of new systems.

3.2.1 Design Rationale in General

The final product of a design process should not only be the artefact being developed, but also design rationale for motivating and explaining the choices made. According to MacLean et. al. [21] it is almost imperative to grasp how the design could have been different, and why the choices that were made are the most suitable in order to fully understand a particular design. Every choice made in the final design describes the artefact, and the alternatives to these, and together with the reasons for the decisions they give support for a better understanding of the final design [21]. They also claim that a compilation of design rationale should be a product of the design just as much as the final artefact is.

It is very important to point out that design rationale is not a documentation of the design process. It is a co-product of the design, and it must be designed also. According to MacLean,

Young and Moran it is very unlikely that a coherent and structured design rationale-document should arise by itself by merely documenting the design process [21]. This is why a good design rationale-document also must be designed carefully. MacLean, Young and Moran concludes that particularly two groups benefit from having structured and well defined design rationale documents [21].

1. **Developers**. Design rationale-documents serve as aids for problems solving as well as support for design deliberations, and as means for communication to ensure that other developers (or system administrators) understand the system.

2. **End Users**. End users also benefit from design rationale-documents, as they provide an understanding of the designers intentions.

By having an explicit representation of design rationale, designers get the possibility to envision the design space in a more structured way, and it gets relatively easy to compare different design choices. [21]. MacLean et. al. also claim that a structured design rationale documentation increases the probability of creating a good design [21].

In practice, it is rare to have to create a design rationale-document from scratch, because a lot in design is borrowed from, or inspired by, existing designs. Explicit design rationale documentation makes it easier to detect inappropriate assumptions, and to see that key assumptions have not been expressed. It also prevents designers tendency to overlook or not consider possible design alternatives when important design choices are made, MacLean et. al. points out [21].

Design rationale-documents have the additional benefit of giving system administrators and maintenance workers the possibility to see what consequences modifications of the system might have.

3.2.2 Background

In the mid to late 1980's, the need for documenting and recording design rationale in a structured way in development processes spawned the approach called Design Space Analysis. Reading the literature in this area, Allan MacLean, Victoria Bellotti, Simon Shum, Richard M. Young and Thomas P. Moran are frequently represented as authors of papers about DSA. They have all had part in the development and evolution of this technique, using it in various case studies and development projects, one of which is the AMODEUS-project [16] that some of these authors have been involved in.

The AMODEUS is an inter-disciplinary project with the goal to investigate and study the interactions between systems and the users, and then develop methods to formulate those in efficient ways for interface designers to use when developing systems.

3.2.3 DSA in Detail

A QOC-representation, or a QOC-diagram, is a schematic sketch representing the design space, with design alternatives, and different Criteria for choosing among these. Figure 3.2 shows a simple QOC-notation, with one Question concerning a specific design issue, having two design Options (i.e. specific solutions to accommodate the question). These Options are judged with respect to the criteria. One of the Options spawn a Question, which in turn can have Options, and so forth. The solid line between the Options and the Criteria denote positive support, and the dotted lines denote negative support for that Option, when assessed against the Criteria.

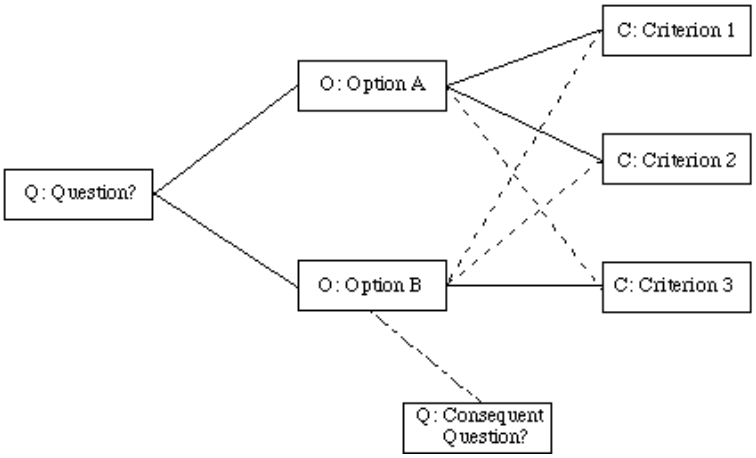


Figure 3.2: The QOC-notation (Bellotti and MacLean, [6])

This is a representation very similar to the discussions that take place during development meetings, but is more logical and coherent. By using the DSA and therefore employing the QOC-notation, the reasoning about design issues becomes more structured and disciplined. It also acts as a very good record for the design deliberations over time [20].

The QOC-representation of a design gives a very concise expression of the final design, by putting it in a wider context and showing how the design might have been different, and why it is like it is. Representations of this kind should give support and make the communication between people with different goals and backgrounds (developers and users) easier [19]. MacLean, Bellotti and Shum [19] have discovered this method to be very useful to more easily understanding design issues during the development.

In MacLean et. al. [20] it is argued that QOC-representations should be useful tools for communication between the designers and the users, as well as the current design team and possible future ones. This due to the fact that they are simple enough to give understanding of the design space to a variety of people, and because they are flexible (suitable for a wide range of system design) and explicit in the way that they bring forth issues that can be judged and questioned by others. MacLean et. al. [20] also states that by using DSA, and its notation, members within the design team understand decisions better, decisions that they may have not been part of. Moreover, the consequences of changes in the design can easily be evaluated, by seeing what trade-offs might be imposed when making a certain decision, or what (earlier already made) decisions will be affected by changes in the design. The QOC-notation deals with central parts of the design. It does not require any advanced technology or equipment – a pen and paper is sufficient [20].

MacLean, Bellotti and Shum [19] point out that the QOC-representation can play an important role in problem solving, as well as being a base for cooperation and shared material in the design team(s) [19]. They conclude that the representation summarizes such things in the design that not even linear text, diagrams or even a prototype can do. The QOC-notation is not only applicable in graphical user interface-design. MacLean, Young and Moran [21] are mainly working

with GUI-related design, but state that the QOC-notation can just as well be used in more general design problems to capture limitations, for example in technical or organizational contexts.

Externalizing and visualizing the design ideas in a QOC-representation is not a top-down process. The creation involves constant revisions and iterations, in which ideas and solutions, i.e. design alternatives are re-evaluated and assessed. The representation evolves through several embedded cycles of representations and evaluations in which the focus is switched between different parts of the design space. Because of this, Shum [28] calls the QOC-representation an *"opportunistic method"*. Renaming and restructuring the design space is a natural part of the QOC-representation, and therefore the design space becomes more and more clear and understandable towards the end of the development. Shum states that studies have shown that design and the creation of QOC-representation is not done in cycles of *"create questions, create options, create criteria, make assessments"*. These processes are more sporadic and the focus is very often changed between different parts of the design space during the process.

MacLean et. al. [20] points out that the QOC-notation is a tool that lets the designers direct their attention to aspects of the design where problems or debatable issues exist. The authors say that this is why the representation (of the design space) should not be considered as a process that can not be revised or modified. It is not a fixed structural relation between design issues. The QOC-representation and the process of developing and exploring the design space are ongoing and open issues. The components can be considered as provocations, with the intention of broadening the scope of ideas for designers. Although it is virtually impossible to represent the entire design space of an artefact, the understanding and insight to the space increases during the design process.

It is pointed out by MacLean, Young and Moran [21] that it should be possible to expand the design space in order to study it in a more detailed level. Having every level of detail visible/accessible, can make the design space representation oversized and hard to manage. A QOC-diagram can quickly become *"messy and difficult to manage"* [20].
One of the main issues in Shum's study [28] is to support the becoming of rational structures in a usually unstructured and sometimes vague mish-mash of ideas in early design deliberations. Designers will not often express ideas in ways that follow the style guide (table 3.3) precisely directly from the beginning of the project. The purpose is to give support and create, order to produce manageable design rationale documents, which are reusable for others, but this creates an extra burden for the designers. This becomes a trade-off between loss of creativity and flexibility, a somewhat common critical point towards formalism.

Shum noticed that certain ground rules for the QOC-representation were needed [28]. These rules had *"proposed themselves"*, as he puts it, throughout the studies he conducted, both in user comments and in his own use of the QOC-notation. The rules of thumb are presented in Table 3.3. Shum points out that these guidelines should be considered merely an early proposal, and those who use them, are free to add and/or remove from the list of guidelines as they please and find appropriate, as experience and empirical facts suggest.

Table 3.3: Styleguide for creating the QOC-representation [28].

QUESTIONS:	
Q1	No YES/NO-questions.
Q2	Every Question considers one and only one design issue.
Q3	Questions must not ask how to fulfill Criteria.
Q4	Questions must not summarize Options.
Q5	Generic and general Questions handle issues that concern the design in general, in the big picture. These are answered once, and subsequent Questions concerning the same aspect will be answered by default.
OPTIONS:	
O1	At least two Options per Question.
O2	Each Option must have positive and negative assessments.
O3	Options are to be considered as potential design choices.
O4	One design alternative per Option.
O5	Options are distinct and contrasting alternatives. They must not include the same alternatives.
CRITERIA:	
C1	Criteria are presented in a positive way. This gives consistent semantics in the QOC-representation, having that a solid line between an Option and a Criterion denotes positive support.
C2	One Criterion only ones per each Question.
ASSESSMENTS:	
A1	Weak assessments are neutralized by creating better Options.

Shum also makes it clear that it is still an issue of when representational limitations (as the guidelines in fact are) constitute a helping compositional structure, and when they form a *"creative straitjacket"*, as he so vividly puts it [28].

An early QOC-representation is not always understandable for others than the members of the design team. When the QOC-representation is used for expressing and evolving ideas, the names, terms and problem decomposition are not often fully correct, but still sufficient for the design team. Shum explains that these types of flaws in early and incomplete QOC-representations are something that designers have to be able to work with in order to evolve ideas. By iteratively remodeling and restructuring the QOC-representation, the representation evolves from being a tool for evolving ideas among design team members, to a means for communicating with others, or as Shum calls it: *"... a move from expression to documentation"* [28].

In his paper, Shum shows two specific problems and difficulties that several designers have encountered [28]:

- The designers usual way of working was already similar to the QOC-representations way of posting alternatives, but they felt that being forced to explore the design space was limiting and acted as a constraint.
- The difference between a QOC-representation and their usual ways of working lie mostly in the representation, and that they felt that the notation was inadequate.

These two problems together slowed the dynamics, normally so characterizing, of the development processes the developers were used to.

The solution to these problems is how the QOC-notation is used. It cannot, and is not intended to, replace brainstorming or other intuitive methods important and pivotal for a creative design process. The important thing to keep in mind is that the roles of documentation and expression have to be separated. If the QOC-notation is used merely for documentation of design rationale, it assumes a passive role and has no part in the reasoning of design ideas. But the QOC-notation can also be used proactively with the purpose of expressing and evolving ideas if the QOC-representation plays a central role in the creation of topics for brainstorming and discussions. The reason the QOC-notation led to the problems and difficulties that some of the designers and users in Shums study experienced, is that the QOC-notation was used as a passive documentation for making decisions recording design rationale, instead of as a medium for expressing and evolving the ideas right from the start [28].

Bellotti and MacLean [6] recommend that one person, or a smaller group of people, should be responsible for the representation of the design space(s) (i.e. the QOC-notation), to reduce redundancy and to increase consistency in the representation. Having too large a group involved in the creation of the representation, can result in different interpretations and mismatching notations, regarding use of terms and focus in the design space.

It is, according to Shum [28], very important to keep the right focus in the representation. Designers should strive to give the Criteria the right focus, so that it is understood how they relate to the Options. For example, the Criteria ”usability” is so general, that it is hard to see how to relate it to an Option and make proper assessment based on that Criteria.

Having distinctive terms in the QOC is also very important, Shum states that two Options cannot be variants of one design alternative. Two Criteria cannot be trade-offs and two Criteria cannot express the same thing in different ways. Distinctive elements and focus are characteristics for a well-formed QOC.

MacLean, Young and Moran makes the point that it is important to formulate Questions and Options as alternative design choices, but that alone is not a way of ensuring a good design [21]. A very important part of the creation of a proper design space is to make sure that the Options are compatible with each other, so that they fit in to the design. The Questions must be appropriate, meaning that they must address the relevant issues, and the Options should be chosen on basic principles.

MacLean, Bellotti, Young and Mora [20] describe the characteristics of their approach to design rationale and the QOC-notation. These are summarized in Table 3.4

The roles that DSA play in creating, evaluating, reflecting over and managing the design is discussed by MacLean et. al. [20]. In the part of creation, they expect DSA to help innovation and reasoning in the development of design. This is because of the ease of generating, representing, evaluating the design options. When using the QOC-notation, assumptions about the design are brought into light. It also spawns new Questions, and challenges Criteria, and guides the developers in deciding if new Options can make use of the strengths, and how to deal with the weaknesses of current Options.

The role of reflection is covered by the uniform format of the notation. It makes it easier to store, cross-reference and index the design deliberations.

Design management is also covered by DSA. With the ability to add, remove and change the elements the design space, design team members can keep track of changes and design explorations over time. As MacLean et. al. put it: "*Because Design Space Analysis explicitly represents a design space, it is well suited for tracking changes*". [20]

3.2.4 Questions

According to MacLean, Young and Moran [21], generating good Questions is more important than generating Options, as the Questions have a central role in the generation and structuring of Options. In some cases though, Questions can be generated from Options if they (and often do) spring in to mind [21]. Once a Question is defined, and having an answering Option to that Question, it is easier to come up with alternative Options and explore the design space more thoroughly.

In MacLean et. al. [20] the role of the Questions is described as being a means of getting a detailed structure of local contexts within the design space. This is, according to the authors, a way of ensuring the Options that address the same issue are compared with each other. This gathering of Options results in a structured set of Options. The authors state that good Questions help in the process of coming up with new Options. They summarize the role of Questions as it being *"generative and structural, not evaluative"*.

In a paper by MacLean, Bellotti and Shum [19], they conclude that when exploring the design space, it is not always clear what Question, Options and Criteria exist or should be included, and if the existing Questions do not cover the relevant and important parts of the design, a significant restructuring of the QOC-representation is needed [19].

There is a notion of internal consistency [20], meaning that a Question in one part of the QOC-representation should be answered in a similar way as a Question in another part of the design space is answered. As discussed earlier, Bellotti and MacLean recommended that one person or a smaller group is responsible for the creation of the QOC-representation. One of the reasons, perhaps the most important one, for that is to keep internal consistency in the design space that the QOC-notation represents.

Table 3.4: Characteristics of the QOC-notation (MacLean, Young, Bellotti and Moran [20].

Design Space Focus
The focus lie on developing a design space, with Options as design alternatives, structured by Questions addressing design issues. The motivations (design rationale) in the DSA are based on the comparison between alternative design choices (Options), and the authors strongly recommend the design space being developed by comparing alternatives, in contrast to other methods only dealing with refining a single design.

Focus on Criteria
The objectives of the design process, represented by the Criteria, are taken into explicit focus in the QOC-notation.

Coproduct of Design
It is important to realize that the DSA itself is an artefact, not a record of the design process, and therefore it also has to be designed.

Embedded in Design Activity
Because the property of being a coproduct of the design process, the Design Space Analysis (specifically the QOC-notation) combines a descriptive representation of the design with the representations of the rationale for the decisions.

Semiformal
The representation of the design space is semiformal. The basic structure of the QOC and the relations between the components is a formal definition, but the descriptions within the components are informal and almost no restrictions apply to them.

Argument based
The authors state that design rationale is based on argumentation and not on proof, and because of this the every element of the QOC-notation can be debated and put up for scrutiny, and evaluated.

Expandable Detail
As discussed earlier, a QOC-representation can quickly become very large and hard to manage. Therefore, it is desirable that parts of the design space can be hidden, and expanded when needed, if more details about that part are needed.

Purposeful
It is not practical to present and include every level of detail in the design space. Even the authors admit that *"a QOC-representation does not need to be a complete specification of a design"*, meaning that it may be unnecessary to include uncontroversial, common and well-understood design solutions. They conclude that it may be sufficient to provide a DSA

- where difficult issues arise
- where nonobviuos solutions are used
- where there is a need for a deeper understanding of the system
- if the part of the design are critical in some senses.

3.2.5 Options

The Options are very important parts for the resulting design, because they represent design alternatives that designers have direct control over. By including some options, and excluding others, the final design space can be very different, than had they been chosen differently.

Options must be distinct. Options cannot be part of, or depend on another Option. They are stand-alone design alternatives, equally plausible when not taking into account the assessments of Criteria.

3.2.6 Criteria

MacLean, Young, and Moran [21] claim that Criteria is the most important part of the design rationale. Criteria are principles or standards that Options are measured against. Criteria are important because they evaluate Options in (local) parts of the design, and the suitability of Options are decided by how well they measure up to the Criteria. Criteria can be considered as desirable properties of the Options they relate to.

Although Criteria are very local, they have a global effect on the design as a whole. The relative importance of Criteria are of great matter in deciding the design [21]. MacLean et. al. [21] compare Criteria to Newmans ”Requirements”, and give the example of ”giving a lot of feedback”. If that requirement more important than, say, ”response speed”, different design choices will be made, than had it been the opposite way.

MacLean, Young, Bellotti and Moran [20] point out that it can be hard to find Criteria, so that they have the desired effect. It is important to make sure that the Criteria are abstracted to a level that suits the Options they concern. They also state that it would not be practical to propose any formal definitions of how Criteria should be formulated, but they offer a set of properties of the Criteria they find helpful when listing and creating criteria.

1. **They are properties of the design that the designers control indirectly.**
   This meaning that if a Criterion such as ”ease of operating with a mouse” is stated, it suggests that it has been decided that a mouse is to be used.

2. **They are unconditional.**
   All other things put aside, the more positive support an Option has for a certain Criteria, the better the design.

3. **They are evaluative.**
   A Criteria is a measure of some property of the of the product being designed, with the effect that the more an Option satisfies the Criteria, the better the design will be.

4. **They potentially yield a quantitative value.**
   This is an extension of property nr 3. It might come in handy to consider the Criteria as terms of judgment that could have a quantitative value. By this extension, a numerical analysis can be performed in order to choose the Options that support the ”best” Criteria (i.e. support the ones of highest order, the most important ones).

Bridging Criteria is a term used for creating ”short cuts” between complex sets of dependencies. MacLean, Young and Moran [21] give the example of ”ease of hitting with the mouse” (the decision to use a mouse had already been made). This Bridging Criteria bridges between

the decision to use a mouse and the more general Criteria of ”speed” and ”accuracy” and encapsulate them into one unit in the design space.

Another interdependency, apart from the bridging criteria, is a cross-question constraint which is a form of interdependency where an Option chosen in one part of the design space in a direct way affects the choice of Options for other Questions. MacLean et. al. [20] introduce the terms exports and imports, as two useful terms to describe this interdependency. Their main purpose is to simplify the representation, by making it easier to break up the design space in modules. Something that has an effect on another part of the system is referred to as an export, and the part where the effect is noticed is called an import. The ability to modularize the design space makes it easier to hide details, and to expand the design space.

To make the interpretation of a QOC-representation easier, Shum [28] proposes a restriction to Criteria: they shall be presented in a positive sense, e.g. ”easy to use”, ”low cost” or ”fast access”. At least, he points out, they should be expressed neutrally like ”usability”. By stating Criteria in a positive way, a consistent semantics in the notation is preserved, because a solid line between an Option and a Criterion denotes positive support.

Since Criteria can be weighted, Shum [28] states that the number of supporting Criteria is not enough, but can give a good indication of the suitability of the Option to which they relate. Having seen that the mere use of supporting links is not enough, Shum states that because of this, weighted Criteria would be preferable. Designing with the respect to a set of predefined Criteria implies that a general Criteria should be weighted in the beginning of the project, and these weights can be changed during development, when more and more design deliberations are made.

MacLean et. al. [20] address the issue of Criteria being too general. They discuss the consequences of finding broader and broader criteria, and explain that by broadening the criteria, and getting them less directly relevant for interface design, the Criteria becomes more and more an issue for financial analysts and marketing strategists. This is illustrated with the examples of the Criteria ”fast user actions” and ”low user errors” (both very general and broad criteria). These two both have impact on the Criteria of ”usability”, and this in turn contributes to the very broad and general Criteria of ”user acceptance”. Clearly, this last Criteria can be formulated as ”product success”. This is not a suitable Criteria for making design decisions, as it has very little to say about the design itself; it only addresses the products/artefacts success properties without taking into account any of the interface design related issues.

MacLean, Bellotti and Shum propose a sequential model (see Table 3.5) for the main activities in DSA and in the creation of a QOC-representation [19]. The model consists of five phases that guides the developers in the tasks involved, ranging from information gathering and organizing, through structuring a QOC-representation and eventually making the design decisions. The goal is to structure and highlight main activities of DSA. This should not be considered as a strict serial model.

Table 3.5: The sequential model of DSA (MacLean, Bellotti and Shum [19])

Phase 1: Identify and gather relevant information
- Get a feel for the main issues.
- Classify the given information as Q, O and C if possible.

Phase 2: Create a rough QOC-representation
- Structure and make sense of the information available.
- Find good Questions

Phase 3: Flesh out a rough design space
- Use current understanding of design to help generate new ideas.
- Generate new Options and Criteria

Phase 4: Reformulate the design space
- Tidy up the descriptions and make the design space more coherent.
- Reword Q, O and C if necessary.
- Reformulate Questions (and/or reorganize O and C) to improve decomposition.

Phase 5: Make design decisions
- Use Criteria to evaluate and select Options.
- Highlight decisions by drawing a box round the chosen Options.

There is also a declarative model, also presented by MacLean, Bellotti and Shum [19]. This is presented in Table 3.6. The declarative model is based on the use of design heuristics that focuses on subparts of the design representation (i.e. the QOC-notation) in order to assist the developers in the development of the design space. It suggests moves that can be used to explore the design space, and it captures ways to have the QOC-notation give cognitive support for reasoning about the design space. The main purpose is to give guidance in a systematic way.

3.2.7 Conclusions

Considering the low complexity and the ease of use, DSA is a very good method for developing systems and making design decisions. With its explicit and logical graphical representation of design rationale, it is a useful tool to incorporate in the design processes. The resulting design and the alternatives (i.e. how the design could have been different) considered during development are presented in an easily understood fashion, that does not require any special knowledge or training to grasp.

Table 3.6: The declarative model of DSA (MacLean, Bellotti and Shum [19])

Use Questions to generate Options
- Main issues are presented as Questions.
- These lead to the issue of having alternative designs (Options).

Use Options to generate Questions
- By brainstorming Options in general, Questions may come up.
- By coming up with Options, Questions that are overlooked or ignored may be discovered.

Use Options to generate Criteria
- The Options imply requirements in the design.
- Use these requirements to generate Criteria
- Criteria give either positive or negative support for a given Option.

Consider Distinctive Options
- The Options should not be ”dependent” of other Options.
- An Option is a stand-alone alternative design choice.

Look for Novel Combinations of Options
- By considering using (new) combinations of Options, positive effects to the design may evolve.
- Do not hesitate to explore unusual combinations of Options.

Represent both Positive and Negative Criteria
- By including and considering both negative and positive Criteria, it is likely that a more balanced and justified design decision is made.
- Helps in the evaluation of Options.

Overcome Negative, but Maintain Positive Criteria
- Try to work out the design space so that the negative Criteria are excluded, replaced or (at least) minimized, so they don’t influence the final design.

Design to a Set of Criteria
- The Criteria that are important, that give positive support for Options are central in the decision making.
- These must be considered at all times, when creating the design

Search for Generic Questions
- If similar instances of Questions arise in the QOC-representation, it is useful to formulate an generic Question, that’s only answered once, and other instances that turn up are answered by default.

With its resemblance to design discussions, the QOC-notation and the design space it represents is a method that is fairly easy to adopt in design work, as the discussions and design delibera-

tions during development often are conducted in ways of judging different design alternatives (Options) against Criteria. The method is certainly well suited for graphical user interface development, but it can also be used for other types of systems in which alternatives can be evaluated against each other [21].

The QOC-notation is semi-formal in the sense that the structure is formal, but the contents of the boxes are informal, which makes the method applicable in a wide range of system development. Because of the explicit and simple representation in the QOC-notation, communication between design team members is aided with the use of DSA.

Chapter 4

A Framework for HCI-design in Systems Development at Land Systems Hägglunds

This chapter describes one part of the main goal of this paper, which is to create and evaluate a framework, by combining analysis methods for design work discussed earlier, that fits into the existing systems development at Land Systems Hägglunds. The framework is meant to be used together with use cases, along with the information about the system being developed as well as the functional requirements.

Based on the conclusions and the observations made in the previous chapters, a structured semi-formal HCI-design process is developed.

The purpose of this framework is to provide a consistent and rather straight forward model for use in system development, as a defined design process, to

- record design rationale in a structured, easy-to-use fashion,
- provide designers with a set of requirements of information needed by the users to operate the system in a satisfying way, and
- put attention on HCI-related issues and aspects in the design.

This chapter concentrates on describing how the framework was created, and gives a detailed description of the framework. The sections in the chapter describe the parts of the framework in detail.

The other part of the main goal of this paper, to evaluate the framework, is described in next chapter.

4.1 Introduction

With the goal to create a framework that provides both a user-centered design approach and a documentation of design rationale as well as a structured way of producing a well defined design space for the artefact, the framework was based on two methods that deal with these issues in very specific ways.

As stated in the conclusions of the Task Analysis study (Chapter 3.1.7), the SGT-method captures, and describes in a explicit way the information a user needs in order to perform tasks to operate the system and to achieve the goals associated with doing so. This makes the SGT-method a suitable form of analysis of tasks in the framework to produce user-centered information requirements, to achieve the balance between the information sets.

The output from the SGT-method, i.e. the user centered information requirements, together with other system information and knowledge, is used as a base for the HCI-design of the system. Having all this information is a good starting point for creating a design space, in which possible design solutions are discussed. This is where the Design Space Analysis method could contribute with useful techniques in the creation of the design space, as a part of the framework. Although this indicates that this framework is a sequential flow, with the SGT-method preceding the Design Space Analysis, this is not the case. The Design Space Analysis can be conducted without the user-centered information (the results from the SGT-method), and only using functional requirements or other functional system knowledge.

Schematic Overview of the Framework

Figure 4.1 describes the framework in a schematic way, showing the relevant parts and how they are used together to get a structured design process. The top parts (the sources of information for ”System Knowledge”) together form the basis of information (input) for the two methods in the framework. The ”Use Cases” part can be used to extract the ”Functional Requirements”. The ”Task Analysis: SGT”-phase generates a specification of the ”User Centered Information Requirements”, that together with the ”System Knowledge” is used in the ”Design Space Analysis”-phase, to ensure that both HCI-related issues and functional requirements are addressed when creating the design space.

Even though the Design Space Analysis can be conducted without the user centered information requirements, it is appropriate to consider the framework as a sort of waterfall model to take the user centered information requirements into account. In essence, the steps of the framework as a waterfall model would be as in Figure 4.2. The waterfall model is a strict sequential process, but the parts of it are not to be considered as strict sequential processes. They are by themselves, in some sense, iterative because they can be done over and over again to get more detailed information or produce ”better” results.

4.2 The framework in detail

Here, the framework and its parts are studied and described in more extensive detail. Figure 4.2 described the overall workflow as a waterfall model, and Figure 4.1 gives a more clear (although not exhaustive) overview of the framework.

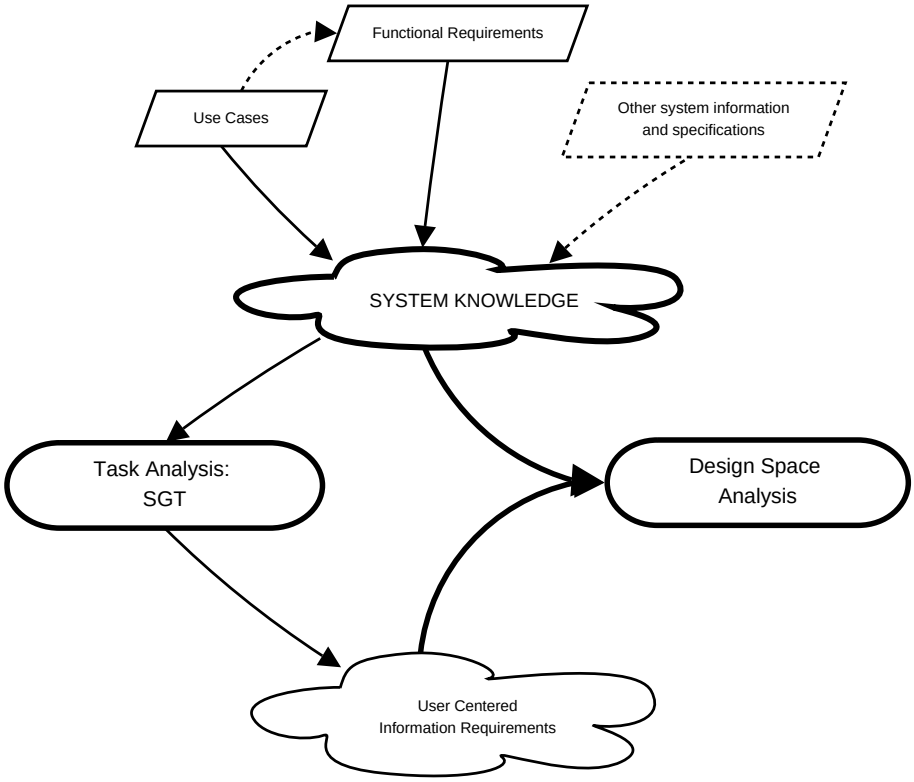


Figure 4.1: A schematic overview of the framework.

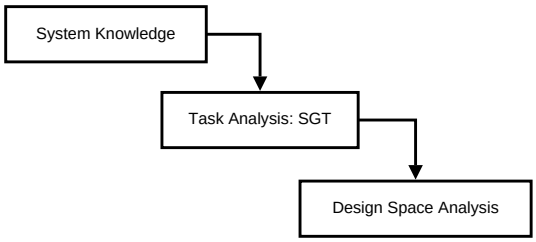


Figure 4.2: The waterfall characteristics of the framework.

The first section explains the work prior to the analysis methods. The following sections deal with the analysis methods used in the framework.

The Task Analysis: SGT-method is a more formal method than the Design Space Analysis, with more rules to be followed and strict definitions. This is obvious in the description of the framework, as the Task Analysis: SGT-method has to be described in more detail to be understandable, whereas the Design Space Analysis is a fairly straight forward method, and consequently the description of it does not have the same depth and detail. The lack of depth and detail in the description of the DSA, compared to the Task Analysis: SGT-method, is also

because the DSA is a method that is a general method of producing a systems design space, regardless of the system and its properties.

4.2.1 System Knowledge

To design a system, it is most important to realize the purpose of the system, its functionality, in what domain it will be used and to know the users needs and resources, and other vital information.

The single most important source of information is the FAD, with its uses cases to describe the system. Not only do the use cases give an understanding about the tasks/goals in the system, they also provide details about the context and the users roles (which user/actor performs what action). This information is more useful in the task decomposition in the SGT-method than in the work concerning the design space. The reason for this is that without the information about the goals and their sub-tasks, that can be derived from the scenarios, the task decomposition will be difficult to perform. By having these documents, this work becomes more easily performed. Another positive aspect of using these documents as a basis of information, is that the information gathering becomes more intuitive and defined.

The functional requirements come from requirements specifications, the users needs and wishes and possible limitations or conditions in the domain of use for the system. To capture the users needs and wishes, meetings and/or interviews can be arranged. It is important in the system development and the Task Analysis process to have good relations with the end users, to capture any changes in their needs or requirements [24]. Since Design Space Analysis is not only intended for handling the graphical aspects of design but is also capable of handling design of any type of design, the functional requirements should be used when stating Questions and Criteria in the design space and eventually when design decisions are made so that the requirements are fulfilled in an appropriate and correct way.

If the framework is used in the process of *re-designing* a system, perhaps when producing a more developed product, based on prototypes or earlier versions, there will most likely be other sources of information available, such as manuals, function trees and other descriptive documents. These will provide with useful information when creating a new (and perhaps a more refined) design space. The reason for this being represented with the dotted lines in Figure 4.1 is to visualize that these other sources of information may not always be available.

To summarize, having extensive system knowledge is useful and imperative in the design process to be able to perform the tasks involved in the framework, and thereby creating a well defined and well thought-out design.

4.2.2 Task Analysis: SGT

The Task Analysis part of the framework is meant to generate more information about the system, that is being developed, that is not specified or can be found by just looking at the description of the functionality of the system. This information is more tightly related towards the users of the system and what they need to know to be able to control and use the system in a successful way. This gives designers more information about constraints and limitations on the design of the system but still encourages innovative design.

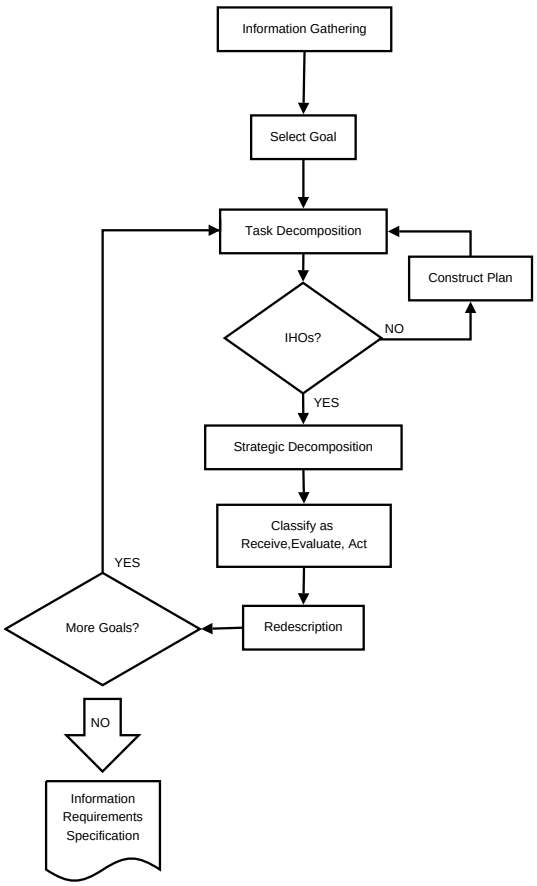


Figure 4.3: Flowchart of the Task Analysis method for information requirements specification.

The flowchart in Figure 4.3 shows how the Task Analysis part of the framework is performed. The method used is the SGT-method which will result in a requirements specification of information needed for the operators of the system. The representation used in Figure 4.3 is based on how Shepherd suggests that a Hierarchical Task Analysis can be represented when used in a framework [27]. This representation is also used when the Design Space Analysis is represented in Figure 4.4.

Information Gathering

Before the analysis of the goals and tasks in the system could be performed, information about these goals and tasks must be collected. As stated earlier, the basis for this information should be the use cases, from the FAD, and the scenarios within them that describes how tasks is performed in the system. These scenarios shall be viewed as descriptions of the overall goals/tasks of the system and also how these goals could be attained.

Information about the goals and tasks of the system can also be collected with other means. A good way of receiving this information is to participate in meetings with system developers and

end-users of the system. This can give better insight in the system and also what the users find to be problematic and what tasks to put extra focus upon in the analysis. The end-users opinions could be useful but they should not be considered as rules that have to be followed.

Task Decomposition

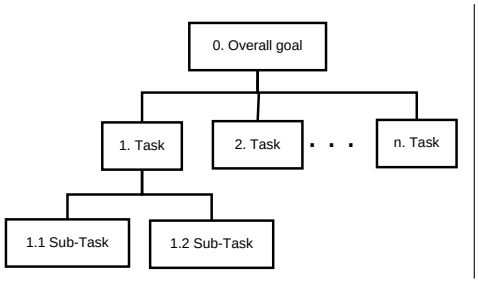
When all use cases related to the system being developed is collected, the decomposition of the tasks can be started. All the use cases should be examined thoroughly, trying to find what uses cases that could be especially important in the system or where a lot is demanded of the operators.

It is up to the person who performs the task decomposition to decide in which order to do the decomposition. One way is to do it in a ”breadth-first” kind of style, where all uses cases are treated in parallel, each level of the decomposition is completed before moving on to the next. In this way, it is easier to get an overview of the tasks and sub-tasks in the system, but it is harder to focus upon each task and a lot of information have to be processed at the same time.

The other way is to concentrate on one use case at the time (depth-first). The name of the use case shall be seen as one of the overall goals of the system. This goal is then decomposed one level at the time until a level is reached where the task can be described as an information handling operation. By focusing on one goal at a time, the decomposition could be easier to perform because of the familiarity with the goal and the tasks that have to be performed.

There are two standard ways of representing the task decomposition, see Table 4.1. One of them is to use a hierarchical tree structure, which gives a good overview of the tasks and their sub-tasks. The other one is more of a tabular representation with indented notions of tasks at different levels.

Table 4.1: Standard representations of the hierarchical task decomposition.



0. Overall Goal
1. Task_1
   1.1 Sub_Task_1
   1.2 Sub_Task_2
2. Task_2
.
.
n. Task_n

The decomposition of a task is done in an iterative fashion. It starts with looking at the overall goal (the use case) and then trying to find what tasks that are associated with achievement of the goal. A goal can often be achieved in more then one way and this shall be shown in the decomposition. Tasks are further decomposed and divided into their subsequent sub-tasks until a level has been reached where some kind of information that has to be handled by the users (IHO) is involved. At this stage the decomposition shall cease.

When a step of the decomposition of a task or sub-task has been made and IHOs cannot be recognized, a plan has to be constructed in order to describe in which order the underlying

sub-tasks of the decomposed task are performed. By using the notation of the SGT-sequensing elements found in Table 3.2, an algorithmic description could be generated. This description (the plan) should be documented within the same document as the representation of the hierarchical task decomposition.

Strategic Decomposition

At this stage in the analysis, a decision has to be made about how the IHO is going to be peformed, i.e. what strategy to use. In order to decide what strategy to use, a negotion with users and other system developers may be needed. Depending on when this analysis is performed in the system development, these strategic decisions may already have been made and could be found within use cases. As stated in Chapter 3.1.6, Information Handling Operations can be divided into parts of *receive*, *evaluate* and *act*. The strategic decision states what tasks to perform to cover these classes of activity. How the information is received, how to evaluate it and how to act upon the information. When this is done the analysis can proceed to the next step.

If a strategic decomposistion has been made and is later judged to be problematic or faulty, a new strategic decomposition can be done directly without having to change anything in the hierarchical structure above the level of the IHO, because the neutrality in terms of implementation is preserved.

Redescription

Now the decomposition of the overall goal has reached a level where no further decomposition is needed. The tasks that are decomposed from the IHO could be seen as basic tasks that could be classified as one of four different Sub-Goal Templates, *Act*, *Exchange*, *Navigate* and *Monitor*. (Chapter 3.1.6). These templates contain the standard sub-task that are carried out to perform the sub-tasks of IHOs. Each SGT contains of a couple of different task elements. These elements is all have the features of the SGT, but the task elements differ from each other in the way they are performed.

Each of the tasks within the receive- evalutate- and act-classes are redescribed as an SGT and a specific SGT-task element is associated with the task. The SGTs and the task elements within are created in as few categories as possible without making the redescription ambiguous.

When a redescription of the tasks within the IHO has been made, it is time to check if there are any more goals or sub-goals that have not yet been decomposed to the level where a redescription to the SGT-notation should be done. When there are no more goals to decompose the task analysis part of the framework is finished, and the results can be handed over to the design team.

Information Requirements

Every SGT has a set of requirements on information associated to them, that the users needs in order to carry out the task in a satisfying way. This applies to the SGT-task elements as well, but these information requirements only consider the tasks that differ from the main template.
By having all the bottom level tasks redescribed as SGT-task elements, a complete specification of information needed for a user to fulfill the overall goal is obtained. This information should be a useful resource in the design part of the framework, to capture user-centered aspects on the design.

4.2.3 Design Space Analysis

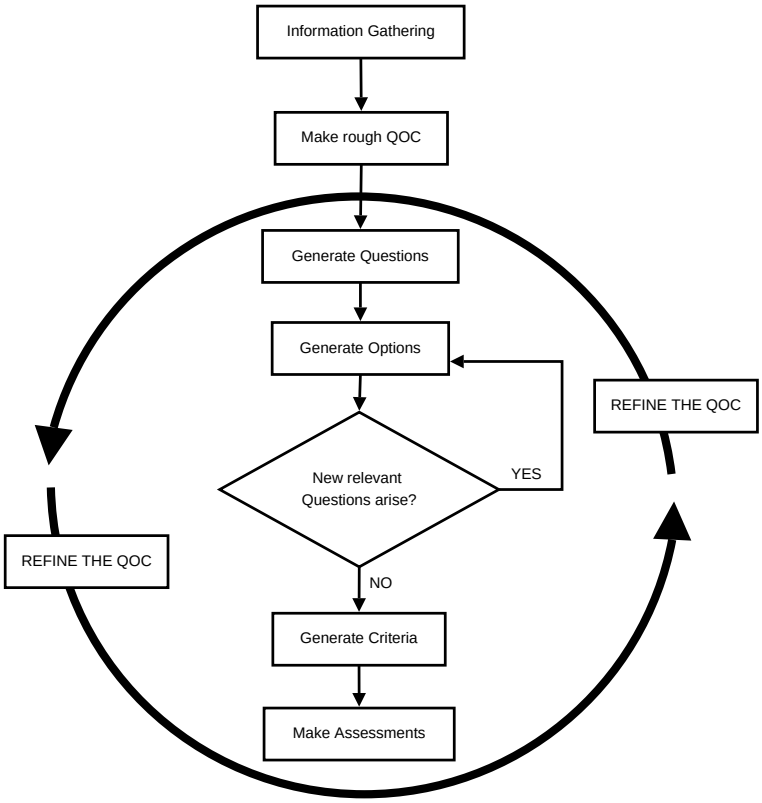


Figure 4.4: A schematic overview of the work flow in DSA.

A schematic overview of the work flow in Design Space Analysis is seen in Figure 4.4. Although it is depicted with a flowchart kind of notation, it is not a flowchart in the sense that it is sequential. The whole process is iterative, and the steps within the circle are allowed to be performed at any time, if and when the issues arise.

The initial step is to gather information about the system. The more the designers know about the system and its intended usage, and the users needs and intentions, the better the resulting analysis will be. The sources of information are the requirements specifications, the resulting set of information requirements produced in the SGT-model, and whatever documents that provide support for understanding the desired design and its use.

When a sufficient amount of system knowledge is gained, it is appropriate to create a rough and sketchy QOC-representation (a design space), to get a feel for the issues at hand. This part consists of coming up with relevant design questions, addressing the design issues at hand, by posing questions that consider specific design related choices. The Questions can not have YES/NO answers.

When Questions have been specified, answers to these are created. These answers are design options, being specific distinct design alternatives. At least two distinct Options (i.e. different design alternatives) per Question must be specified. When generating Options, new Questions may arise (and most probably will do so), and these are included in the notation, simply by adding them into the QOC-representation.

Having generated Options, Criteria used for assessing/judging Options are formulated. The Options fitnesses are judged based on their support for the Criteria stated. If an Option gives negative (or poor) support for a certain Criteria, it will be noted in the resulting design space, and these assessments are finally used for the resulting design. The Criteria are presented in a positive way, to preserve the semantics of the notation, being that a solid line denotes positive support. This is the part of making assessments.

Since the DSA-analysis and the creation of the QOC-representation are not strictly sequential, the generation of new Questions, Options and Criteria can be done at anytime in the process of generating the notation (the design space). If either new Questions or new Options arise during the creation, these are simply included in the design space whenever they spring into mind.

As the creation of the design space proceeds, and Questions, Options and Criteria are stated, a refinement of the QOC-representation is made. Being an iterative method, the QOC-representation (the design space) is constantly refined, approaching a final design space with a final resulting design of the artefact.

Information Gathering

This phase is a very abstract part of the Design Space Analysis. It involves activities that contribute to the overall knowledge of the system, its users and their requirements, in order to get a feel on the design and how and why it should be in a certain way.

By acquiring as much information and knowledge about the system as possible, the better the design deliberations and resulting design space will ultimately be. Having knowledge about the domain of use for the system, intended context and so forth, the designers are more likely to make better design decisions when being able to put the design options into context and judging the Options against the Criteria.

The information can be of various kind. Any information that helps developers to understand the system, its purposes and the users requirements and needs, is of great value and should be taken into account and examined.

Meetings with end users, studies and evaluations and development of prototypes are activities that are useful in coming up with design ideas, finding out what restrictions (if any) apply on the requirements on the final design.

Make a Rough QOC

In this step, an initial design space is created. It is very likely to be far from the final representation, but the intention is to get a starting point and being a ”igniting spark” in the creation of the design space. There is no need for this first generation to be anywhere near a ”correct” design space (of course, the rules that apply to the respective parts of the QOC-notation should

be followed). Due to the constant refining and restructuring of the design space, the QOC-representation will over time move towards a more and more detailed and structured, coherent version.

When creating this ”low-fi” QOC-representation, it might be convenient to use only paper and pen, or other simple tools. There is no need to use any more advanced tools, such as computer software (such as DREAM, see chapter 5.4) for creating the notation at this point, as it is only a first-step-version, with the purpose of of starting the design deliberations and to get a feel for the design space.

Generate Questions

The Questions, the elements that address specific design issues, are important for the resulting design. Stating relevant Questions lead to a structured and defined design space overall. The Questions can not be stated in a way so that they ask how to fulfill requirements. This would result in the Options being stated not to provide alternative designs, but as more general answers to how to meet the requirements.

To obtain and preserve the internal consistency (see Chapter 3.2.4) in the design space, a Question shall be answered in the same way as other Questions in the design space.

The Questions must not to stated be as how to solve a problem, or how to perform a task. For example, ”How to manage the Fire Missions” is not a valid Question in this method, at least not if the design space is covering the graphical design of the system. If the Design Space Analysis is used to create a design space that deals with the tasks or goals in the system, it is appropriate to formulate task-specific question, as the example of managing fire missions. In this framework, however, this is not the case, and therefore the Questions cannot be stated in such way.

Generate Options

Options are the elements that in a very direct way control the resulting design, as the designers explicitely influence the design by including or excluding design alternatives. That is, the designers should include more or less every (reasonable options to a reasonable extent, of course) design option they can think of, even if they are considered to have no realistic possibility of being included in the final artefact. By doing this, the designers cannot, when the design of the artefact is considered done (and the design space is complete and final), be accused of not considering this or that design option.

Obviously, it is a fine line between really exhausting the alternative design choices and adding them into the design space, and including ridiculous and unreasonable options that most likely will not even be considered, whatever the conditions or circumstances may be. Nevertheless, the designers should try to ”think outside the box” and try to find new design choices. All in the name of getting a design space that could be argued to be thoroughly explored.

Options can certainly lead to new Questions regarding design issues. This is very likely to happen, as a part of the refining and re-iteration in the generation of the design space.

Generate Criteria

The Criteria are elements in the design space that ultimately have a great influence on the final design, since the Options are judged against the Criteria. Criteria can be derived from the functional requirements, the users needs or other sources of information that provide information about what aspects are important in the final artefact.

Always state the Criteria in a positive way. This gives the notation consistent semantics: a solid line from an option supports the Criteria.

By including certain Criteria, the design will be influenced by them, which makes it important to have in mind what consequences the inclusion of this or that Criteria has for the resulting design. Designers can easily include Criteria that makes Certain Options unsuitable as alternatives for the design.

Although the designers themselves alone are not the ones who decide what Criteria are important, or even to be included in the design space, they are able to decide what aspects of the design are important.

The level of abstraction of criteria, or how general they are, has a significant effect on the Options. By stating Criteria in general ways, the Options to which they relate become more and more of other departments concerns (see Chapter 3.2.6).

Make Assessments

This is the part of evaluating the Options and their suitability in the design, based on how well they support the Criteria. Because the Criteria are desired properties of the design addressing the specific design alternative, and stated in a positive way, if an Option is assessed to provide the proper support for a Criterion, the ”assessment making” is done by relating the supported Criteria and the Option with each other with a solid line. Similarly, if negative support is found for the Criteria at hand, a dotted line connects the Option and the Criteria.

The assessments does not have to binary; they do not only have to denote support or no support. There is always the possibility of defining degrees of support. For example, instead of having NOT SUPPORTED and SUPPORTED as the only choices for an option regarding a Criteria, it might come in handy to let the support be expressed in a, say, six-step fashion (as is the case in the DREAM-application. See section 5.4). Having different degrees of support is a more dynamic and natural way of looking at assessments. But it is very difficult to determine what level of support a certain Options provides, i.e. how to differ a ”very much supported” Option from a ”supported” Option.

Refine the QOC

Design Space Analysis is not a sequential process. It is very iterative and opportunistic. Opportunistic in the sense that whenever new design elements (i.e. Questions, Options or Criteria) are thought of, they can (and should) be incorporated in the design space. There is no restrictions or rules limiting the creativity in that way; stating when designers are allowed to add/create design elements and when they are not.

The refinement of the QOC is an on-going process, and is in some sense the essence of the creation of the design space. By constantly evolving the design space, for each iteration it (probably and most likely) approaches an optimal design for the artefact.

4.3 Summary – Using the Framework

In Table 4.2 a concise ”how-to”, that in a very brief way describes the framework and phases associated with it, is presented.

Table 4.2: Summary of the workflow in the framework

1. **Acquire system knowledge**
   Realize the purposes of the system, gather information about goals and tasks involved in operating the system and information vital for the intended functionality.
2. **Create a hierarchical structure of tasks**
   Use information, found in use cases, about goals and tasks and decompose them into a hierarchy of tasks. Create a plan describing in which order sub-tasks may be performed. Stop the decomposition when an information handling operation can be recognized.
3. **Classify sub-tasks as SGTs**
   Find strategies for receiving, evaluating and acting based on the IHOs. Redescribe the strategies tasks as SGT-task elements.
4. **Combine user-centered and system knowledge**
   Use the resulting specification of user-centered information requirements combined with additional knowledge about the system to form a foundation for the design.
5. **Brainstorming and discussions concerning the design**
   Explore different design alternatives and their consequences and influences on the design, iteratively to approach a refined and optimal design.
6. **Create the design space representation**
   Generate the QOC-representation representing the different design alternatives in the design space,

Chapter 5

Evaluating the Framework in a Case Study

In this chapter, the second part of the main goal of this paper, the evaluation of the framework is presented. The evaluation is done to discover the pros and cons of the framework and to find out whether or not it is appropriate for the context and domain. This is done in a case study.

In the first section, the industrial context is described, giving an understanding of the domain and context in which the framework is supposed to be used. The methods are described in the following section, by stating how information is both gathered and how the acquired information is analyzed. In the remaining sections, a detailed description of how the evaluation was conducted is given, and the results of the evaluation are discussed and conclusions are drawn.

5.1 Study Context

The case study was performed at Land Systems Hägglunds, which is a defense contractor. The company develops combat and all terrain vehicles. The company also develops systems for the components in these vehicles. One of the systems is AMOS. It is an abbreviation for Advanced Mortar System, which is a grenade launcher turret. The turret can be mounted on a multitude of vehicles, such as the Combat Vehicle 90 and other wheeled or tracked vehicles, but also on naval vehicles, such as Combat Boat 90. MIS, the Mortar Information System, is a subsystem responsible for controlling AMOS.

Functional Architectural Description – FAD

A central part of the systems development at Hägglunds is the Functional Architectural Description model, also known as FAD. It is based on use cases, but it is not a *use case model*. It is a *use cases realization model*. The difference between a use case model and the realizations of use cases, is that the realization model includes both the black-box and white-box steps, whereas a ”plain” use case model only include the black box steps.

The purpose of the FAD-model is to be a tool for analyzing functional system requirements, and a way of allocating these requirements onto the subparts of the systems.

The specification of use cases within the FAD-model is constructed around use case scenarios, with relevant background information about them. A use case represents a complete goal, that a user has to achieve to when operate the system.

Every use case specification in the FAD-model consists of exactly one *main success scenario*, every *alternative scenario* (alternative ways of achieving the goal of the use case) and every *exceptional scenario* (describing exceptions that can occur, and how the system handles these). The use cases are mostly interesting when dealing with HCI-issues.

5.2 Methods

Having no prior knowledge about MIS, or any part of the system, the first steps towards system knowledge and understanding MIS, and the domain of its use, was taken by exploring the prototype WCS. The exploration was done while documents describing the functionality for the prototype and the AMOS as a whole were studied (e.g. manuals, brochures, function trees etc). The FAD-documents were used to provide a view of the tasks and goals in each part of the system. By studying function trees, an understanding of what functions were needed and wanted in the system was gained. Conducting this exploration created a base for system understanding and knowledge about it.

Manuals, descriptions and guides (being useful and important sources of information for the system knowledge) were read, and thereby the system become more familiar. Another important factor in getting to know the system and its use, was to see it being used in the intended context and domain. An opportunity was given to visit a military training facility, where a prototype of AMOS using WCS, was used and tested. This was the first time the system as a whole was seen used in its intended domain and context. This trip also included a first contact with the end users and a chance to see them operate the WCS. It is worth noting that when observing users and the system, it may affect the behavior of the users and how they use the system. This is referred to

as the Hawthorne-effect. [22][25][12] In addition to that, a brief initial round of user comments concerning the system, and its functionality and problems and limitations they had noticed, took place. The purpose of this trip was mainly to observe the system to get more familiar with its domain of use and functionality, but also to be able to apply and associate the information acquired prior to the trip to concrete contexts. This created a greater understanding what MIS is all about.

Because the system developers at Hägglunds have continuous contact and an on-going negotiations concerning the systems design, a meeting was held with high-ranking officers, that were expert users of AMOS and WCS. During this meeting general discussions about the functionality in WCS and its use in AMOS, possible design alternatives were held. This meeting increased the awareness of problems in WCS and captured the users views and visions not specified in the system requirements documents.

Having gathered more information, acquired additional knowledge and deeper understanding for the system, a second trip to the training facilities was scheduled. On this trip, interviews and discussions with expert users were planned. A questionnaire was crafted, and handed out the expert users, to get their opinions recorded. The users who filled out the questionnaire were both high-ranking officers, with a lot of experience in the system, and privates, also experienced in the system. Both of these groups of people were considered expert users. After the questionnaire was handed out, and the users had filled them out, discussions about the design and the system took place. This turned out to be a very informal and open-hearted meeting. Users were able to express their feelings about the design, and even propose alternative design solutions.

Since a broader knowledge and a deeper understanding for WCS and the system as a whole was gained prior to the second trip, the design discussions were held on a whole other level than on the previous trip. The design solutions the users suggested were more easily grasped, and related to the specific tasks and goals in the system in a more detailed level. During this meeting, the developers also suggested design solutions, and were able to receive and discuss the users comments and reactions to these suggestions.

The period of information gathering, with meetings with users and developers involved in the AMOS and MIS/WCS project, and studies of the systems involved, was a useful experience in how the development process works, giving an insight in the tasks involved in systems development.

The acquired information was analyzed to find out what main tasks are involved in the system. It was clear that the information could be categorized into two separate domains. These being on one hand how the WCS (the prototype for MIS) works, how its used, its functions and its functionality in its context to be (AMOS), and on the other hand a more goal driven information, more neutral in terms of implementations.

The MIS was considered as a non-existing system, even though there was a prototype for it (the WCS). The part of the information addressing the prototype and its use, was, because of the decision to view the MIS as a non-existing system, used for capturing functionality in the system. This information would, when developing new systems, normally come from negotiations with the users. The parts of the information derived from meetings, interviews and questionnaires with end users reflect the users opinions about the functionality in the system. Normally when a new system, without prototypes or prior version, this kind of extensive information is obviously not available. Clearly, users always have opinions about the functionality in a system being

developed, but by having had the opportunity to actually *use* the system (or a prototype), their opinions about it are more valuable.

The use cases in the FAD were analyzed to find out what the overall goals in the system were. The use cases were suitable for this purpose, because they are goal driven and believed to be neutral to the implementation. Therefore, tasks necessary to be performed to achieve the goals can be discovered. By analyzing the steps in the task decomposition are possible to perform.

All information about the system is analyzed to be used in the design space analysis. When analyzing the gathered information, design options and alternative design are proposed, discussed and rejected or refined.

5.3 Applying the Framework

The workflow of the framework presented in Table 4.2 is used as outline for this section, where the parts of the framework are applied in the case study.

5.3.1 Acquiring System Knowledge

By studying the use cases in the FAD-model, a more functional view of the system and its requirements was gained. A short introduction to the FAD-model was given, to understand its parts, and how they relate to each other and to the development of MIS. Relevant use cases for MIS, those directly dealing with the goals in MIS, were collected.

The MIS-related use cases were examined, to relate them to the functionality issues observed and discussed during meetings and system (prototype) observations.

The use cases in the FAD, assumed to be goal driven, were used to extract the main tasks and goals associated with MIS. These tasks and goals in the use cases are ways of describing the functional requirements for the system. The actions of the actors in each step of the scenarios are considered as descriptions of (sub-) tasks needed to be performed to achieve the goal(s).

5.3.2 Creating a Hierarchical Structure of Tasks

The information in the use cases, specifically in the scenarios, was used for creating a hierarchical structure of the tasks involved in MIS, necessary to be carried out to achieve the goal(s) associated with the specific use cases.

Together with Karin Fossum[1], MIS-related use cases were examined, and the work with the SGT-method was initiated.

The overall goals/tasks in using MIS were defined by the use cases related to MIS, and the names of the use cases were used to label the main goals/tasks in MIS. Based on this, an assumption was made. The assumption was that each scenario in the use cases describe the tasks involved in achieving the goal, see Figure 5.1.

[1] An Interaction Design-student at Umeå University, also writing her Master Thesis at Land Systems Hägglunds. Her Master Thesis is about producing a new prototype GUI for the MIS

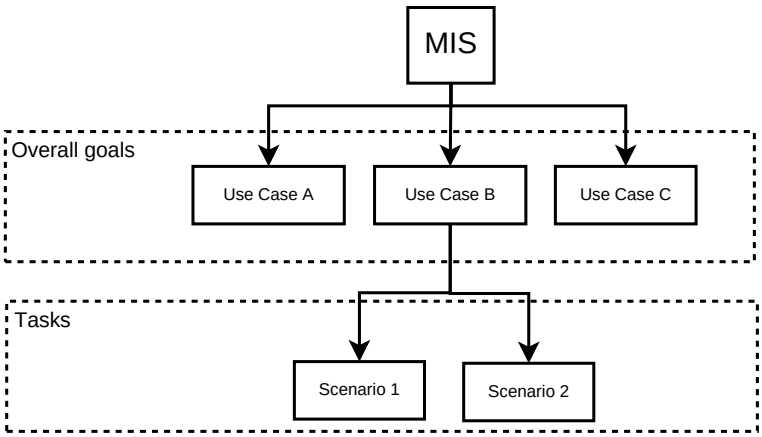


Figure 5.1: The goal/task hierarchy based on use cases and scenarios

A choice was made, to first and foremost focus on the MIS-related use cases that were felt to be important parts of the MIS, both in terms of the actual use of the functionality they represent, and in terms of how much information about the use case was provided. Some of them were rejected, because they described the task in a very abstract way; on a ”high level”, and this made the decomposition of them much harder, since it was difficult to recognize sub-tasks.

The use cases were examined one by one, one scenario at a time. This was done using the depth-first style, described in Section 4.2.2. The hierarchy of tasks in the decomposition was represented with the indented tabular kind-of style, seen in Table 4.1. This was done in a regular word processor application.

The task decomposition of a use case was done more than once. The resulting task decompositions differed from each other every time. None of the decompositions was felt to be ”correct” and ”complete”; none of them covered or included all tasks. It was felt that the decompositions were insufficient. It was hard to pin-point the reason for this feeling of incompleteness in the decompositions. Although the apparent shortcoming in the decomposition, several use cases were decomposed, and the feeling of incompleteness was found out to be not just a coincidence.

The assumption made, that the scenarios in the use cases described the task involved, was probably one of the reasons that the feeling of incompleteness arose. Instead of having that assumption, to view the scenarios as separate units to be explored one at a time, it was decided that the tasks involved in achieving the goal should be derived from examining *all* scenarios of the use case. Using this new approach, different opinions on what sub-tasks the decompositions should result in came up; more room for different interpretations appeared, but doing it this way, it was felt that the decompositions were less incomplete.

The decompositions were done until the tasks could be characterized as IHOs. This turned out to be very problematic. The reason for this was that the elements in the *first* decomposition, and sometimes even *the very name of the use case*, could be characterized as IHOs. This had the consequence that it was difficult to determine whether or not the decomposition had gone too far, even though only *one* step, or even *no step at all* had been taken in the decomposition.

5.3.3 Classifying Sub-tasks as SGTs

Because the development of the MIS had already gone on for a quite long time, and the MIS is influenced by a prototype that has been used and tested by the users that will use the MIS as well, it was assumed that decisions about how to perform the IHOs already had been made, i.e. the strategic decomposition had already in a sense been made and it should be possible to trace the tasks from the decomposition in the scenarios. Performing the negotiations and discussions with users to determine the strategies for performing the IHOs was not realistic in the time available for this work, which also was a factor in making the assumption. When an IHO had been found, the tasks derived from the strategic decomposition were to be found within the scenarios. These tasks were labeled as receive, evaluate or act. At this level in the task decomposition, the tasks were redescribed using the SGT-notion. By doing this, the information requirements for the corresponding tasks could be found in the SGT-scheme. The decision about what SGT-task element to use for description of the bottom-level task were also widely discussed. This was not so unambiguous as it seemed to be. But in order to go on with the case study, a choice was made and an interpretation was decided to be the correct one.

5.3.4 Combining User-centered Knowledge and System Knowledge

Having gathered relevant background information, carried out the task decompositions and generated the information requirements, the foundation for a design space analysis was built.

5.3.5 Brainstorming and Discussions Concerning the Design

A brainstorming kind of activity of coming up with design ideas and different design solutions started. At first, the hardware issues were discussed. Proposals and design alternatives were debated, pros and cons of each proposal were analyzed and deliberations of their suitability were held.

When concentrating on each part of the system, questions concerning the design were generated. Often times, they led to more questions about other parts of the system. Answers to the questions were debated, i.e. different design alternatives and options on how to design the system in order to cope with the design issues addressed by the questions were compiled. The options were compared to each other, and judged for suitability in the resulting design of the artefact, by making assessments on how well they met with different criteria.

To formulate the criteria, the functional requirements (the user centered information requirements in the framework) were first used. This was not so successful, because the functional requirements were generated by elements in the task decomposition that were felt to be on the ”wrong level”; they did not seem to handle the issues at hand in a level appropriate for design issues. They were concerning issues that were not of design specific concerns; they were in fact neutral to implementation. Therefore, the use of the functional requirements as ways of making design decisions was not useful in this case study.

When it became obvious that the functional requirements were not applicable for generating criteria, other criteria was generated. The new set of criteria were created by discussing what was felt to be important aspects of the specific design issues. This new set of criteria was clearly influenced by the options that were included, in the sense that the options that were in one way or another preferable had the most impact and influence on the criteria, so that criteria that supported the desired option were created. Criteria that caused negative support were also included,

but they were often created and included because negative supporting links from options to criteria had to be there, according to the framework and the design space analysis. The generation of criteria turned to be not very simple, and often times felt a bit forced.

5.3.6 Creating the Design Space Representation

The design space representations were first created using pen and paper only. After a few iterations, the representations got messy. The decision was made to use a computer application called DREAM to facilitate in the generation of the design space. (See Chapter 5.4 for more details about DREAM).

5.4 Generating a Design Space Using DREAM

A very useful tool for creating a QOC-representation is the DREAM -application [18]. DREAM is an acronym for Design Rationale Environment for Argumentation and Modeling. Figure 5.2 shows an overview of the application. DREAM is implemented in Java, and is therefore compatible with most operating systems. Using DREAM is a fairly straight forward process, since the program is not very complicated and has limited functionality.

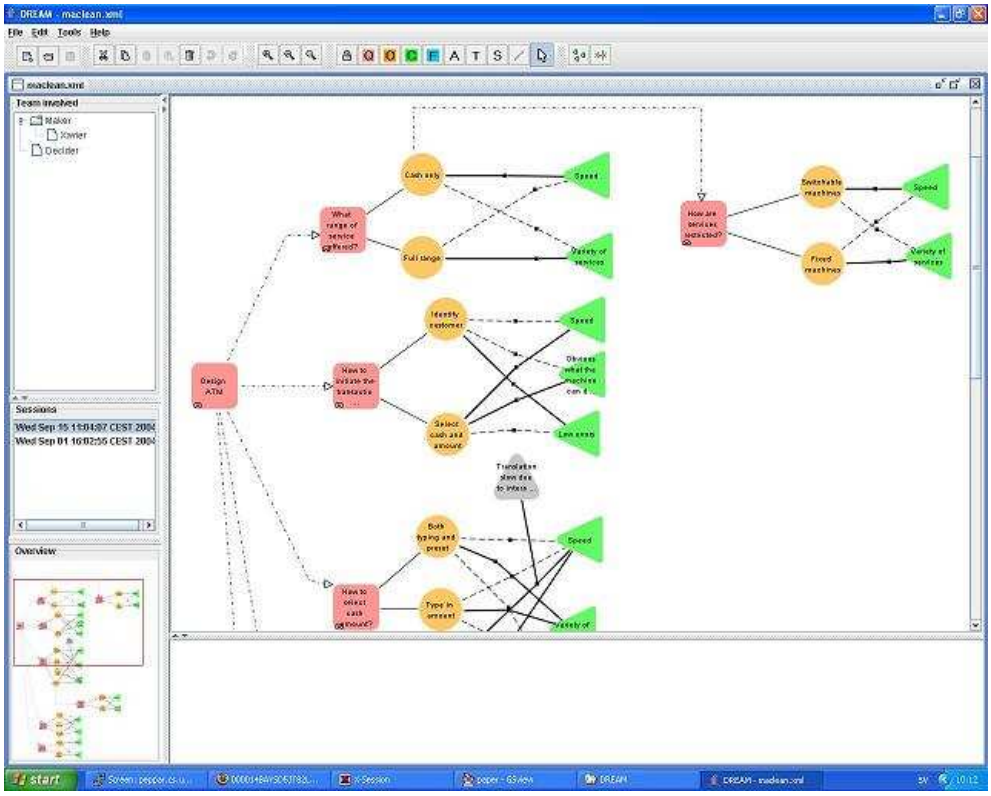


Figure 5.2: Screenshot of the DREAM application

There are buttons for Questions, Options and Criteria. Additionally, there are also buttons for Factors, Arguments, Task Models and Scenarios. The Factors are connected to Criteria as a description of how to assess an Option as being favorable. See the example in Figure 5.3 below, in which the factor states that when assessing the Criteria of cost, the more favorable a lower cost is (i.e. the cheapest is best).

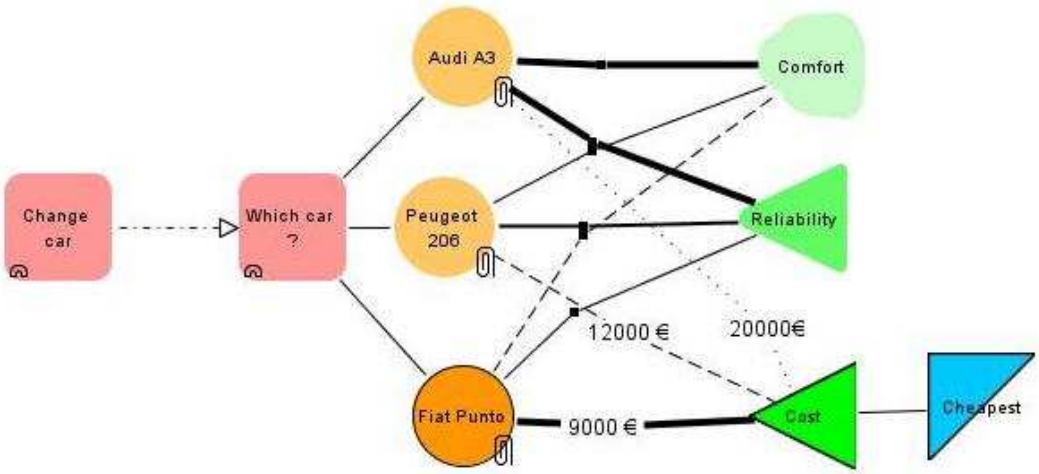


Figure 5.3: The use of factors in DREAM.

Arguments (the grey triangle in Figure 5.4) are used to explain why a certain Option is (or is not) supported by a certain Criteria. This feature is useful when it is not clear why assessments between Criteria and Options are the way they are.

DREAM offers the possibility to give Criteria weights. In Figure 5.3 the Criteria ”Comfort”, ”Reliability” and ”Cost” are given the weights Optional, Neutral and Important respectively. The Criteria that are optional are represented by a cloudlike shape and the most important Criteria has an triangle with a sharp outline.

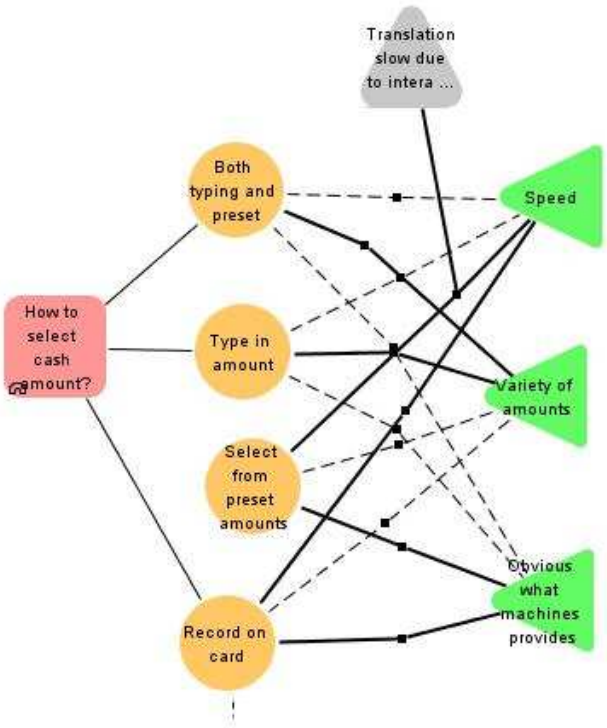


Figure 5.4: The use of arguments in DREAM.

The support that a Options gives to a Criteria is not only represented in a binary supporting-or-not fashion. When making assessments, six levels of support are available, giving a more dynamic and natural process of assessing the Criteria:

- Strongly Denied
- Denied
- Neutral
- Supported
- Strongly Supported

Even though DREAM offers an extended variety of functions than described in previous chapters, it is by no means intended that all of the extra notations have to be used. The standard Question-Option-Criteria is sufficient for most purposes.

When a design project is opened in DREAM, the program offers the possibility to make it a ”new session”, meaning that previous design work will be saved as earlier version, giving a chronological flow of the design process, making is easy to see what revisions are made compared to earlier versions of the design space. This results in a traceable design process, having the design rationale recorded in dated and time stamped versions.

In DREAM, the projects are stored in XML-format, which makes it easy to export them into other applications, since the XML-standard is a well-know format that has a variety of areas of use.

5.5 Results and Observations

Much of the information that was gathered and created the basis of knowledge about the system was associated to the prototype WCS. Because MIS was considered a non-existing system and not an improvement of WCS, this could lead to problems of neutrality towards implementation.

The only source of information, used in the case-study, that described the functional parts of MIS in a neutral way by focusing on goals within MIS and how these goals could be achieved, was the descriptions of the different possible scenarios that were stated in the MIS-related use cases used in the FAD-model. The overall goals in MIS were difficult to discover, and the scenarios were not that neutral in terms of implementation as hoped. The reason for that is that the use cases were in fact not as goal driven as they should have been and influences from the WCS could also be traced in the use cases and the scenarios.

It was felt that, step by step, as the case study progressed and more information was gathered, more knowledge and understanding of the different aspects of MIS was gained. But it was still difficult to grasp, define and fully understand the overall goals and connections between the tasks within the system.

By using the proposed framework in the phase of development where MIS was, i.e. when there is a prototype that is used for testing and learning the system in its natural domain and context and a lot of functionality and implementation have already been undertaken, had both positive and negative aspects. The positive part is that it makes it is easier to gather information about a system and its parts and its functionality. It is easy to establish a wide range basis of information in correspondence to a system which have not yet been developed at all. But, as said earlier, this information is colored by the prototype and can be ”too” detailed which leads to many limitations regarding how the system can be designed.

The task decomposition became a process with a lot of discussion and negotiation about how the goals and tasks should be decomposed. The assumption that the use case specific scenarios could be directly associated with the tasks that must be performed to achieve to goal of the use case was discarded. The reason for this was that by using each scenario as a main task, a lot of information was lost. By examining all use case specific scenarios together resulted in a clearer description of the main tasks to achieve the use case goal, but it also lead to an increased amount of interpretations and decision making which made the procedure more complex. If a description of a task within a scenario was directly associated with another use case, this was shown with a reference to that use case. By doing it in this fashion, unnecessary redundancy could be reduced.

The recognition that a task/sub-task was an IHO was made at a very early step in the decomposition, which lead to a shallow hierarchical structure of tasks and sub-tasks. Because the IHOs often were found in the first step of the decomposition, it was difficult to know if this really was the first occurrence of an IHO when performing the main tasks of MIS. In other words, had the

point where the neutrality towards the implementation and interaction technique first is challenged already been passed? It seemed like, by using the use cases as descriptions of the main goals for MIS, the decomposition was started levels too low in the task hierarchy, and information about tasks/goal of the higher levels is lost. Because no back-tracking in the hierarchy was possible, the recognition of the IHO had to be done at this level.

The assumption made about the strategic decomposition, because of the progression of the development of MIS and the tasks within, was necessary in order to conduct this case study in the short time available. If time had not been a factor, this strategic decomposition of IHOs should be done by negotiating with the users, trying to determine how the operations receive, evaluate and act should be performed.

By studying the tasks classified as IHOs, the tasks derived from the strategic decomposition were tried to be found. It was not always easy to find these information related sub-tasks just by studying the scenario specifications. A reason for this could be the fact that the IHO is recognized too low in the task hierarchy, which made it hard to realize what strategy has been used.

In the final phase of the task analysis, when the decomposition of tasks has reached the stopping point and the redescription of the bottom-level tasks using the SGT-notation was conducted, more problems were encountered. The redescription was not as straight forward as first believed. To decide which SGT-task element to use was not easy, because of the ambiguities in the SGT-scheme. Depending on the interpretation, different decisions were made. One common problem was when to use the task elements A1: Activate and E1: Enter; these were often mixed-up.

The plans in the decomposition that state in which order the tasks must be performed in order to fulfill the goal in this level, was described using the SGT-sequencing notation. This work was relatively straight forward because of the shallow structure of the task hierarchy.

The resulting documents from the Task Analysis phase (using the SGT-method) can be viewed as a specification of information that MIS users need in order to carry out the tasks derived from the use cases scenarios. The choice to separate the decompositions of the use cases so a clearer view of what their task hierarchy was made .

When studying the documents containing the specifications of the information requirements and the final task decomposition, it was discovered that *Navigate* and *Monitor* sub-goals templates were seldomly used. Even though these sub-goals were added with the purpose of making it possible to analyze any kind of interactive systems and not only systems of supervision and process control.

Another reflection made was that in the description of the Navigate sub-goal it is stated that a navigation is (see Table 3.1):

> *To move to an informational state for exchange, action or monitoring.*

But if all these sub-goal should be preceded by a Navigation, the description would be ”overly” detailed and the information requirements associated with the navigation tasks would be redundant. Thomas Ormerod, one of the founders of the SGT-method, was contacted and this reflection was brought to his attention. He agreed with it having some ambiguities in his response addressing the issue:

> ”*I agree that there is an issue with unnecessary burden. In some respects, we might have been better of calling it search (i.e., where the location is in some sense unknown or hidden) rather than navigate. In practice, the analyst makes a judgement about which elements are needed for design consideration and which are not, so you are right to drop unnecessary detail.*”

Having the support in this matter from a prominent researcher in this area, the elements that were felt to provide only with unnecessary information were excluded.

The design space analysis was intended to cover the whole of MIS, but when trying to create the design space for the whole system, it turned out to be difficult to get down to a level where it felt comfortable to work with design specific issues. ”MIS” was the root node, and to it, a number of Questions were related. At this level, being quite high and abstract, it felt appropriate to state Questions such as ”What kind of display will be used”, having Options like ”LCD”, ”Touch screen”. This was clearly a level too high up in abstraction compared to the level where the user specific information requirements that the SGT-method produced can be applied. Instead of trying to generate an overall design space, more local design spaces were produced. These were created using use cases as grounds for the new design spaces, that were expected to be more manageable in both size and in ”level of abstraction”.

When generating the design spaces, the creation of good Questions was not easy. They tended to become formulated in styles of, for example, ”How to manage Fire Missions”. This made the generation of Options difficult, because by having Questions stated in that way, the corresponding Options will be answers that state how to perform tasks (which is not even allowed, according to the method), and do not present design alternatives. The Questions were eventually formulated as ”How to represent Fire Missions”, which is a more design related Question, than the previously formulated Questions. These Questions, when formulated in this manner, helped spawn the different Options concerning design specific alternatives.

Options that seemed unrealistic to implement, or use in a system (in this, specifically) were often left out of the design space, although this is a very direct way a violation of the essence of the design space analysis, which is to include as much information as possible, to create good design rationale. A probable reason for not thinking of and including outrageous, ”think out side the box”-kind of design options was that the work with MIS was so much closely coupled to the WCS and influenced by it (and the grounds for the design as far as deciding what kind of hardware to use, etc were already established), so they were discarded more or less unconsciously. The design spaces produced became quite uncontroversial, and quite similar to the WCS and its design solutions.

The Criteria that were formulated were also influenced by the Options that seemed favorable for the final design. This led to a biased set of Criteria, expressing the views of what a good design should be according to those who stated the Criteria relating to Options. Criteria that led to certain Options getting negative assessments were included in the design space. But often times, due to the fact that negative assessments must exist, and not so much because the Options

themselves had this consequence or that the Criteria simply happened to result in some negative assessments.

The user centered information requirements being results from the SGT-method were of little or no help when formulating Criteria. Of course, they could have been useful in the sense that they stated (for example) what information a user needs to have when performing tasks in the system. Still, this information, and the Criteria, felt like they were ”far apart” and hard to relate to each other to get appropriate Criteria. Generating and formulating the Criteria turned out to be more difficult than expected.

The design space analysis, and its design spaces, did not end up very detailed or fully explored, much due to lack of time and because the gathered information and the system knowledge that was gained was simply not enough, not to mention the fact that this was done by people who are not designers.

5.6 Summary of Results

The main problem in the use of the proposed framework in the case study of MIS is the problems associated with decompositions of the goals and tasks related to MIS. As stated in the last section, the level where a task is recognized as an IHO was often made in the first step of the task decomposition and in some cases the use case itself could be recognized as an IHO, and a feeling that this perhaps was not the first time that the neutrality in terms of implementation was challenged. It was sensed almost as if these IHOs were derived from a strategic decomposition already made. It was difficult to see if the recognized IHO was the ”first”, see Figure 5.5.

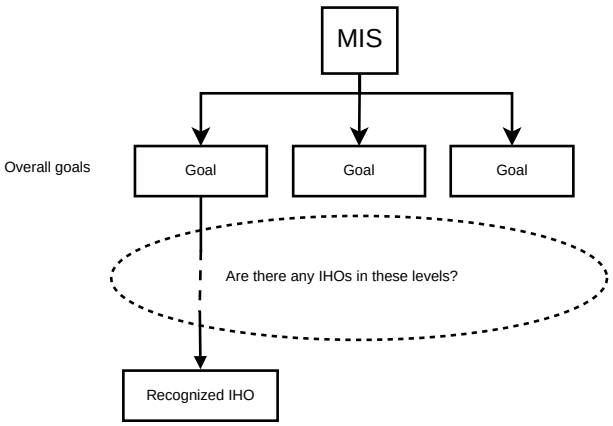


Figure 5.5: Representation of the main problem.

Only using the information in scenarios in MIS-related use cases, together with other attained knowledge of the system, was not enough to be able to discover the overall goals in MIS. The names of these use cases could *not* be used to describe these overall goals.

Because the problem arises in an early stage in the work with the framework, it will propagate throughout the case-study and will have consequences in other parts of the framework, namely:

- Tasks derived from the strategic decomposition of the recognized IHO were difficult to trace in the scenarios.
- Tasks at the bottom-level sometimes generated low quality information requirements.
- It was difficult to use the information requirements to create design space.

This also lead to the possibility that other potential problems with using the framework were overshadowed and became more difficult to detect. It also made it harder to evaluate the actual analysis methods, but the purpose of the case-study was to evaluate these methods in the framework and not the methods themselves.

Some other problems were also discovered. Using the SGT-scheme to redescribe tasks in the lowest level of the task hierarchy felt to be a bit ambiguous. It was hard to find out if it was the SGT-scheme itself that was the reason for this or if the problem arose because of other reasons.

When the design space analysis in the case study was performed, it was felt that neither the knowledge about the system and its intended functionality, nor the time available for performing the analysis, was enough. Creating a ”complete” design space is not done with the greatest of ease. It is a work that requires hard work, both in the phases prior to the actual creation of the design space, and during it to include novel, although thoroughly examined and discussed design proposals that address relevant design aspects and Criteria that handles and cover the requirements in an adequate way.

Although the problems encountered using Design Space Analysis in the case study, observations were made indicating that DSA is a good method for recording design rationale.

- The recording design rationale is easily done using DSA, especially using an application such as DREAM.
- The decision making over time can be recorded in the representation and can be traced. When using DREAM, a new session can be started each time a design space is loaded, making the revisions and refinements obvious when comparing the different versions of the design space.
- Being a semi-formal notation, and the fact that the design space representation consists of very few elements, makes it easy for almost anyone to use the QOC-representation in design deliberations. However, to ensure a practical and feasible design, a designer should be involved.
- The design options chosen and those rejected are explicitly expressed in the representation of the design space.

The main question concerning the problems encountered is: What could be the reason for the IHO being recognized so early in the decomposition phase and why could not the overall goals be found within the specifications of the scenarios in the MIS-related use cases?

A possible answer to the question is that MIS is influenced by the prototype WCS. The work with the MIS related use cases was performed when a prototype already had been developed. Though these use cases should describe a goal and the steps of actions to reach this goal in a

neutral way, this description is easily influenced when there is a prototype available, and this is probably the case with MIS. The tasks that were derived from studying the scenarios could then be at a level where some assumptions of implementation already had been made and this does not suit the methods of the proposed framework.

Therefore, the use cases should be goal driven, and not created based on a prototype and its implementation. This prevents any unwanted influences from the prototype which could place constraints on the design. Instead, focus can be put on describing how to reach goals in a neutral way in terms of implementation. If the use cases were created before any prototyping was performed, this would not be an issue.

Chapter 6

Accomplishments and Conclusions

One major difficulty in the work in this paper, was investigating Task Analysis, being a large area of many different methods. It is difficult to determine which method is suitable for a specific domain or context if not every method is examined to discover their strengths and weaknesses, which is absurd to even suggest.

The accomplishments and conclusions in this paper can be summarized with these points:

- A framework based on two analysis methods was created, that addresses HCI-related issues in design processes.
- This framework has been evaluated in a case study. This was done by applying the framework to a system in the intended domain and context.
  - ◇ The evaluation showed that there were problems with getting the framework to fit in to the system development process involved in developing the MIS.
  - ◇ Some of the reasons for the problems have been recognized, and solutions for them have been proposed.

Evaluating a framework like the one in the paper, is not an easy process. The suitability and applicability in its intended context and domain for a framework is determined by many different factors.

Overall, it is difficult to view problems as results. When problems arise and setbacks occur, they are easily considered to be signs of the work having failed. In fact, the problems themselves, can be viewed as results worth investigating and mentioning.

The framework as such is, in an ideal case, a good framework that captures and addresses HCI-aspects of a system development process. It focuses on information requirements that a user needs in order to operate/handle the system being developed. The framework also gives the developers deeper understanding and knowledge about the system and its intended purposes. The deeper knowledge and understanding should lead to a better design in the end, both concerning usability and functionality.

Chapter 7

Future Work

Interesting future work would be to more thoroughly investigate the reasons for the problems; why some of the elements in the framework did not work as expected, what the causes were for the ambiguities in the SGT-scheme.

A future study would be to test and examine the framework, with all its pros and cons, by doing this on a non-existing system, having no prototypes, without assumptions made of its functionality, and having no decisions taken about any implementations. This to ensure that no prior versions or planned and decided functionality can influence the system being developed.

It would be interesting to investigate improved tool support for the proposed framework, and the representations in it. For example, having a database that stores the elements in the design space analysis (the Questions, Options and Criteria), and their relations and the assessments, could be a useful tool when working with larger projects. This database could be used for storing the (large) design space, and the relations would be easy to trace.

Chapter 8

Acknowledgments

Here we thank those who have helped us in any way to write this paper, in no particular order.

Magnus Eriksson
Our supervisor at Umeå University. You've provided with great help. We should have used it a lot more.

Daniel Granholm
Our travel companion. Visiting Cloetta is not the same without you.

Anders "Albertville" Mannelqvist
Our supervisor at Land Systems Hägglunds, for being so helpful and kind in so many ways. Keep on running!

Karin Fossum.
You were great help. Fore!

Thomas Ormerod
For helping us with the SGT-method that he founded.

All the end users representatives. Your opinions and comments were invaluable.

... and everyone else who supported and helped us in any way.

References

[1] S. Adolph, P. Bramble, A. Cockburn, and A. Pols. *Patterns for Effective Use Cases*. Addison-Wesley, 2002.

[2] J. Annett. Theoretichal and Pragmatic Influences on Task Analysis Methods. *Cognitive Task Analysis*, 2000.

[3] J. Annett. Hierarchical Task Analysis. *The Handbook of Task Analysis for Human-Computer Interaction*, 2004.

[4] J. Annett and K. D. Duncan. Task Analysis and training design. *Occupational Psychology*, 1967.

[5] P. Barnard and J. May. Cognitive Task Analysis (CTA). http://www.vuw.ac.nz/acis99/articles/articleCalway-137.pdf (visited 2005-03-03), 2000.

[6] V. Bellotti and A. MacLean. Design Space Analysis (DSA). http://www.mrc-cbu.cam.ac.uk/amodeus/summaries/DSAsummary.html (visited 2005-03-03).

[7] S. Card, T. Moran, and A. Newell. The keystroke-level model for user performance time with interactive systems. *Communications of the ACM*, 1980.

[8] S. Card, T. Moran, and A. Newell. *The psychology of human-computer interaction*. Lawrence Erlbaum Associates, 1983.

[9] S. Chipman, J. Schraagen, and V. Shalin. Introduction to Cognitive Task Analysis. *Cognitive Task Analysis*, 2000.

[10] D. Diaper and N. A. Stanton. *The Handbook of Task Analysis for Human-Computer Interaction*. Lawrence Earlbaum Associates, 2004.

[11] P. M. Fitts and M. I. Posener. *Human Performance*. Brooks/Cole, 1967.

[12] R. Gillespie. *Manufacturing knowledge : a history of the Hawthorne experiments*. Cambridge University Press, 1991.

[13] D. K. Goldstein. The impact of development aids on the systems development process. *Conference on Human Factors in Computing Systems*, 1982.

[14] W. D. Gray, B. E. John, and M. E. Atwood. Project Ernestine: A validation of GOMS for prediction and explanation of real-world task performance. *Human-Computer Interaction*, 1993.

[15] D. Kieras. GOMS Models for Task Analysis. *The Handbook of Task Analysis for Human-Computer Interaction*, 2004.

[16] S. B. Shum L. Nigay, D. Salber and J. Coutaz. Teaching Trainee and Professional Designers to use the PAC-AMODEUS Software Architecture Modelling Technique. *http://kmi.open.ac.uk/people/sbs/docs/SBS-TA-WP42.pdf*, 1995.

[17] S. Lauesen. *Software Requirements - Styles and Techniques*. Addison-Wesley, 2002.

[18] LIIHS. The DREAM application. http://liihs.irit.fr/dream/.

[19] A. MacLean, V. Bellotti, and S. Shum. Developing the Design Space with Design Space Analysis. *Computers, Communication and Usability: Design issues, research and methods for integrated services.*, 1993.

[20] A. MacLean, R. M. Young, V. M. E. Bellotti, and T. P. Moran. Question, Options, and Critera: Elements of Design Space Analysis. *Human-Computer Interaction*, vol 6, 1991.

[21] A. MacLean, R. M. Young, and T. P. Moran. Design Rationale: The argument behind the artifact. http://portal.acm.org/ft_gateway.cfm?id=67497&type=pdf, 1989.

[22] E. Mayo. *The human problems of an industrial civilization*. MacMillan, 1933.

[23] T. Ormerod. Using Task Analysis as a Primary Design Method: SGT-approach. *Cognitive Task Analysis*, 2000.

[24] T. Ormerod and A. Shepherd. Using Task Analysis for Information Requirements Specification: The Sub-Goal Template (SGT) Method. *The Handbook of Task Analysis for Human-Computer Interaction*, 2004.

[25] F.J. Roethlisberger and W.J. Dickson. *Management and the Worker*. Harvard University Press, 1939.

[26] J. M. Schraagen, S. F. Chipman, and V. L. Shalin. *Cognitive Task Analysis*. Lawrence Earlbaum Associates, 2000.

[27] A. Shepherd. Task Analysis as a framework for examining HCI tasks. *Perspectives on HCI: Diverse Approaches*, 1995.

[28] S. J. Shum. A Cognitive Analysis of Design Rationale Representation. Technical report, Department of Psychology, University of York, 1991.

[29] I. Sommerville. *Software Engineering*. Addison-Wesley, 2000.

[30] Usability First. Usability Glossary. http://www.usabilityfirst.com/.

[31] M van Welie. Task-based User Interface Design. Technical report, University of Vrije, Amsterdam, 2001.